Corso di Algoritmi e Strutture Dati

ESERCIZI E COMPLEMENTI

Ottobre 2007

Corso di laurea triennale di Informatica
Università degli Studi di Milano

Alberto Bertoni
Massimiliano Goldwurm

Indice

0.1 Informazioni generali 2
0.1.1 Programma del corso (I e II parte) 2
0.1.2 Dispense del corso 3
0.1.3 Testi adottati o di riferimento 3

1 Esercizi e problemi di base **4**
1.1 Notazioni asintotiche 4
1.2 Stima di somme 4

2 Funzioni generatrici **7**

3 Equazioni di ricorrenza **11**

4 Criteri di costo uniforme e logaritmico **15**
4.1 Analisi di procedure AG 15
4.2 Esercizi 19

5 Procedure ricorsive **22**

6 Algoritmi su alberi e grafi **27**

7 Il metodo "divide et impera" **34**

8 Algoritmi greedy **42**

9 Algoritmi particolari **45**

A Temi d'esame svolti **48**

0.1 Informazioni generali

Al sito http://homes.dsi.unimi.it/∼goldwurm/algo/ si trova una pagina web del Corso e del Laboratorio di Algoritmi e Strutture Dati, dove si possono trovare tutte le informazioni e il materiale didattico messo a disposizione dai docenti: le modalità d'esame, le date degli appelli, i risultati dell'ultima prova scritta, gli orari dei corsi e di ricevimento degli studenti, i testi dei progetti e dei temi d'esame degli appelli già svolti.

0.1.1 Programma del corso (I e II parte)

INTRODUZIONE
Nozione intuitiva di problema e algoritmo. I principali tipi di problemi. La fase di progetto di un algoritmo e quella di analisi. La complessità di un algoritmo. La classificazione dei problemi.

MODELLO DI CALCOLO
Macchina ad accesso casuale (RAM). Risorse in spazio e tempo. Criteri di costo uniforme e logaritmico. Analisi nel caso peggiore e in quello medio. Schema di linguaggio ad alto livello (pseudocodice) e sua esecuzione sul modello RAM.

NOZIONI MATEMATICHE
Notazioni asintotiche : le relazioni o grande, o piccolo, ugual ordine di grandezza, asintotico, omega grande.

Strutture combinatorie elementari e relativa enumerazione: permutazioni, disposizioni, combinazioni. Relazioni d'ordine e di equivalenza. Algebre eterogenee e omomorfismi.

Valutazione di somme e di serie. Sommatorie fondamentali. Equazioni di ricorrenza. Numeri di Fibonacci. Funzioni generatrici ordinarie. Teorema generale per la soluzione di equazioni di ricorrenza del tipo divide et impera.

STRUTTURE DATI ELEMENTARI
Strutture elementari: vettori, liste, pile, code e relative operazioni fondamentali. Esecuzione iterativa delle chiamate ricorsive: record di attivazione delle chiamate, loro gestione mediante una pila e analisi dello spazio di memoria utilizzato.

Grafi diretti e indiretti, grafi non etichettati, alberi, alberi ordinati, alberi binari. Metodi elementari di enumerazione. Enumerazione di alberi binari. Rappresentazione di alberi e grafi.

Procedure di esplorazione di alberi: attraversamento in ordine anticipato, posticipato e simmetrico. Algoritmi di attraversamento di grafi in profondità e in ampiezza e relativo albero di copertura. Algoritmo di calcolo delle distanze dei nodi da una sorgente.

ALGORITMI DI ORDINAMENTO
Generalità sul problema dell'ordinamento. Ordinamento interno per confronti: numero minimo di confronti necessari per ordinare n elementi. L'algoritmo di ordinamento mediante inserimento. La struttura di heap e il relativo algoritmo di costruzione. L'algoritmo Heapsort. L'algoritmo Quicksort. Implementazione iterativa di Quicksort e ottimizzazione della memoria.

ALGORITMI E STRUTTURE DATI PER LA MANIPOLAZIONE DI INSIEMI
Strutture di dati astratti e implementazione corretta. Le strutture elementari come strutture di dati astratti. Operazioni fondamentali sugli insiemi e le strutture dati associate. Esecuzioni on-line e off-line.

Tabelle hash. Ricerca binaria. Alberi di ricerca binaria. Alberi bilanciati: alberi 2-3 e B-alberi.

Operazioni union-find su partizioni: algoritmi basati su liste; algoritmi basati su alberi con bilanciamento e compressione.

TECNICHE DI PROGETTO
Schema generale degli algoritmi "Divide et impera" e relativa analisi dei tempi di calcolo. Algoritmo "Divide et impera" per il prodotto di interi. Algoritmo Mergesort. L'algoritmo di Strassen per il prodotto di matrici.

Programmazione dinamica: chiusura transitiva di un grafo; calcolo delle lunghezze minime di cammino; costo minimo del prodotto di n matrici.

Tecnica greedy: sistemi di indipendenza, matroidi e teorema di Rado; l'algoritmo di Kruskal; gli algoritmi di Prim e Dijkstra; i codici di Huffman.

CLASSIFICAZIONE DI PROBLEMI
Le classi P e NP. Riduzione polinomiale molti-uno. I problemi NP-completi. Il problema della soddisfacibilità e il teorema di Cook.

0.1.2 Dispense del corso

A. Bertoni, M. Goldwurm, *Progetto e analisi di algoritmi*, Rapporto interno n. 230-98, Dip. Scienze dell'Informazione, Università degli Studi di Milano, Settembre 2007.

0.1.3 Testi adottati o di riferimento

T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, *Introduzione agli algoritmi e strutture dati*, Seconda edizione, McGraw-Hill, 2005.

A.V. Aho, J.E. Hopcroft, J.D. Ullman, *The design and analysis of computer algorithms*, Addison-Wesley, 1974.

A. Bertossi, *Algoritmi e strutture di dati*, UTET Libreria, 2000.

C. Demetrescu, I. Finocchi, G.F. Italiano, *Algoritmi e strutture dati*, McGraw-Hill, 2004.

R. Sedgewick, P. Flajolet, *An introduction to the analysis of algorithms*, Addison-Wesley, 1996.

Capitolo 1

Esercizi e problemi di base

1.1 Notazioni asintotiche

Dato l'insieme delle funzioni $\{f \mid f : \mathbb{N} \longrightarrow \mathbb{R}^+\}$, si ricordano le relazioni binarie O, Ω, Θ, $\sim$ e o:

$$f(n) = O(g(n)) \quad \text{se e solo se} \quad \exists C > 0, \exists n_0 \in \mathbb{N} : \quad f(n) \leq Cg(n) \quad \forall n \geq n_0$$

$$f(n) = \Omega(g(n)) \quad \text{se e solo se} \quad \exists C > 0, \exists n_0 \in \mathbb{N} : \quad f(n) \geq Cg(n) \quad \forall n \geq n_0$$

$$f(n) = \Theta(g(n)) \quad \text{se e solo se} \quad \exists C_1, C_2 > 0,\ \exists n_0 \in \mathbb{N} : \quad C_1 g(n) \leq f(n) \leq C_2 g(n) \quad \forall n \geq n_0$$

$$f(n) \sim g(n) \quad \text{se e solo se} \quad \lim_{n \to +\infty} \frac{f(n)}{g(n)} = 1$$

$$f(n) = o(g(n)) \quad \text{se e solo se} \quad \lim_{n \to +\infty} \frac{f(n)}{g(n)} = 0$$

Formula di Stirling: $n! = n^n e^{-n} \sqrt{2\pi n} \left(1 + O(\frac{1}{n})\right)$

1.2 Stima di somme

Problema 1.1 *Data la successione* $\{f(n)\}$, *calcolare la successione*

$$S(n) = \sum_{k=0}^{n} f(k) = f(0) + f(1) + \cdots + f(n)$$

`Esempi fondamentali`:

1. somma geometrica $\displaystyle\sum_{k=0}^{n} x^k = \begin{cases} n+1 & \text{se } x = 1 \\ \frac{x^{n+1}}{x-1} & \text{se } x \neq 1 \end{cases}$

2. somma binomiale $\sum_{k=0}^{n} \binom{n}{k} x^k a^k b^{n-k} = (ax+b)^n$

3. valutazione per derivazione

$$\text{se} \quad \sum_{k=0}^{n} f(k)x^k = F(x) \quad \text{allora} \quad \sum_{k=0}^{n} kf(k)x^k = x\frac{d}{dx}F(x)$$

`Avvertenza`: nel seguito quando utilizzeremo derivate o integrali, supporremo sempre soddisfatte le relative condizioni di derivabilità o integrabilità .

Esercizio 1.1

Dimostrare che

$$\sum_{k=0}^{n} 2^k = 2^{n+1} - 1, \qquad \sum_{k=0}^{n} k2^k = (n-1)2^{n+1} + 2, \qquad \sum_{k=0}^{n} \binom{n}{k} k2^k = 2n3^{n-1}$$

Problema 1.2 *Data la successione* $\{f(n)\}$*, fornire una stima asintotica della successione* $\{S(n)\}$ *dove*

$$S(n) = \sum_{k=0}^{n} f(k)$$

Nel caso di sequenze $\{f(n)\}$ monotone non decrescenti si verifica:

1. $S(n) = O(n \cdot f(n)$,
2. se $f(n) = o\left(\int_0^n f(x)dx\right)$ allora
$$S(n) \sim \int_0^n f(x)dx$$

Esercizio 1.2

Determiniamo la stima asintotica delle somme

$$\sum_{k=0}^{n} k^\alpha, \qquad \sum_{k=1}^{n} \log k$$

Svolgimento.

$\int_0^n x^\alpha dx = \frac{1}{\alpha+1} \cdot n^{\alpha+1}$; quindi, poiché $n^\alpha = o(n^{\alpha+1})$, segue

$$\sum_{k=0}^{n} k^\alpha \sim \frac{1}{\alpha+1} \cdot n^{\alpha+1}$$

$\int_1^n \log x dx = n\log n - n + 1$; quindi, poiché $\log n = o(n \log n)$, segue

$$\sum_{k=1}^{n} \log k \sim n \log n$$

∎

Esercizio 1.3

Calcolare in modo esatto le seguenti somme:

$$\sum_{k=0}^{n} k, \qquad \sum_{k=0}^{n} k^2, \qquad \sum_{k=0}^{n} k3^k, \qquad \sum_{k=0}^{n} \binom{n}{k} k^2$$

Esercizio 1.4

Fornire una stima asintotica delle seguenti somme:

$$\sum_{k=0}^{n} k^{\frac{1}{2}}, \qquad \sum_{k=1}^{n} k \log k, \qquad \sum_{k=1}^{n} \log^2 k, \qquad \sum_{k=1}^{n} \frac{1}{k}$$

Capitolo 2

Funzioni generatrici

La funzione generatrice di una sequenza $\{a_n\}$ è la funzione

$$A(x) = \sum_{k=0}^{+\infty} a_k x^k$$

La funzione $A(x)$ è quindi definita dalla serie di potenze con centro in 0 i cui coefficienti coincidono con gli elementi della sequenza $\{a_n\}$. Nel seguito per indicare che $A(x)$ è la funzione generatrice di $\{a_n\}$ useremo l'espressione

$$a_n \longleftrightarrow A(x)$$

Problema 2.1 *Data la sequenza $\{a_n\}$ calcolare la funzione generatrice $A(x)$.*

Gli esempi fondamentali sono i seguenti:

	Sequenze		Funzioni generatrici
1.	a^n	$\longleftrightarrow$	$\frac{1}{1-ax}$
2.	$\binom{\alpha}{n}$	$\longleftrightarrow$	$(1+x)^\alpha$
3.	$\frac{1}{n}$	$\longleftrightarrow$	$-\log(1-x)$

Esercizio 2.1

Determinare la funzione generatrice di $\{1\}$.

Supponiamo ora che $A(x)$ e $B(x)$ siano le funzioni generatrici di $\{a_n\}$ e $\{b_n\}$, e c sia una costante; allora la seguente tabella fornisce la corrispondenza tra operazioni sulle sequenze e operazioni sulle funzioni generatrici.

	Sequenze		Funzioni generatrici
1.	$a_n + b_n$	$\longleftrightarrow$	$A(x) + B(x)$
2.	$c \cdot a_n$	$\longleftrightarrow$	$c \cdot A(x)$
3.	$\sum_{k=0}^{n} a_k b_{n-k}$	$\longleftrightarrow$	$A(x) \cdot B(x)$
4.	$\sum_{k=0}^{n} a_k$	$\longleftrightarrow$	$\frac{A(x)}{1-x}$
5.	a_{sn}	$\longleftrightarrow$	$A(x^s)$
6.	$n \cdot a_n$	$\longleftrightarrow$	$x\frac{d}{dx}A(x)$
7.	a_{n+1}	$\longleftrightarrow$	$\frac{A(x) - a_0}{x}$
8.	a_{n+2}	$\longleftrightarrow$	$\frac{A(x) - a_0 - a_1 x}{x^2}$

Esercizio 2.2

Determinare le funzioni generatrici delle sequenze:

$$a_n = 2^n + (-1)^n; \quad b_n = n; \quad c_n = \sum_{k=0}^{n} 2^{n-k}k; \quad a_{n+1}; \quad b_{n+2}.$$

Svolgimento. Si verifica $2^n \longleftrightarrow \frac{1}{1-2x}$ mentre $(-1)^n \longleftrightarrow \frac{1}{1-(-x)}$.

Quindi, per la 1., abbiamo

$$a_n = 2^n + (-1)^n \longleftrightarrow \frac{1}{1-2x} + \frac{1}{1+x},$$

per la 7., $a_{n+1} \longleftrightarrow \left(\frac{1}{1-2x} + \frac{1}{1+x} - 2\right)\frac{1}{x} = \frac{1+2x}{(1-2x)(1+x)}$.

Inoltre, per la 6., abbiamo

$$b_n = n \cdot 1 \longleftrightarrow x\frac{d}{dx}\left(\frac{1}{1-x}\right) = \frac{x}{(1-x)^2},$$

mentre, per la 8., $b_{n+2} \quad \longleftrightarrow \left(\frac{x}{(1-x)^2} - 0 - 1x\right)\frac{1}{x^2} = \frac{2-x}{(1-x)^2}$.

Infine, poiché $n \longleftrightarrow \frac{x}{(1-x)^2}$ e $2^n \longleftrightarrow \frac{1}{1-2x}$, per la 3. otteniamo

$$\sum_{k=0}^{n} k2^{n-k} \quad \longleftrightarrow \quad \frac{x}{(1-x)^2} \cdot \frac{1}{1-2x}$$

∎

Esercizio 2.3

Trovare le funzioni generatrici delle sequenze

$$a_n = \frac{2^n}{n} \quad (n > 0), \qquad b_n = \sum_{k=1}^{n} \frac{2^k}{k}, \qquad c_n = \sum_{k=0}^{n} \binom{100}{k}$$

Problema 2.2 *Data la funzione generatrice $A(x)$, trovare la sequenza $\{a_n\}$.*

Nel caso di funzioni razionali $A(x) = \frac{P(x)}{Q(x)}$, con P e Q polinomi, si procede nel modo seguente:

1. si suppone che P e Q non abbiano radici in comune e che grado(P) < grado(Q), altrimenti si scrive $A(x)$ nella forma
$$A(x) = S(x) + \frac{R(x)}{Q(x)}$$
dove grado(R) < grado(Q), e si prosegue il ragionamento per la funzione $A(x) - S(x)$;

2. si determinano le radici $x_1, x_2, \ldots, x_m$ di Q ottenendo la decomposizione
$$Q(x) = Q(0)(1 - \alpha_1 x) \cdots (1 - \alpha_m x)$$
dove $\alpha_k = \frac{1}{x_k}$ per ogni k;

3. se tali radici sono distinte, si decompone $\frac{P(x)}{Q(x)}$ in frazioni parziali, ottenendo
$$\frac{P(x)}{Q(x)} = \frac{C_1}{1 - \alpha_1 x} + \frac{C_1}{1 - \alpha_1 x} + \cdots + \frac{C_m}{1 - \alpha_m x}$$
dove $C_1, C_2, \ldots, C_m$ sono costanti;

4. allora, per ogni intero $n \in \mathbb{N}$, abbiamo
$$a_n = C_1 \cdot \alpha_1^n + C_2 \alpha_2^n + \cdots + C_m \alpha_m^n$$

Esercizio 2.4

Determinare la successione $\{a_n\}$ associata alla funzione $\frac{3-4x}{1-3x+2x^2}$.

Svolgimento.

Le radici di $1-3x+2x^2$ sono $x_1 = 1/2$ e $x_2 = 1$. Allora abbiamo $1-3x+2x^2 = (1-x)(1-2x)$. Determiniamo due costanti A e B tali che

$$\frac{3 - 4x}{1 - 3x + 2x^2} = \frac{A}{1 - x} + \frac{B}{1 - 2x}$$

A tal fine poniamo

$$A = \lim_{x \to 1} \frac{3 - 4x}{1 - 2x} = 1$$
$$B = \lim_{x \to \frac{1}{2}} \frac{3 - 4x}{1 - x} = 2$$

Di conseguenza otteniamo

$$\frac{1}{1 - x} + \frac{2}{1 - 2x} \longleftrightarrow 1^n + 2 \cdot 2^n = 2^{n+1} + 1$$

∎

Esercizio 2.5

Determinare le successioni associate alle seguenti funzioni:

$$\frac{1+x}{2-3x+2x^2}, \quad \frac{3+x}{(1+x)(1-3x)(1+2x)}, \quad \frac{1}{(1-x)^5}$$

Problema 2.3 *Data la funzione generatrice $A(x)$, stimare il comportamento asintotico di $\{a_n\}$ per $n \to +\infty$.*

Nel caso di funzioni razionali $A(x) = \frac{P(x)}{Q(x)}$ (con P e Q polinomi primi tra loro) in cui il denominatore $Q(x)$ ha un'unica radice di minor modulo $\overline{x}$ di molteplicità t, si ottiene

$$a_n = \Theta\left(n^{t-1} \cdot \frac{1}{\overline{x}^n}\right)$$

Esercizio 2.6

Determinare l'ordine di grandezza della sequenza associata alla funzione $\frac{3x^3+2x-7}{(1-x)^5(2-x)}$.
Svolgimento. Il polinomio $(1-x)^5(2-x)$ ha radici 1 (di molteplicità 5) e 2 (di molteplicità 1). La radice di minor modulo è 1, quindi $a_n = \Theta(n^{5-1} \cdot 1^n) = \Theta(n^4)$ ∎

Esercizio 2.7

Determinare l'ordine di grandezza delle sequenze associate alle funzioni generatrici seguenti:

$$\frac{1+x}{(2-3x-2x^2)^2(1-4x)^3}, \quad \frac{3-4x}{(1-3x+2x^2)^5}$$

Capitolo 3

Equazioni di ricorrenza

Problema 3.1 *Determinare la funzione generatrice* $Y(x)$ *della successione* $\{y(n)\}$ *tale che*

$$a \cdot y(n+2) + b \cdot y(n+1) + c \cdot y(n) + g(n) = 0$$

sapendo che $y(0) = \alpha$, $y(1) = \beta$, *dove* $g(n)$ *è nota e* a, b *e* c *sono costanti.*

Si può procedere come segue:

1. Si calcola la funzione generatrice G di $\{g(n)\}$ (vedi il problema 2.1).
2. Ricordando le regole 7. e 8. della sezione precedente, si ottiene:

$$a \cdot \frac{Y(x) - \alpha - \beta x}{x^2} + b \cdot \frac{Y(x) - \alpha}{x} + c \cdot Y(x) + G(x) = 0$$

3. Si risolve rispetto a $Y(x)$ la precedente equazione di primo grado.

Esercizio 3.1

Determinare la funzione generatrice $Y(x)$ di $y(n)$ dove

$$y(n+2) = y(n) + n, \quad \text{con} \quad y(0) = y(1) = 0$$

Svolgimento.
La funzione generatrice di $\{n\}$ è $\frac{x}{(1-x)^2}$ (vedi regola 6. della sezione precedente). Si ottiene allora:

$$\frac{Y(x)}{x^2} = \frac{Y(x)}{x} + Y(x) + \frac{x}{(1-x)^2}$$

Risolvendo l'equazione si ricava

$$Y(x) = \frac{x^3}{(1-x)^2(1-x-x^2)}$$

∎

Esercizio 3.2

Determinare le funzioni generatrici associate alle successioni

$$\begin{cases} y(n+1) = 3y(n) + n2^n \\ y(0) = 1 \end{cases} \qquad \begin{cases} y(n+2) = y(n+1) - y(n) + 3^n \\ y(0) = y(1) = 0 \end{cases}$$

Problema 3.2 *Risolvere l'equazione di ricorrenza lineare a coefficienti costanti :*

$$\begin{cases} a \cdot y(n+2) + b \cdot y(n+1) + c \cdot y(n) + g(n) = 0 \\ y(0) = \alpha, \quad y(1) = \beta \end{cases}$$

Si può procedere come segue:

1. Determina la funzione generatrice $Y(x)$ con il metodo usato per risolvere il problema 3.1.
2. Determina la successione $y(n)$ associata a $Y(x)$ come nella soluzione al problema 2.2.

Esercizio 3.3

Risolvere le equazioni di ricorrenza

$$\begin{cases} y(n+1) = 3y(n) + 2^n \\ y(0) = 1 \end{cases} \qquad \begin{cases} y(n+1) = 2 \cdot y(n) + 5 \\ y(0) = 0 \end{cases}$$

Problema 3.3 *Determinare l'ordine di grandezza di una sequenza $y(n)$ che verifica una equazione di ricorrenza lineare a coefficienti costanti.*

Si può procedere come segue:

1. Si determina la funzione generatrice $Y(x)$ con il metodo studiato per risolvere il problema 3.1.
2. Si determina l'ordine di grandezza di $\{y(n)\}$ con il metodo proposto per risolvere il problema 2.3.

Esercizio 3.4

Determinare l'ordine di grandezza delle successione definite dalle seguenti equazioni di ricorrenza:

$$\begin{cases} y(n+2) = 3y(n+1) - y(n) + 3n \\ y(0) = y(1) = 0 \end{cases} \qquad \begin{cases} y(n+1) = 2y(n) + n^2 \\ y(0) = 3 \end{cases}$$

Problema 3.4 *Risolvere l'equazione del tipo "divide et impera" definita da*

$$\begin{cases} y(n) = a \cdot y\left(\frac{n}{b}\right) + g(n) \\ y(1) = \alpha \end{cases}$$

Si può procedere come segue (per n potenza di b):

1. Si pone $n = b^k$ e $z(k) = y(b^k)$; per $z(k)$ si ottiene la seguente equazione lineare a coefficienti costanti
$$\begin{cases} z(k) = az(k-1) + g(b^k) \\ z(0) = \alpha \end{cases}$$

2. Si ottiene l'espressione esplicita di $z(k)$ applicando il metodo usato per il problema 3.2.

3. Ricordando ora che $k = \log_b n$ e $z(k) = y(b^k)$, si ottiene $y(n) = z(\log_b n)$.

Esercizio 3.5

Determinare l'ordine di grandezza della successione $\{y(n)\}$ che verifica la seguente equazione

$$\begin{cases} y(n) = 2y\left(\frac{n}{2}\right) + 1 \\ y(1) = 0 \end{cases}$$

Svolgimento. Poiché $n = 2^k$ e $z(k) = y(2^k)$, si ha

$$\begin{cases} z(k) = 2z(k-1) + 1 \\ z(0) = 0 \end{cases}$$

Quindi

$$\begin{cases} z(k+1) = 2z(k) + 1 \\ z(0) = 0 \end{cases}$$

La funzione generatrice $Z(x)$ di $\{z(k)\}$, ottenuta risolvendo il problema 3.1, è

$$Z(x) = \frac{x}{(1-2x)(1-x)}$$

e quindi $z(k) = \Theta(2^k)$. Poiché $k = \log_2 n$, otteniamo

$$y(n) = z(\log_2 n) = \Theta(2^{\log_2 n}) = \Theta(n)$$

∎

Esercizio 3.6

Determinare l'ordine di grandezza delle successioni $\{y(n)\}$ che verificano le seguenti equazioni:

$$\begin{cases} y(n) = 3y\left(\frac{n}{3}\right) + n \\ y(1) = 0 \end{cases} \qquad \begin{cases} y(n) = 5y\left(\frac{n}{3}\right) + n \\ y(1) = 0 \end{cases} \qquad \begin{cases} y(n) = 3y\left(\frac{n}{4}\right) + n^2 \\ y(1) = 0 \end{cases}$$

Esercizio 3.7

Determinare l'ordine di grandezza della successione $\{y(n)\}$ che verifica la seguente equazione

$$\begin{cases} y(n) = 2y\left(\frac{n}{3}\right) + n\log_2 n \\ y(1) = 0 \end{cases}$$

Svolgimento. Ponendo $n = 3^k$ e $g(k) = y(3^k)$, abbiamo

$$g(k+1) = 2g(k) + (\log_2 3)(k+1)3^{k+1}$$

Una stima asintotica di $\{g(k)\}$ può essere ottenuta passando alle funzioni generatrici e applicando i metodi illustrati nella sezione 2. Denotando quindi con $G(x)$ la funzione generatrice di $\{g(k)\}$, si ottiene

$$\frac{G(x)-1}{x} = 2G(x) + (\log_2 3)\sum_{k=0}^{+\infty}(k+1)3^{k+1}x^k$$

e quindi

$$G(x)(1-2x) = (\log_2 3)\sum_{k=0}^{+\infty} k3^k x^k$$

Osserva ora che la funzione generatrice di $\{k3^k\}$ è $x\frac{d}{dx}\frac{1}{1-3x} = \frac{3x}{(1-3x)^2}$. Allora, la funzione generatrice di $\{g(k)\}$ è

$$G(x) = \frac{3(\log_2 3)x}{(1-2x)(1-3x)^2};$$

questa ammette una sola radice di modulo minimo $\frac{1}{3}$ la cui molteplicita è 2. Di conseguenza otteniamo $g(k) = \Theta(k3^k)$ e quindi

$$y(n) = g(\log_3 n) = \Theta(n\log n)$$

∎

Esercizio 3.8

Determinare l'ordine di grandezza delle successioni $\{y(n)\}$ che verificano le seguenti equazioni:

$$\begin{cases} y(n) = 5y\left(\frac{n}{3}\right) + \log_2 n \\ y(1) = 0 \end{cases} \quad \text{[soluzione: } \Theta(n^{\log_3 5})\text{]},$$

$$\begin{cases} y(n) = 2y\left(\frac{n}{2}\right) + n\log_2 n \\ y(1) = 0 \end{cases} \quad \text{[soluzione: } \Theta(n\log^2 n)\text{]},$$

$$\begin{cases} y(n) = 3y\left(\frac{n}{4}\right) + n^2\log_2 n \\ y(1) = 0 \end{cases} \quad \text{[soluzione: } \Theta(n^2\log n)\text{]}.$$

Capitolo 4

Criteri di costo uniforme e logaritmico

4.1 Analisi di procedure AG

Gli algoritmi considerati in questo corso sono descritti mediante un linguaggio chiamato AG (vedi sez.3.4 delle dispense). Per valutare il tempo di calcolo e lo spazio di memoria richiesto dai suoi comandi è sufficiente considerare la complessità in tempo e spazio dei corrispondenti programmi RAM. Se siamo interessati solo all'ordine di grandezza della complessità in tempo si possono utilizzare le definizioni dei tempi di calcolo dei comandi AG che presentiamo in questa sezione. Queste tengono conto delle ipotesi fatte sul modello RAM e possono essere viste come una naturale estensione delle nozioni introdotte nel capitolo 3 delle dispense.

