Algoritmi e Strutture dati - ANNO ACCADEMICO 2016/17

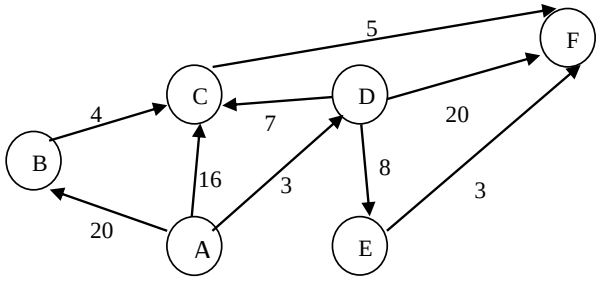
13 giugno 2017

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

COGNOME E NOME : **MATRICOLA:**

Esercizio 1

a) Descrivere l'algoritmo di Dijkstra: a cosa serve, su quale ragionamento è basato, come è implementato, qual è la sua complessità
b) Applicarlo al grafo seguente a partire dal nodo A.



	A	B	C	D	E	F
A, B, C, D, E, F	0 | -	inf | -	inf | -	inf | -	inf | -	inf | -
B, C, D, E, F	0 | -	20 | A	16 | A	3 | A	inf | -	inf | -
B, C, E, F	0 | -	20 | A	10 | D	3 | A	11 | D	23 | D
B, E, F	0 | -	20 | A	10 | D	3 | A	11 | D	15 | C
B, F	0 | -	20 | A	10 | D	3 | A	11 | D	14 | E
B	0 | -	20 | A	10 | D	3 | A	11 | D	14 | E

Cammini:

AB, ADC, AD, ADE, ADEF

Esercizio 2

c) Descrivere l'algoritmo di Huffman: a cosa serve, su quale ragionamento è basato, come è implementato, qual è la sua complessità.

a) Applicarlo all'alfabeto seguente in cui ad ogni carattere corrisponde la sua frequenza percentuale nel testo, indicando l'albero risultante.

A	12
B	16
C	7
D	10
E	9
F	30
G	5
H	11

Esercizio 3

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,

a) supponendo che t sia quasi bilanciato.
b) supponendo che t sia completamente sbilanciato

Indicare per esteso numero di iterazioni e complessità di ogni iterazione per i comandi ripetitivi.

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

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

```
int f(Node* tree) {
  if (!tree) return 1;
  int x=0;
  for (int i=1;i<= Nodes(tree);i++)
  x+=i;
  int b = 4*f(tree->left);
  return x+b;
}
```

```
int g(Node * tree) {
  if (!tree) return 1;
  int a=0;
  for (int i=1;i<=f(tree)/Nodes(tree);i++)
    { a++;
      cout << a;
    }
  return 3;
  }
```

a) Funzione f

Numero iterazioni del for: O(n)
Complessità di una iterazione: O(n)
Complessità del for: O(n^2)

Tf(0)=a
Tf(n)=dn^2+Tf(n/2) O(n^2)

Rf(0)=1
Rf(n)=dn^2+4Rf(n/2) O(n^2 logn)

Funzione g

Numero iterazioni del for: O(nlogn)
Complessità di una iterazione: O(n^2)
Complessità del for: O(n^3 logn)

Tg(n)= O(n^3logn)

Rg = O(1)

For esterno

Numero iterazioni del for: O(n^2 logn)
Complessità di una iterazione: O(n) + O(n^2)+ O(n^3 logn) = O(n^3 logn)
Complessità del for: O(n^5 log^2n)

b) consideriamo il caso peggiore

Funzione f

Numero iterazioni del for: O(n)
Complessità di una iterazione: O(n)
Complessità del for: O(n^2)

Tf(0)=a
Tf(n)=dn^2+Tf(n-1) O(n^3)

Rf(0)=1
Rf(n)=dn^2+4Rf(n-1) O(4^n) esponenziale

Funzione g

Numero iterazioni del for: O(4^n/n)
Complessità di una iterazione: O(n^3)
Complessità del for: O(n^2 4^n)

Tg(n)= O(n^2 4^n)

Rg = O(1)

For esterno

Numero iterazioni del for: O(n^3)
Complessità di una iterazione: O(n^2 4^n)
Complessità del for: O(n^5 4^n)

Esercizio 4 Scrivere un algoritmo in c++ che, dato un array, costruisce un albero binario quasi bilanciato che contiene tutti gli elementi dell'array. Descrivere a parole l'algoritmo e implementarlo in c++. Calcolare la complessità.

```
Node * build(int A, int i, int j) {
if (i>j) return NULL;
Node * t=new Node;
int k=(i+j)/2;
t->label=A[k];
t->left=build(A,i,k-1);
t->right=build(A,k+1,j);
return t;
};

void costruisci (int* A, int n) {
 build (A, 0, n-1);
}
```

Complessità: O(n)

Esercizio 5 dato il seguente programma c++.

a) Indicare la sua uscita
b) Indicare l'uscita togliendo la parola "virtual"
c) Indicare l'uscita togliendo la parola "protected"

Spiegare le eventuali differenze fra i casi sopra citati.

```
class C {
public:
      C(){cout << "nuovo C" <<  endl; };
      void f(){cout << "f di C" <<  endl;
}
};

class A {
protected:
      C obj_x;
public:
      A(){cout << "nuovo A" <<  endl;
        obj_x.f();
        };
      void virtual f(){cout << "f di A"
<<  endl; }
};
```

```
class B: public A {
       A obj_y;
public:
B(){cout << "nuovo B" <<  endl;
    obj_x.f();
    };
void f(){cout << "f di B" <<  endl; }
};


int main(){
    int i;
       A* obj1= new B;
       obj1->f();
       cin >> i;
}
```

a) Esecuzione del programma

```
Nuovo C
Nuovo A
F di C
Nuovo C
Nuovo A
F di C
Nuovo B
F di C
F di B
```

b) Togliendo il virtual

```
Nuovo C
Nuovo A
F di C
Nuovo C
Nuovo A
F di C
Nuovo B
F di C
F di A
```

c) togliendo la linea `protected`: errore di compilazione perché B usa obj_x che è privato in A

Esercizio 6

Scrivere una funzione **`int conta (Node*tree)`** che , dato un albero generico memorizzato figlio-fratello, conta quanti nodi hanno un numero pari di figli.

```
int contafigli (Node*tree) {
  if (! tree) return 0;
  return 1 + contafigli( tree->right);
}

int conta (Node*tree) {
  if (! tree) return 0;
  return (contafigli (tree->left)%2==0)
        + conta( tree->left) + conta( tree->right);
}
```