Fondamenti di Informatica II – Modulo di Algoritmi e Strutture dati

ANNO ACCADEMICO 2015/16 - 26 luglio 2016

Esercizio 1

a. Dare la definizione di complessità, cioè quando una funzione è O di un'altra.
b. Utilizzando la definizione, dimostrare che: se f(n) è O (g(n)) e g(n) è O(h(n)), allora f(n) è O(h(n)).

b.

Se f(n) è O (g(n)), allora esistono n_1 e c_1 tali che : per ogni n >=n_1 f(n) <= c_1*g(n)

Se g(n) è O (h(n)), allora esistono n_2 e c_2 tali che: per ogni n >=n_2 g(n) <= c_2*h(n)

Se n_3=max(n_1, n_2) e c_3=c_1*c_2 vale che : per ogni n >=n_3, f(n) <= c_3*h(n)

e questo dimostra che f(n) è O(h(n)).

Esercizio 2

Calcolare la complessità del blocco (indicando le relazioni di ricorrenza di tempo e risultato per ogni funzione) in funzione del numero di nodi dell'albero t, supponendo che sia quasi bilanciato e che Nodes sia la funzione definita a lezione.

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

Le funzioni **f** e **g** sono definite come segue:

```
int f(Node* tree) {
  if (!tree) return 1;
  int a=Nodes(tree),j=a;
  for (int i=1;i<=j;i++) a+=j;
  int b = f(tree->left)
          +f(tree->right);
  return a+b;
}
```

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

Tf(0)=cost
Tf(n)=dn+2Tf(n/2) O(nlogn)

Rf(0)=1
Rf(n)=dn^2+2Rf(n/2) O(n^2)

Tg(0)=cost
Tg(n)=d+4Tg(n/2) O(n^2)

Rg(0)=1
Rg(n)=1+Rg(n/2) O(logn)

Numero iterazioni for: Rg(Rf(n)) = (O(log(n^2))= O(logn)
Complessità di una iterazione: C[f(t)] + C[g(n^2)] + C[Nodes(t)]= O(nlogn) + O(n^4) + O(n) = O(n^4)
Complessità del blocco: O(n^4 * logn)

Esercizio 3

Si scriva una funzione che, dati un albero generico ad etichette intere (memorizzato figlio-fratello) con puntatore alla radice **t** e due interi **a** e **b**, ritorna true se nell'albero c'è un nodo con etichetta **a** che ha fra i suoi discendenti un nodo con etichetta **b**. Se ne calcoli la complessità in funzione del numero di nodi dell'albero.

```
bool findabG (Node* t, int a,
              int b) {
      if (!t)
            return false;
      if (t->label == a) {
            if (findNode(t->left,b))
                  return true;
      }
      return (findabG(t->left) ||
             findabG(t->right))
}

O(N log N)
```

```
bool findab (Node* t, int a, int b,
             bool a_found=false) {
if (!t)
    return false;
bool a_old = a_found;
if ((t->label == b) && a_found)
            return true;
if (t->label == a)
            a_found = true;
return (findab(t->left,a, b, a_found)
              ||
        findab(t->right,a, b, a_old));
}

O(N)
```

Esercizio 4

a) Descrivere l'algoritmo PLSC: a cosa serve, su quale ragionamento è basato, come funziona, qual è la sua complessità.
b) Applicarlo alle due sequenze: BBACB e BABBC e indicare la lunghezza della PLSC e la/le PLSC.

b)

		B	B	A	C	B
	0	0	0	0	0	0
B	0	1	1	1	1	1
A	0	1	1	2	2	2
B	0	1	2	2	2	2
B	0	1	2	2	2	3
C	0	1	2	2	3	3

PLSC: BBB, BBC, BAC, BAB

Esercizio 5

Indicare l'output del programma seguente

```
#include <iostream.h>
template <class T>
class A{
T a;
public:
A(T x){
  a=x;
  cout << 'A' << endl;
  cout << a << endl;}
 void fun(T x) {
  cout << a  <<  endl;
  cout << x << endl;
}
};
void main(){
      A<int> obj(8);
      obj.fun(3.5);
}
```

A
8
8
3

Il risultato è lo stesso del programma seguente? Spiegare.

```
#include <iostream.h>
template <class T>
class A{
T a;
public:
A(T x){
  a=x;
  cout << 'A' << endl;
  cout << a << endl;}
template<class T1>
void fun(T1 x) {
  cout << a  <<  endl;
  cout << x << endl;
}
};
void main(){
      A<int> obj(8);
      obj.fun(3.5);
}
```

Nel primo caso viene instanziata la funzione fun<int> perché fun ha lo stesso template della classe che viene istanziata con int (con conseguente conversione), nel secondo la funzione fun<double> perché il template della funzione è indipendente da quello della classe. Di conseguenza l'output del secondo programma è:

A
8
8
3,5

Punteggio esercizi: 6,7,7,7,6