FONDAMENTI DI INFORMATICA II – Complessità, Algoritmi e Strutture dati

19 settembre 2008 - ANNO ACCADEMICO 2007/08

NOME: **COGNOME:** **MATRICOLA:**

SCRITTO RIDOTTO: **SI NO (barrare SI se si intende svolgere lo scritto ridotto)**

Siano date le due stringhe "STARTS" e "DISTRUST". Trovare, utilizzando l'algoritmo di programmazione dinamica visto a lezione

a) la lunghezza della più lunga sottosequenza comune alle due stringhe

b) tutte le sottosequenze di lunghezza massima

		S	T	A	R	T	S
	0	0	0	0	0	0	0
D	0	0	0	0	0	0	0
I	0	0	0	0	0	0	0
S	0	1	1	1	1	1	1
T	0	1	2	2	2	2	2
R	0	1	2	2	3	3	3
U	0	1	2	2	3	3	3
S	0	1	2	2	3	3	4
T	0	1	2	2	3	4	4

Lunghezza massima = ____4____

Sottosequenze di lunghezza massima = STRT, STRS

Esercizio 2

Confrontare le due funzioni F e G dal punto di vista della complessità: dire se una è O dell'altra e viceversa. In caso affermativo, indicare una coppia (n_0,c). In caso negativo, giustificare la risposta.

F(x)=	x^4+2x^2+1	se x<20	G(x)=	6x	se x pari
	$2x^2-1$	se x>=20 e primo		$5x^3$	se x dispari
	3x	altrimenti			

F è O(G) n_0 = 20, c=1 (i numeri primi maggiori di 19 sono tutti dispari)

G non è O(F) Sui numeri dispari superiori a 20 (che sono infiniti) G è cubica mentre F è lineare o quadratica

Situazione	*Esercizi*	*Tempo*
Senza progetto	**1,2,3,4,5**	80 min
Con progetto	**1,2,3**	40 min

1	*2*	*3*	*4*	*5*

Esercizio 3

Sia **l** il puntatore ad una lista singola di interi. Calcolare la complessità dell'istruzione

```
for(int i=0;i<f(l); i++) cout << "!";
```

in funzione della lunghezza **n** di **l** (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) con le funzioni **f** e **g** definite come segue:

```
int f(Elem* h) {
   if (!h) return 1;
   int a=0;
   for(Elem* e=h;e;e=e->next){
      e->inf++;
      a++;
   }
   return g(a)*g(a)+f(h->next);
}
```

```
int g(int x) {
   if (x<=1) return 1;
   int b = 1+g(x/2);
   return 2*b;
}
```

Tempo di g

$T_g(x) = cost.$ per $x<=1$

$T_g(x) = cost+T_g(x/2)$ altrimenti

$T_g(x)$ è $O(\log x)$

Tempo di f (in funzione della lunghezza n di h)

$T_f(n) = cost.$ per $n=0$

$T_f(n) = cost*n+2T_g(n)+T_f(n-1) = cost*n+T_f(n-1)$ altrimenti

$T_f(n)$ è $O(n^2)$

la variabile a alla fine del for contiene la lunghezza di h quindi n

Risultato di g

$R_g(x) = cost$ per $x <=1$

$R_g(x) = 2+2*R_g(x/2)$ altrimenti

$R_g(x)$ è $O(x)$

Risultato di f

$R_f(n) = cost$ per $n<=1$

$R_f(n) = R_g(n)*R_g(n) + R_f(n-1) = cost*n^2$ altrimenti

Rf(n) è O(n3)

Calcolo del for

Numero iterazioni: $R_f(n)$ che è $O(n^3)$

Complessità singola iterazione = cost + $T_f(n) = O(n^2)$

complessità del for: $O(n^3)*O(n^2)=O(n^5)$

Esercizio 4

Sia dato un grafo orientato contenente N nodi numerati da 0 a N-1 e memorizzato tramite liste di adiacenza. Si scriva la dichiarazione di una struct Grafo che lo realizza. Definiamo gradi di connessione di un nodo k il numero di nodi raggiungibili (direttamente o indirettamente) da k. Si scriva una funzione **raggiungi** che, dato un Grafo g e un intero 0<=i<N, restituisce il grado di connessione del nodo i.

```
struct Graph {
 int N;             // numero di nodi
 struct Node {
 int nodeNumber ;   // nodo destinazione dell'arco
 Node* next ;       // arco successivo
 };
 Node **graph;      // array di liste di adiacenza
};

void visita(Graph &g, int i, bool* m) {
 m[i] = true;
 Graph::Node* p = g.graph[i];
 while (p) {
   int nodeId = p->nodeNumber;
   if (!m[nodeId])
      visita(g,nodeId,m);
   p = p->next;
 }
}

int raggiungi(Graph& g, int i) {
 int N=g.N;
 int j;
 bool* mark = new bool[N];
 for (j=0;j<N;j++)
    mark[j]=0;
 visita(g,i,mark);
 int raggiungibili = 0;
 for (j=0;j<N;j++)
    if (mark[j])
       raggiungibili++;
 delete[] mark;
 return raggiungibili;
}
```

Esercizio 5

Modificare la struttura dati albero binario di ricerca ad etichette intere in maniera che ogni nodo contenga anche un campo **`liv`** che rappresenta il livello del nodo. Riscrivere quindi la procedure di inserimento e calcolarne la complessità in funzione del numero di nodi, considerando alberi quasi bilanciati.

```
struct Node {
  int label;
  Node *left, *right;
  int liv;
};

void insert(Node*& t, int info, int livello=0) {
  if (!t) {
    t = new Node;
    t->left = t->right = NULL;
    t->liv = livello;
    return;
  }
  if (t->label < info)
    insert(t->right,info,livello+1);
  if (t->label > info) {
    insert(t->left,info,livello+1);
  }
}
```

la complessità è pari a O(log n)