Algoritmi e Strutture dati - ANNO ACCADEMICO 2016/17

15 settembre 2017

1	**2**	**3**	**4**	**5**	**6**
6	5	6	5	6	5

Esercizio 1

a) Descrivere l'algoritmo PLSC: a cosa serve, su quale ragionamento è basato, come è implementato, qual è la sua complessità e come viene calcolata (scrivere sul retro del foglio).
b) Applicarlo alle due sequenze di caratteri : MUMMIA e MIAMIMA, indicando il contenuto della matrice in figura e la/le PLSC.

		M	U	M	M	I	A
	0	0	0	0	0	0	0
M	0	1	1	1	1	1	1
I	0	1	1	1	1	2	2
A	0	1	1	1	1	2	3
M	0	1	1	2	2	2	3
I	0	1	1	2	2	3	3
M	0	1	1	2	3	3	3
A	0	1	1	2	3	3	4

Lunghezza massima : 4

PLSC : MMIA MMMA

c) Fare un esempio di 2 sequenze di 3 caratteri ciascuna in cui la matrice contiene 1 in tutte le righe e le colonne a parte la prima riga e la prima colonna:

		a	d	e
	0	0	0	0
a	0	1	1	1
b	0	1	1	1
c	0	1	1	1

Esercizio 2 Scrivere una funzione c++ che, dato un albero binario con etichette intere e un intero x, conti il numero di nodi che hanno fra i discendenti una e una sola etichetta uguale a x. Calcolare la complessità.

```
int conta(Node* tree, int x, int & quanti_x) {
  if (!tree) { quanti_x=0; return 0;}
  int cl, cr, ql, qr;
  cl=conta(tree->left, x, ql);
  cr=conta(tree->right, x, qr);
  quanti_x=ql+qr+(tree->label==x);
  return cl+cr+(ql+qr==1);
}
```

Esercizio 3 Scrivere una funzione c++ che, dato un albero generico memorizzato figlio-fratello, per ogni nodo sposta l'ultimo sottoalbero del nodo al primo posto lasciando inalterati gli altri sottoalberi (per ogni nodo l'ultimo sottoalbero diventa il primo).

```
void modifica(Node* & tree) {
if (!tree) return;
if (tree->left) scambia(tree->left);
modifica(tree->left);
modifica(tree->right);
}

void scambia(Node* & tree) {
if (!tree->right) return;
Node* a=tree;
for(Node * penultimo=tree; penultimo->right->right;
penultimo=penultimo->right);
Node* ultimo=penultimo->right;
penultimo->right=0;
tree=ultimo;
ultimo->right=a;
}
```

Esercizio 4

Calcolare la complessità in funzione di **n**>0 delle espressioni **f(g(n)) e g(f(n))**.

con le funzioni **f** e **g** definite come segue. Indicare per esteso le relazioni di ricorrenza e, per ogni comando ripetitivo, il numero di iterazioni e la complessità della singola iterazione.

```
int g(int x) {
  if (x<=0) return 1;

  int a = 0;
  int b = 4*g(x/2);

  cout << a + 3*g(x/2);

  return 1 + 2*b; }
```

```
int f(int x) {

 if (x<=0) return 1;

  for (int i=0; i <= g(x); i++) cout << i;

  return f(x-1);

}
```

Tg(0)= d
Tg(n)= c + 2Tg(n/2) Tg(n) è O(n)

Rg(n)= 1 Rg(n) è O(n^3)
Rg(n)= 1 +8 Rg(n/2)

Funzione f
Calcolo for:
numero iterazioni: O(n^3)
Complessità della singola iterazione: O(n)
Complessità del for: O(n^4)

Tf(0)= d Tf(n) è O(n^5)
Tf(n) = n^4+ Tf(n-1)

Rf(n) è O(1)

Calcolo f(g(n)) : Tg(n) + Tf(n^3) = O(n)+ O(n^15) = O(n^15)

Calcolo g(f(n)) : Tf(n) + Tg(1) = O(n^5)+ O(1) = O(n^5)

Esercizio 5

a) Considerare gli algoritmi di ordinamento visti a lezione, indicando per ciascuno di essi una descrizione a parole e la complessità.
b) Applicare il quicksort all'array con il contenuto seguente, indicando tutte le chiamate a quicksort generate dalla chiamata iniziale e, per ciascuna di esse, il contenuto dell'array e il perno.

4	15	2	3	21	7

						inf	sup	perno
4	15	2	3	21	7	**0**	**5**	**2**
2	15	4	3	21	7	**1**	**5**	**3**
2	3	4	15	21	7	**2**	**5**	**15**
2	3	4	7	21	15	**2**	**3**	**4**
2	3	4	7	21	15	**4**	**5**	**21**

a) Indicare un array di 3 elementi contenente i numeri 1, 2, 3 per la quale il quicksort raggiunge la massima efficienza

1	2	3

b) Indicare un array di 3 elementi contenente i numeri 1, 2, 3 per la quale il quicksort raggiunge la minima efficienza.

3	1	2

Esercizio 6

Indicare l'output del programma seguente:

```
class ecc1{
protected:
int x;
public:
ecc1(){ x= 5; cout << "ecc1" <<
endl;};
int h(){ return x; }
};

class ecc2: public ecc1{
protected:
int x;
public:
ecc2(){ x= 6; cout << "ecc2" <<
endl;};
int h(){ return x; }
};

int g(int x) {
  try {
  cout << 100 << endl;
  if (x==2) { ecc2* e=new ecc2; throw
e;};
  if (x==3) throw 3;
  }
  catch (ecc2* e) { cout << e->h() +1
<< endl; }
  catch (...) { throw; }
  cout << "fine g" << endl;
  };

int f(int x) {
  try {
  if (x==1) { ecc1* e=new ecc1; throw
e;};
  cout << 600 << endl;
  g(x);
  if (x==2) { ecc2* e=new ecc2; throw
e;};
  }
  catch (ecc2* e) { cout << e->h()<<
endl; }
  catch (ecc1* e) { cout << 90 <<
endl; }
  catch(int) { cout << 80 << endl; }
  cout << "fine f" << endl;
  };

int main () {
f(1);
cout << endl;
f(2);
cout << endl;
f(3);
}
```

```
ecc1
90
fine f

600
100
ecc1
ecc2
7
fine g
ecc1
ecc2
6
fine f

600
100
80
fine f
```