Dato un programma AG P, dotato di un insieme di variabili V, chiamiamo *stato di calcolo* una funzione $S : V \longrightarrow \mathbb{Z}$. Per ogni $a \in V$, $S(a)$ rappresenta appunto il valore della variabile a nello stato S. L'effetto di un comando AG è quindi quello di trasformare lo stato di calcolo (secondo la nota semantica operazionale dei comandi considerati).

Esempio. Determiniamo gli stati S_1 e S_2 ottenuti applicando allo stato S (con $S(a) = 1$ e $S(b) = 2$) rispettivamente i comandi:

1. `for` $k = 1, 2, 3$ `do` $a := a + b$
2. `begin`
   $a := a + b$
   $b := a \cdot b$
   `end`

È facile verificare che $S_1(a) = 7$, $S_1(b) = 2$, $S_2(a) = 3$, $S_2(b) = 6$.

Dato un comando AG C e uno stato S, definiamo i seguenti tempi di calcolo:

$T^U_{C,S}$ = tempo di calcolo richiesto dal comando C nello stato S assumendo il criterio di costo *uniforme*

$T^L_{C,S}$ = tempo di calcolo richiesto dal comando C nello stato S assumendo il criterio di costo *logaritmico*

Osserva che, a meno di un fattore costante, questi sono i tempi richiesti (rispettivamente nell'ipotesi uniforme e in quella logaritmica) dal corrispondente programma RAM che implementa il comando considerato.

Comando di assegnamento. Considera $a := b + c$ (con S stato qualsiasi). Allora:

$$T^U_{a:=b+c,S} = 1$$

$$T^L_{a:=b+c,S} = \log S(b) + \log S(c)$$

Esempio. Se $S(b) = n$ e $S(c) = 2^n$, allora

$$T^L_{a:=b+c,S} = \log n + \log 2^n = \Theta(n)$$

Comando composto. Siano C_1 e C_2 due comandi qualsiasi e sia S uno stato generico. Denotiamo con C il comando composto $C \equiv \texttt{begin}\ C_1; C_2\ \texttt{end}$. Allora

$$T^U_{C,S} = T^U_{C_1,S} + T^U_{C_2,S_1}$$

$$T^L_{C,S} = T^L_{C_1,S} + T^L_{C_2,S_1}$$

dove S_1 è lo stato ottenuto applicando C_1 a S.
Esempio. Se $S(a) = n$ e $C \equiv \texttt{begin}\ a := a \cdot a;\ \ a := a \cdot a\ \texttt{end}$ allora

$$\begin{aligned} T^L_{C,S} &= T^L_{a:=a^2,S} + T^L_{a:=a^2,S_1} \\ &= \log n + \log n + \log n^2 + \log n^2 = 6 \log n \end{aligned}$$

Comando For. Se $C \equiv \texttt{for}\ k = 1, \ldots, n\ \texttt{do}\ C_1$ e S uno stato qualsiasi, allora

$$T^U_{C,S} = \sum_{k=1}^{n} 1 + T^U_{C_1,S_k}$$

$$T^L_{C,S} = \sum_{k=1}^{n} \log k + T^L_{C_1,S_k}$$

dove S_k è lo stato ottenuto da S dopo k iterazioni.
Esempio. Assumendo il criterio di costo logaritmico, determiniamo il tempo di calcolo $T(n)$ richiesto dal programma

```
begin
      a := 1
      for j = 1, ..., n do a := a · j
end
```

Dopo k iterazioni la variabile j assume il valore k e la variabile a assume il valore $k!$. Di conseguenza

$$T(n) = 1 + \sum_{k=1}^{n} (2 \log k + \log(k - 1!)) = \Theta(n^2 \log n)$$

Comando While. Definiamo $C \equiv \texttt{while}\ cond\ \texttt{do}\ C_1$ e sia S uno stato qualsiasi. Allora $T^U_{C,S}$ è definito come nel caso precedente a patto di tenere conto del numero (questa volta variabile) delle iterazioni eseguite e del tempo necessario per valutare la condizione *cond*. Anche la definizione di $T^L_{C,S}$ è analoga.

Esempio. Assumendo il criterio di costo logaritmico, stimiamo il tempo di calcolo $T(n)$ richiesto dal programma

```
begin
    a := 0
    c := 0
    while c < b do { c := c + a
                   { a := a + 1
end
```

a partire dallo stato S, dove $S(b) = n$. Dopo k iterazioni la variabile a assume il valore k e la variabile c assume il valore $1 + 2 + \cdots + k$. Il ciclo viene ripetuto t volte, con

$$1 + 2 + \cdots + (t - 1) < n \leq 1 + 2 + \cdots + t$$

e quindi $n \sim \frac{t^2}{2}$, da cui si ricava $t \sim \sqrt{2n}$. Di conseguenza

$$T(n) = \Theta\left(\sum_{k=1}^{\sqrt{2n}} (2\log(1 + 2 + \cdots + k) + \log n + \log k)\right) = \Theta(n^{\frac{1}{2}} \log n)$$

Chiamata di procedura. Consideriamo una chiamata di procedura della forma:

```
      .
      .
      .            Procedure F(λ)
  a := F(b)              C
      .
      .
      .
```

Come per i comandi precedenti possiamo definire:

$$T^U_{a:=F(b),S} = 1 + T^U_{C,S_0}$$

$$T^L_{a:=F(b),S} = \log f(S(b)) + T^L_{C,S_0}$$

dove S_0 è lo stato di calcolo per F ottenuto attribuendo al parametro formale λ il valore $S(b)$ e $f(c)$ è la funzione calcolata da F su input c.

Esempio. Assumendo il criterio uniforme, determinare il tempo di calcolo $T(n)$ dell'esecuzione del programma MAIN su input $n \in \mathbb{N}$:

```
Procedure MAIN
begin
    S := 0
    for j = 0,...,n do { a := F(j)
                       { S := a + S
end
```

```
ProcedureF(λ)
begin
    c := 0
    for k = 0,...,λ do c := c + k
    return c
end
```

Se al parametro formale λ si attribuisce il valore x, la procedura F restituisce il valore

$$f(x) = \frac{x(x+1)}{2} \sim \frac{x^2}{2}$$

Sempre in tali ipotesi, il tempo di esecuzione di Procedura$F$ è

$$T_F(x) = 3x \log x$$

Allora:

$$T(n) = 1 + \sum_{j=0}^{n} [T_F(j) + \log f(j) + \log(f(0) + \cdots + f(j)] = \Theta(n^2 \log n)$$

Procedure ricorsive. Come sappiamo le procedure ricorsive sono implementate su RAM in maniera iterativa costruendo la pila dei record di attivazione delle varie chiamate (vedi capitolo 5 delle dispense). Il tempo di calcolo e lo spazio di memoria richiesti da una procedura di questo genere si ottengono considerando l'implementazione iterativa associata.

Consideriamo una famiglia di procedure ricorsive $P_1, P_2, \ldots, P_m$ e, per ogni $k \in \{1, 2, \ldots, m\}$, denotiamo con $T_k(n)$ il tempo di calcolo richiesto da P_k su un input di dimensione n. Allora è possibile esprimere tali funzioni mediante un sistema di m equazioni di ricorrenza in $T_1, T_2, \ldots, T_m$. La sua soluzione consente di valutare il tempo di calcolo richiesto da ciascuna procedura. La valutazione dello spazio di memoria utilizzato deve inoltre tenere conto della memoria occupata dalla pila dei record di attivazione.

Esempio. Consideriamo le procedure $F(x)$ e $G(y)$ definite dai seguenti programmi:

```
Procedure F(x)
if x = 0 then return 0
     else { a := x − 1
          { b := F(a)
          { c := G(a)
          { return (b + c)
```

```
Procedure G(y)
if y ≤ 1 then return 0
     else { d := ⌊y/2⌋
          { d := G(d)
          { return (d + 1)
```

I tempi di calcolo (secondo il criterio uniforme) richiesti dalle due procedure, su input $n \in \mathbb{N}$, sono definiti dal seguente sistema di equazioni di ricorrenza:

$$\begin{cases} T_F(0) = 2 & \\ T_F(n) = 3 + T_F(n-1) + T_G(n-1) & \text{se } n \geq 1 \\ T_G(n) = 2 & \text{se } n \leq 1 \\ T_G(n) = 2 + T_G(\lfloor \frac{n}{2} \rfloor) & \text{se } n \geq 2 \end{cases}$$

Applicando i metodi illustrati nelle sezioni precedenti otteniamo

$$T_G(n) = \Theta(\log n)$$
$$T_F(n) = \Theta(n \log n)$$

Per quanto riguarda lo spazio (secondo il criterio uniforme) è facile verificare che

$$S_G(n) = \Theta(\log n)$$
$$S_F(n) = \Theta(n)$$

4.2 Esercizi

Esercizio 4.1

Considera il seguente problema:

> Istanza : una n-pla di numeri interi $a_1, a_2, \ldots, a_n$, $n \geq 2$;
> Soluzione : il valore massimo e il valore minimo in $\{a_1, a_2, \ldots, a_n\}$.

a) Descrivere ad alto livello una procedura per risolvere il problema.

b) Assumendo il criterio di costo uniforme, determinare l'ordine di grandezza del tempo di calcolo e dello spazio di memoria richiesti dalla procedura in funzione di n.

c) Supponiamo che ogni a_i sia un intero di k bit e assumiamo il criterio di costo logaritmico; determinare l'ordine di grandezza del tempo e dello spazio richiesti dalla procedura in funzione dei parametri n e k.

Svolgimento.

a)

```
begin
    b := read(a_1)
    c := b
    for i = 2, 3, ..., n do
        begin
            d := read(a_i)
            if b < d then b := d
                     else if c > d then c := d
        end
end
```

b) Secondo il criterio uniforme il tempo di calcolo è $\Theta(n)$ mentre lo spazio di memoria è $\Theta(1)$.

c) Secondo il criterio logaritmico il tempo di calcolo è $\Theta(kn)$ mentre spazio occupato è $\Theta(k)$.

∎

Esercizio 4.2

Considera il seguente problema:

Istanza : una n-pla di numeri interi $a_1, a_2, \ldots, a_n$, $n \geq 1$;
Soluzione : la somma $a_1 + a_2 + \cdots + a_n$.

a) Descrivere ad alto livello una procedura per risolvere il problema.
b) Assumendo il criterio di costo uniforme, determinare l'ordine di grandezza del tempo di calcolo e dello spazio di memoria richiesti dalla procedura in funzione di n.
c) Supponiamo che ogni a_i sia un intero di k bit e assumiamo il criterio di costo logaritmico; determinare l'ordine di grandezza del tempo e dello spazio richiesti dalla procedura in funzione dei parametri n e k.

Svolgimento.
a)

begin
$s := 0$
for $i = 1, 2, \ldots, n$ **do** $\begin{cases} b := read(a_i) \\ s := s + b \end{cases}$
end

b) Secondo il criterio uniforme il tempo di calcolo è $\Theta(n)$ mentre lo spazio di memoria è $\Theta(1)$.

c) Assumiamo ora il criterio logaritmico. Osserva che dopo l'esecuzione del ciclo i-esimo la variabile s assume un valore minore di $2^k i$ e quindi la sua dimensione è $\Theta(k + \log i)$ nel caso peggiore. Di conseguenza il tempo di calcolo complessivo risulta

$$\Theta\left(\sum_{i=1}^{n} k + \log i\right) = \Theta(kn + n \log n)$$

mentre lo spazio occupato è $\Theta(k + \log n)$. ∎

Esercizio 4.3

Considera il seguente problema:

Istanza : una n-pla di numeri interi $a_1, a_2, \ldots, a_n$, $n \geq 2$;
Domanda : esistono due indici distinti $i, j \in \{1, 2, \ldots, n\}$ tali che $a_i = a_j$?

a) Descrivere ad alto livello una procedura per risolvere il problema.
b) Assumendo il criterio di costo uniforme, determinare l'ordine di grandezza del tempo di calcolo richiesto dalla procedura nel caso migliore e nel caso peggiore.
c) Supponendo che ogni a_i sia un intero di k bit e assumendo il criterio di costo logaritmico, svolgere l'analisi richiesta al punto precedente.

Svolgimento.
a)

```
begin
        i := 1
        j := 2
        while i < n ∧ a_i ≠ a_j do
                if j < n  then j := j + 1
                          else { i := i + 1
                                 j := i + 1
        if  a_i = a_j     then return si
                          else return no
end
```

b) Il caso migliore si verifica quando $a_1 = a_2$, mentre quello peggiore quando gli elementi a_i sono tutti distinti tra loro. Quindi, secondo il criterio uniforme, nel caso migliore la procedura può essere eseguita in tempo $\Theta(1)$ mentre, in quello peggiore, in tempo $\Theta(n^2)$.

c) Secondo il criterio logaritmico il tempo di calcolo nel caso migliore è $\Theta(k)$ mentre, in quello peggiore, è $\Theta(n^2(k + \log n))$. ∎

Capitolo 5

Procedure ricorsive

Esercizio 5.1

Considera le seguenti procedure G e F che calcolano rispettivamente i valori $G(n)$ e $F(n)$ su input $n \in \mathbb{N}$:

```
Procedure G(n)
begin
        b := 0
        for k = 0, 1, 2, ..., n do
            begin
                    u := F(k)
                    b := b^2 + u
            end
        return b
end

Procedure F(n)
if n = 0 then return 1
         else  return 2F(n − 1) + n
```

a) Valutare l'ordine di grandezza del tempo di calcolo e dello spazio di memoria richiesti dalla procedura G su input n assumendo il criterio di costo uniforme.

b) Stimare il tempo di calcolo richiesto dalla procedura G assumendo il criterio di costo logaritmico.

c) Definire una procedura per il calcolo di $G(n)$ che richieda tempo $O(n)$ e spazio $O(1)$ secondo il criterio di costo uniforme.

Svolgimento.

a) Assumendo il criterio uniforme denotiamo con $T_F(n)$ e $T_G(n)$ il tempo di calcolo richiesto rispettivamente dalle procedure F e G su input n. Analogamente siano $S_F(n)$ e $S_G(n)$ le quantità di spazio utilizzate dalle due procedure.

Si verifica facilmente che $T_F(n) = \Theta(n)$ e $S_F(n) = \Theta(n)$.

Inoltre, per una opportuna costante $c > 0$, abbiamo

$$T_G(n) = \sum_{k=0}^{n} c + T_F(k) = \Theta(n^2)$$

Per quanto riguarda lo spazio si verifica $S_G(n) = \Theta(n)$.

b) Assumiamo ora il criterio di costo logaritmico e siano $T_F^\ell(n)$, $T_G^\ell(n)$, $S_F^\ell(n)$, $S_G^\ell(n)$ le corrispondenti quantità di tempo e spazio.

Osserviamo subito che il valore $F(n)$ è dato da $F(n) = n + 2((n-1) + 2((n-2) + \cdots = \sum_{j=0}^{n-1}(n-j)2^j$; di conseguenza $F(n) = \Omega(2^n)$ e $F(n) = O(n2^n)$. Questo significa che la dimensione di $F(n)$ verifica la relazione

$$|F(n)| = \Theta(n)$$

Possiamo allora scrivere la relativa ricorrenza per $T_F^\ell(n)$ (con $c > 0$ costante opportuna):

$$T_F^\ell(n) = \begin{cases} c & \text{se } n = 0 \\ \Theta(n) + T_F^\ell(n-1) & \text{altrimenti} \end{cases}$$

ricavando $T_F^\ell(n) = \Theta(n^2)$.

Per quanto riguarda lo spazio di memoria richiesto dalla procedura F osserviamo che, su input n, la pila raggiunge altezza n e l'i-esimo record di attivazione occupa uno spazio $\Theta(\log(n-i))$. Sappiamo inoltre che lo spazio necessario per mantenere il risultato è $\Theta(n)$ e quindi (per un'altra $c > 0$ costante) si ottiene un valore complessivo

$$S_F^\ell(n) = \Theta(n) + \sum_{i=1}^{n} c \log i = \Theta(n \log n)$$

Valutiamo ora le risorse utilizzate da G. Innanzitutto osserviamo che, per ogni $n > 0$, $G(n) = F(n) + (G(n-1))^2$ e quindi $|G(n)| \leq \Theta(n) + 2|G(n-1)| \leq O(n2^n)$. Questo significa che, per opportune costanti $c_1, c_2 > 0$ e trascurando termini di ordine inferiore, abbiamo

$$T_G^\ell(n) = \sum_{k=1}^{n}(T_F^\ell(k) + c_1 k + c_2|G(n-1)|) = \Theta(n^3) + O\left(\sum_{k=1}^{n} k2^k\right) = O(n2^n)$$

Analogamente otteniamo $S_G^\ell(n) = O(n2^n)$.

c)

```
Procedure G(n)
begin
    b := 0
    u := 1
    for k = 0, 1, 2, ..., n do
        begin
            b := b^2 + u
            u := 2u + k + 1
        end
    return b
end
```

∎

Esercizio 5.2

Considera la seguente procedura ricorsiva che calcola il valore $F(n) \in Z$ su input $n \in \mathbb{N}$.

```
Procedure F(n)
begin
        if n ≤ 1 then return n
               else begin
                        A = F(n − 1)
                        B = F(n − 2)
                        return (5B − A)
                    end
end
```

a) Determinare l'ordine di grandezza del tempo di calcolo e dello spazio di memoria richiesti dalla procedura F su input n assumendo il criterio di costo uniforme.

b) Svolgere l'esercizio precedente assumendo il criterio di costo logaritmico.

c) Definire una procedura per il calcolo di $F(n)$ che richieda tempo $O(n)$ e spazio $O(1)$ secondo il criterio di costo uniforme.

Svolgimento.

a) Denotiamo con $T_F(n)$ il tempo richiesto dalla procedura F su input n assumendo il criterio di costo uniforme. Allora esistono due interi positivi b, c tali che

$$T_F(n) = \begin{cases} b & \text{se } n \leq 1 \\ T_F(n-1) + T_F(n-2) + c & \text{se } n \geq 2 \end{cases}$$

L'equazione può essere risolta usando le funzioni generatrici. Esprimiamo innanzitutto i termini di $\{T_F(n)\}_n$ mediante gli operatori definiti sulle sequenze:

$$E^2(T_F(n)) = E(T_F(n)) + T_F(n) + c$$

Passiamo ora alle corrispondenti equazioni sulle funzioni generatrici. Denotando con $T_F(z)$ la funzione generatrice di $\{T_F(n)\}_n$, si ricava

$$\frac{T_F(z) - b - bz}{z^2} = \frac{T_F(z) - b}{z} + T_F(z) + \frac{c}{1-z}$$

e quindi

$$T_F(z) = \frac{b + (c-b)z}{(1-z)(1-z-z^2)}$$

Ponendo ora $P(z) = b + (c-b)z$ e $Q(z) = (1-z)(1-z-z^2)$, possiamo scrivere

$$T_F(z) = \frac{P(z)}{Q(z)}$$

Valutiamo ora l'ordine di grandezza di $\{T_F(n)\}_n$ considerando le radici dei due polinomi. È facile verficare che $P(z)$ e $Q(z)$ non possono avere radici comuni. Inoltre $Q(z)$ possiede una sola radice di modulo minimo: $\frac{\sqrt{5}-1}{2}$. Di conseguenza si ottiene

$$T_F(n) = \Theta\left(\left(\frac{2}{\sqrt{5}-1}\right)^n\right)$$

Lo spazio di memoria richiesto dalla procedura è essenzialmente dato dalla dimensione della pila necessaria per implementare la ricorsione. Tale pila contiene al più n record di attivazione e quindi lo spazio occupato, assumendo il criterio uniforme, è $\Theta(n)$.

b) Valutiamo prima l'ordine di grandezza della sequenza di valori $\{F(n)\}_n$. Applicando lo stesso procedimento utilizzato per risolvere il punto a), otteniamo:

$$F(n) = \begin{cases} n & \text{se } n \leq 1 \\ 5F(n-2) - F(n-1) & \text{se } n \geq 2 \end{cases}$$

$$E^2(F(n)) = 5E(F(n)) - F(n)$$

$$F(z) = \frac{z}{1 - 5z + z^2}$$

$$F(n) = \Theta\left(\left(\frac{2}{\sqrt{21} - 5}\right)^n\right)$$

Osserva che la dimensione di $F(n)$ è dell'ordine $\Theta(n)$.

Denotiamo ora con $T_F^\ell(n)$ il tempo richiesto dalla procedura assumendo il criterio di costo logaritmico. L'equazione di ricorrenza corrispondente è

$$T_F^\ell(n) = \begin{cases} b & \text{se } n \leq 1 \\ T_F^\ell(n-1) + T_F^\ell(n-2) + g(n) & \text{se } n \geq 2 \end{cases}$$

dove b è un intero positivo e $g(n)$ è il tempo necessario per eseguire le varie operazioni aritmetiche che compaiono nella procedura. Poiché $|F(n)| = \Theta(n)$ otteniamo $g(n) = \Theta(n)$ e possiamo così sostituire nell'equazione precedente il termine $g(n)$ con l'espressione $c \cdot n$, con c costante positiva.

Riapplicando il metodo delle funzioni generatrici otteniamo:

$$E^2(T_F^\ell(n)) = E(T_F^\ell(n)) + T_F^\ell(n) + cn$$

$$\frac{T_F^\ell(z) - b - bz}{z^2} = \frac{T_F^\ell(z) - b}{z} + T_F^\ell(z) + \frac{cz}{(1-z)^2}$$

e quindi

$$T_F^\ell(z) = \frac{b(1-z)^2 + cz^3}{(1-z)(1-z-z^2)}$$

dalla quale si ricava

$$T_F^\ell(n) = \Theta\left(\left(\frac{2}{\sqrt{5} - 1}\right)^n\right)$$

Per quanto riguarda lo spazio di memoria richiesto secondo il criterio logaritmico, osserva che ogni chiamata ricorsiva deve mantenere almeno il parametro di ingresso. Così l'i-esimo record di attivazione della pila occupa uno spazio $\Theta(\log(n-i))$. Inoltre la dimensione del valore di uscita è $|F(n)| = \Theta(n)$. Di conseguenza otteniamo una valutazione complessiva

$$S_F^\ell(n) = \Theta(n) + \Theta\left(\sum_{i=1}^{n} \log i\right) = \Theta(n \log n)$$

c)

```
Procedure F(n)
if n ≤ 1 then return n
        else
            begin
                b := 0
                a := 1
                                          { c := 5b − a
                for  i = 2, ..., n do     { b := a
                                          { a := c
                return a
            end
```

∎

Esercizio 5.3

Considera le seguenti procedure G e F che calcolano rispettivamente i valori $G(n)$ e $F(n)$ su input $n \in \mathbb{N}$:

```
Procedure G(n)
begin
    b := 0
    for k = 0, 1, 2, ..., n do
        begin
            u := F(k)
            b := b + 2^k u
        end
    return b
end
```

```
Procedure F(n)
if n = 0 then return 1
        else  return F(n − 1) + n
```

a) Determinare l'ordine di grandezza del tempo di calcolo e dello spazio di memoria richiesti dalla procedura G su input n assumendo il criterio di costo uniforme.

b) Svolgere l'esercizio precedente assumendo il criterio di costo logaritmico.

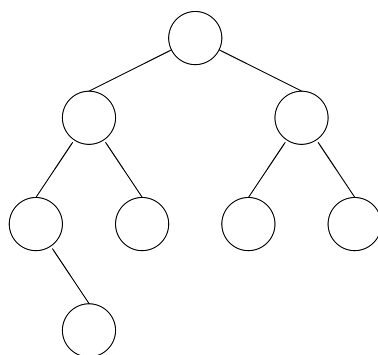
c) Definire una procedura per il calcolo di $G(n)$ che richieda tempo $O(n)$ e spazio $O(1)$ secondo il criterio di costo uniforme.

Capitolo 6

Algoritmi su alberi e grafi

Esercizio 6.1

Si consideri l'albero binario T descritto nella seguente figura:

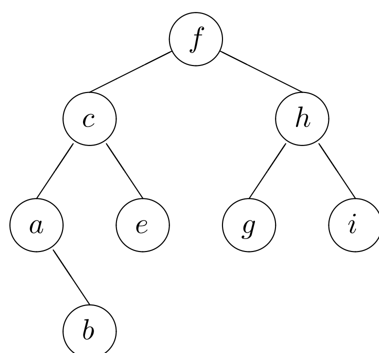


a) Etichettare i nodi di T in modo da ottenere un albero binario di ricerca per l'insieme di lettere $\{a,b,c,e,f,g,h,i\}$ rispetto all'ordine alfabetico.

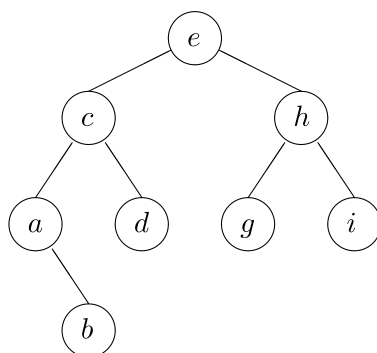
b) Eseguire su T le istruzioni INSERT(d) e DELETE(f) una dopo l'altra e disegnare l'albero risultante.

Svolgimento.

a)



b)



∎

Esercizio 6.2

Dato un albero con radice T, chiamiamo *peso* di un nodo v in T il numero dei suoi discendenti.

a) Descrivere una procedura ricorsiva per risolvere il seguente problema:

> Istanza : un albero con radice T;
> Soluzione : la somma dei pesi dei nodi di T.

b) Descrivere una procedura *non* ricorsiva per risolvere lo stesso problema.

Svolgimento.

a) Rappresentiamo l'albero di input usando la variabile r per la radice e, per ogni nodo v, la lista $L(v)$ dei suoi figli. Definiamo un programma principale e una procedura ricorsiva Discendenti(v) che utilizzano la variabile globale S che fornisce il risultato. La procedura Discendenti(v) esegue una visita in ordine posticipato dell'albero di input; essa restituisce il peso t del nodo v dopo aver incrementato S proprio del valore t calcolato. Tale procedura è ricorsiva e richiama sé stessa su tutti i discendenti di v. In questo modo l'esecuzione di Discendenti(v) incrementa la variabile globale S di una quantità pari alla somma dei pesi dei nodi contenuti nel sottoalbero di radice v.

```
begin
        S := 0
        u := Discendenti(r)
        return S
end

Procedure Discendenti(v)
if IS_EMPTY(L(v)) = 1 then return 0
                           else
                               begin
                                   t := 0;
                                   for w ∈ L(v) do { u := Discendenti(w);
                                                   { t := t + u + 1;
                                   S := S + t;
                                   return t;
```

end

b) La versione iterativa del medesimo algoritmo mantiene una pila, per la gestione della ricorsione, i cui elementi sono coppie della forma (v, t): v è il nome del nodo associato al record considerato mentre t contiene il valore del suo peso man mano che questo viene calcolato. Quando il record (v, t) viene inserito nella pila per la prima volta il valore di t è 0, mentre quando viene tolto il valore di t è proprio il peso del nodo v.

