FONDAMENTI DI INFORMATICA II – Complessità, Algoritmi e Strutture dati

1 febbraio 2010 - ANNO ACCADEMICO 2008/09

NOME: COGNOME: MATRICOLA:

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

Esercizio 1

Esercizio 2

Sia dato il seguente vettore A di interi:

[91 73 97 0 32 11 61]

Scrivere tutte le chiamate a quicksort generate dalla chiamata quicksort(A, 0, 6). Per ogni chiamata indicare lo stato dell'array e il valore dei parametri attuali e del perno.

Array A	*inf*	*sup*	*perno*
[91 73 97 0 32 11 61]	0	6	0
[0 73 97 91 32 11 61]	1	6	91
[0 73 61 11 32 91 97]	1	4	61
[0 32 11 61 73 91 97]	1	2	32
[0 11 32 61 73 91 97]	3	4	61
[0 11 32 61 73 91 97]	5	6	91
[0 11 32 61 73 91 97]			

Esercizio 2

Sia dato un grafo orientato, in cui ogni arco è etichettato con un intero strettamente positivo rappresentato su 2 byte. Anche il numero d'ordine del nodo occupa 2 byte. Si supponga che il grafo abbia n nodi e m archi. Calcolare la complessità dell'operazione di cancellazione dell'arco che congiunge i nodi i e j nei seguenti scenari:

2-a) il grafo è memorizzato tramite una matrice di adiacenza

2-b) il grafo è memorizzato tramite liste di adiacenza

2-c) Supponendo inoltre che un puntatore occupi 4 byte e che il numero di nodi n sia fissato a 6, indicare per quali valori di m la matrice di adiacenza occupa meno spazio rispetto alle liste di adiacenza.

2-a) O(1)
2-b) O(m)
2-c) m > 6 Infatti la matrice occupa 2n^2 mentre le liste 4n+8m

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 dell'istruzione

```
for (int i=0; i<=f(t); i++) cout << g(t->left) + g(t->right);
```

(indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) in funzione del numero di nodi di t. Supporre che t sia un albero binario bilanciato e le funzioni **f** e **g** siano definite come segue:

```
int f(Node* t) {
   if (! t)
     return 1;
  return g(t->left) + f(t->left) +
f(t->right);
}
```

```
int g(Node * t) {
  int a=0;
  int n = nodes(t);
  for (int i=0; i<n; i++)
     a++;
  return a*a;
}
```

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

Calcolo Tg(n)
Il for ha numero iterazioni n, quindi O(n) Complessità singola iterazione: O(1)
complessità dei for: O(n)
Tg(n) è O(n)

Rg(n) è O(n^2)

Tf(0)= d
Tf(n)= cn + 2 Tf(n/2) Tf(n) è O(nlogn)

Rf(0)= d
Rf(n)= cn^2 + 2 Rf(n/2) Rf(n) è O(n^2)

Numero di iterazioni del for: O(n^2)
Complessità di una iterazione: Tf(n) + 2Tg(n/2)= O(nlogn) + O(n)= O(nlogn)

Complessità del for: O(n^3logn)

Esercizio 4
Sia dato un albero generico ad etichette intere memorizzato figlio fratello. Si scriva una funzione pota che prende in ingresso un intero positivo k e provvede a cancellare dall'albero tutti i nodi di livello maggiore di k.

```
void cancella(Node*& t) {
  if (t) {
    cancella(t->left);
    cancella(t->right);
    delete t;
    t=0;
  }
}

void pota(Node* t, int k, int liv=0) {
  if (t) {
    if (liv==k) {
      cancella(t->left);
      pota(t->right,k,liv);
    } else {
      pota(t->left,k,liv+1);
      pota(t->right,k,liv);
    }
  }
}
```

Esercizio 5
Sia dato un albero binario di ricerca non vuoto contenente etichette intere. Si scriva una funzione tutte_minori che prende in ingresso due interi h e k, con h<=k e restituisce true se tutte le etichette dell'albero sono comprese nell'intervallo [h,k], false altrimenti. Si calcoli la complessità della soluzione proposta in funzione del numero di nodi dell'albero e ipotizzando che l'albero sia bilanciato.

```
int min(Node* t) {
  if (!(t->left)) return t->label;
  return min(t->left);
}

int max(Node* t) {
  if (!(t->right)) return t->label;
  return max(t->right);
}

bool comprese(Node* t, int h, int k) {
  return ((min(t)>=h) && (max(t)<=k));
}
```

Complessità: O(log n)