FONDAMENTI DI INFORMATICA II – Complessità, Algoritmi e Strutture dati

18 settembre 2009 - 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 "LATRAI" e "AREATI". 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

		L	A	T	R	A	I
	0	0	0	0	0	0	0
A	0	0	1	1	1	1	1
R	0	0	1	1	2	2	2
E	0	0	1	1	2	2	2
A	0	0	1	1	2	3	3
T	0	0	1	2	2	3	3
I	0	0	1	2	2	3	4

Lunghezza massima = ____4____

Sottosequenze di lunghezza massima = ARAI

Esercizio 2

Si supponga di dover memorizzare in una tabella hash 900 numeri telefonici. Un certo numero telefonico può essere presente nella tabella al più una volta. Si supponga che un numero telefonico sia una sequenza di 6 cifre comprese fra 0 e 9, la cui prima cifra è 7. Il numero è rappresentato come un unsigned int su 4 byte. Si scelga a) una funzione hash iniettiva opportuna e b) una funzione non iniettiva e tabella a indirizzamento aperto con capacità di carico pari a 75%. Si indichi inoltre la memoria occupata dalla tabella nei due casi.

a)	**b)**
h(x) = x-700000 Dim. tabella = 100000*1 bit = circa 12KB	h(x) = x % 1200 Dim. Tabella = 1200 * 4 byte = 4800 byte = circa 5KB

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
Calcolare la complessità del blocco (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) in funzione di n:

```
{
  int a = 0;
  for (int i=0; i <= g(n)/n; i++)
     a += f(n);
}
```

con le funzioni **f** e **g** definite come segue:

```
int f(int x) {
   if (x<=1)
     return 1;
   cout << f(x/3) + f(x/3);
   return f(x/3) + 1;
}
```

```
int g(int x) {
  int a=0,i;
  for (i=0; i <= f(x); i++)
     a++;
  for (i=0; i <= 2*f(x); i++)
     a+=i;
  return a;
}
```

Indicare per esteso le relazioni di ricorrenza e, per ogni comando ripetitivo, il numero di iterazioni e la complessità dell'iterazione singola.

$T_f(0,1)= k_1$
$T_f(n)= k_2 + 3\ T_f(n/3)$ $T_f(n)$ è $O(n)$
$R_f(0,1)= 1$
$R_f(n)= 1 + R_f(n/3)$ $R_f(n)$ è $O(\log n)$

Calcolo $T_g(n)$
Entrambi i for hanno
numero iterazioni: $O(\log n)$ Complessità singola iterazione: $O(n)$
complessità dei for: $O(n \log n)$

il valore di alla fine e' $O(\log^2 n)$ (somma dei primi log n numeri naturali)
$T_g(n)$ è $O(n \log n)$
$R_g(n)$ è $O(\log^2 n)$

Calcolo for del blocco:
numero iterazioni: $R_g(n)/n = O((\log^2 n)/n))$
Complessità singola iterazione: $T_g(n)+ T_f(n)+= O(n \log n) + O(n)=O(n \log n)$
Complessità totale blocco = $O(\log^3 n)$

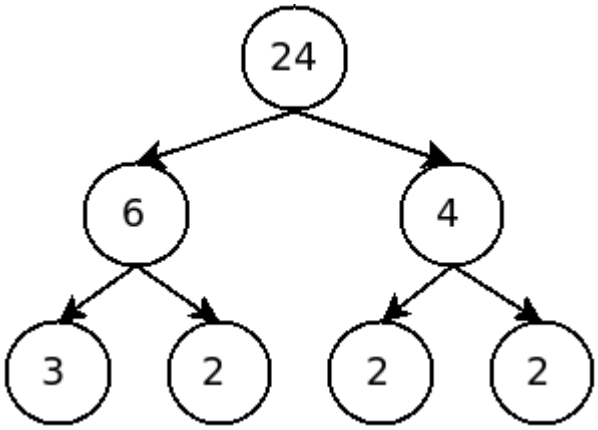
Esercizio 4

Scrivere una funzione **`bool potatura(Node* t)`** che cancella il primo sottoalbero di ogni nodo. Si supponga che l'albero generico sia non vuoto e memorizzato figlio-fratello.

```
void potatura (Node* t) {
  if (!t) return;
  if (!t->left) {
      potatura(t->right);
      return;
  }
  Node* temp=t->left;
  t->left= t->left->right;
  temp->right = NULL;
  deltree(temp);
  potatura (t->left);
  potatura (t->right);
}

void deltree (Node*& t) {
  if (!t) return;
  deltree (t->left);
  deltree (t->right);
  delete (t);
  t = NULL;
}
```

Esercizio 5



Si supponga di disporre di una funzione

```
unsigned int divisore_proprio(unsigned
                int n)
```

che restituisce 0 se n è primo, un divisore proprio di n altrimenti. I divisori propri sono i divisori diversi da 1 e dal numero stesso. Utilizzando la funzione **`divisore_proprio`**, scrivere una funzione

```
Node* scomponi(unsigned int n)
```

che restituisce un albero binario ad etichette intere con etichetta della radice n e tale che le foglie contengono numeri primi e ogni nodo che non è foglia con etichetta h ha come figli due nodi con etichetta h1 e h2 tali che siano divisori propri di h e che h1*h2 sia uguale a h. Per esempio, un possibile albero generato da scomponi(24) potrebbe essere come quello in figura

```
Node* scomponi(unsigned int k) {
  Node* t = new Node;
  t->label = k;
  unsigned int div = divisore_proprio(k);
  if (div){
    t->left = scomponi(div);
    t->right = scomponi(k/div);
  } else {
    t->left = NULL;
    t->right = NULL;
  }
  return t;
}
```