L'algoritmo è il seguente:

```
begin
    S := 0;
    Pila := Push(Λ, (r, 0));
    v := r;
    while Pila ≠ Λ do
        begin
            while IS_EMPTY(L(v)) = 0 do { w := TESTA(L(v));
                                          Pila := Push(Pila, (w, 0));
                                          L(v) := TOGLI_IN_TESTA(L(v));
                                          v := w,
            (v, u) := Top(Pila));
            Pila:= Pop(Pila);
            S := S + u;
            if Pila ≠ Λ then { (v, t) := Top(Pila);        % ovvero: incrementa
                               t := t + u + 1;             % di u + 1 la seconda
                               Pila := Pop(Pila);          % componente di Top(Pila)
                               Pila := Push(Pila,(v, t));
        end
    return S;
end
```

∎

Esercizio 6.3

a) Descrivere un algoritmo per il seguente problema:

Istanza : un grafo diretto $G = \langle V, E\rangle$ rappresentato mediante liste di adiacenza.
Soluzione : per ogni nodo $v \in V$ il numero dei nodi w raggiungibili da v (ovvero il numero dei nodi $w \in V$ per i quali esiste un cammino orientato da v a w).

b) Svolgere l'analisi dell'algoritmo valutando il tempo di calcolo e lo spazio di memoria richiesti su un input di n nodi e m lati (si assuma il criterio uniforme).

Svolgimento.

a) Supponiamo che il grafo di ingresso G sia rappresentato da un insieme di nodi V e dalla famiglia di liste di adiacenza $\{L(v) \mid v \in V\}$. Per ogni $v \in V$, L(v) è la lista dei nodi w per i quali esiste in G il lato (v, w).

L'algoritmo che presentiamo calcola per ogni $v \in V$ il valore $N[v]$ che rappresenta il numero di nodi in G raggiungibili da v. L'algoritmo esegue una esplorazione in ampiezza del grafo diretto G a partire da ciascun nodo v tenendo conto del numero di vertici incontrati.

for $v \in V$ **do**
  begin
    $c := 1$;
    $Q :=$ Enqueue(Λ, v);
    for $w \in V \backslash \{v\}$ **do** marca w come "nuovo"
    marca v come "vecchio"
    while $Q \neq \Lambda$ **do**
      begin
        $u :=$ Front(Q);
        $Q :=$ Dequeue(Q);
        for $w \in L(u)$ **do**
          if w marcato nuovo **then** $\begin{cases} \text{marca } w \text{ come "vecchio"} \\ c := c + 1; \\ Q := \text{Enqueue}(Q, w); \end{cases}$
      end
    $N[v] := c$;
  end

b) Sia n il numero dei nodi di G e sia m il numero dei suoi lati. Allora l'algoritmo sopra descritto lavora in tempo $O(n^2 + nm)$ e utilizza uno spazio $O(n + m)$. ∎

Esercizio 6.4

Chiamiamo *albero ternario* un albero con radice nel quale ogni nodo interno possiede al più tre figli e ogni figlio è distinto come *figlio sinistro*, *figlio centrale* oppure *figlio destro*.

a) Descrivere una struttura dati per rappresentare un albero ternario formato da n nodi.

b) Definire una procedura ricorsiva per attraversare un albero ternario che, partendo dalla radice, visiti ciascun nodo v applicando il seguente criterio:

prima attraversa il sottoalbero che ha per radice il figlio centrale di v,
poi visita v,
quindi attraversa il sottoalbero che ha per radice il figlio sinistro di v,
infine attraversa il sottoalbero che ha per radice il figlio destro di v.

c) Descrivere una versione iterativa dell'algoritmo precedente.

Svolgimento.

a) Supponiamo che l'albero contenga n nodi e rappresentiamo ogni nodo mediante un intero compreso tra 1 e n. La variabile r indica la radice. Allora possiamo rappresentare l'intero albero mediante tre vettori L, C, R di dimensione n ciascuno, le cui componenti sono così definite per ogni $v \in \{1, 2 \ldots, n\}$:

$$L[v] = \begin{cases} 0 & \text{se } v \text{ non possiede figlio sinistro} \\ u & \text{se } u \text{ è figlio sinistro di } v \end{cases}$$

$$C[v] = \begin{cases} 0 & \text{se } v \text{ non possiede figlio centrale} \\ u & \text{se } u \text{ è figlio centrale di } v \end{cases}$$

$$R[v] = \begin{cases} 0 & \text{se } v \text{ non possiede figlio destro} \\ u & \text{se } u \text{ è figlio destro di } v \end{cases}$$

Questi tre vettori sono sufficienti per implementare le procedure previste nei punti b) e c) successivi. Si può eventualmente aggiungere un ulteriore vettore F che associa a ogni nodo il padre corrispondente.

Un'altra struttura dati che svolge lo stesso ruolo può essere definita associando a ogni nodo un record contenente tutte le informazioni necessarie: nome del nodo, puntatore al record del figlio sinistro, puntatore al record del figlio centrale, puntatore al record del figlio destro e (eventualmente) puntatore al record del padre.

b) L'algoritmo consiste di un programma principale ATTRAVERSA che utilizza le variabili r, L, C, R sopra definite e richiama una procedura ricorsiva VISITA(v).

Procedure ATTRAVERSA
begin
  $v := r$;
  VISITA(v);
end

Procedure VISITA(v)
begin
  if $C[v] \neq 0$ **then** $\begin{cases} u := C[v] \\ \text{VISITA}(u) \end{cases}$
  visita il nodo v;
  if $L[v] \neq 0$ **then** $\begin{cases} u := L[v] \\ \text{VISITA}(u) \end{cases}$
  if $R[v] \neq 0$ **then** $\begin{cases} u := R[v] \\ \text{VISITA}(u) \end{cases}$
end

c) Definiamo ora una versione iterativa della procedura illustrata nel punto precedente. Anche in questo caso il procedimento utilizza le variabili r, L, C, R definite sopra insieme a una pila S che serve per mantenere traccia della ricorsione.

L'algoritmo è una semplice "traduzione iterativa" della procedura VISITA. Nota che ogni record di attivazione della pila S contiene il nome del nodo "chiamante" e l'indirizzo di ritorno.

Procedure VISITA_R
begin
  $S := \Lambda$; % Λ rappresenta la pila vuota %
  $v := r$;
1) **while** $C[v] \neq 0$ **do** $\begin{cases} S := \text{Push}(S, (v, 0)); \\ v := C[v]; \end{cases}$
2) visita il nodo v;

```
               if L[v] ≠ 0 then { S := Push(S, (v, 1));
                                  v := L[v];
                                  go to 1)
3)             if R[v] ≠ 0 then { v := R[v]
                                  go to 1)
               if S ≠ Λ then { (v, u) := Top(S);
                               S := Pop(S);
                               if u = 0 then go to 2)
                               if u = 1 then go to 3)
    end
```

Completiamo la soluzione descrivendo un'altra versione del medesimo algoritmo nella quale la pila mantiene solo nodi non ancora visitati; per questo motivo non si tratta della mera traduzione iterativa della procedura VISITA. Nota che in questa versione il nodo corrente si trova sempre in testa alla pila e il vettore C viene modificato durante l'esecuzione. Questa procedura non contiene istruzioni *go to* e mantiene nella pila solo il nome dei nodi (senza l'informazione relativa all'indirizzo di ritorno prevista nella versione precedente). Questo consente un certo risparmio di memoria che in alcuni casi può essere significativo.

```
Procedure ATTRAVERSA_R
begin
      S := Push(Λ, r);
      while S ≠ Λ do
                begin
                      v := Top(S);
                                             u := C[v];
                      while C[v] ≠ 0 do {    C[v] := 0;
                                             v := u;
                                             S := Push(S, v);
                      visita il nodo v;
                      S := Pop(S, v);
                      if R[v] ≠ 0 then S := Push(S, R[v]);
                      if L[v] ≠ 0 then S := Push(S, L[v]);
                end
end
```

∎

Esercizio 6.5

a) Determina l'albero di ricerca binaria che si ottiene inserendo la seguente sequenza di parole in un albero inizialmente vuoto (si adotti il tradizionale ordinamento alfabetico):

$$e, c, d, a, b, g, f, h, cb, ea$$

b) Determinare l'albero di ricerca binaria che si ottiene dall'albero precedente eseguendo le seguenti operazioni: delete(e), delete(g), insert(da), insert(eb), delete(f).

Esercizio 6.6

Dato un albero con radice, chiamiamo *altezza minimale* di un nodo v la minima distanza di v da una foglia.

a) Descrivere una procedura ricorsiva che riceve in input un albero con radice e calcola la somma delle altezze minimali dei suoi nodi.

b) Descrivere una versione iterativa dello stesso algoritmo.

Esercizio 6.7

Ricordiamo che la *lunghezza di cammino* in un albero con radice è la somma delle distanze dei nodi dalla radice.

a) Descrivere una procedura ricorsiva per calcolare la lunghezza di cammino in un albero con radice qualsiasi.

b) Descrivere una versione non ricorsiva dello stesso algoritmo.

Esercizio 6.8

a) Descrivere un algoritmo ricorsivo per risolvere il seguente problema:

Istanza : un albero con radice T di n nodi e un intero k, $0 \leq k \leq n-1$.
Soluzione : l'insieme dei nodi di T che hanno altezza k.

b) Descrivere una procedura *non* ricorsiva per risolvere lo stesso problema.

Esercizio 6.9

a) Descrivere un algoritmo ricorsivo per risolvere il seguente problema:

Istanza : un albero con radice T di n nodi e un intero k, $0 \leq k \leq n-1$.
Soluzione : il numero di nodi di T che hanno k discendenti.

b) Descrivere una procedura *non* ricorsiva per risolvere lo stesso problema.

Capitolo 7

Il metodo “divide et impera”

Esercizio 7.1

a) Assumendo come modello di calcolo la macchina RAM, descrivere ad alto livello una procedura iterativa (la più semplice intuitivamente) per calcolare il valore 5^n su input $n \in \mathbf{N}$.

b) Descrivere una procedura ricorsiva, basata su una tecnica divide et impera, per eseguire lo stesso calcolo.

c) Svolgere l'analisi dei tempi di calcolo delle due procedure secondo il criterio uniforme.

d) Svolgere l'analisi dei tempi di calcolo delle due procedure secondo il criterio logaritmico.

e) Valutare lo spazio di memoria richiesto dalla procedura ricorsiva secondo il criterio uniforme e secondo quello logaritmico.

Svolgimento.

a)

```
begin
    a := 1
    if n > 0 then for i = 1, 2, ..., n do
                      a := 5 · a
    return a
end
```

b)

```
Procedure Calcola(n)
if n = 0 then return 1
        else
            begin
                k := ⌊n/2⌋
                a := Calcola(k);
                if n pari  then return a · a
                           else return 5 · a · a
            end
```

c) È evidente che l'algoritmo descritto al punto a) richiede, secondo il criterio uniforme, un tempo di calcolo di ordine $\Theta(n)$ su input n.

Denotiamo ora con $T_b(n)$ il tempo richiesto (sempre secondo il criterio uniforme) dall'algoritmo descritto al punto b) su input n. Chiaramente tale quantità soddisfa una equazione della forma

$$T_b(n) = \begin{cases} c & \text{se } n = 0 \\ d_1 + T_b\left(\lfloor \frac{n}{2} \rfloor\right) & \text{se } n \text{ è dispari} \\ d_2 + T_b\left(\lfloor \frac{n}{2} \rfloor\right) & \text{se } n > 0 \text{ è pari} \end{cases}$$

dove c, d_1 e d_2 sono costanti. Usando opportune maggiorazioni e minorazioni si ottiene $T(n) = \Theta(\log n)$.

d) Assumiamo ora il criterio logaritmico. Su input n l'algoritmo descritto al punto a) richiede un tempo dell'ordine di

$$\sum_{i=1}^{n} c \cdot i = \Theta(n^2)$$

poiché all'i-esimo ciclo si eseguono un numero constante di operazioni su interi di $\Theta(i)$ bit.

Denotiamo ora con $T_b^\ell(n)$ il tempo richiesto (secondo il criterio logaritmico) dall'algoritmo descritto al punto b) su input n. Tale quantità soddisfa una equazione della forma

$$T_b^\ell(n) = \begin{cases} c & \text{se } n = 0 \\ \Theta(n) + T_b^\ell\left(\lfloor \frac{n}{2} \rfloor\right) & \text{altrimenti} \end{cases}$$

dove c è una costante opportuna. Sviluppando l'equazione si ottiene $T(n) = \Theta(n)$.

e) Su input n, l'altezza massima raggiunta dalla pila che implementa la ricorsione è $\log n$. Quindi $\Theta(\log n)$ è lo spazio richiesto dalla procedura secondo il criterio uniforme.

Assumendo invece quello logaritmico si può verificare che l'i-esimo record di attivazione della pila richiede uno spazio di ordine $O(\log n)$; inoltre, il risultato mantenuto sul record di attivazione in testa alla pila ha dimensione $O(n)$ (e raggiunge $\Theta(n)$ al termine della computazione). In totale quindi lo spazio richiesto è $O(\log^2 n) + \Theta(n) = \Theta(n)$. ∎

Esercizio 7.2

Considera il problema di calcolare l'espressione

$$C = b_1 \cdot b_2 \cdot \; \cdots \; \cdot b_n$$

avendo in input una sequenza $b_1, b_2, \ldots, b_n$ di numeri interi.

a) Descrivere un algoritmo iterativo che risolve il problema in tempo $O(n)$ secondo il criterio uniforme.

b) Svolgere l'analisi dell'algoritmo precedente assumendo il criterio di costo logaritmico, nell'ipotesi che ogni b_i sia un intero di m bit.

c) Descrivere un algoritmo ricorsivo del tipo *divide et impera* per risolvere lo stesso problema.

d) Svolgere l'analisi del tempo di calcolo e dello spazio di memoria richiesti dall'algoritmo precedente assumendo il criterio di costo uniforme.

e) Svolgere l'esercizio precedente assumendo il criterio di costo logaritmico e supponendo che ogni b_i sia un intero di n bit.

Svolgimento.

a)

```
begin
    C := 1;
    for k = 1, ..., n do
        begin
            b := read(b_k);
            C := C · b;
        end
end
```

b) Durante l'esecuzione del k-esimo ciclo il valore di b è un intero di m bit mentre C contiene il prodotto di k interi di m bit ciascuno; di conseguenza la dimensione di quest'ultimo risulta $|C| = \Theta(km)$. Ne segue che il tempo di calcolo $T(n, m)$ sull'input considerato verifica la seguente uguaglianza:

$$T(n,m) = \Theta\left(\sum_{k=1}^{n} km\right) = \Theta(mn^2)$$

c) L'idea dell'algoritmo è quella di applicare una procedura ricorsiva che spezza l'input in due parti (circa) uguali, richiama se stessa ricorsivamente su queste ultime e quindi restituisce il prodotto dei due risultati ottenuti. Formalmente l'algoritmo si riduce alla chiamata

$$C := \text{Prodotto}(1, n)$$

della procedura Prodotto(i, j), definita nel seguito, che restituisce il prodotto degli elementi

$$b_i, b_{i+1}, \ldots, b_j$$

con $1 \leq i \leq j \leq n$.

```
Procedure Prodotto(i, j)
if i = j then { a := read(b_i)
              { return a
    else
        begin
            k := ⌊(j+i)/2⌋;
            b := Prodotto(i, k);
            c := Prodotto(k + 1, j);
            return b · c;
        end
```

d) È chiaro che il tempo di calcolo richiesto per eseguire Prodotto(i, j) dipende dal numero di elementi che occorre moltiplicare tra loro. Definiamo allora $u = j - i + 1$ e denotiamo con $T(u)$ il tempo di calcolo richiesto da Prodotto(i, j) secondo il criterio uniforme. Otteniamo quindi la seguente equazione di ricorrenza:

$$T(u) = \begin{cases} c_1 & \text{se } u = 1 \\ T\left(\lfloor \frac{u}{2} \rfloor\right) + T\left(\lceil \frac{u}{2} \rceil\right) + c_2 & \text{se } u \geq 2 \end{cases}$$

dove c_1 e c_2 sono due costanti opportune. Applicando le regole di soluzione si ricava $T(u) = \Theta(u)$. Pertanto il tempo di calcolo complessivo è $\Theta(n)$.

Inoltre lo spazio di memoria richiesto è essenzialmente quello utilizzato dalla pila che implementa la ricorsione. Osserviamo che, su un input di n elementi, la pila raggiunge una altezza massima pari a $1 + \lceil \log_2 n \rceil$: questo è dovuto al fatto che ogni chiamata ricorsiva riduce della metà la dimensione $(j - i + 1)$ della porzione di input corrente. Poiché nel nostro caso ogni record di attivazione occupa uno spazio costante, lo spazio complessivo richiesto dall'algoritmo è $\Theta(\log n)$.

e) Assumiamo ora il criterio logaritmico. Per ipotesi sappiamo che ogni b_i è un intero di n bit. Applicando i ragionamenti svolti nel punto precedente il tempo di calcolo $T^\ell(u)$ risulta determinato da una equazione della forma

$$T^\ell(u) = \begin{cases} \Theta(n) & \text{se } u = 1 \\ T^\ell\left(\lfloor \frac{u}{2} \rfloor\right) + T^\ell\left(\lceil \frac{u}{2} \rceil\right) + \Theta(u) \cdot \Theta(n) & \text{se } u \geq 2 \end{cases}$$

Applicando ora la regola di soluzione (e considerando n come constante rispetto al parametro u) si ottiene

$$T^\ell(u) = \Theta(n) \cdot \Theta(u \log u)$$

Di conseguenza, il tempo di calcolo sull'input considerato risulta $\Theta(n^2 \log n)$.

Per quanto riguarda lo spazio di memoria, valutiamo lo spazio P necessario per mantenere la pila che implementa la ricorsione. Supponendo che n sia una potenza di 2, durante il funzionamento dell'algoritmo, P assume un valore massimo pari all'espressione seguente nella quale a è una costante opportuna e i termini di ordine minore sono trascurati:

$$P = a \cdot n \cdot \frac{n}{2} + a \cdot n \cdot \frac{n}{4} + a \cdot n \cdot \frac{n}{8} + \ldots = a \cdot n^2 \sum_{i=0}^{\log n} 2^{-i} = \Theta(n^2)$$

Quindi, secondo il criterio logaritmico, lo spazio richiesto è dell'ordine $\Theta(n^2)$. ∎

Esercizio 7.3

Dati tre interi $a, b, n \in \mathbb{N}$ considera l'espressione

$$F(a,b,n) = \begin{cases} 1 & \text{se } n = 0 \\ a^n + a^{n-1}b + a^{n-2}b^2 + \cdots + ab^{n-1} + b^n & \text{se } n \geq 1 \end{cases}$$

a) Descrivere un algoritmo del tipo *divide et impera* che calcola $F(a,b,n)$ su input a, b, n.

b) Assumendo il criterio di costo uniforme, valutare l'ordine di grandezza del tempo di calcolo e dello spazio di memoria richiesti in funzione del parametro n.

Svolgimento.

La soluzione che qui proponiamo si ispira alla classica procedura divide et impera per il calcolo della potenza di un numero reale. Essa permette di risolvere il problema in un tempo $O(\log^2 n)$ applicando semplicemente le definizioni e senza l'uso di strumenti relativamente complicati non previsti dal nostro programma. Avvertiamo tuttavia che tale soluzione non è l'unica possibile e neppure la più efficiente.

Anzitutto ricordiamo che il calcolo di una potenza di un intero può essere eseguito attraverso la seguente nota procedura che su input $x, y \in \mathbb{N}$ restituisce il valore x^y.

```
Procedure Potenza(x, y)
if y = 0 then return 1
        else
            begin
                u := ⌊y/2⌋;
                v := Potenza(x, u);
                v := v · v;
                if y pari then return v
                          else return v · x;
            end
```

Torniamo ora al problema di calcolare $F(a,b,n)$. Possiamo calcolare tale valore sfruttando la naturale simmetria della sommatoria che lo definisce. L'idea è quella di spezzare la sommatoria di $n+1$ elementi in due somme di (circa) $n/2$ elementi ciascuna; raccogliendo $a^{n/2}$ nella prima somma e $b^{n/2}$ nella seconda, in entrambi i casi il valore rimanente è quasi uguale a $F(a,b,n/2)$. In questo modo possiamo calcolare $F(a,b,n)$ conoscendo il valore $F(a,b,n/2)$ e possiamo definire una procedura ricorsiva del tipo divide et impera basata su una sola chiamata ricorsiva.

Più precisamente, si verifica che il legame tra $F(a,b,n)$ e $F(a,b,n/2)$ è il seguente:

1. se n è dispari (per cui $\lfloor \frac{n}{2} \rfloor = \frac{n-1}{2}$) allora

   $$F(a,b,n) = a^{\lfloor \frac{n}{2} \rfloor + 1} F(a,b,\lfloor n/2 \rfloor) + b^{\lfloor \frac{n}{2} \rfloor + 1} F(a,b,\lfloor n/2 \rfloor) = \left\{ a^{\lfloor \frac{n}{2} \rfloor + 1} + b^{\lfloor \frac{n}{2} \rfloor + 1} \right\} \cdot F(a,b,\lfloor n/2 \rfloor)$$

   nota che l'uguaglianza vale anche per $n = 1$;

2. se n è pari e maggiore di 0 allora

   $$F(a,b,n) = a^{\frac{n}{2}+1} F(a,b,\frac{n}{2}-1) + a^{\frac{n}{2}} b^{\frac{n}{2}} + b^{\frac{n}{2}+1} F(a,b,\frac{n}{2}-1) = \quad (7.1)$$

   $$= \left\{ a^{\frac{n}{2}+1} + b^{\frac{n}{2}+1} \right\} \cdot F(a,b,\frac{n}{2}-1) + a^{\frac{n}{2}} b^{\frac{n}{2}} \quad (7.2)$$

   nota che l'uguaglianza vale anche per $n = 2$.

L'algoritmo può essere descritto da un programma principale e da una procedura ricorsiva che riportiamo di seguito.

```
Procedure Main
begin
    leggi a, b e n
    r := Calcola(n)
    stampa r
end
```

```
Procedure Calcola(y)
if y = 0 then return 1
        else
            begin
                k := ⌊y/2⌋;
```

$\alpha :=$ Potenza(a, k);
$\beta :=$ Potenza(b, k);
$u := \alpha a + \beta b$;
if y dispari **then** $\begin{cases} t := \text{Calcola}(k) \\ \text{return } \; u \cdot t \end{cases}$
else $\begin{cases} t := \text{Calcola}(k-1) \\ \text{return } \; ut + \alpha\beta \end{cases}$

end

b) Assumendo il criterio di costo uniforme si verifica facilmente che il tempo di calcolo richiesto dalla procedura Potenza su input x, y è di ordine $\Theta(\log y)$. Denotiamo ora con $T(n)$ il tempo di calcolo richiesto dalla procedura Calcola su input n. Usando la definizione della procedura possiamo maggiorare $T(n)$ mediante la seguente relazione dove d è una costante positiva opportuna.

$$T(n) \leq \begin{cases} d & \text{se } n = 0 \\ d \log n + T(\lfloor n/2 \rfloor) & \text{altrimenti} \end{cases}$$

Di conseguenza

$$T(n) \leq \sum_{i=0}^{\lceil \log n \rceil} d \log \left(\frac{n}{2^i} \right) = d \left\{ \sum_{i=0}^{\lceil \log n \rceil} \log n - \sum_{i=0}^{\lceil \log n \rceil} i \right\} \leq (\frac{d}{2} + 1)(\log^2 n)$$

In modo analogo si prova che per una opportuna costante positiva f vale la relazione $T(n) \geq f(\log^2 n)$ e quindi si ottiene $T(n) = \Theta(\log^2 n)$.

Lo spazio richiesto dalla procedura Potenza su input x, y è dell'ordine $\Theta(\log y)$ e quindi anche lo spazio richiesto da Calcola su input n è dell'ordine $\Theta(\log n)$. ∎

Esercizio 7.4

Considera la sequenza $\{F(n)\}_n$ definita dalla seguente equazione :

$$F(n) = \begin{cases} n & \text{se } 0 \leq n \leq 2 \\ 2^n F\left(\lfloor \frac{n}{3} \rfloor\right) + F\left(\lceil \frac{n}{3} \rceil\right) & \text{se } n > 2 \end{cases}$$

a) Descrivere un algoritmo del tipo "divide et impera" per calcolare $F(n)$ su input $n \in \mathbb{N}$.

b) Valutare l'ordine di grandezza del tempo di calcolo e dello spazio di memoria richiesti sull'input n, assumendo il criterio di costo uniforme.

c) Svolgere l'esercizio precedente assumendo il criterio di costo logaritmico.

Svolgimento.

a) L'algoritmo consiste di due procedure. La prima calcola il valore 2^n su input $n \in \mathbb{N}$, la seconda calcola $F(n)$ sul medesimo input.

Procedure Potenza(n)
if $n = 0$ **then return** 1
else

```
begin
    k := ⌊n/2⌋;
    u := Potenza(k);
    if n pari then return u · u
              else return 2 · u · u;
end
```

```
Procedure F(n)
if n ≤ 2 then return n
     else
       begin
           i := ⌊n/3⌋;
           j := ⌈n/3⌉;
           u := F(i);
           v := F(j);
           p := Potenza(n);
           return p · u + v;
       end
```

b) È facile verificare che il tempo di calcolo e lo spazio di memoria richiesti dalla procedura Potenza su input n sono di ordine $\Theta(\log n)$ assumendo il criterio uniforme.

Denotiamo ora con $T(n)$ il tempo di calcolo richiesto dalla procedura F su input n. Per ogni n potenza di 3 $T(n)$ verifica la seguente equazione

$$T(n) = \begin{cases} c & \text{se } n \leq 2 \\ 2T\left(\frac{n}{3}\right) + c_1 \log n + c_2 & \text{altrimenti} \end{cases}$$

dove c, c_1 e c_2 sono costanti. Risolvendo l'equazione, con il metodo usato per il problema 3.4, si ottiene $T(n) = \Theta(n^{\log_3 2})$.

