FONDAMENTI DI INFORMATICA II – Complessità, Algoritmi e Strutture dati

18 luglio 2008 - ANNO ACCADEMICO 2007/08

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

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

Esercizio 1

Si supponga di dover memorizzare un insieme di elementi, ciascuno composto da due attributi di cui uno è la chiave e l'altro l'informazione. Confrontare le seguenti soluzioni di memorizzazione interna valutando la complessità delle operazioni richieste in funzione del numero n di elementi.
Nelle ipotesi in cui nell'insieme siano presenti al più 1000 elementi e mediamente 500, che un puntatore occupi 1 cella di memoria, e che la chiave occupi 1 cella di memoria, l'informazione 10, k=100, calcolare lo spazio in memoria richiesto (medio) in numero di celle di memoria.

Struttura dati	*Ricerca*	*Inserimento*	*Cancellazione*	*Memoria*
Array non ordinato	O(n)	O(1)	O(n)	11*1000=11000
Array ordinato	O(log n)	O(n)	O(n)	11*1000=11000
Lista semplice non ordinata	O(n)	O(1)	O(n)	12*500=6000
Lista semplice ordinata	O(n)	O(n)	O(n)	12*500=6000
Albero binario di ricerca	O(log n)	O(log n)	O(log n)	13*500=6500
Tab. Hash con concatenazione e k dimensione del vettore	O(n/k)	O(1)	O(n/k)	12*500+100=6100

Esercizio 2

Sia **`t`** un albero binario ad etichette intere. Scrivere una funzione che restituisce la somma delle etichette delle foglie di **`t`**. Calcolarne la complessità in funzione del numero di nodi *n* di **`t`**.

```
int sommafoglie(Node* t) {
   if (!t) return 0;
   if ((!t->left)&&(!t->right))
     return t->label;
   return sommafoglie(t->left)+sommafoglie(t->right);
}
```

Complessità: O(n)

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

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

Esercizio 3

Calcolare la complessità della chiamata **`g(n)`** in funzione di **`n`** (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) con le funzioni **`f`** e **`g`** definite come segue::

```
int f(int x) {
   if (x<=1) return 1;
   int b=f(x/2);
   return 1+3*f(x/2)+b;
}
```

```
void g(int x) {
   if (x<=1) return;
   for(int i=1;i<=f(x)*f(x);i++)
      cout << f(x);
   g(x-1);
}
```

$T_f(1)=a$
$T_f(n)=b+2T_f(n/2)$

$O(n)$

$R_f(1)=1$
$R_f(n)=1+4R_f(n/2)$
$O(n^2)$

for di g:

Numero di iterazioni: $O(m^4)$
Complessità di una iterazione: $O(m)$

Complessità: $O(m^5)$

$Tg(1)=a$

$Tg(m)= m^5 + Tg(m-1)$
$O(m^6)$

Esercizio 4

Sia dato il seguente heap:

[90 56 88 53 31 81 0]

mostrare lo stato dello stesso e le chiamate ad up e down:

A) dopo l'inserzione dell'intero 99

B) dopo l'estrazione di un elemento dallo heap ottenuto al passo A

A	[99 90 88 56 31 81 0 53] up(7) up(3) up(1) up(0)
B	[90 56 88 53 31 81 0] down(0) down(1) down(3)

Esercizio 5

Si scriva un programma che prende in ingresso un intero n e due sequenze S1 e S2 composte ciascuna da n interi distinti. Il programma deve stampare tutte le coppie (x1,x2) con x1 appartenente a S1 e x2 appartenente a x2, tali che x1+x2=x, dove x è un intero noto a priori. Calcolare la complessità del programma in funzione di n. **N.B.** Soluzioni complete ma con complessità maggiore o uguale a $O(n^2)$ otterranno la metà del punteggio previsto. Riportare il corpo delle funzioni usate.

```
#include <iostream>
using namespace std;
struct Node {
  int label;
  Node *left, *right;
};
void insert(Node *&t, int k) {
  if (!t) {
    t = new Node();
    t->label = k;
    t->left = t->right = NULL;
  }
  if (t->label<k)
    insert(t->right,k);
  else if (t->label>k)
    insert(t->left,k);
}
void deletet(Node *&t) {
  if (t) {
    deletet(t->left);
    deletet(t->right);
    delete t;
  }
}
bool find(Node *t, int k) {
  if (!t) return false;
  if (t->label == k)
      return true;
  if (t->label < k)
    return find(t->right,k);
  return find(t->left,k);
}
```

```
int main() {
  int n,i;
  cout << "Inserisci n:";
  cin >> n;
  int *s1 = new int[n];
  for (i=0;i<n;i++)
    cin >> s1[i];
  Node *s2 = NULL;
  for (i=0;i<n;i++) {
    int k;
    cin >> k;
    insert(s2,k);
  }

  for (i=0;i<n;i++) {
    int to_find = x-s1[i];
    if (find(s2,to_find))
      cout << "(" << s1[i]
        << "," << to_find
        << ")" << endl;
  }

  delete s1;
  deletet(s2);
}
```

Complessità O(n log n)