Si può inoltre verificare facilmente che lo spazio richiesto dalla procedura F su input n è $\Theta(\log n)$.

c) Assumiamo ora il criterio logaritmico e sia $T_P^\ell(n)$ il tempo richiesto dalla procedura Potenza su input n; per ogni n pari abbiamo $T_P^\ell(n) = T_P^\ell(n/2) + \Theta(n)$ e quindi si verifica

$$T_P^\ell(n) = \Theta\left(\sum_{k=0}^{\lceil \log n \rceil} \frac{n}{2^k}\right) = \Theta(n)$$

Denotiamo ora con $T_F^\ell(n)$ il tempo richiesto dalla procedura F su input n. Si verifica innanzitutto che la dimensione del valore $F(n)$ è di ordine $\Theta(n)$. Quindi possiamo scrivere la seguente equazione di ricorrenza per ogni n potenza di 3:

$$T_F^\ell(n) = \begin{cases} c & \text{se } n \leq 2 \\ 2T_F^\ell\left(\frac{n}{3}\right) + \Theta(n) & \text{altrimenti} \end{cases}$$

La soluzione dell'equazione fornisce il valore $T_F^\ell(n) = \Theta(n)$.

Per quanto riguarda lo spazio di memoria richiesto da F, osserviamo che $S_F^\ell(n) \geq |F(n)| = \Theta(n)$ e inoltre $S_F^\ell(n) \leq T_F^\ell(n) = \Theta(n)$; quindi anche lo spazio richiesto è di ordine $\Theta(n)$. ∎

Esercizio 7.5

Considera il problema di calcolare l'espressione

$$b = (a_1 + a_2) \cdot (a_2 + a_3) \cdots (a_{n-1} + a_n)$$

avendo in input una sequenza $a_1, a_2, \ldots, a_n$ di numeri interi.

a) Descrivere un algoritmo del tipo *divide et impera* per risolvere il problema.

b) Svolgere l'analisi del tempo di calcolo e dello spazio di memoria richiesti dall'algoritmo assumendo il criterio di costo uniforme.

c) Svolgere l'esercizio precedente assumendo il criterio di costo logaritmico supponendo che ogni a_i sia un intero di n bit.

Esercizio 7.6

Considera la seguente procedura che calcola il valore

$$F(\underline{a}) = 3 \cdot a_1 + 3^2 \cdot a_2 + \cdots + 3^n \cdot a_n$$

avendo in input una sequenza $\underline{a}$ di n interi, (ovvero $\underline{a} = (a_1, a_2, \ldots, a_n)$ con $a_i \in Z$ per ogni $i = 1, 2, \ldots, n$).

```
begin
        b = 1
        F = 0
        for i = 1, 2, ..., n do
            b = 3 · b
            F = F + a_i · b
        return F
end
```

a) Determinare l'ordine di grandezza (al variare di n) del tempo di calcolo e dello spazio di memoria richiesto dalla procedura assumendo il criterio di costo uniforme.

b) Svolgere l'esercizio precedente assumendo il criterio di costo logaritmico nell'ipotesi che $a_i \in \{1, 2, \ldots, n\}$ per ogni $i = 1, 2, \ldots, n$.

c) Descrivere un algoritmo del tipo *divide et impera* che esegue la medesima computazione.

d) Determinare l'ordine di grandezza del tempo di calcolo richiesto dall'algoritmo precedente assumendo il criterio di costo logaritmico.

Capitolo 8

Algoritmi greedy

Esercizio 8.1

Dato un grafo indiretto $G = \langle V, E \rangle$ (dove V è l'insieme dei nodi e E quello dei lati) definiamo la famiglia F_G di sottoinsiemi di E (matching) nel modo seguente:

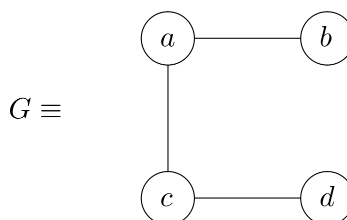
$$F_G = \{A \subseteq E \mid \forall u, v \in A,\ u \neq v \Rightarrow\ u \text{ e } v \text{ non hanno nodi in comune}\}$$

a) La coppia $\langle E, F_G \rangle$ forma un sistema di indipendenza?
b) La coppia $\langle E, F_G \rangle$ forma un matroide?
c) Dato un grafo indiretto $G = \langle V, E \rangle$ e una funzione peso $w : E \longrightarrow \mathbb{R}^+$, ogni insieme $A \subseteq E$ ammette un peso $w(A)$ definito da

$$w(A) = \sum_{x \in A} w(x).$$

Descrivere una procedura greedy che cerca di determinare un insieme $C \in F_G$ di peso massimo in F_G.
d) La soluzione prodotta dall'algoritmo è sempre ottima?
e) Assumendo il criterio di costo uniforme valutare il tempo di calcolo richiesto dall'algoritmo su un input G formato da n nodi e m lati.
Svolgimento.
a) La coppia $\langle E, F_G \rangle$ forma un sistema di indipendenza per ogni grafo G. Infatti, ogni insieme $A \in F_G$ è formato da lati che non hanno nodi in comune: di conseguenza ogni suo sottoinsieme gode della stessa proprietá e quindi appartiene a F_G.

b) In generale $\langle E, F_G \rangle$ non è un matroide. Consideriamo per esempio il grafo

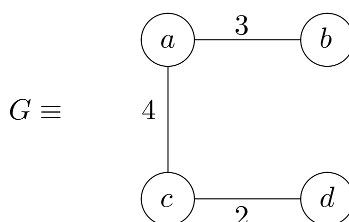
$G \equiv$ 

e definiamo $B = \{\{a,b\},\{c,d\}\}$, $A = \{\{a,c\}\}$. È immediato verificare che $A, B \in F_G$, $|B| = |A| + 1$ e, tuttavia, non esiste alcun $u \in B$ tale che $A \cup \{u\} \in F_G$.

c) L'algoritmo greedy è descritto dalla seguente procedura nella quale si suppone di poter marcare opportunamente i nodi del grafo di ingresso. Inizialmente nessun nodo è marcato.

```
begin
    S := ∅
    Q := E
    while Q ≠ ∅ do
        begin
            determina l'elemento u di peso massimo in Q
            cancella u da Q
            siano a e b gli estremi di u
            if a e b non sono marcati then { marca a e b
                                           { S := S ∪ {u}
        end
    return S
end
```

d) La soluzione fornita dall'algoritmo in generale non è ottima perché il sistema di indipendenza non sempre è un matroide. Per esempio consideriamo il grafo descritto al punto b) opportunamente pesato:



In questo caso la soluzione ottenuta dall'algoritmo è $\{\{a,c\}\}$ di peso 4, mentre quella ottimale è $\{\{a,b\},\{c,d\}\}$ di peso 5.

e) Definendo Q come una coda di priorità (per esempio uno heap), sono necessari $\Theta(m \log m)$ passi per determinare i valori di minimo e per eseguire le cancellazioni. Inoltre occorrono $O(n)$ passi per marcare opportunamente i nodi del grafo e per aggiornare la soluzione S. In totale l'algoritmo richiede quindi un tempo $O(n + m \log m)$. ∎

Esercizio 8.2

Dati due interi k, n, $1 \leq k \leq n$, sia E_n l'insieme dei primi n interi positivi:

$$E_n = \{1, 2, \ldots, n\},$$

e sia F_k la famiglia dei sottoinsiemi di E_n che hanno al più k elementi:

$$F_k = \{A \subseteq E_n \mid \sharp A \leq k\}.$$

a) Per quali interi k, n, $1 \leq k \leq n$, la coppia $\langle E_n, F_k \rangle$ forma un sistema di indipendenza?
b) Per quali interi k, n, $1 \leq k \leq n$, la coppia $\langle E_n, F_k \rangle$ forma un matroide?
c) Definire un algoritmo greedy per il problema seguente:

Istanza : due interi k, n, $1 \leq k \leq n$, e una funzione peso $w : E_n \to \mathbb{R}^+$.
Soluzione : il sottoinsieme $A \in F_k$ di peso massimo.

d) Per quali k e n l'algoritmo determina sempre la soluzione ottima?

Esercizio 8.3

Dato un grafo non orientato $G = \langle V, E \rangle$ nel quale V è l'insieme dei nodi ed E quello degli archi, definiamo la seguente famiglia di sottoinsiemi di E:

$$F = \{A \subseteq E \mid \exists v \in V \text{ tale che ogni lato } \alpha \in A \text{ è incidente a } v\}$$

Per ipotesi assumiamo $\emptyset \in F$.
a) La coppia $\langle E, F \rangle$ forma un sistema di indipendenza?
b) La coppia $\langle E, F \rangle$ forma un matroide?
c) Definire un algoritmo greedy per il problema seguente:

Istanza : un grafo non orientato $G = \langle V, E \rangle$ e una funzione peso $w : E \to \mathbb{R}^+$.
Soluzione : il sottoinsieme $A \in F$ di peso massimo.

d) Assumendo il criterio di costo uniforme, svolgere l'analisi dell'algoritmo supponendo che il grafo di ingresso abbia n nodi e m lati.
e) Mostrare con un esempio che l'algoritmo descritto non è ottimale.

Esercizio 8.4

Dato un grafo non orientato $G = \langle V, E \rangle$ nel quale V è l'insieme dei nodi ed E quello degli archi, denotiamo con F_G la famiglia delle clique di G, ovvero:

$$F_G = \{A \subseteq V \mid \forall a, b \in A, a \neq b, \Longrightarrow \{a, b\} \in E\}$$

a) La coppia $\langle V, F_G \rangle$ forma un sistema di indipendenza?
b) La coppia $\langle V, F_G \rangle$ forma un matroide?
c) Definire un algoritmo greedy per il problema seguente:

Istanza : un grafo non orientato $G = \langle V, E \rangle$ e una funzione peso $w : V \to \mathbb{R}^+$.
Soluzione : il sottoinsieme $A \in F_G$ di peso massimo.

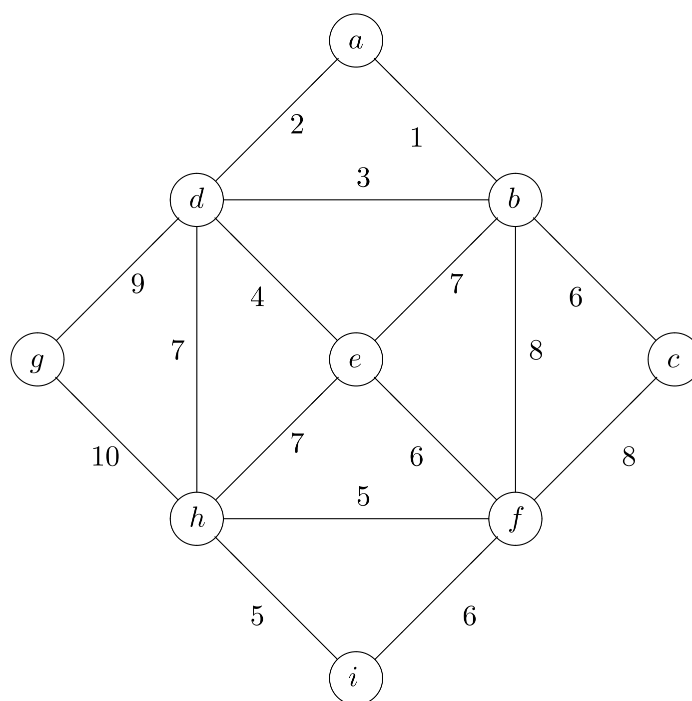
d) Assumendo il criterio di costo uniforme, svolgere l'analisi dell'algoritmo supponendo che il grafo di ingresso abbia n nodi e m lati.
e) Mostrare con un esempio che l'algoritmo descritto non è ottimale.

Capitolo 9

Algoritmi particolari

Esercizio 9.1

Applicare l'algoritmo di Kruskal al grafo pesato descritto in figura. Mostrare le operazioni eseguite sulle varie strutture dati usate dall'algoritmo.



Soluzione (Cenni)

Sia $G = \langle V, E \rangle$ il grafo indicato in figura. L'algoritmo di Kruskal gestisce una partizione P sull'insieme dei nodi V, una coda di priorità Q sull'insieme dei lati E pesati, e un insieme T di lati che fornisce la soluzione.

Inizialmente $P = \{\{a\}, \{b\}, \ldots, \{i\}\}$, Q contiene tutti gli elementi di E e $T = \emptyset$. L'esecuzione del ciclo principale dell'algoritmo consiste nel determinare l'elemento ℓ di peso minimo in Q, cancellare ℓ da Q, eseguire FIND sugli estremi di ℓ nella partizione P e, se i due valori ottenuti sono diversi, eseguire la corrispondente operazione UNION su P aggiungendo ℓ a T.

Nelle righe della seguente tabella riportiamo il risultato delle operazioni compiute nell'esecuzione di ciascun ciclo. $find_1$ e $find_2$ indicano le operazioni FIND sui due estremi di ℓ.

$\ell = \min(Q)$	$Q = \text{delete}(\ell, Q)$	$find_1$	$find_2$	P	T
$\{a,b\}$	$E\backslash\{\{a,b\}\}$	a	b	$\{\{a,b\},\{c\},\{d\},\ldots,\{i\}\}$	$\{\{a,b\}\}$
$\{a,d\}$	$E\backslash\{\{a,b\},\{a,d\}\}$	a	d	$\{\{a,b,d\},\{c\},\{e\},\ldots,\{i\}\}$	$\{\{a,b\},\{a,d\}\}$
...	...	...	...	...	...

Esercizio 9.2

Esegui l'algoritmo *Quicksort* sulla sequenza

$$5, 2, 3, 1, 6, 8, 3, 9, 8, 7$$

scegliendo sempre come pivot il primo elemento del vettore considerato e applicando la versione iterativa che minimizza lo spazio di memoria richiesto. Si mettano in evidenza gli scambi compiuti e le operazioni eseguite sulla pila.

Esercizio 9.3

Esegui l'algoritmo *Quicksort* sulla sequenza

$$4, 5, 2, 6, 7, 1, 6, 3, 9$$

scegliendo sempre come pivot il primo elemento del vettore considerato e mettendo in evidenza i confronti e gli scambi compiuti. Applicare poi sullo stesso input la versione iterativa dell'algoritmo mostrando il comportamento della pila durante l'esecuzione.

Esercizio 9.4

Costruire il codice di Huffman di una sequenza di 100 caratteri definita sull'alfabeto $\{a, b, c, d, e, f, g\}$, nella quale i vari simboli compaiono con la seguente frequenza: 35 occorrenze di a, 20 di b, 5 di c, 30 di d, 5 di e, 2 di f, e infine 3 occorrenze di g.

Qual è la lunghezza della codifica dell'intera sequenza?

Esercizio 9.5

Eseguire la seguente sequenza di operazioni *Union-Find* sull'insieme $\{2, 3, 4, 5, 6, 7, 8, 9\}$ a partire dalla partizione iniziale (identità) usando una rappresentazione ad albero con bilanciamento: Union$(3, 4)$, Union$(5, 6)$, Union$(3, 5)$, Union$(2, 3)$, Union$(9, 7)$, Union$(7, 8)$, Find(3), Find(6), Union$(7, 3)$, Find(9).

Per convenzione, nell'esecuzione delle operazioni Union, a parità di dimensione dei due alberi, la radice del nuovo insieme sia quella di minor valore. Si metta in evidenza la foresta che si ottiene dopo l'esecuzione di ogni operazione.

Esercizio 9.6

Esegui l'algoritmo *Heapsort* sulla sequenza

$$4, 1, 2, 0, 5, 7, 2, 8, 7, 6$$

mettendo in evidenza i confronti e gli scambi compiuti.

Esercizio 9.7

Applicare Mergesort alla sequenza di interi

$$8, 3, 4, 2, 5, 6, 7, 1, 10, 9$$

mettendo in evidenza i confronti eseguiti.

Qual è l'altezza massima raggiunta dalla pila che implenta la ricorsione?

Esercizio 9.8

a) Determinare l'albero di ricerca binaria che si ottiene inserendo la seguente sequenza di lettere in un albero inizialmente vuoto (si adotti il tradizionale ordinamento alfabetico):

$$g, b, n, f, r, s, h, l, a, c, d, i$$

b) Determinare l'albero di ricerca binaria che si ottiene dall'albero precedente cancellando nell'ordine i seguenti elementi:

$$g, n, r, d, h$$

Appendice A

Temi d'esame svolti

ALGORITMI E STRUTTURE DATI

Svolgimento della prova scritta del 6.9.2006

Esercizio 1.

Determinare, al crescere di $n \in \mathbb{N}$ a $+\infty$, l'espressione asintotica di

$$\sum_{i=0}^{\lfloor \log n \rfloor} \log \frac{n^2}{2^i}$$

supponendo che il logaritmo sia in base 2.

Soluzione proposta

Per le proprietà dei logaritmi possiamo scrivere

$$\sum_{i=0}^{\lfloor \log n \rfloor} \log \frac{n^2}{2^i} = \sum_{i=0}^{\lfloor \log n \rfloor} (2\log n - i) = 2(\log n)(\lfloor \log n \rfloor + 1) - \sum_{i=0}^{\lfloor \log n \rfloor} i \qquad \text{(A.1)}$$

Ricordiamo che, per ogni $k \in \mathbb{N}$, la somma dei primi k interi positivi è $k(k+1)/2$, ovvero $\sum_{i=1}^{k} i = \frac{k(k+1)}{2}$ (vedi per esempio la sezione 2.4.2 delle dispense). Quindi l'ultima sommatoria nell'equazione (A.1) è asintotica a $\lfloor \log n \rfloor^2/2$.

Inoltre, poiché $\log x = \lfloor \log x \rfloor + \varepsilon$ con $0 \leq \varepsilon < 1$ (per ogni $x > 0$), vale la relazione $\lfloor \log n \rfloor \sim \log n$. Di conseguenza l'espressione ottenuta in (A.1) è asintotica a

$$2(\log n)^2 - \frac{(\log n)(1 + \log n)}{2} \sim \frac{3}{2}(\log n)^2$$

Esercizio 2.

Dato un albero con radice T, con insieme dei nodi V, e una funzione peso $p : V \longrightarrow \mathbb{R}$, chiamiamo *costo* di un nodo $v \in V$ il valore

$$c(v) = \begin{cases} p(v) & \text{se } v \text{ e' la radice} \\ p(v) + c(padre(v)) & \text{altrimenti} \end{cases}$$

a) Definire un algoritmo per il seguente problema:

Istanza : un albero con radice T, rappresentato mediante liste di adiacenza, e per ogni nodo v di T il peso $p(v) \in \mathbb{R}$.

Soluzione : l'insieme delle foglie di T che hanno costo minimo.

b) Svolgere l'analisi dell'algoritmo valutando il tempo di calcolo e lo spazio di memoria richiesti in funzione del numero n di nodi dell'albero (si assuma il criterio uniforme).

Soluzione proposta

Il problema può essere risolto con una visita in preordine dell'albero T nella quale, per ogni nodo v diverso dalla radice, il valore $c(v)$ viene calcolato conoscendo il costo del padre di v. Descriviamo quindi un programma principale che richiama una procedura ricorsiva, chiamata

Visita($\cdot$), che calcola i valori $c(v)$ di ogni nodo v diverso dalla radice. Tale procedura calcola anche una lista F contenente le foglie di T accompagnate dal relativo costo e determina il valore minimo m di tali costi. Il programma principale poi scorrerà la lista F e determinerà l'elenco delle foglie che hanno costo m. Nella sua computazione la chiamata Visita(v) mantiene, oltre alle variabili globali F e m, anche la variabile globale C che rappresenta il costo $c(v)$ del nodo corrente v.

Nella descrizione che segue supponiamo che V sia l'insieme dei nodi di T, r sia la radice di T e, per ogni nodo $v \in V$, $L(v)$ sia la lista dei figli di v.

```
Procedure Main
begin
        leggi e memorizza l'input
        m := +∞
        F := Λ (lista vuota)
        C := p(r)
        Visita(r)
        S := Λ (lista vuota)
        for (v, x) ∈ F do
                        if x = m then S := Inserisci_in_testa(S, v)
        return S
end

Procedure Visita(v)
begin
        if L(v) = Λ (ovvero v e' una foglia)
                then { F := Inserisci_in_testa(F, (v, C))
                     { if C < m then m := C
                else  for w ∈ L(v) do { C := C + p(w)
                                      { Visita(w)
        C := C − p(v)
end
```

b) Valutiamo ora il tempo di calcolo richiesto dalla procedura assumendo il criterio di costo uniforme. La procedura Visita viene chiamata esattamente una volta su ogni nodo dell'albero e, per ogni nodo, esegue un numero di operazioni limitato da una costante. Quindi il tempo richiesto dalla visita in preordine dell'albero è $\Theta(n)$. Anche il tempo necessario per leggere e memorizzare l'input è $\Theta(n)$, mentre le altre operazioni eseguite dal Main richiedono un tempo $O(n)$. Quindi il tempo complessivo richiesto dall'algoritmo è $\Theta(n)$.

Per quanto riguarda lo spazio di memoria l'algoritmo richiede uno spazio $\Theta(n)$ per mantenere l'input (cioé l'albero T). L'esecuzione della chiamata Visita(r) richiede uno spazio $O(n)$ per mantenere la pila che implementa la ricorsione. Una analoga quantità è richiesta per mantenere le liste F e S. Quindi anche lo spazio di memoria richiesto è dell'ordine $\Theta(n)$.

Esercizio 3.

a) Descrivere un algoritmo del tipo *divide et impera* che su input $n, k, a_1, a_2, \ldots, a_n \in \mathbb{N}$ calcoli il valore

$$a_1^k + a_2^k + \cdots + a_n^k \ .$$

b) Assumendo il criterio di costo uniforme, valutare l'ordine di grandezza del tempo di calcolo e dello spazio di memoria richiesti in funzione dei parametri n e k.

Soluzione proposta

a) L'algoritmo consiste di due procedure. La prima calcola il valore a^k su input $a, k \in \mathbb{N}$, la seconda permette il calcolo del valore richiesto.

```
Procedure Potenza(a, k)
if k = 0 then return 1
        else
            begin
                t := ⌊k/2⌋;
                u := Potenza(a, t);
                if k pari  then return u · u
                           else return a · u · u;
            end
```

```
Procedure F(i, j)
                 ⎧ a := read(a_i)
if i = j then   ⎨ b := Potenza(a, k)
                 ⎩ return b;
       else
           begin
               t := ⌊(i+j)/2⌋;
               u := F(i, t);
               v := F(t + 1, j);
               return u + v;
           end
```

L'algoritmo si riduce quindi alla lettura dei parametri k, n e alla chiamata di $F(1, n)$.

b) È facile verificare che il tempo di calcolo richiesto dalla procedura Potenza su input a, k è di ordine $\Theta(\log k)$ assumendo il criterio uniforme. Infatti, denotando con $P(k)$ tale quantità si verifica che

$$P(k) = \begin{cases} O(1) & \text{se } k = 0 \\ P(\lfloor k/2 \rfloor) + O(1) & \text{se } k > 0 \end{cases}$$

Quindi esistono due costanti positive r, s tali che, per ogni k potenza di 2, si può scrivere

$$P(k) = r + P(k/2) = 2r + P(k/4) = 3r + P(k/8) = \cdots = jr + P(k/2^j) = \cdots = (\log_2 k)r + s$$

Questo prova $P(k) = \Theta(\log k)$.

Denotiamo ora con $T(u)$ il tempo di calcolo richiesto dalla chiamata $F(i, j)$ tale che $u = j - i + 1$. Per ogni u potenza di 2, $T(u)$ verifica la seguente equazione

$$T(u) = \begin{cases} O(\log k) & \text{se } u = 1 \\ 2T\left(\frac{u}{2}\right) + O(1) & \text{se } u > 1 \end{cases}$$

Sviluppando l'equazione si ottiene (per due costant opportune c, d)

$$T(u) = c\left(\sum_{i=0}^{\log u - 1} 2^i\right) + duO(\log k) = O(u \log k)$$

Ponendo $u = n$ si ottiene quindi il tempo complessivo $O(n \log k)$.

Si può inoltre verificare che lo spazio richiesto dalle due procedure ricorsive è principalmente dovuto al mantenimento della pila che implementa la ricorsione. Esso è quindi dell'ordine $\Theta(\log n) + \Theta(\log k)$.

ALGORITMI E STRUTTURE DATI

Svolgimento della prova scritta del 7.2.2006

Esercizio 1.

Determinare, in funzione di $n \to +\infty$, l'espressione asintotica della somma

$$\sum_{i=0}^{n-1} \frac{i}{n-i}$$

giustificando la risposta.

Soluzione

Poiché il termine $n-i$ a denominatore è scomodo, operiamo la sostituzione $j = n-i$. Ovvero, chiamiamo j il valore $n-i$ (e quindi $i = n-j$). Di conseguenza si ottiene

$$\sum_{i=0}^{n-1} \frac{i}{n-i} = \sum_{j=1}^{n} \frac{n-j}{j} = \sum_{j=1}^{n} \frac{n}{j} - \sum_{j=1}^{n} 1 = n \sum_{j=1}^{n} \frac{1}{j} - n$$

Dobbiamo quindi valutare $\sum_{j=1}^{n} 1/j$. Usando l'approssimazione mediante integrali è facile verificare che $\sum_{j=1}^{n} 1/j \sim \log n$. Infatti, la funzione $1/x$ tra 1 e n è continua, positiva e monotona decrescente, per cui abbiamo

$$\int_1^{n+1} \frac{1}{x} dx \leq \sum_{j=1}^{n} \frac{1}{j} \leq 1 + \int_1^{n} \frac{1}{x} dx$$

che implica appunto $\sum_{j=1}^{n} 1/j = (\log n)(1 + o(1))$. Sostituendo questo valore nella equazione precedente si ricava

$$\sum_{i=0}^{n-1} \frac{i}{n-i} \sim n \log n.$$

Esercizio 2.

Dato un intero positivo n, diamo le seguenti definizioni:

1. sia $U = \{1, 2, \ldots, n\}$;
2. sia $c : U \longrightarrow \mathbb{Z}$ una funzione costo, dove $\mathbb{Z}$ è l'insieme dei numeri interi relativi;
3. per ogni $A \subseteq U$, sia $c(A) = \sum_{i \in A} c(i)$;
4. dato un valore $H > 0$, sia $C_H = \{A \subseteq U \mid c(A) \leq H\}$.

Rispondere alle seguenti domande giustificando le risposte:

a) La coppia $\langle U, C_H \rangle$ forma un sistema di indipendenza?

b) Supponiamo ora che la funzione c assuma solo valori positivi, cioè $c : U \longrightarrow \mathbb{N}$. In questo caso $\langle U, C_H \rangle$ è un sistema di indipendenza? Nelle stesse ipotesi $\langle U, C_H \rangle$ è un matroide?

c) Consideriamo ora una nuova funzione valore $v : U \longrightarrow \mathbb{N}$ e supponiamo sempre che c assuma solo valori positivi. Definire una procedura greedy per determinare un sottoinsieme di U di valore massimo in C_H su input n, H, c, v.

d) L'algoritmo determina sempre un elemento di C_H di valore massimo?

e) Assumendo il criterio di costo uniforme, valutare in funzione di n l'ordine di grandezza del tempo di calcolo richiesto dall'algoritmo nel caso peggiore.

Soluzione

a) In generale la coppia (U, C_H) non forma un sistema di indipendenza perché la funzione c può assumere valori negativi e quindi se un insieme A appartiene a C_H (e di conseguenza $c(A) \leq H$), non è detto che i suoi sottoinsiemi soddisfino la stessa proprietà (per esempio, banalmente, se $c(1) = \lfloor H \rfloor + 1$ e $c(2) = -2$, allora l'insieme $\{1, 2\}$ appartiene a C_H mentre l'insieme $\{1\}$ non vi appartiene).

b) Se c assume solo valori in $\mathbb{N}$ allora (U, C_H) è sempre un sistema di indipendenza perché se A appartiene a C_H ogni suo sottoinsieme appartiene a C_H.

In generale invece (U, C_H) non è un matroide. Per esempio, supponiamo che $H = 8$, $c(1) = 5$, $c(2) = c(3) = 4$; allora gli insiemi $B = \{2, 3\}$ e $A = \{1\}$ sono in C_H, ma non è possibile trovare in B un elemento b tale che $A \cup \{b\}$ appartenga a C_H.

c) Un semplice algoritmo per il problema proposto è definito dalla seguente procedura

```
Procedure GREEDY(n, H, c, v)
begin
    S := ∅
    Q := {1, 2, ..., n}
    t := 0
    while Q ≠ ∅ do
            begin
                determina l'elemento k in Q tale che v(k) sia massimo
                togli k da Q
                if t + c(k) ≤ H then { S := S ∪ {k}
                                     { t := t + c(k)
            end
    return S
end
```

d) Poiché (U, C_H) non è un matroide, per il teorema di Rado l'algoritmo Greedy non determina sempre la soluzione ottima.

e) Per implementare l'algoritmo possiamo rappresentare Q mediante una coda di priorità , per esempio uno heap rovesciato. In questo caso la costruzione dello heap (implicita nella inizializzazione di Q) richiede un tempo $\Theta(n)$, la scelta dell'elemento massimo in Q e la sua cancellazione richiedono un tempo $\Theta(\log n)$ nel caso peggiore, l'esecuzione della istruzione `if` $t + c(k) \leq H$ `then` ... richiede tempo $\Theta(1)$. Complessivamente otteniamo quindi un tempo $\Theta(n \log n)$ nel caso peggiore.

Un'altra possibile implementazione prevede la rappresentazione di Q come vettore di n elementi; possiamo quindi inizialmente ordinare tali elementi secondo il valore di v e, successivamente, eseguire il ciclo `for` scorrendo il vettore con un puntatore a partire dall'elemento di

valore massimo. Usando un algoritmo ottimale di ordinamento (come heapsort o mergesort) l'intera procedura richiede anche in questo caso un tempo $\Theta(n \log n)$ nel caso peggiore.

ALGORITMI E STRUTTURE DATI
Svolgimento della prova scritta del 1.4.2003

Esercizio 1.

Considera le sequenze $\{f_n\}$ e $\{g_n\}$ così definite:

$$f_n = 7n^2 \log n + n \log n$$

$$g_n = \begin{cases} n^2 + 5n & \text{se } n \text{ è pari} \\ n\lfloor \log n \rfloor + 3n\lfloor n \rfloor & \text{se } n \text{ è dispari} \end{cases}$$

Tra le seguenti relazioni determinare le relazioni vere e quelle false:

$$1) f_n \sim n^2 \log n, \quad 2) f_n = \Theta(n^2), \quad 3) f_n = O(n^3), \quad 4) f_n = o(n^2),$$

$$5) g_n \sim n \log n, \quad 6) g_n = o(n^3), \quad 7) g_n = O(n^2), \quad 8) g_n = \Theta(n^2).$$

Soluzione

1) falsa, 2) falsa, 3) vera, 4) falsa, 5) falsa, 6) vera, 7) vera, 8) falsa.

Esercizio 2.

a) Descrivere un algoritmo per risolvere il seguente problema:

Istanza : un albero binario T di n nodi rappresentato mediante vettori *sin* e *des* e un intero k, $0 \leq k \leq n-1$.
Soluzione : il numero di nodi di T che hanno altezza k.

b) Assumendo il criterio di costo uniforme, valutare in funzione di n il tempo di calcolo e lo spazio di memoria richiesti dall'algoritmo nel caso peggiore.

Soluzione

a)

Ricordiamo che in un albero binario T, l'altezza di un nodo v è è la massima distanza di v da una foglia. Vogliamo descrivere un algoritmo che calcoli l'altezza di v, per ogni nodo v dell'albero T. È chiaro che l'altezza di una foglia è 0; invece, l'altezza di un nodo interno v si calcola facilmente una volta note le altezze dei suoi figli: è sufficiente considerare la massima tra queste e incrementarla di 1 Quindi, per risolvere il nostro problema è sufficiente descrivere un algoritmo che esegue una visita di T in ordine posticipato.

A tale scopo, rappresentiamo con r la radice dell'albero e, per ogni nodo v, denotiamo con $sin(v)$ e $des(v)$ il figlio sinistro e destro di v, rispettivamente. Se v non possiede figlio sinistro allora $sin(v) = 0$ e lo stesso vale per il figlio destro. Definiamo una procedura ricorsiva Visita(v) che, su input v, restituisce l'altezza di v. Tale procedura restituisce il valore $d = 0$ se v è una foglia; altrimenti richiama se stessa sui figli di v determinando il valore massimo ottenuto, incrementa di 1 tale valore e lo assegna alla variabile d. Se inoltre d coincide con k si incrementa un opportuno contatore rappresentato da una variabile globale S (inizialmente posta a 0). Infine, la procedura Visita(v) restituisce proprio d e il valore finale di S sarà l'output dell'algoritmo.

Il programma principale, dopo aver letto l'input e inizializzato S, richiama la procedura sulla radice r. Le variabili S e k sono qui considerate variabili globali.

```
begin
    leggi e memorizza l'input
    S := 0
    u := Visita(r)
    return S
end

Procedure Visita(v)
begin
    if sin(v) ≠ 0
            then a := Visita(sin(v))
            else a := −1
    if des(v) ≠ 0
            then b := Visita(des(v))
            else b := −1
    if a ≤ b
            then d := 1 + b
            else d := 1 + a
    if d = k
            then S := S + 1
    return d;
end
```

b) Il tempo di calcolo richiesto è $\Theta(n)$, dove n è il numero di nodi, poiché l'algoritmo esegue un numero limitato di istruzioni RAM per ciascun vertice dell'albero. Anche lo spazio di memoria richiesto è dello stesso ordine di grandezza poiché l'input richiede uno spazio $\Theta(n)$ per essere mantenuto in memoria e la pila che implementa la ricorsione raggiunge una altezza massima $O(n)$.

Esercizio 3.

Considera il problema di calcolare l'espressione

$$(a_1 \cdot a_2 + a_2 \cdot a_3 + \cdots + a_{n-1} \cdot a_n) \pmod{k}$$

assumendo come input una sequenza di interi $a_1, a_2, \ldots, a_n$ preceduta dal valore $k \geq 1$ e dallo stesso parametro $n \geq 2$ ($k, n \in N$).

a) Descrivere un algoritmo del tipo *divide et impera* per risolvere il problema.

b) Assumendo il criterio di costo uniforme, valutare l'ordine di grandezza del tempo di calcolo e dello spazio di memoria richiesti su un input di lunghezza n.

c) Assumiamo il criterio di costo logaritmico e supponiamo che ogni a_i, $i = 1, 2, \ldots, n$, sia un intero di m bits. Determinare una stima O-grande del tempo di calcolo e dello spazio di memoria in funzione dei parametri n, m e k.

Soluzione

a) L'algoritmo consiste di un programma principale che richiama una procedura per il calcolo del valore richiesto dopo aver letto i parametri k, n e il primo valore a_1 che viene assegnato ad

una variabile a. Nel seguito k, n e a sono considerate variabili globali e si suppone $n \geq 2$. Il programma principale può essere descritto nel modo seguente:

begin
$k :=$ read k
$n :=$ read n
$a :=$ read a_1
$f :=$ Calcola$(1, n)$
write f
end

La procedura Calcola su input (i, j), con $1 \leq i \leq j \leq n$, restituisce il valore dell'espressione richiesta tra gli indici i e j. Tale procedura è chiaramente del tipo *divide et impera*. Da notare che, per calcolare l'espressione richiesta, è necessario richiamare la procedura sugli intervalli (i, t) e (t, j), dove $t := \lfloor \frac{i+j}{2} \rfloor$: in questo caso dunque i due intervalli sono parzialmente sovrapposti.

Procedure Calcola(i, j)

if $i + 1 = j$ **then** $\begin{cases} b := \text{read}(a_j) \\ u := a \cdot b \\ v := u (\text{mod } k) \\ a := b \\ \text{return } v \end{cases}$
else
begin
$t := \lfloor \frac{i+j}{2} \rfloor$;
$u :=$ Calcola(i, t);
$v :=$ Calcola(t, j);
$s := u + v (\text{mod } k)$;
return s;
end

Denotiamo ora con $T(u)$ il tempo di calcolo richiesto dalla procedura Calcola(i, j) tale che $u = j - i$. Per ogni u potenza di 2, $T(u)$ verifica la seguente equazione

$$T(u) = \begin{cases} O(1) & \text{se } u = 1 \\ 2T\left(\frac{u}{2}\right) + O(1) & \text{se } u > 1 \end{cases}$$

Sviluppando l'equazione si ottiene (per due costant opportune c d)

$$T(u) = c \left(\sum_{i=0}^{\log u - 1} 2^i \right) + du = \Theta(u)$$

La valutazione rimane anche per valori di u che non sono potenze di 2. Ponendo $u = n - 1$ si ottiene quindi il tempo complessivo $\Theta(n)$.

Si può inoltre verificare che lo spazio richiesto dalla procedura è principalmente dovuto al mantenimento della pila che implementa la ricorsione e quindi dell'ordine $\Theta(\log n)$.

c) Assumendo il criterio di costo logaritmico, denotiamo con $T^\ell(u)$ il tempo richiesto da Calcola(i, j) con $u = j - i$. Per ogni u potenza di 2, si verifica che

$$T^\ell(u) = \begin{cases} O(m + \log k + \log n) & \text{se } u = 1 \\ 2T^\ell\left(\frac{u}{2}\right) + O(\log n + \log k) & \text{se } u > 1 \end{cases}$$

dalla quale si ricava

$$T^\ell(u) = O(\log n + \log k) \left(\sum_{i=0}^{\log u - 1} 2^i \right) + uO(m + \log k + \log n)$$

Chiaramente la valutazione ottenuta è valida anche per valori di u diversi dalle potenze di 2. Ponendo così $u = n - 1$ si ottiene un tempo complessivo $O(n(m + \log n + \log k))$.

Per quanto riguarda lo spazio di memoria, notiamo che il record di attivazione di ciascuna chiamata ricorsiva occupa uno spazio $O(m + \log n + \log k)$. Poiché l'altezza massima raggiunta dalla pila è $O(\log n)$, otteniamo una valutazione complessiva $O(\log^2 n + (m + \log k) \log n)$.

ALGORITMI E STRUTTURE DATI
Svolgimento della prova scritta del 4.2.2003

Esercizio 1.

Eseguire l'algoritmo Heapsort sull'input

$$4, 5, 2, 6, 7, 0, 5, 3, 8, 9$$

mettendo in evidenza gli scambi eseguiti.

.... Facile: consultare le dispense.....

La sequenza delle coppie scambiate è comunque la seguente: 7-9, 6-8, 2-5, 5-9, 5-7, 4-9, 4-8, 4-6 (a questo punto la costruzione dello heap è terminata), 5-9, 8-5, 7-5, 8-4, 7-4, 4-6, 7-3, 3-6, 3-5, 6-2, 2-5, 5-0, 0-5, 0-4, 5-3, 3-4, 4-0, 0-3, 3-2, 2-0.

Esercizio 2.

Considera il problema di calcolare l'espressione

$$a_1 + 2a_2 + 3a_3 + \cdots + na_n (\text{mod } k)$$

assumendo come input una sequenza di interi $a_1, a_2, \ldots, a_n$ preceduta dal valore $k \geq 1$ e dallo stesso parametro $n \geq 1$ ($k, n \in N$).

a) Descrivere un algoritmo del tipo *divide et impera* per risolvere il problema.

b) Assumendo il criterio di costo uniforme, valutare l'ordine di grandezza del tempo di calcolo e dello spazio di memoria richiesti su un input di lunghezza n.

c) Assumiamo il criterio di costo logaritmico e supponiamo che ogni a_i, $i = 1, 2, \ldots, n$, sia un intero di m bits. Determinare una stima O-grande del tempo di calcolo e dello spazio di memoria in funzione dei parametri n, m e k.

Soluzione

a) L'algoritmo consiste di un programma principale per la lettura dei parametri k e n e di una procedura per il calcolo del valore richiesto. Il programma principale può essere descritto nel modo seguente:

```
begin
    read k;
    read n;
    f := Calcola(1, n);
    write f;
end
```

La procedura Calcola su input (i, j), con $1 \leq i \leq j \leq n$, restituisce invece il valore dell'espressione richiesta tra gli indici i e j. Tale procedura è chiaramente del tipo *divide et impera* e può essere descritta come segue:

Procedure Calcola(i, j)

if $i = j$ **then** $\begin{cases} a := \text{read}(a_i) \\ u := a \cdot i; \\ v := u (\text{mod } k); \\ \text{return } v \ ; \end{cases}$

else

begin

$t := \lfloor \frac{i+j}{2} \rfloor$;

$u :=$ Calcola(i, t);

$v :=$ Calcola$(t + 1, j)$;

$s := u + v (\text{mod } k)$;

return s;

end

Denotiamo ora con $T(u)$ il tempo di calcolo richiesto dalla procedura Calcola(i, j) tale che $u = j - i + 1$. Per ogni u potenza di 2, $T(u)$ verifica la seguente equazione

$$T(u) = \begin{cases} O(1) & \text{se } u = 1 \\ 2T\left(\frac{u}{2}\right) + O(1) & \text{se } u > 1 \end{cases}$$

Sviluppando l'equazione si ottiene (per due costant opportune c d)

$$T(u) = c \left(\sum_{i=0}^{\log u - 1} 2^i \right) + du = \Theta(u)$$

La valutazione rimane anche per valori di u che non sono potenze di 2. Ponendo $u = n$ si ottiene quindi il tempo complessivo $\Theta(n)$.

Si può inoltre verificare che lo spazio richiesto dalla procedura è principalmente dovuto al mantenimento della pila che implementa la ricorsione e quindi dell'ordine $\Theta(\log n)$.

c) Assumendo il criterio di costo logaritmico, denotiamo con $T^\ell(u)$ il tempo richiesto da Calcola(i, j) con $u = j - i + 1$. Per ogni u potenza di 2, si verifica che

$$T^\ell(u) = \begin{cases} O(m + \log k + \log n) & \text{se } u = 1 \\ 2T^\ell\left(\frac{u}{2}\right) + O(\log n + \log k) & \text{se } u > 1 \end{cases}$$

dalla quale si ricava

$$T^\ell(u) = O(\log n + \log k) \left(\sum_{i=0}^{\log u - 1} 2^i \right) + uO(m + \log k + \log n)$$

Chiaramente la valutazione ottenuta è valida anche per valori di u diversi dalle potenze di 2. Ponendo così $u = n$ si ottiene un tempo complessivo $O(n(m + \log n + \log k))$.

Esercizio 3.

a) Descrivere un algoritmo per risolvere il seguente problema:

Istanza : un albero ordinato T di n nodi rappresentato mediante liste di adiacenza e un intero k, $0 \leq k \leq n - 1$.

Soluzione : il numero di nodi di T che hanno k discendenti.

b) Assumendo il criterio di costo uniforme, valutare in funzione di n il tempo di calcolo e lo spazio di memoria richiesti dall'algoritmo nel caso peggiore.

Soluzione

a)

Ricordiamo che in un albero ordinato T, un nodo z è discendente di un nodo v se $z \neq v$ e z si trova nel sottoalbero di T che ha per radice v. Vogliamo descrivere un algoritmo che calcoli, per ogni nodo $v \in V$, il numero dei discendenti di v. È chiaro che il numero di discendenti di una foglia è 0; invece, il numero di discendenti di un nodo interno v si calcola facilmente una volta noti il numero di discendenti dei suoi figli. Quindi, per risolvere il nostro problema è sufficiente descrivere un algoritmo che esegue una visita di T in ordine posticipato.

A tale scopo, rappresentiamo con r la radice dell'albero e, per ogni nodo v, denotiamo con $L(v)$ la lista dei figli di v. Definiamo una procedura ricorsiva Visita(v) che, su input v, restituisce il numero di discendenti del nodo v. Tale procedura restituisce il valore $d = 0$ se v è una foglia, altrimenti richiama se stessa sui figli di v, ne somma i valori incrementandoli di uno e restituisce la quantità d ottenuta. Se inoltre d coincide con k si incrementa un opportuno contatore rappresentato da una variabile globale S (inizialmente posta a 0). Il valore finale di S sarà proprio l'output dell'algoritmo.

Il programma principale, dopo aver letto l'input e inizializzato S, richiama la procedura sulla radice r.

```
begin
    leggi e memorizza l'input
    S := 0
    u := Visita(r)
    return S
end

Procedure Visita(v)
begin
    d := 0
    if IS_EMPTY(L(v)) = 0
          then for z ∈ L(v) do { b := Visita(z)
                               { d := d + 1 + b
    if d = k then S := S + 1
    return d;
end
```

b) Il tempo di calcolo richiesto è $\Theta(n)$, dove n è il numero di nodi, poiché l'algoritmo esegue un numero limitato di istruzioni RAM per ciascun vertice dell'albero. Anche lo spazio di memoria richiesto è dello stesso ordine di grandezza poiché l'input richiede uno spazio $\Theta(n)$ per essere mantenuto in memoria e la pila che implementa la ricorsione raggiunge una altezza massima $O(n)$.

ALGORITMI E STRUTTURE DATI
Svolgimento della prova scritta del 19.12.2002
TEMA N. 1

Esercizio 1.

Considera le sequenze $\{f_n\}$ e $\{g_n\}$ così definite:

$$f_n = \begin{cases} 3n^2 + n & \text{se } n \equiv 0 \pmod 3 \\ 4n^3 \log\left(1 + \frac{1}{n}\right) & \text{se } n \equiv 1 \pmod 3 \\ \frac{n^2}{2}\left(1 + \frac{1}{n}\right)^n & \text{se } n \equiv 2 \pmod 3 \end{cases}$$

$$g_n = \begin{cases} n^2 + 2n & \text{se } n \text{ è pari} \\ n^3 + n \log n & \text{se } n \text{ è dispari} \end{cases}$$

Tra le seguenti relazioni determinare le relazioni vere e quelle false:

$$1) f_n \sim 3n^2, \quad 2) f_n = \Theta(n^2), \quad 3) f_n = O(n^3), \quad 4) f_n = o(n^2),$$

$$5) g_n = o(n^3), \quad 6) g_n = \Theta(n^3), \quad 7) g_n = O(n^2), \quad 8) g_n = O(n^3).$$

Soluzione

1) falsa, 2) vera, 3) vera, 4) falsa, 5) falsa, 6) falsa, 7) falsa, 8) vera.

Esercizio 2.

Dato un albero binario $T =< V, E >$, sia $h_T : V \longrightarrow \mathbb{Z}$ la funzione così definita:

$$h_T(v) = \begin{cases} 0 & \text{se } v \text{ è una foglia} \\ 2 + h_T(sin(v)) & \text{se } v \text{ possiede solo il figlio sinistro} \\ 1 - h_T(des(v)) & \text{se } v \text{ possiede solo il figlio destro} \\ h_T(sin(v)) - h_T(des(v)) & \text{se } v \text{ possiede sia il figlio sinistro che il figlio destro} \end{cases} \tag{A.2}$$

a) Definire una procedura ricorsiva che su input T (rappresentato mediante una coppia di vettori sin e des) calcoli i valori $h_T(v)$ per tutti i nodi v di T.

b) Definire una procedura non ricorsiva per risolvere lo stesso problema.

Soluzione

a) L'algoritmo svolge una visita dell'albero in ordine posticipato. Si denota con r la radice e con V l'insieme dei nodi dell'albero. I valori calcolati vengono mantenuti in un vettore H di dimensione pari al numero dei nodi dell'albero. Alla fine del calcolo, per ogni nodo v $H[v]$ contiene il valore desiderato. Nelle procedure seguenti H, sin e des sono variabili globali.

```
Main
  begin
      leggi e memorizza l'input
      for v ∈ V do H[v] := 0
      Visita(r)
      return H
  end
```

Procedure Visita(v)
begin

$$\textbf{if } sin(v)=0 \textbf{ then } \{ \textbf{ if } des(v)\neq 0 \textbf{ then } \begin{cases} \text{Visita}(des(v)) \\ H[v] := 1 - H[des(v)] \end{cases} \}$$

$$\textbf{else if } des(v)=0 \textbf{ then } \begin{cases} \text{Visita}(sin(v)) \\ H[v] := 2 + H[sin(v)] \end{cases}$$

$$\textbf{else } \begin{cases} \text{Visita}(sin(v)) \\ \text{Visita}(des(v)) \\ H[v] := H[sin(v)] - H[des(v)] \end{cases}$$

end

b) La procedura iterativa implementa di nuovo una visita in ordine posticipato nella quale ogni nodo viene visitato dopo gli eventuali figli. Anche in questo caso usiamo il vettore H per mantenere i valori calcolati e si suppone che inizialmente tutti i suoi valori siano 0. Per comodità definiamo una procedura Calcola(v) che determina effettivamente il valore $h_T(v)$ una volta noti e valori relativi ai figli. Tale procedura è la seguente:

Procedure Calcola(v)
begin
  if $sin(v) = 0$ **then** { **if** $des(v) \neq 0$ **then** $H[v] := 1 - H[des(v)]$ }
    else if $des(v) = 0$ **then** $H[v] := 2 + H[sin(v)]$
      else $H[v] := H[sin(v)] - H[des(v)]$
end

La visita in post-ordine viene implementata usando un ciclo repeat until nel quale si distinguono due fasi diverse rappresentate dal valore di una variabile booleana f: quando $f = 0$ la procedura sta scendendo lungo l'albero memorizzando nella pila i nodi interni visitati; quando $f = 1$ la procedura risale l'albero prelevando dalla pila i nodi accantonati i cui figli sono stati già visitati. Ogni nodo v nella pila viene mantenuta assieme ad un altro valore booleano k (la pila contiene così coppie di valori booleani (v, k)) con il seguente significato: si inserisce (v, k) nella pila con $k = 0$ se $sin(v)$ non è stato ancora visitato, si inserisce invece (v, k) con $k = 1$ se $sin(v)$ è già stato visitato (e il il valore $h_T(sin(v))$ calcolato) mentre $des(v)$ non è stato ancora considerato.

```
Procedure Visita
begin
    v := r
    P := Λ
    esci := 0
    f := 0
    repeat
        while f = 0 do
            if sin(v) ≠ 0  then { P := Push(P, (v, 0))
                                 { v := sin(v)
                           else if des(v) ≠ 0  then { P := Push(P, (v, 1))
                                                    { v := des(v)
                                               else f := 1
        if P ≠ Λ then
            begin
              (v, k) := TOP(P)
              P := POP(P)
              if k = 0 ∧ des(v) ≠ 0        then { P := Push(P, (v, 1))
                                                 { v := des(v)
                                                 { f := 0
                                           else { Calcola(v)
                                                { f := 1
            end
        else esci := 1
    until esci = 1
end
```

ALGORITMI E STRUTTURE DATI

Svolgimento della prova scritta del 8.2.2002

Esercizio 1.

Eseguire l'algoritmo Quicksort sull'input

$$3, 4, 1, 5, 6, 0, 5, 3, 8, 9$$

scegliendo sempre come pivot il primo elemento del vettore considerato e mettendo in evidenza gli scambi eseguiti.

... Facile, consultare le dispense ....

Esercizio 2.

a) Descrivere un algoritmo per risolvere il seguente problema:

Istanza : un albero con radice T di n nodi, rappresentato mediante liste di adiacenza, e un intero k, $0 \leq k \leq n-1$.
Soluzione : il numero di nodi di T che hanno k discendenti.

b) Assumendo il criterio di costo uniforme, valutare in funzione di n il tempo di calcolo e lo spazio di memoria richiesti dall'algoritmo nel caso peggiore.

Soluzione

a)

Ricordiamo che in un albero con radice T, un nodo z è discendente di un nodo v se $z \neq v$ e z si trova nel sottoalbero di T che ha per radice v. Vogliamo descrivere un algoritmo che calcoli, per ogni nodo $v \in V$, il numero dei discendenti di v. È chiaro che il numero di discendenti di una foglia è 0; invece, il numero di discendenti di un nodo interno v si calcola facilmente una volta noti il numero di discendenti dei suoi figli. Quindi, per risolvere il nostro problema è sufficiente descrivere un algoritmo che esegue una visita di T in ordine posticipato.

A tale scopo, rappresentiamo con r la radice dell'albero e, per ogni nodo v, denotiamo con $L(v)$ la lista dei figli di v. Definiamo una procedura ricorsiva Visita(v) che, su input v, restituisce il numero di discendenti del nodo v. Tale procedura restituisce il valore $d = 0$ se v è una foglia, altrimenti richiama se stessa sui figli di v, ne somma i valori incrementandoli di uno e restituisce la quantità d ottenuta. Se inoltre d coincide con k si incrementa un opportuno contatore rappresentato da una variablile globale S (inizialmente posta a 0). Il valore finale di S sarà proprio l'output dell'algoritmo.

Il programma principale, dopo aver letto l'input e inizializzato S, richiama la procedura sulla radice r.

begin
  leggi e memorizza l'input
  $S := 0$
  $u :=$ Visita(r)
  return S
end

Procedure Visita(v)
begin
  $d := 0$
  if IS_EMPTY($L(v)$) $= 0$
    then for $z \in L(v)$ **do** $\begin{cases} b := \text{Visita}(z) \\ d := d + 1 + b \end{cases}$
  if $d = k$ **then** $S := S + 1$
  return d;
end

b) Il tempo di calcolo richiesto è $\Theta(n)$, dove n è il numero di nodi, poiché l'algoritmo esegue un numero limitato di istruzioni RAM per ciascun vertice dell'albero.

Esercizio 3.

Dato un grafo non orientato $G = \langle V, E \rangle$ nel quale V è l'insieme dei nodi ed E quello degli archi, denotiamo con F_G la seguente famiglia di sottoinsiemi di V:

$$F_G = \{A \subseteq V \mid \forall a, b \in A, a \neq b \Longrightarrow \{a, b\} \notin E\}$$

Rispondere alle seguenti domande giustificando le risposte:
a) La coppia $\langle V, F_G \rangle$ forma un sistema di indipendenza?
b) La coppia $\langle V, F_G \rangle$ forma un matroide?
c) Definire un algoritmo greedy per il problema seguente:

Istanza : un grafo non orientato $G = \langle V, E \rangle$ e una funzione peso $w : V \to \mathbb{R}^+$.
Soluzione : il sottoinsieme $A \in F_G$ di peso massimo.

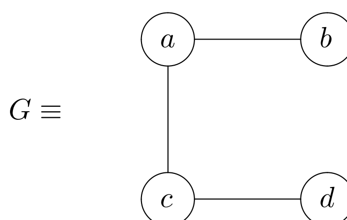
d) Assumendo il criterio di costo uniforme, svolgere l'analisi dell'algoritmo supponendo che il grafo di ingresso abbia n nodi e m lati.
e) L'algoritmo descritto è ottimale?

Soluzione

a) La coppia $\langle V, F_G \rangle$ forma un sistema di indipendenza per ogni grafo G. Infatti, ogni insieme $A \in F_G$ è formato da nodi che non hanno lati in comune: di conseguenza ogni suo sottoinsieme gode della stessa proprietá e quindi appartiene a F_G.

b) In generale $\langle V, F_G \rangle$ non è un matroide. Consideriamo per esempio il grafo

$G \equiv$



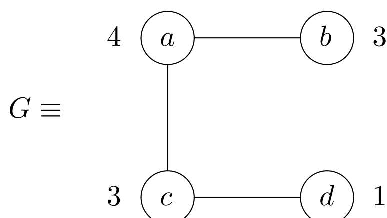
e definiamo $A = \{a\}$ e $B = \{c, d\}$. È immediato verificare che $A, B \in F_G$, $|B| = |A| + 1$ e, tuttavia, non esiste alcun $u \in B$ tale che $A \cup \{u\} \in F_G$.

c) Un algoritmo greedy per il problema considerato è descritto dalla seguente procedura:

```
begin
    S := ∅
    Q := V
    while Q ≠ ∅ do
        begin
            determina l'elemento u ∈ Q con w(u) massimo
            cancella u da Q
            if S = ∅ then S := {u}
                    ⎧ h := 0
               else ⎨ for z ∈ S do if {u, z} ∈ E then h := 1
                    ⎩ if h = 0 then S := S ∪ {u}
        end
    return S
end
```

d) Calcoliamo il tempo richiesto dall'esecuzione di ciascun ciclo while. Definendo Q come una coda di priorità (per esempio uno heap), sono necessari $\Theta(\log n)$ passi per determinare il nodo u di peso massimo in Q e per cancellarlo. Inoltre, posso implementare S come una lista ordinata rispetto ai nomi dei nodi e, analogamente, mantenere le liste $L(v)$ con lo stesso ordine. Così il ciclo for più interno può essere eseguito in $O(n)$ passi. Complessivamente quindi il corpo del ciclo while richiede un tempo $O(n)$. Poiché il ciclo while viene ripetuto n volte il tempo totale richiesto dall'algoritmo è $O(n^2)$.

e) La soluzione fornita dall'algoritmo in generale non è ottima perché il sistema di indipendenza non sempre è un matroide. Per esempio consideriamo il grafo descritto al punto b) opportunamente pesato:



In questo caso la soluzione ottenuta dall'algoritmo è $\{a, d\}$ di peso 5, mentre quella ottimale è $\{b, c\}$ di peso 6.

ALGORITMI E STRUTTURE DATI

Svolgimento della prova scritta del 9.1.2002

Esercizio 1.

Eseguire l'algoritmo Heapsort sull'input

$$4, 1, 5, 8, 7, 6, 9, 10, 2, 3$$

mettendo in evidenza gli scambi eseguiti.

... Facile, consultare le dispense ....

Esercizio 2.

Ricordiamo che in un grafo orientato, dati due nodi s e v, si dice che v è raggiungibile da s se esiste un cammino da s a v e, in questo caso, la distanza di v da s è la lunghezza del più corto cammino da s a v.

a) Descrivere una procedura per risolvere il seguente problema:

Istanza : un grafo orientato G di n nodi e m lati, e due nodi s e u di G;
Soluzione : il numero di nodi di G raggiungibili da s che si trovano alla stessa distanza da s e da u.

b) Assumendo il criterio di costo uniforme, valutare in funzione di n e m il tempo di calcolo e lo spazio di memoria richiesti dall'algoritmo nel caso peggiore.

Soluzione

a)

Supponiamo che il grafo di ingresso G sia rappresentato da un insieme di nodi V e dalla famiglia di liste di adiacenza $\{L(v) \mid v \in V\}$. Per ogni $v \in V$, L(v) è la lista dei nodi w per i quali esiste in G il lato (v, w).

L'algoritmo esegue una esplorazione in ampiezza del grafo diretto G a partire dal nodo s calcolando così le distanze di tutti i nodi da s. Tali distanze sono memorizzate in un vettore A che ha per indici i nodi del grafo. Successivamente con una analoga visita in ampiezza si determinano le distanze dei nodi da u calcolando un vettore delle distanze B. Infine l'algoritmo confronta le componenti corrispondenti dei vettori A e B contando il numero dei nodi che hanno uguale valore. L'algoritmo è descritto dal seguente schema nel quale si utilizza la procedura Ampiezza(G, v) che restituisce il vettore D delle distanze dei nodi di G dal vertice v. Per convenzione, $D[w]$ risulterà uguale a -1 se e solo se il nodo w non è rangiungibile da v, altrimenti $D[w]$ sarà proprio la distanza di w da v.

```
begin
    leggi e memorizza l'input
    A := Ampiezza(G, s)
    B := Ampiezza(G, u)
    R := 0
    for w ∈ V do
        if A[w] ≥ 0 ∧ A[w] = B[w] then R := R + 1
    return R
end
```

Il calcolo delle distanze si ottiene applicando la seguente procedura che esegue un attraversamento in ampiezza. Il procedimento è basato sull'utilizzo della coda Q (Λ rappresenta qui la coda vuota).

Procedure Ampiezza(G, v)
begin
  $Q :=$ Enqueue(Λ, v)
  for $w \in V$ **do** $D[w] := -1$
  $D[v] := 0$
  while $Q \neq \Lambda$ **do**
    begin
      $w :=$ Front(Q)
      $Q :=$ Dequeue(Q)
      for $z \in L(w)$ **do**
        if $D[z] := -1$ **then** $\begin{cases} D[z] := D[w] + 1 \\ Q := \text{Enqueue}(Q, z) \end{cases}$
    end
  return D
end

b)

Nel caso peggiore, sia il tempo di calcolo che lo spazio di memoria richiesti dall'algoritmo sono $O(n + m)$.

Esercizio 3.

Considera il problema di calcolare l'espressione

$$(x_1 + x_2 + \cdots + x_m)\text{mod}(k)$$

assumendo come input un intero positivo k e una sequenza di interi $x_1, x_2, \ldots, x_m$.

a) Descrivere un algoritmo del tipo *divide et impera* per risolvere il problema.

b) Assumendo il criterio di costo uniforme, valutare l'ordine di grandezza del tempo di calcolo e dello spazio di memoria in funzione di m.

c) Assumiamo il criterio di costo logaritmico e supponiamo che $x_i \in \{0, 1, 2, \ldots, k-1\}$, per ogni $i = 1, 2, \ldots, m$. Determinare una stima O-grande del tempo di calcolo e dello spazio di memoria in funzione dei parametri m e k.

Soluzione

a)

L'idea dell'algoritmo è quella di applicare una procedura ricorsiva che spezza l'input in due parti (circa) uguali, richiama se stessa ricorsivamente su queste ultime e quindi restituisce la somma (modulo k) dei due risultati ottenuti. Definiamo la procedura Somma(i, j), che su input $1 \leq i \leq j \leq m$, calcola

$$(x_i + x_{i+1} + \cdots + x_j) mod(k)$$

Il programma principale consiste semplicemente nella lettura dei parametri $m, k \geq 1$ e nella chiamata Somma$(1, m)$.

Procedure Somma(i, j)

$$\textbf{if } i = j \textbf{ then} \begin{cases} a := read(x_i) \\ \text{return } (a)\text{mod}(k) \end{cases}$$

else
begin
$\ell := \lfloor \frac{j+i}{2} \rfloor$
$b :=$ Somma(i, ℓ)
$c :=$ Somma$(\ell + 1, j)$
return $(b + c)\text{mod}(k)$
end

b)

Valutiamo ora il tempo di calcolo secondo il criterio uniforme. È chiaro che il tempo di calcolo richiesto per eseguire Somma(i, j) dipende dal numero di elementi della sequenza di input compresi tra gli indici i e j. Definiamo allora $u = j - i + 1$ e denotiamo con $T(u)$ il tempo di calcolo richiesto da Somma(i, j) secondo il criterio uniforme. Otteniamo quindi la seguente equazione di ricorrenza:

$$T(u) = \begin{cases} c_1 & \text{se } u = 1 \\ T\left(\lfloor \frac{u}{2} \rfloor\right) + T\left(\lceil \frac{u}{2} \rceil\right) + c_2 & \text{se } u \geq 2 \end{cases}$$

dove c_1 e c_2 sono due costanti opportune. Applicando le regole di soluzione si ricava $T(u) = \Theta(u)$. Pertanto il tempo di calcolo complessivo è $\Theta(m)$.

Inoltre lo spazio di memoria richiesto è essenzialmente quello utilizzato dalla pila che implementa la ricorsione. Osserviamo che, su un input di m elementi, la pila raggiunge una altezza massima pari a $1 + \lfloor \log_2 m \rfloor$: questo è dovuto al fatto che ogni chiamata ricorsiva riduce della metà la dimensione $(j - i + 1)$ della porzione di input corrente. Poiché nel nostro caso ogni record di attivazione occupa uno spazio costante, lo spazio complessivo richiesto dall'algoritmo è $\Theta(\log m)$.

c)

Assumiamo ora il criterio logaritmico. Per ipotesi sappiamo che ogni x_i è un intero di $\log k$ bit al più . Applicando i ragionamenti svolti nel punto precedente il tempo di calcolo $T^\ell(u)$ risulta determinato da una equazione della forma

$$T^\ell(u) = \begin{cases} O(\log k + \log m) & \text{se } u = 1 \\ T^\ell\left(\lfloor \frac{u}{2} \rfloor\right) + T^\ell\left(\lceil \frac{u}{2} \rceil\right) + O(\log k + \log m) & \text{se } u \geq 2 \end{cases}$$

Infatti, se $u = 1$, $O(\log m)$ è il tempo richiesto per verificare $i = j$ mentre $O(\log k)$ è il tempo per calcolare $x_i\text{mod}(k)$. Se invece $u > 1$, oltre al tempo $O(\log m)$ per verificare $i < j$ e calcolare ℓ , occorre un tempo $O(\log k)$ per determinare $(b + c)\text{mod}(k)$.

Applicando ora la regola di soluzione (e considerando $O(\log k + \log m)$ come costante rispetto al parametro u) si ottiene

$$T^\ell(u) = O(\log k + \log m) \cdot \Theta(u)$$

e fissando $u = m$ si ricava $T^\ell(m) = O(m \cdot (\log k + \log m))$.

Per quanto riguarda lo spazio di memoria, valutiamo lo spazio P necessario per mantenere la pila che implementa la ricorsione. Si può provare che ogni record di attivazione mantiene al più

interi di dimensione complessiva $O(\log k + \log m)$. Di conseguenza lo spazio richiesto, secondo il criterio logaritmico, risulta $O((\log k + \log m) \log m)$.

ALGORITMI E STRUTTURE DATI

Prova scritta del 5.12.2001

TEMA N. 2

Esercizio 1.

Ricordiamo che in un albero con radice T, l'*altezza* di un nodo v in T è la massima distanza di v da una foglia.

a) Descrivere una procedura (eventualmente ricorsiva) che, avendo per input un albero con radice T (rappresentato mediante liste di adiacenza), calcoli la somma delle altezze dei nodi di T cercando di minimizzare lo spazio di memoria usato dall'algoritmo.

b) Assumendo il criterio di costo uniforme, valutare il tempo di calcolo e lo spazio di memoria richiesti in funzione del numero n di nodi dell'albero.

Soluzione proposta

a)

È chiaro che l'altezza di una foglia è 0, mentre l'altezza di un nodo interno è il massimo tra le altezze dei suoi figli, incrementato di 1. Di conseguenza, le altezze di tutti i nodi dell'albero possono essere calcolate mediante una semplice visita in ordine posticipato.

A tale scopo, rappresentiamo con r la radice dell'albero e, per ogni nodo v, denotiamo con $L(v)$ la lista dei figli di v. Definiamo una procedura ricorsiva Altezza(v) che, su input v, restituisce l'altezza del nodo v; tale procedura restituisce il valore 0 se v è una foglia, altrimenti richiama se stessa sui figli di v e determina il massimo dei valori così ottenuti: tale quantità, incrementata di 1, fornisce proprio il valore cercato. La procedura utilizza la variabile globale S per mantenere la somma delle altezze calcolate. Il programma principale, dopo aver letto l'input e inizializzato S, richiama la procedura sulla radice r.

```
begin
        leggi e memorizza l'input
        S := 0
        u := Altezza(r)
        return S
end

Procedure Altezza(v)
if IS_EMPTY(L(v)) = 1  then return 0
else
     begin
            h := 0;
                               { b := Altezza(w);
            for w ∈ L(v) do    { if h < b then h := b
            h := h + 1;
            S := S + h;
            return h;
     end
```

b) Il tempo di calcolo richiesto è $\Theta(n)$, dove n è il numero di nodi, poiché l'algoritmo esegue

un numero limitato di istruzioni per ciascun vertice dell'albero. Analogamente, lo spazio di memoria richiesto è $\Theta(n)$. Tale valutazione si ottiene considerando lo spazio necessario per mantenere l'albero di ingresso e per eseguire la procedura ricorsiva appena descritta.

Esercizio 2.

Considera il seguente problema:

Istanza : due sequenze finite di numeri interi $a_0, a_1, \ldots, a_n$, $b_0, b_1, \ldots, b_n$.
Soluzione : il valore dell'espressione $c = (a_0 + b_n) \cdot (a_1 + b_{n-1}) \cdot \cdots \cdot (a_n + b_0)$.

a) Stimare il tempo di calcolo e lo spazio di memoria richiesti dalla seguente procedura assumendo il criterio di costo uniforme:

begin
  leggi e memorizza l'input
  $s = 1$
  for $i = 0, 1, \ldots, n$ **do**
    $\begin{cases} t := a_{n-i} + b_i \\ s := s \cdot t \end{cases}$
  return s
end

b) Supponendo che ogni a_i e ogni b_i abbia dimensione m, svolgere l'esercizio precedente assumento il criterio di costo logaritmico.

c) Descrivere un algoritmo del tipo *divide et impera* per risolvere lo stesso problema.

d) Assumendo il criterio di costo uniforme, valutare l'ordine di grandezza del tempo di calcolo e dello spazio di memoria richiesti dalla procedura descritta nell'esercizio precedente.

e) Nelle stesse ipotesi dell'esercizio b) determinare una stima O-grande del tempo di calcolo assumendo il criterio di costo logaritmico.

Soluzione proposta

a) Assumendo il criterio di costo uniforme la procedura descritta richiede un tempo $\Theta(n)$ per memorizzare l'input; anche l'esecuzione del ciclo for richiede tempo $\Theta(n)$ perchè ogni iterazione può essere eseguita mediante un numero costante di operazioni RAM. La procedura richiede inoltre uno spazio $\Theta(n)$ per mantenere l'input e uno spazio $\Theta(1)$ per gli altri valori. Quindi, sia lo spazio che il tempo richiesti sono di ordine di grandezza $\Theta(n)$.

b) Assumiamo ora il criterio di costo logaritmico. Poiché ogni intero dell'input ha dimensione m, il tempo necessario per memorizzare l'input è di ordine di grandezza $\Theta(nm)$. Il ciclo for viene eseguito $n + 1$ volte; la i-esima iterazione richiede un tempo $\Theta(im + \log n)$ per ciascun $i \in \{1, 2, \ldots, n + 1\}$. Quindi il tempo necessario per l'esecuzione del ciclo for risulta

$$\sum_{i=1}^{n+1} \Theta(im + \log n) = \Theta(n^2 m)$$

che fornisce anche l'ordine di grandezza il tempo complessivo richiesto dalla procedura.

Lo spazio di memoria necessario per mantenere l'input è $\Theta(nm)$ e una valutazione analoga si ottiene per la variabile s. Di conseguenza lo spazio richiesto dalla procedura secondo il criterio logaritmico è dell'ordine di grandezza di $\Theta(nm)$.

c) L'algoritmo legge e memorizza l'input in due vettori di variabili globali che per comodità chiamiamo $A = (A[0], A[1], \dots, A[n])$ e $B = (B[0], B[1], \dots, B[n])$; quindi richiama la funzione Calcola$(0, n)$ e restituisce il valore ottenuto.

Definiamo la procedura Calcola(i, j), che su input $0 \le i \le j \le n$, restituisce il valore

$$(a_i + b_{n-i}) \cdot (a_{i+1} + b_{n-i-1}) \cdot \ \cdots \ \cdot (a_j + b_{n-j})$$

```
Procedure Calcola(i, j)
if i = j then
        { r := A[i] + B[n − i]
        { return r
        else
            begin
                k := ⌊(i+j)/2⌋
                s := Calcola(i, k)
                u := k + 1
                t := Calcola(u, j)
                v := s · t
                return v
            end
```

d) Assumiamo il criterio di costo uniforme. Il tempo richiesto per memorizzare l'input è chiaramente dell'ordine di grandezza $\Theta(n)$. Denotiamo ora con $T(n)$ il tempo di calcolo richiesto da Calcola$(0, n-1)$ (in questo modo n è il numero di componenti di A e di B). Supponiamo inoltre che n sia una potenza di 2. Otteniamo allora la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} c_1 & \text{se } n = 1 \\ 2T\left(\frac{n}{2}\right) + c_2 & \text{se } n \ge 2 \end{cases}$$

dove c_1 e c_2 sono costanti opportune. Applicando le regole di soluzione si ricava $T(n) = \Theta(n)$. Pertanto il tempo di calcolo complessivo è $\Theta(n)$.

Inoltre lo spazio di memoria richiesto è dato dallo spazio necessario per mantenere l'input e da quello utilizzato dalla pila che implementa la ricorsione. Il primo è dell'ordine di $\Theta(n)$ mentre il secondo risulta $\Theta(\log n)$. Quindi anche in questo caso otteniamo un ordine di grandezza $\Theta(n)$.

e)

Assumiamo ora il criterio logaritmico. Per ipotesi sappiamo che ogni intero di input possiede al più m bit. Occorre quindi un tempo $\Theta(nm)$ per memorizzare l'input. Denotiamo inoltre con $T^\ell(u)$ il tempo necessario per eseguire Calcola(i, j) con $u = j - i + 1$. Supponendo u una potenza di 2, si ottiene la seguente equazione di ricorrenza

$$T^\ell(u) = \begin{cases} O(m + \log n) & \text{se } u = 1 \\ 2T^\ell\left(\frac{u}{2}\right) + O(um + \log n) & \text{se } u \ge 2 \end{cases}$$

la cui soluzione fornisce (considerando m e $\log n$ come costanti rispetto al parametro u)

$$T^\ell(u) = O(u(m \log u + \log n))$$

Tale valutazione può essere chiaramente estesa al caso in cui u non è una potenza di 2 e quindi, fissando $u = n$, si ricava $T^\ell(n) = O(mn \log n))$.

ALGORITMI E STRUTTURE DATI
Svolgimento della prova scritta del 6.2.2001
TEMA N. 1

Esercizio 1.

a) Descrivere un algoritmo per risolvere il seguente problema:

Istanza : un grafo non orientato G di n nodi e m lati;
Soluzione : il numero di componenti connesse di G.

b) Valutare in funzione di n e m il tempo di calcolo e lo spazio di memoria richiesti dall'algoritmo nel caso peggiore.

Soluzione proposta

a)

Il numero di componenti connesse del grafo G può essere ottenuto mediante una semplice visita in profondità che tenga conto del numero di nodi sorgente incontrati. L'algoritmo è descritto dalle seguenti procedure nelle quali V rappresenta l'insieme dei nodi di G e, per ogni $v \in V$, $L(v)$ è la lista dei vertici adiacenti a v.

Main
begin
leggi e memorizza l'input
for $v \in V$ **do** marca v come *nuovo*
$c := 0$
for $v \in V$ **do if** v marcato *nuovo* **then** $\begin{cases} c := c+1 \\ \text{Visita}(v) \end{cases}$
return c
end

Procedure Visita(v)
begin
marca v come *vecchio*
for $w \in L(v)$ **do if** w marcato *nuovo* **then** Visita(w)
end

b) Il tempo di calcolo richiesto è $\Theta(n+m)$, poiché l'algoritmo esegue un numero limitato di istruzioni per ciascun vertice e ciascun lato del grafo. Analogamente, anche lo spazio di memoria richiesto è $\Theta(n+m)$. Tale valutazione si ottiene considerando lo spazio necessario per mantenere il grafo di ingresso e per eseguire la procedura ricorsiva appena descritta.

Esercizio 2.

Dato un intero $q \in \mathbf{N}$ e una matrice 2×2 a componenti intere, $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$, denotiamo con $M\text{mod}(q)$ la matrice che si ottiene da M sostituendo le componenti con i relativi resti modulo q:

$$M\text{mod}(q) = \begin{pmatrix} a \text{ mod}(q) & b \text{ mod}(q) \\ c \text{ mod}(q) & d \text{ mod}(q) \end{pmatrix}$$

a) Descrivere un algoritmo del tipo *divide et impera* per risolvere il sequente problema:

Istanza : due interi positivi n, q e una matrice di numeri interi $M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$;

Soluzione : le componenti della matrice $M^n \text{mod}(q)$.

b) Assumendo il criterio di costo uniforme, valutare l'ordine di grandezza del tempo di calcolo e dello spazio di memoria in funzione di n.

c) Assumiamo il criterio di costo logaritmico e supponiamo che i valori di input a, b, c, d siamo minori di q in modulo. Determinare una stima O-grande del tempo di calcolo e dello spazio di memoria in funzione dei parametri n e q.

Soluzione proposta

a)

Innanzitutto descriviamo una procedura per il calcolo del prodotto di due matrici 2×2 modulo q. Tale procedura riceve in input due matrici $A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$, $B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$, e restituisce la matrice $C = A \cdot B \text{mod}(q)$. Si suppone di poter utilizzare una funzione $MOD_q(x)$ che, per ogni intero x, restituisce il resto della divisione di x per q.

Procedure Prodotto(A, B)

begin

  for $i = 1, 2$ **do**

    for $j = 1, 2$ **do**

      $c_{ij} := MOD_q(a_{i1}b_{1j} + a_{i2}b_{2j})$

  return $C = \begin{pmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{pmatrix}$

end

L'algoritmo richiesto legge e memorizza i parametri n, q e la matrice M come variabili globali e quindi chiama la seguente procedura Calcola(n) che restituisce $M^n \text{mod}(q)$.

Procedure Calcola(i)

if $i = 0$ **then return** $I = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$

  else

    begin

      $k := \lfloor \frac{i}{2} \rfloor$

      $A :=$ Calcola(k)

      $C :=$ Prodotto(A, A)

      if i pari **then return** C

        else return Prodotto(C, M)

    end

b)

È facile verificare che il tempo di calcolo richiesto dalla procedura prodotto è costante secondo il criterio uniforme. Quindi denotiamo con $T(i)$ il tempo di calcolo richiesto da Calcola(i) sempre secondo il medesimo criterio. Otteniamo la seguente equazione di ricorrenza:

$$T(i) = \begin{cases} c_1 & \text{se } i = 0 \\ T\left(\lfloor \frac{i}{2} \rfloor\right) + c_2 & \text{se } i \geq 1 \text{ e } i \text{ pari} \\ T\left(\lfloor \frac{i}{2} \rfloor\right) + c_3 & \text{se } i \geq 1 \text{ e } i \text{ dispari} \end{cases}$$

dove c_1, c_2 e c_3 sono costanti opportune. Applicando le regole di soluzione si ricava $T(i) = \Theta(\log i)$. Pertanto il tempo di calcolo complessivo è $\Theta(\log n)$.

Inoltre lo spazio di memoria utilizzato è principalmente quello richiesto dalla pila che implementa la ricorsione. Osserviamo che, su un input n, q, M , la pila raggiunge una altezza massima pari a $1 + \lfloor \log_2 n \rfloor$: questo è dovuto al fatto che ogni chiamata ricorsiva riduce della metà l'esponente della potenza da calcolare. Poiché nel nostro caso ogni record di attivazione occupa uno spazio costante, lo spazio complessivo richiesto dall'algoritmo è $\Theta(\log n)$.

c)

Assumiamo ora il criterio logaritmico e osserviamo che il tempo di calcolo richiesto dalla procedura Prodotto è $\Theta(\log q)$.

e denotiamo con $T^\ell(i)$ il tempo di calcolo richiesto da Calcola(i). Allora $T^\ell(i)$ risulta determinato da una equazione della forma

$$T^\ell(i) = \begin{cases} O(\log q) & \text{se } i = 1 \\ T^\ell\left(\lfloor \frac{i}{2} \rfloor\right) + O(\log q + \log i) & \text{se } i \geq 2 \end{cases}$$

Infatti, se $i > 1$, $O(\log i)$ è il tempo richiesto per verificare $i > 1$ e calcolare k; mentre $O(\log q)$ è il tempo necessario per calcolare i vari prodotti di matrici.

Sviluppando direttamente l'equazione si ottiene

$$T^\ell(n) = O\left(\sum_{i=0}^{\log n} \log q + \log \frac{n}{2^i}\right) = O((\log q + \log n) \log n)$$

Per quanto riguarda lo spazio di memoria, valutiamo lo spazio P necessario per mantenere la pila che implementa la ricorsione. Ogni record di attivazione richiede uno spazio $O(\log n)$ per mantenere le variabili k e i; il record di attivazione corrente richiede uno spazio $(\log q)$ per mantenere le matrici A e C (ricorda che n, q, M sono variabili globali). Di conseguenza lo spazio richiesto, secondo il criterio logaritmico, risulta $O(\log^2 n + \log q)$.

Esercizio 3.

Eseguire l'algoritmo Quicksort sull'input

$$4, 2, 5, 8, 7, 6, 9, 10, 3$$

assumendo sempre come pivot il primo elemento della sottosequenza corrente: si mettano in evidenza i confronti e gli scambi eseguiti.

Soluzione

.... Facile: consultare appunti ed eserciziario ....

ALGORITMI E STRUTTURE DATI
Svolgimento della prova scritta del 12.12.2000
TEMA N. 1

Esercizio 1.

Dato un albero con radice, chiamiamo *altezza media* di un nodo v il numero intero $A(v)$ così definito:

$$A(v) = \begin{cases} 0 & \text{se } v \text{ è una foglia} \\ 1 + \frac{A(w_1)+A(w_2)+\cdots+A(w_\ell)}{\ell} & \text{se } v \text{ è un nodo interno e} \\ & w_1, w_2, \ldots, w_\ell \text{ sono i suoi figli} \end{cases}$$

a) Descrivere una procedura che, avendo in input un albero con radice (rappresentato mediante liste di adiacenza), calcoli l'altezza media della radice.

b) Assumendo il criterio di costo uniforme, valutare il tempo di calcolo e lo spazio di memoria richiesti in funzione del numero n di nodi dell'albero.

Soluzione

a)

L'altezza media di un nodo interno può essere calcolata una volta note le altezze medie dei figli. Di conseguenza le altezze medie di tutti i nodi si ottengono mediante una semplice visita in ordine posticipato.

A tale scopo, rappresentiamo con r la radice dell'albero e, per ogni nodo v, denotiamo con $L(v)$ la lista dei figli di v. Definiamo una procedura ricorsiva Altezza_media(v) che, su input v, restituisce $A(v)$; ovvero, restituisce il valore 0 se v è una foglia, altrimenti richiama se stessa sui figli di v e determina la media dei valori così ottenuti.

```
begin
    leggi e memorizza l'input
    h := Altezza_media(r)
    return h
end
```

```
Procedure Altezza_media(v)
if IS_EMPTY(L(v)) = 1  then return 0
else
    begin
        ℓ := 0
        a := 0
                        ⎧ b := Altezza_media(w)
        for w ∈ L(v) do ⎨ a := a + b
                        ⎩ ℓ := ℓ + 1
        a := a/ℓ
        return a + 1
    end
```

b) Il tempo di calcolo richiesto è $\Theta(n)$, dove n è il numero di nodi, poiché l'algoritmo esegue un numero limitato di istruzioni per ciascun vertice dell'albero. Analogamente, anche lo spazio di memoria richiesto è $\Theta(n)$. Tale valutazione si ottiene considerando lo spazio necessario per mantenere l'albero di ingresso e per eseguire la procedura ricorsiva appena descritta.

Esercizio 2.

Considera il problema di calcolare l'espressione

$$(a^n) mod(r)$$

assumendo come input tre interi positivi a, n, r.

a) Descrivere un algoritmo del tipo *divide et impera* per risolvere il problema.

b) Assumendo il criterio di costo uniforme, valutare l'ordine di grandezza del tempo di calcolo e dello spazio di memoria in funzione di n.

c) Assumiamo il criterio di costo logaritmico e supponiamo che $0 < a < r$. Determinare una stima O-grande del tempo di calcolo e dello spazio di memoria in funzione dei parametri n e r.

Soluzione

a)

Definiamo la procedura Calcola(i), che su input $0 \leq i \leq n$, restituisce il valore $(a^i) mod(r)$. Il programma principale consiste semplicemente nella lettura dei tre parametri di ingresso a, r, n e nella chiamata Calcola(n).

```
Procedure Calcola(i)
if i = 0 then return 1
      else
          begin
              k := ⌊i/2⌋
              b := Calcola(k)
              c := (b · b)mod(r)
              if i pari   then return c
                          else return (c · a)mod(r)
          end
```

b)

Denotiamo con $T(i)$ il tempo di calcolo richiesto da Calcola(i) secondo il criterio uniforme. Otteniamo quindi la seguente equazione di ricorrenza:

$$T(i) = \begin{cases} c_1 & \text{se } i = 0 \\ T\left(\lfloor \frac{i}{2} \rfloor\right) + c_2 & \text{se } i \geq 1 \text{ e } i \text{ pari} \\ T\left(\lfloor \frac{i}{2} \rfloor\right) + c_3 & \text{se } i \geq 1 \text{ e } i \text{ dispari} \end{cases}$$

dove c_1, c_2 e c_3 sono costanti opportune. Applicando le regole di soluzione si ricava $T(i) = \Theta(\log i)$. Pertanto il tempo di calcolo complessivo è $\Theta(\log n)$.

Inoltre lo spazio di memoria richiesto è essenzialmente quello utilizzato dalla pila che implementa la ricorsione. Osserviamo che, su un input a, n, r , la pila raggiunge una altezza massima pari a $1 + \lfloor \log_2 n \rfloor$: questo è dovuto al fatto che ogni chiamata ricorsiva riduce della metà l'esponente della potenza da calcolare. Poiché nel nostro caso ogni record di attivazione occupa uno spazio costante, lo spazio complessivo richiesto dall'algoritmo è $\Theta(\log n)$.

c)

Assumiamo ora il criterio logaritmico e denotiamo con $T^\ell(i)$ il tempo di calcolo richiesto da Calcola(i). Allora $T^\ell(i)$ risulta determinato da una equazione della forma

$$T^\ell(i) = \begin{cases} O(\log r) & \text{se } i = 1 \\ T^\ell\left(\lfloor \frac{i}{2} \rfloor\right) + O(\log r + \log i) & \text{se } i \geq 2 \end{cases}$$

Infatti, se $i > 1$, $O(\log i)$ è il tempo richiesto per verificare $i > 1$ e calcolare k; mentre $O(\log r)$ è il tempo necessario per calcolare $c^2 \text{mod}(r)$ (o eventualmente $c^2 a \text{mod}(r)$).

Sviluppando direttamente l'equazione si ottiene

$$T^\ell(n) = O\left(\sum_{i=0}^{\log n} \log r + \log \frac{n}{2^i}\right) = O((\log r + \log n) \log n)$$

Per quanto riguarda lo spazio di memoria, valutiamo lo spazio P necessario per mantenere la pila che implementa la ricorsione. Ogni record di attivazione richiede uno spazio $O(\log n)$ per mantenere le variabili k e i; il record di attivazione corrente richiede uno spazio $(\log r)$ per mantenere b e c (possiamo assumere a, n, r variabili globali). Di conseguenza lo spazio richiesto, secondo il criterio logaritmico, risulta $O(\log^2 n + \log r)$.

Esercizio 3.

Eseguire l'algoritmo Heapsort sull'input

$$3, 1, 4, 7, 6, 5, 8, 9, 2$$

mettendo in evidenza gli scambi eseguiti.

Soluzione

.... Facile: consultare appunti ed eserciziario ....

ALGORITMI E STRUTTURE DATI
Prova scritta del 12.12.2000
TEMA N. 2

Esercizio 1. Considera il problema di calcolare l'espressione

$$(a_1 \cdot a_2 \cdot \ \cdots \ \cdot a_n) mod(r)$$

assumendo come input un intero positivo r e una sequenza di interi $a_1, a_2, \ldots, a_n$.

a) Descrivere un algoritmo del tipo *divide et impera* per risolvere il problema.

b) Assumendo il criterio di costo uniforme, valutare l'ordine di grandezza del tempo di calcolo e dello spazio di memoria in funzione di n.

c) Assumiamo il criterio di costo logaritmico e supponiamo che $a_i \in \{0, 1, 2, \ldots, r-1\}$, per ogni $i = 1, 2, \ldots, n$. Determinare una stima O-grande del tempo di calcolo e dello spazio di memoria in funzione dei parametri n e r.

Soluzione

a)

L'idea dell'algoritmo è quella di applicare una procedura ricorsiva che spezza l'input in due parti (circa) uguali, richiama se stessa ricorsivamente su queste ultime e quindi restituisce il prodotto (modulo r) dei due risultati ottenuti. Definiamo la procedura Calcola(i, j), che su input $1 \leq i \leq j \leq n$, calcola

$$(a_i \cdot a_{i+1} \cdots a_j) mod(r)$$

Il programma principale consiste semplicemente nella lettura del parametro $n \geq 1$ e nella chiamata Calcola$(1, n)$.

Procedure Calcola(i, j)

if $i = j$ **then** $\begin{cases} a := read(a_i) \\ \text{return } (a)\text{mod}(r) \end{cases}$

else

begin

$\ell := \lfloor \frac{j+i}{2} \rfloor$

$b :=$ Calcola(i, ℓ)

$c :=$ Calcola$(\ell + 1, j)$

return $(b \cdot c)$mod(r)

end

b)

Valutiamo ora il tempo di calcolo secondo il criterio uniforme. È chiaro che il tempo di calcolo richiesto per eseguire Calcola(i, j) dipende dal numero di elementi della sequenza di input compresi tra gli indici i e j. Definiamo allora $u = j - i + 1$ e denotiamo con $T(u)$ il tempo di calcolo richiesto da Calcola(i, j) secondo il criterio uniforme. Otteniamo quindi la seguente equazione di ricorrenza:

$$T(u) = \begin{cases} c_1 & \text{se } u = 1 \\ T\left(\lfloor \frac{u}{2} \rfloor\right) + T\left(\lceil \frac{u}{2} \rceil\right) + c_2 & \text{se } u \geq 2 \end{cases}$$

dove c_1 e c_2 sono due costanti opportune. Applicando le regole di soluzione si ricava $T(u) = \Theta(u)$. Pertanto il tempo di calcolo complessivo è $\Theta(n)$.

Inoltre lo spazio di memoria richiesto è essenzialmente quello utilizzato dalla pila che implementa la ricorsione. Osserviamo che, su un input di n elementi, la pila raggiunge una altezza massima pari a $1 + \lfloor \log_2 n \rfloor$: questo è dovuto al fatto che ogni chiamata ricorsiva riduce della metà la dimensione $(j - i + 1)$ della porzione di input corrente. Poiché nel nostro caso ogni record di attivazione occupa uno spazio costante, lo spazio complessivo richiesto dall'algoritmo è $\Theta(\log n)$.

c)

Assumiamo ora il criterio logaritmico. Per ipotesi sappiamo che ogni a_i è un intero di $\log r$ bit al più . Applicando i ragionamenti svolti nel punto precedente il tempo di calcolo $T^\ell(u)$ risulta determinato da una equazione della forma

$$T^\ell(u) = \begin{cases} O(\log r + \log n) & \text{se } u = 1 \\ T^\ell\left(\lfloor \frac{u}{2} \rfloor\right) + T^\ell\left(\lceil \frac{u}{2} \rceil\right) + O(\log r + \log n) & \text{se } u \geq 2 \end{cases}$$

Infatti, se $u = 1$, $O(\log n)$ è il tempo richiesto per verificare $i = j$ mentre $O(\log r)$ è il tempo per calcolare $a_i \text{mod}(r)$. Se invece $u > 1$, oltre al tempo $O(\log n)$ per verificare $i < j$ e calcolare ℓ , occorre un tempo $O(\log r)$ per determinare $(b \cdot c)\text{mod}(r)$.

Applicando ora la regola di soluzione (e considerando $O(\log r + \log n)$ come costante rispetto al parametro u) si ottiene

$$T^\ell(u) = O(\log r + \log n) \cdot \Theta(u)$$

e fissando $u = n$ si ricava $T^\ell(n) = O(n \cdot (\log r + \log n))$.

Per quanto riguarda lo spazio di memoria, valutiamo lo spazio P necessario per mantenere la pila che implementa la ricorsione. Si può provare che ogni record di attivazione mantiene al più interi di dimensione complessiva $O(\log r + \log n)$. Di conseguenza lo spazio richiesto, secondo il criterio logaritmico, risulta $O((\log r + \log n) \log n)$.

Esercizio 2.

Ricordiamo che in un grafo orientato, dati due nodi s e v, si dice che v è raggiungibile da s se esiste un cammino da s a v e, in questo caso, la distanza di v da s è la lunghezza del più corto cammino da s a v.

a) Descrivere una procedura per risolvere il seguente problema:

> Istanza : un grafo orientato G di n nodi e m lati, e un nodo s di G;
> Soluzione : l'insieme dei nodi di G raggiungibili da s che si trovano a distanza massima da s.

b) Valutare in funzione di n e m il tempo di calcolo e lo spazio di memoria richiesti dall'algoritmo nel caso peggiore.

Soluzione

a)

Supponiamo che il grafo di ingresso G sia rappresentato da un insieme di nodi V e dalla famiglia di liste di adiacenza $\{L(v) \mid v \in V\}$. Per ogni $v \in V$, L(v) è la lista dei nodi w per i quali esiste in G il lato (v, w).

L'algoritmo esegue una esplorazione in ampiezza del grafo diretto G a partire dal nodo s. Come è nodo tale esplorazione mantiene una coda Q nella quale i nodi vengono inseriti in ordine

di distanza da s. Per determinare i nodi alla stessa distanza da s si introduce in Q il simbolo speciale $\sharp$. Tale simbolo separa i nodi in Q che si trovano a una distanza data da quelli che si trovano alla distanza successiva. Nell'insieme S si mantengono i nodi che si trovano alla distanza corrente da s; tale insieme viene aggiornato costantemente durante la visita. In questo modo al termine dell'attraversamento S contiene l'insieme dei nodi alla massima distanza dalla sorgente.

L'algoritmo è descritto dalla seguente procedura nella quale Λ denota la coda vuota.

begin
  $S := \emptyset$
  $Q :=$ Enqueue(Λ, s)
  for $w \in V$ **do** marca w come "nuovo"
  marca s come "vecchio"
  $Q :=$ Enqueue$(Q, \sharp)$
  while $Q \neq \Lambda$ **do**
    begin
      $u :=$ Front(Q)
      $Q :=$ Dequeue(Q)
      if $u = \sharp$ **then** $\left(\text{if } Q \neq \Lambda \text{ then } \begin{cases} S := \emptyset \\ Q := \text{Enqueue}(Q, \sharp) \end{cases}\right)$
        else $S := S \cup \{u\}$
          for $w \in L(u)$ **do**
            if w marcato nuovo **then** $\begin{cases} \text{marca } w \text{ come "vecchio"} \\ Q := \text{Enqueue}(Q, w) \end{cases}$
    end
  return S
end

b)

Nel caso peggiore, sia il tempo di calcolo che lo spazio di memoria richiesti dall'algoritmo sono $O(n + m)$.

Esercizio 3.

Eseguire l'algoritmo Quicksort sull'input

$$3, 1, 4, 7, 6, 5, 8, 9, 2$$

assumendo sempre come pivot il primo elemento della sottosequenza corrente: si mettano in evidenza i confronti e gli scambi eseguiti.

Soluzione

.... Facile: consultare appunti ed eserciziario ....

ALGORITMI E STRUTTURE DATI
Svolgimento della prova scritta del 15.12.1999
TEMA N. 1

Esercizio 2.

Dato un albero con radice T, chiamiamo *altezza minima* di un nodo v in T la minima distanza di v da una foglia.

a) Descrivere una procedura (eventualmente ricorsiva) che, avendo per input un albero con radice T (rappresentato mediante liste di adiacenza), calcoli la somma delle altezze minime dei nodi di T.

b) Assumendo il criterio di costo uniforme, valutare il tempo di calcolo richiesto in funzione del numero n di nodi dell'albero.

Soluzione

a)

È chiaro che l'altezza minima di una foglia è 0, mentre l'altezza minima di un nodo interno è il minimo tra le altezze minime dei suoi figli, incrementato di 1. Di conseguenza, le altezze minime di tutti i nodi dell'albero possono essere calcolate mediante una semplice visita in ordine posticipato.

A tale scopo, rappresentiamo con r la radice dell'albero e, per ogni nodo v, denotiamo con $L(v)$ la lista dei figli di v. Definiamo una procedura ricorsiva Altezza_min(v) che, su input v, restituisce l'altezza minima del nodo v; tale procedura restituisce il valore 0 se v è una foglia, altrimenti richiama se stessa sui figli di v e determina il minimo dei valori così ottenuti: tale quantità, incrementata di 1, fornisce proprio il valore cercato. La procedura utilizza la variabile globale S per mantenere la somma delle altezze minime calcolate. Il programma principale, dopo aver letto l'input e inizializzato S, richiama la procedura sulla radice r.

```
begin
    leggi e memorizza l'input
    S := 0
    u := Altezza_min(r)
    return S
end
```

```
Procedure Altezza_min(v)
if IS_EMPTY(L(v)) = 1  then return 0
else
    begin
        h := n;
        for w ∈ L(v) do { b := Altezza_min(w);
                        { if b < h then h := b
        h := h + 1;
        S := S + h;
        return h;
    end
```

b) Il tempo di calcolo richiesto è $\Theta(n)$, dove n è il numero di nodi, poiché l'algoritmo esegue un numero limitato di istruzioni RAM per ciascun vertice dell'albero.

Esercizio 3.

Considera il problema di calcolare l'espressione

$$a_1 + 2a_2 + 3a_3 + \cdots + na_n$$

assumendo come input una sequenza di interi $a_1, a_2, \ldots, a_n$ preceduta dallo stesso parametro $n \geq 1$.

a) Descrivere un algoritmo del tipo *divide et impera* per risolvere il problema.

b) Assumendo il criterio di costo uniforme, valutare l'ordine di grandezza del tempo di calcolo e dello spazio di memoria richiesti su un input di lunghezza n.

c) Assumiamo il criterio di costo logaritmico e supponiamo che ogni a_i, $i = 1, 2, \ldots, n$, sia un intero di k bits. Determinare una stima O-grande del tempo di calcolo e dello spazio di memoria in funzione dei parametri n e k.

Soluzione

a)

L'idea dell'algoritmo è quella di applicare una procedura ricorsiva che spezza l'input in due parti (circa) uguali, richiama se stessa ricorsivamente su queste ultime e quindi restituisce la somma dei due risultati ottenuti. Definiamo la procedura Calcola(i, j), che su input $1 \leq i \leq j \leq n$, restituisce l'espressione

$$ia_i + (i+1)a_{i+1} + \cdots + ja_j$$

Il programma principale consiste semplicemente nella lettura del parametro $n \geq 1$ e nella chiamata Calcola$(1, n)$.

Procedure Calcola(i, j)

$$\textbf{if } i = j \textbf{ then} \begin{cases} a := read(a_i) \\ \text{return } i \cdot a \end{cases}$$

else

begin

$\ell := \lfloor \frac{j+i}{2} \rfloor$;
$b :=$ Calcola(i, ℓ);
$c :=$ Calcola$(\ell + 1, j)$;
return $b + c$;

end

b)

Valutiamo ora il tempo di calcolo secondo il criterio uniforme. È chiaro che il tempo di calcolo richiesto per eseguire Calcola(i, j) dipende dal numero di elementi della sequenza di input compresi tra gli indici i e j. Definiamo allora $u = j - i + 1$ e denotiamo con $T(u)$ il tempo di calcolo richiesto da Calcola(i, j) secondo il criterio uniforme. Otteniamo quindi la seguente equazione di ricorrenza:

$$T(u) = \begin{cases} c_1 & \text{se } u = 1 \\ T\left(\lfloor \frac{u}{2} \rfloor\right) + T\left(\lceil \frac{u}{2} \rceil\right) + c_2 & \text{se } u \geq 2 \end{cases}$$

dove c_1 e c_2 sono due costanti opportune. Applicando le regole di soluzione si ricava $T(u) = \Theta(u)$. Pertanto il tempo di calcolo complessivo è $\Theta(n)$.

Inoltre lo spazio di memoria richiesto è essenzialmente quello utilizzato dalla pila che implementa la ricorsione. Osserviamo che, su un input di n elementi, la pila raggiunge una altezza massima pari a $1 + \lceil \log_2 n \rceil$: questo è dovuto al fatto che ogni chiamata ricorsiva riduce della metà la dimensione $(j - i + 1)$ della porzione di input corrente. Poiché nel nostro caso ogni record di attivazione occupa uno spazio costante, lo spazio complessivo richiesto dall'algoritmo è $\Theta(\log n)$.

c)

Assumiamo ora il criterio logaritmico. Per ipotesi sappiamo che ogni a_i è un intero di k bit. Applicando i ragionamenti svolti nel punto precedente il tempo di calcolo $T^\ell(u)$ risulta determinato da una equazione della forma

$$T^\ell(u) = \begin{cases} O(k + \log n) & \text{se } u = 1 \\ T^\ell\left(\lfloor \frac{u}{2} \rfloor\right) + T^\ell\left(\lceil \frac{u}{2} \rceil\right) + O(k + \log n) & \text{se } u \geq 2 \end{cases}$$

Infatti, se $u = 1$, $O(\log n)$ è il tempo richiesto per verificare $i = j$ mentre $O(k + \log n)$ è il tempo per calcolare $i \cdot a_i$. Se invece $u > 1$, oltre al tempo $O(\log n)$ per verificare $i < j$, occorre un tempo $O(\log n)$ per calcolare ℓ e un tempo $O(k + \log u)$ per determinare la somma $b + c$ (ricorda che $u \leq n$).

Applicando ora la regola di soluzione (e considerando $O(k + \log n)$ come costante rispetto al parametro u) si ottiene

$$T^\ell(u) = O(k + \log n) \cdot \Theta(u)$$

e fissando $u = n$ si ottiene $T^\ell(u) = O(n \cdot (k + \log n))$.

Per quanto riguarda lo spazio di memoria, valutiamo lo spazio P necessario per mantenere la pila che implementa la ricorsione. Si può provare che ogni record di attivazione matiene al più interi di dimensione complessiva $O(k + \log n)$. Di conseguenza lo spazio richiesto, secondo il criterio logaritmico, risulta $O((k + \log n) \log n)$.

ALGORITMI E STRUTTURE DATI
Svolgimento della prova scritta del 15.12.1999
TEMA N. 2

Esercizio 2.

Ricordiamo che in un albero con radice T, l'*altezza* di un nodo v in T è la massima distanza di v da una foglia.

a) Descrivere una procedura (eventualmente ricorsiva) che, avendo per input un albero con radice T (rappresentato mediante liste di adiacenza), calcoli la somma delle altezze dei nodi di T.

b) Assumendo il criterio di costo uniforme, valutare il tempo di calcolo richiesto in funzione del numero n di nodi dell'albero.

Soluzione

a)

È chiaro che l'altezza di una foglia è 0, mentre l'altezza di un nodo interno è il massimo tra le altezze dei suoi figli, incrementato di 1. Di conseguenza, le altezze di tutti i nodi dell'albero possono essere calcolate mediante una semplice visita in ordine posticipato.

A tale scopo, rappresentiamo con r la radice dell'albero e, per ogni nodo v, denotiamo con $L(v)$ la lista dei figli di v. Definiamo una procedura ricorsiva Altezza(v) che, su input v, restituisce l'altezza del nodo v; tale procedura restituisce il valore 0 se v è una foglia, altrimenti richiama se stessa sui figli di v e determina il massimo dei valori così ottenuti: tale quantità, incrementata di 1, fornisce proprio il valore cercato. La procedura utilizza la variabile globale S per mantenere la somma delle altezze calcolate. Il programma principale, dopo aver letto l'input e inizializzato S, richiama la procedura sulla radice r.

```
begin
    leggi e memorizza l'input
    S := 0
    u := Altezza(r)
    return S
end

Procedure Altezza(v)
if IS_EMPTY(L(v)) = 1  then return 0
else
    begin
        h := 0;
        for w ∈ L(v) do { b := Altezza(w);
                          if h < b then h := b
        h := h + 1;
        S := S + h;
        return h;
    end
```

b) Il tempo di calcolo richiesto è $\Theta(n)$, dove n è il numero di nodi, poiché l'algoritmo esegue un numero limitato di istruzioni RAM per ciascun vertice dell'albero.

ALGORITMI E STRUTTURE DATI

Prova scritta del 29.9.1999

Esercizio 2. Ricordiamo che la *lunghezza di cammino* in un albero con radice è la somma delle distanze dei nodi dalla radice.

a) Descrivere una procedura ricorsiva per calcolare la lunghezza di cammino in un albero con radice qualsiasi.

b) Assumendo il criterio uniforme, valutare il tempo di calcolo richiesto in funzione del numero di nodi dell'albero.

Soluzione

a) Supponiamo che l'albero di ingresso sia rappresentato da una famiglia di nodi V e, per ogni $v \in V$, dalla lista $L(v)$ dei figli di v. Denotiamo inoltre con r la radice dell'albero. L'algoritmo esegue una semplice visita in profondità dell'albero, definita mediante una procedura ricorsiva che chiamiamo `Calcola`; quest'ultima su input $v \in V$ richiama se stessa su tutti i figli di v e aggiorna due variabili globali ℓ e R che rappresentano rispettivamente la distanza dalla radice del nodo corrente v e la somma delle distanze dei nodi già visitati.

Formalmente l'algoritmo è descritto dalle seguenti procedure:

```
begin
    R := 0
    ℓ := 0
    Calcola(r)
    return R
end

Procedure Calcola(v)
begin
    ℓ := ℓ + 1
    R := R + ℓ
    for w ∈ L(v) do Calcola(w)
    ℓ := ℓ − 1
end
```

b) Denotando con n il numero di nodi dell'albero, il tempo di calcolo è $\Theta(n)$.

ALGORITMI E STRUTTURE DATI

Prova scritta del 1°.7.1999

Esercizio 2.

a) Descrivere un algoritmo per risolvere il seguente problema:

Istanza : un grafo diretto $G = \langle V, E \rangle$ di n nodi e m lati, un vertice $s \in V$ e un intero k, $0 \leq k \leq n-1$.

Soluzione : l'insieme dei nodi di G la cui distanza da s è minore o uguale a k.

b) Valutare, in funzione di n e m, il tempo di calcolo e lo spazio di memoria richiesti dall'algoritmo assumendo il criterio di costo uniforme.

c) Svolgere l'esercizio precedente assumendo il criterio di costo logaritmico.

Soluzione

a) Possiamo risolvere il problema mediante un algoritmo di visita in ampiezza che mantiene nella coda solo i nodi incontrati che si trovano a una distanza da s minore di k. Rappresentiamo inoltre il grafo mediante liste di adiacenza: per ogni nodo v, $L(v)$ sia la lista dei nodi adiacenti a v.

L'algoritmo è descritto dalla seguente procedura nella quale R rappresenta la soluzione calcolata, Q rappresenta la coda che mantiene i nodi visitati, D è un vettore di dimensione n tale che, per ogni nodo raggiunto v, $D[v]$ rappresenta la distanza di v da s.

```
begin
    R := {s}
    Q := Enqueue(Λ, s)
    for v ∈ V\{s} do D[v] := ∞
    D[s] := 0
    while Q ≠ Λ do
        begin
            v := Front(Q)
            Q := Dequeue(Q)
            for u ∈ L(v) do
                                    { D[u] := D[v] + 1
                if D[u] = ∞ then    { R := R ∪ {u}
                                    { if D[u] < k then Q := Enqueue(Q, u)
        end
    return R
end
```

b) Assumendo il criterio uniforme, il tempo di calcolo richiesto dall'algoritmo è $O(n + m)$ mentre lo spazio di memoria è $O(n + m)$.

c) Assumendo il criterio logaritmico, il tempo di calcolo richiesto dall'algoritmo è $O((n + m)\log n)$ e la stessa valutazione si ottiene per lo spazio di memoria.

Esercizio 3.

a) Assumendo il criterio di costo uniforme, valutare il tempo di calcolo richiesto dalla seguente procedura che, su input $n \in \mathbb{N}$, determina il valore $L(n)$.

Procedure $L(n)$
begin
$a := 1$
$\ell := 0$
while $a \leq n$ **do** $\begin{cases} a := a \cdot 2 \\ \ell := \ell + 1 \end{cases}$
return ℓ
end

b) Usando la procedura precedente, descrivere un algoritmo basato sul metodo divide et impera per calcolare il valore dell'espressione

$$a_1 L(a_1) + a_2 L(a_2) + \cdots + a_n L(a_n)$$

su input $a_1, a_2, \ldots, a_n \in \mathbb{N}$

c) Supponendo che $a_i \in \{1, 2, \ldots, m\}$ per ogni $i = 1, 2, \ldots, n$, valutare in funzione di n e m il tempo di calcolo e lo spazio di memoria richiesti dall'algoritmo assumendo il criterio di costo uniforme.

Soluzione

a) Il tempo di calcolo richiesto è $O(\log n)$ mentre lo spazio è $O(1)$.

b) L'algoritmo può essere descritto dalla semplice chiamata

$$\text{Calcola}(1, n)$$

della procedura definita nel modo seguente per ogni $1 \leq i \leq j \leq n$:

Procedure Calcola(i, j)
if $i = j$ **then** $\begin{cases} a := read(a_i) \\ \text{return } aL(a) \end{cases}$
else
begin
$k := \lfloor \frac{j+i}{2} \rfloor$;
$b :=$ Calcola(i, k);
$c :=$ Calcola$(k + 1, j)$;
return $b + c$;
end

c)

Sia $T(n)$ il tempo di calcolo richiesto dalla procedura su input di n elementi. Si verifica che, per ogni $n \in \mathbb{N}$ potenza di 2, $T(n)$ soddisfa la seguente equazione di ricorrenza:

$$T(n) = \begin{cases} O(\log m) & \text{se } n = 1 \\ b + 2T\left(\frac{n}{2}\right) & \text{se } n \geq 2 \end{cases}$$

dove b è una opportuna costante maggiore di 0.

Risolvendo l'equazione si ottiene $T(n) = \Theta(n \log m)$.

Per quanto riguarda lo spazio si ottiene invece una valutazione $\Theta(\log n)$.

ALGORITMI E STRUTTURE DATI
Prova scritta del 3.6.1999

Esercizio 2.

a) Descrivere un algoritmo per risolvere il seguente problema:

Istanza : ūn albero ordinato T di n nodi e un intero k, $0 \leq k \leq n-1$.
Soluzione : l'insieme dei nodi di T che si trovano a profondità k.

b) Valutare, in funzione di n, il tempo di calcolo e lo spazio di memoria richiesti dall'algoritmo assumendo il criterio di costo uniforme.

c) Svolgere l'esercizio precedente assumendo il criterio di costo logaritmico.

Soluzione

a) Possiamo risolvere il problema mediante un qualunque algoritmo di visita dell'albero in preordine, mantenendo in un registro opportuno la profondità del nodo corrente che stiamo visitando: il nodo viene quindi inserito nella soluzione se il valore della profondità è proprio k.

Per semplicità descriviamo l'algoritmo mediante un programma principale che richiama una procedura ricorsiva; quest'ultima visita tutti i nodi dell'albero a partire dalla radice. I parametri p e S, che rappresentano rispettivamente la profondità corrente e la soluzione costruita, sono considerati variabili globali.

L'albero di ingresso T viene rappresentato mediante una lista di nodi, il primo dei quali è la radice r, e ogni nodo v è associato alla lista $L(v)$ dei suoi figli.

```
begin
    p := 0
    S := ∅
    v := r
    Esplora(v)
    return S
end
```

```
Procedure Esplora(v)
if p = k  then S := S ∪ {v}
          else
               begin
                    p := p + 1
                    for w ∈ L(v) do Esplora(w)
                    p := p − 1
               end
```

b) Il caso peggiore si verifica quando k è maggiore o uguale alla profondità dell'albero. In questo caso l'algoritmo esplora l'albero completamente ed esegue un numero costante di operazioni per ogni nodo dell'albero. Di conseguenza, assumendo il criterio di costo uniforme, il tempo di calcolo richiesto è $O(n)$.

Lo spazio di memoria è invece essenzialmente determinato dall'altezza massima raggiunta dalla pila che implementa la ricorsione. Nel caso peggiore tale altezza risulta proporzionale al numero di nodi dell'albero e ogni record di attivazione richiede uno spazio costante (secondo il criterio uniforme). Quindi, anche in questo caso abbiamo una valutazione $O(n)$.

c) Applicando i ragionamenti precedenti nel caso di modello di costo logaritmico si ottiene un tempo di calcolo $O(n \log n)$ e uno spazio di memoria $O(n \log n)$.

ALGORITMI E STRUTTURE DATI

Correzione della prova scritta del 14.4.1999

Esercizio 2. Considera il problema di calcolare l'espressione

$$b = (a_1 + a_2) \cdot (a_2 + a_3) \cdots (a_{n-1} + a_n)$$

avendo in input una sequenza $a_1, a_2, \ldots, a_n$ di numeri interi.

a) Descrivere un algoritmo del tipo *divide et impera* per risolvere il problema.

b) Svolgere l'analisi del tempo di calcolo e dello spazio di memoria richiesti dall'algoritmo assumendo il criterio di costo uniforme.

c) Supponendo che $a_i \in \{1, 2, \ldots, n\}$ per ogni indice i, valutare in funzione di n il tempo e lo spazio richiesti dall'algoritmo secondo il criterio logaritmico.

Soluzione

a) Supponiamo $n \geq 2$ e definiamo una procedura $\texttt{Calcola}(i, j)$ che riceve in ingresso due indici i, j tali che $1 \leq i < j \leq n$ e restituisce il valore

$$(a_i + a_{i+1}) \cdot (a_{i+1} + a_{i+2}) \cdots (a_{j-1} + a_j)$$

```
Procedure Calcola (i, j)
begin
        if j = i + 1 then { b := read(a_i)
                          { c := read(a_{i+1})
                          { return b + c
                  else begin
                          k := ⌊(i+j)/2⌋
                          b := Calcola(i, k)
                          c := Calcola(k, j)
                          return b · c
                       end
end
```

l'algoritmo consiste nella semplice chiamata della procedura $\texttt{Calcola}(1, n)$

b) Assumendo il criterio di costo uniforme, denotiamo con $T(u)$ e $S(u)$ il tempo e lo spazio richiesti dalla chiamata $\texttt{Calcola}(i, j)$, dove $u = j - i$. Allora, la sequenza $\{T(u)\}$ verifica una equazione di ricorrenza della forma

$$T(u) = \begin{cases} a & \text{se } u = 1 \\ b + T\left(\lfloor \frac{u}{2} \rfloor\right) + T\left(\lceil \frac{u}{2} \rceil\right) & \text{se } u > 1 \end{cases}$$

per opportune costanti positive a, b. Risolvendo la ricorrenza otteniamo $T(u) = \Theta(u)$. Quindi il tempo di calcolo complessivo risulta $T(n-1) = \Theta(n)$.

Inoltre, lo spazio richiesto è essenzialmente dato dalla dimensione della pila che implementa la ricorsione. Poiché ogni chiamata ricorsiva all'interno della procedura $\texttt{Calcola}(i, j)$ riduce di circa la metà la dimensione del vettore corrente $a_i, a_{i+1}, \ldots, a_j$, la pila può contenere al più $\lceil \log_2 n \rceil$ record di attivazione. Lo spazio richiesto risulta quindi $S(n-1) = \Theta(\log n)$.

c) Assumendo il criterio di costo logaritmico, denotiamo con $T^\ell(u)$ e $S\ell(u)$ il tempo e lo spazio richiesti dalla chiamata `Calcola`(i, j), dove $u = j - i$. Allora, la sequenza $\{T^\ell(u)\}$ verifica una equazione di ricorrenza della forma

$$T^\ell(u) = \begin{cases} \Theta(\log n) & \text{se } u = 1 \\ T^\ell\left(\lfloor \frac{u}{2} \rfloor\right) + T^\ell\left(\lceil \frac{u}{2} \rceil\right) + \Theta(u)\Theta(\log n) & \text{se } u > 1 \end{cases}$$

Risolvendo la ricorrenza otteniamo $T^\ell(u) = \Theta(u \log u)\Theta(\log n)$. Quindi il tempo di calcolo complessivo risulta $T^\ell(n-1) = \Theta(n \log^2 n)$.

Anche in questo caso, lo spazio richiesto è essenzialmente dato dalla dimensione della pila che implementa la ricorsione. Tuttavia lo spazio occupato da ciascun record dipende ora dalla dimensione dei valori che contiene. Si ricava quindi l'espressione

$$S^\ell(n-1) = \Theta\left(\sum_{i=1}^{\log_2 n} \frac{n}{2^i} \log n\right) = \Theta(n \log n).$$

Esercizio 3. Ricordiamo che la *lunghezza di cammino* in un albero con radice è la somma delle distanze dei nodi dalla radice.

a) Descrivere una procedura ricorsiva per calcolare la lunghezza di cammino in un albero con radice qualsiasi.

b) Assumendo il criterio uniforme, valutare il tempo di calcolo richiesto in funzione del numero di nodi dell'albero.

Soluzione

Supponiamo che l'albero sia rappresentato da liste di adiacenza e denotiamo con $L(v)$ la lista dei nodi adiacenti al vertice v. Denotiamo inoltre con r la radice dell'albero.

L'algoritmo consiste di un programma principale e una procedura ricorsiva che esplora l'albero in ordine anticipato. Entrambi utilizzano due variabili globali S e d che rappresentano rispettivamente la soluzione parziale finora calcolata e la distanza dalla radice del nodo corrente.

```
Procedure Principale
begin
      S := 0
      d := 0
      Lunghezza(r)
end
```

La seguente procedura esplora il sottoalbero che ha per radice il nodo v. Al momento della sua chiamata il valore di d rappresenta già la distanza di v dalla radice r.

```
Procedure Lunghezza (v)
begin
      S := S + d
      d := d + 1
      for u ∈ L(v) do Lunghezza(u)
      d := d − 1
end
```

b) Il tempo di calcolo richiesto risulta proporzionale (al più) al numero n di nodi dell'albero e quindi otteniamo una valutazione di ordine $\Theta(n)$.

ALGORITMI E STRUTTURE DATI

Correzione della prova scritta del 24.2.1999

Esercizio 1.

Considera le seguenti procedure A e B che calcolano rispettivamente i valori $A(n)$ e $B(n)$ su input $n \in \mathbb{N}$:

Procedure $B(n)$
begin
  $S = 0$
  for $i = 0, 1, 2, \ldots, n$ **do**
    $S = S + A(i)$
  return S
end

Procedure $A(n)$
if $n = 0$ **then return** 2
  else return $2 + nA(n-1)$

a) Determinare l'ordine di grandezza del tempo di calcolo e dello spazio di memoria richiesti dalla procedura B su input n assumendo il criterio di costo uniforme.

Denotiamo con $T_A(n)$ e $S_A(n)$ il tempo e lo spazio richiesti dalla procedura A su input n, secondo il criterio uniforme. Allora, la sequenza $\{T_A(n)\}$ verifica una equazione di ricorrenza della forma

$$T_A(n) = \begin{cases} a & \text{se } n = 0 \\ b + T_A(n-1) & \text{se } n > 0 \end{cases}$$

per opportune costanti positive a, b. Sviluppando la ricorrenza otteniamo $T_A(n) = \Theta(n)$. Inoltre, la procedura A su input n richiama se stessa n volte e ogni record di attivazione (nelle nostre ipotesi) occupa uno spazio costante. Di conseguenza abbiamo $S_A(n) = \Theta(n)$.

Denotiamo ora con $T_B(n)$ e $S_B(n)$ il tempo e lo spazio richiesti dalla procedura B su input n, secondo il criterio uniforme. Si verifica facilmente che

$$T_B(n) = c + \sum_{i=0}^{n} (d + T_A(i))$$

Quindi, usando la valutazione di $T_A(n)$ ottenuta sopra, si ricava $T_B(n) = \Theta(n^2)$. Infine, per quanto riguardo lo spazio, si verifica che $S_B(n)$ è dello stesso ordine di grandezza di $S_A(n)$ e quindi $S_B(n) = \Theta(n)$.

b) Svolgere l'esercizio richiesto al punto a) assumendo il criterio di costo logaritmico.

Per determinare la complessità delle procedure secondo il criterio logaritmico valutiamo la dimensione di valori calcolati (cioè il numero di bit necessari per rappresentarli). Abbiamo

$$A(n) = \begin{cases} 2 & \text{se } n = 0 \\ 2 + nA(n-1) & \text{se } n > 0 \end{cases}$$

e quindi, per ogni $n \geq 1$,

$$A(n) = 2 + n(2 + (n-1)A(n-2)) = \cdots = 2\{1 + n + n(n-1) + n(n-1)(n-2) + \cdots + n!\}$$

Di conseguenza si verifica la disuguaglianza

$$2(n!) \leq A(n) \leq 2(n+1)(n!)$$

Passando ora ai logaritmi, otteniamo

$$n \log n + O(n) \leq \log A(n) \leq n \log n + O(n)$$

e quindi le dimensione di $A(n)$ e $B(n)$ risultano rispettivamente

$$|A(n)| = \Theta(n \log n)$$

$$|B(n)| = |\sum_{i=0}^{n} A(i)| = \Theta(n \log n)$$

Denotiamo ora con $T_A^l(n)$ e $S_A^l(n)$ il tempo e lo spazio richiesti dalla procedura A su input n, secondo il criterio logaritmico (useremo poi la stessa notazione per B). Si verifica allora

$$T_A^l(n) = \begin{cases} h & \text{se } n = 0 \\ k + T_A^l(n-1) + \Theta(n \log n) & \text{se } n > 0 \end{cases}$$

e quindi, sviluppando l'equazione, si ricava

$$T_A^l(n) = \Theta\left(\sum_{i=1}^{n} i \log i\right) = \Theta(n^2 \log n)$$

$$T_B^l(n) = \Theta\left(\sum_{i=1}^{n} T_A^l(i) + i \log i\right) = \Theta(n^3 \log n)$$

Per quanto riguarda lo spazio richiesto osserva che nella procedura A il record di attivazione relativo alla chiamata su input j occupa uno spazio dell'ordine $\Theta(j)$; inoltre, occorre uno spazio $\Theta(n \log n)$ per mantenere il risultato del calcolo e quindi:

$$S_A^l(n) = \Theta(n \log n) + \sum_{i=1}^{n} \Theta(\log i) = \Theta(n \log n)$$

$$S_B^l(n) = \Theta(S_A^l(n)) = \Theta(n \log n)$$

c) Definire una procedura per il calcolo di $B(n)$ che richieda tempo $O(n)$ e spazio $O(1)$ secondo il criterio di costo uniforme.

```
begin
    S := 0
    a := 0
    for i = 0, 1, 2, ..., n do
        a := 2 + ia
        S := S + a
    return S
end
```

Esercizio 2.

Chiamiamo *albero ternario* un albero con radice nel quale ogni nodo interno possiede al più tre figli e ogni figlio è distinto come *figlio sinistro*, *figlio centrale* oppure *figlio destro*.

a) Descrivere una struttura dati per rappresentare un albero ternario formato da n nodi.

Rappresentiamo ogni nodo mediante un intero compreso tra 1 e n. Inoltre definiamo 4 vettori a componenti in $\mathbb{N}$, ciascuno di dimensione n: P, S, C, D.

$$P[v] = \begin{cases} 0 & \text{se } v \text{ e' radice} \\ u & \text{se } u \text{ e' il padre di } v \end{cases}$$

$$S[v] = \begin{cases} 0 & \text{se } v \text{ non ha figlio sinistro} \\ u & \text{se } u \text{ e' il figlio sinistro di } v \end{cases}$$

$$C[v] = \begin{cases} 0 & \text{se } v \text{ non ha figlio centrale} \\ u & \text{se } u \text{ e' il figlio centrale di } v \end{cases}$$

$$D[v] = \begin{cases} 0 & \text{se } v \text{ non ha figlio destro} \\ u & \text{se } u \text{ e' il figlio destro di } v \end{cases}$$

b) Definire una procedura ricorsiva per attraversare un albero ternario che, partendo dalla radice, visiti ciascun nodo v applicando il seguente criterio:

prima attraversa il sottoalbero che ha per radice il figlio sinistro di v,
poi attraversa il sottoalbero che ha per radice il figlio destro di v,
quindi visita il nodo v,
infine, attraversa il sottoalbero che ha per radice il figlio centrale di v.

Denotando con r la radice dell'albero, l'algoritmo consiste nella semplice chiamata `Attraversa`(r) della procedura definita nel modo seguente (sul generico nodo v):

Procedure `Attraversa`(v)
begin
  if $S[v] \neq 0$ **then** `Attraversa`($S[v]$)
  if $D[v] \neq 0$ **then** `Attraversa`($D[v]$)
  visita v;
  if $C[v] \neq 0$ **then** `Attraversa`($C[v]$)
end

c) Descrivere una versione iterativa dell'algoritmo precedente.

begin
  $P := \Lambda$ (P pila vuota)
  $v := r$

1) **while** $S[v] \neq 0$ **do** $\begin{cases} P := \text{Push}(P, (v, d)) \\ v := S[v] \end{cases}$

2) **if** $D[v] \neq 0$ **then** $\begin{cases} P := \text{Push}(P, (v, c)) \\ v := D[v] \\ \text{go to 1)} \end{cases}$

```
3)        visita v
          if C[v] ≠ 0      then { v := C[v]
                                { go to 1)
                                { (v,h) := Top(P)
          if P ≠ Λ         then { P := Pop(P)
                                {  if h=d then go to 2) else go to 3)
     end
```

ALGORITMI E STRUTTURE DATI
Correzione della prova scritta del 13.1.1999

Esercizio 3.
Dati due interi k, n, $1 \leq k \leq n$, sia E_n l'insieme dei primi n interi positivi:

$$E_n = \{1, 2, \ldots, n\},$$

e sia F_k la famiglia dei sottoinsiemi di E_n che hanno al più k elementi:

$$F_k = \{A \subseteq E_n \mid \sharp A \leq k\}.$$

a) Per quali interi k, n, $1 \leq k \leq n$, la coppia $\langle E_n, F_k \rangle$ forma un sistema di indipendenza?
b) Per quali interi k, n, $1 \leq k \leq n$, la coppia $\langle E_n, F_k \rangle$ forma un matroide?
c) Definire un algoritmo greedy per il problema seguente:

Istanza : due interi k, n, $1 \leq k \leq n$, e una funzione peso $w : E_n \rightarrow \mathbb{R}^+$.
Soluzione : il sottoinsieme $A \in F_k$ di peso massimo.

d) Per quali k e n l'algoritmo determina sempre la soluzione ottima?

Soluzione
a) La coppia $\langle E_n, F_k \rangle$ forma un sistema di indipendenza per ogni coppia di interi k, n tali che $1 \leq k \leq n$.
b) La coppia $\langle E_n, F_k \rangle$ forma un matroide per ogni coppia di interi k, n tali che $1 \leq k \leq n$.
c)

```
begin
    A := ∅
    E := E_n
    for i = 1, 2, ..., k do
        begin
            calcola l'elemento u di peso massimo in E ;
            A := A ∪ {u};
            E := E\{u};
        end
    return A
end
```

d) L'algoritmo determina la soluzione ottima per tutti gli interi k, n, $1 \leq k \leq n$.

ALGORITMI E STRUTTURE DATI

Correzione della prova scritta del 14.12.1998

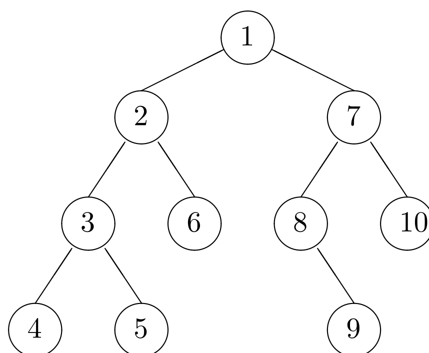
Esercizio 1.

Rappresentare graficamente un albero binario di 10 nodi e numerare i vertici secondo l'ordine di visita ottenuto attraversando l'albero in preordine (ordine anticipato).

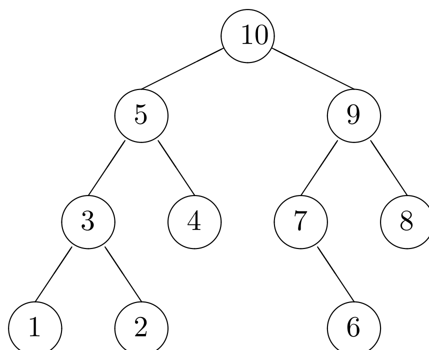
Ripetere l'esercizio considerando gli attraversamenti in postordine (ordine posticipato) e in ordine simmetrico (inorder).

Soluzione

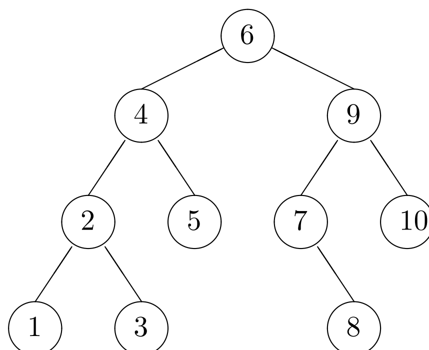
Numerazione in preordine:



Numerazione in postordine:



Numerazione in ordine simmmetrico



Esercizio 2.

a) Descrivere un algoritmo per il seguente problema:

Istanza : un grafo orientato $G = \langle V, E \rangle$ rappresentato mediante liste di adiacenza e un nodo $s \in V$.

Soluzione : per ogni nodo $v \in V$ raggiungibile da s, la distanza tra s e v (ovvero la lunghezza del più corto cammino che congiunge s a v).

b) Svolgere l'analisi dell'algoritmo valutando il tempo di calcolo e lo spazio di memoria richiesti su un input di n nodi e m lati (si assuma il criterio uniforme).

Soluzione

a) Supponiamo che il grafo di ingresso G sia rappresentato da un insieme di nodi V e dalla famiglia di liste di adiacenza $\{L(v) \mid v \in V\}$. Per ogni $v \in V$, L(v) è la lista dei nodi w per i quali esiste in G il lato (v, w).

L'algoritmo che presentiamo calcola per ogni $v \in V$ il valore $N[v]$ che rappresenta la distanza di s da v (se v non è raggiungibile da s $N[v]$ assume il valore convenzionale $+\infty$). L'algoritmo esegue una esplorazione in ampiezza del grafo diretto G a partire dal nodo s.

```
begin
    for w ∈ V\{s} do N[w] := +∞
    N[s] := 0
    Q := Enqueue(Λ, s);
    while Q ≠ Λ do
        begin
            v := Front(Q);
            Q := Dequeue(Q);
            for w ∈ L(v) do
                if N[w] = +∞ then { N[w] := N[v] + 1
                                  { Q := Enqueue(Q, w);
        end
end
```

b) Sia n il numero dei nodi di G e sia m il numero dei suoi lati. Allora l'algoritmo sopra descritto lavora in tempo $O(n + m)$ e utilizza uno spazio $O(n + m)$.

Esercizio 3.

a) Descrivere un algoritmo del tipo "divide et impera" per risolvere il seguente problema:

Istanza : una sequenza di $n \geq 1$ numeri interi $a_1, a_2, \ldots, a_n$.

Soluzione : la somma $b = a_1 + a_2 + \cdots + a_n$.

b) Valutare (in funzione di n) il tempo di calcolo e lo spazio di memoria richiesti dall'algoritmo assumendo il criterio di costo uniforme.

c) Supponendo che ogni a_i abbia m bit e assumendo il criterio di costo logaritmico, valutare (in funzione di n e m) il tempo di calcolo e lo spazio di memoria richiesti dall'algoritmo.

Soluzione

a) L'idea dell'algoritmo è quella di applicare una procedura ricorsiva che spezza l'input in due parti (circa) uguali, richiama se stessa ricorsivamente su queste ultime e quindi restituisce la somma dei due risultati ottenuti. Formalmente l'algoritmo si riduce alla chiamata

$$C := \text{Somma}(1, n)$$

della procedura Somma(i, j), definita nel seguito, che restituisce la somma degli elementi

$$a_i, a_{i+1}, \ldots, a_j$$

con $1 \leq i \leq j \leq n$.

```
Procedure Somma(i, j)
if i = j then { b := read(a_i)
              { return b
      else
          begin
                k := ⌊(j+i)/2⌋;
                b := Somma(i, k);
                c := Somma(k + 1, j);
                return b + c;
          end
```

b) È chiaro che il tempo di calcolo richiesto per eseguire Somma(i, j) dipende dal numero di elementi che occorre moltiplicare tra loro. Definiamo allora $u = j - i + 1$ e denotiamo con $T(u)$ il tempo di calcolo richiesto da Somma(i, j) secondo il criterio uniforme. Otteniamo quindi la seguente equazione di ricorrenza:

$$T(u) = \begin{cases} c_1 & \text{se } u = 1 \\ T\left(\lfloor \frac{u}{2} \rfloor\right) + T\left(\lceil \frac{u}{2} \rceil\right) + c_2 & \text{se } u \geq 2 \end{cases}$$

dove c_1 e c_2 sono due costanti opportune. Applicando le regole di soluzione si ricava $T(u) = \Theta(u)$. Pertanto il tempo di calcolo complessivo è $\Theta(n)$.

Inoltre lo spazio di memoria richiesto è essenzialmente quello utilizzato dalla pila che implementa la ricorsione. Osserviamo che, su un input di n elementi, la pila raggiunge una altezza massima pari a $1 + \lceil \log_2 n \rceil$: questo è dovuto al fatto che ogni chiamata ricorsiva riduce della metà la dimensione $(j - i + 1)$ della porzione di input corrente. Poiché nel nostro caso ogni record di attivazione occupa uno spazio costante, lo spazio complessivo richiesto dall'algoritmo è $\Theta(\log n)$.

c) Assumiamo ora il criterio logaritmico. Per ipotesi sappiamo che ogni a_i è un intero di m bit. Ponendo quindi $u = j - i + 1$, la procedura Somma(i, j) richiede un tempo di calcolo costituito dalla somma delle seguenti quantità :

1. un tempo $O(\log_2 n)$ per manipolare gli interi i, j e calcolare k,
2. il tempo necessario per eseguire le due chiamate ricorsive,
3. il tempo necessario per calcolare la somma $b + c$.

Il costo della terza operazione è dell'ordine $\Theta(m + \log_2 u)$ poiché b e c assumono al più il valore $2^m \cdot \frac{u}{2}$.

Allora, poiché $u \leq n$, il tempo di calcolo $T^\ell(u)$ di Somma(i, j) risulta maggiorato da una equazione della forma

$$T^\ell(u) \leq \begin{cases} am + b\log_2 n & \text{se } u = 1 \\ T^\ell\left(\lfloor \frac{u}{2} \rfloor\right) + T^\ell\left(\lceil \frac{u}{2} \rceil\right) + cm + d\log_2 n & \text{se } u \geq 2 \end{cases}$$

dove a, b, c, d sono costanti opportune. Sviluppando ora la relazione di ricorrenza nell'ipotesi $u = 2^k$, si ottiene

$$T^\ell(u) = (O(m) + O(\log_2 n)) \sum_{i=0}^{k} 2^i = O(mu) + O(u \log n)$$

Di conseguenza, il tempo di calcolo sull'input considerato risulta $O(nm) + O(n \log n)$.

Per quanto riguarda lo spazio di memoria, valutiamo lo spazio necessario per mantenere la pila che implementa la ricorsione. Supponendo che n sia una potenza di 2, tale quantità assume un valore massimo pari all'espressione seguente nella quale si tiene conto degli indici i, j, k e dei risultati parziali b, c mantenuti in ogni record di attivazione:

$$O(m \log_2 n) + O((\log_2 n)^2)$$

Chiaramente l'ordine di grandezza resta il medesimo anche per valori di n diversi da potenze di 2.