Algoritmi e Strutture dati - ANNO ACCADEMICO 2017/18

12 giugno 2018

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

Esercizio 1

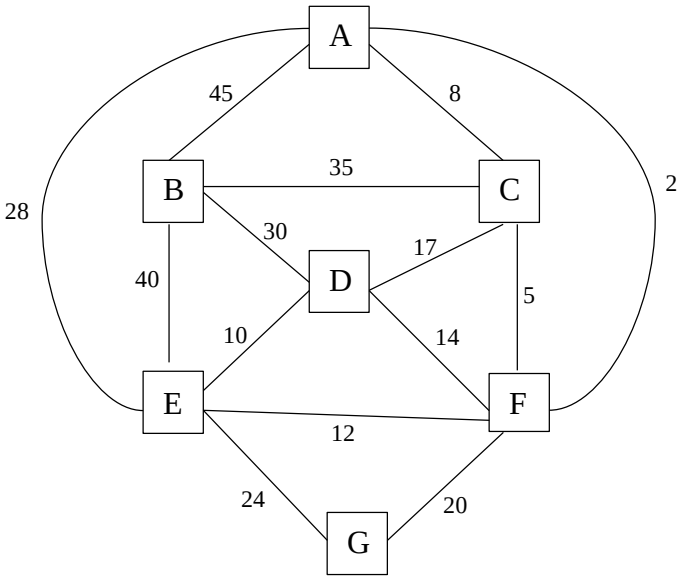
a) Descrivere l'algoritmo di Huffman: a cosa serve, su quale ragionamento è basato, come è implementato, qual è la sua complessità.
b) Applicarlo all'alfabeto in figura in cui ad ogni carattere corrisponde la sua frequenza percentuale nel testo, indicando l'albero risultante e la codifica di ogni carattere (per ogni nodo dell'albero etichettare con **0** l'arco che lo congiunge al figlio con etichetta minore)
c) Se un alfabeto ha **2^n** simboli e tutti hanno la stessa frequenza, che caratteristica hanno le relative codifiche? **Sono tutte di lunghezza uguale a n**

A	13
B	16
C	7
D	15
E	17
F	20
G	4
H	8

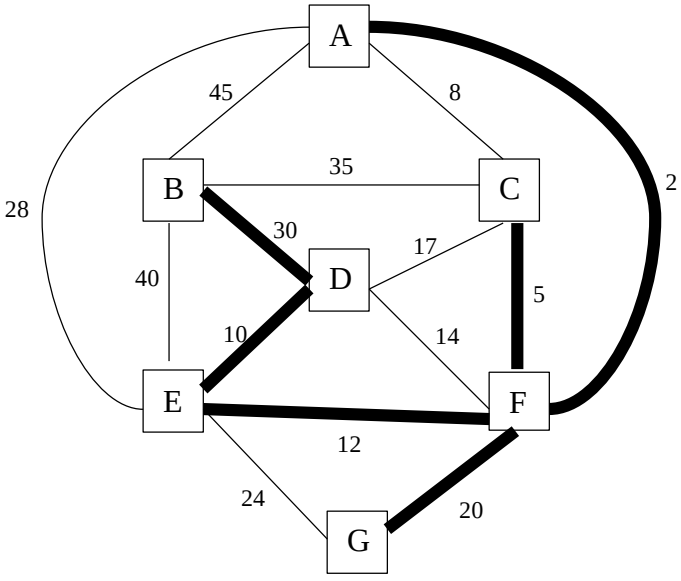
A	100
B	110
C	0011
D	101
E	111
F	01
G	0010
H	000

Esercizio 2

a) Dare la definizione di minimo albero di copertura di un grafo non orientato.
b) Descrivere a parole l'algoritmo di Kruskal per trovare il minimo albero di copertura. Descrivere come è implementato, indicare e spiegare la sua complessità.
c) Applicarlo al grafo seguente indicando l'ordine in cui sono presi gli archi appartenenti al minimo albero di copertura.
d) Dato un minimo albero di copertura per un grafo G questo contiene un cammino minimo per ogni coppia di nodi in G? Dimostrare la risposta. **NO**
e) Tutti gli alberi di copertura (sia minimi che non) di un grafo G hanno lo stesso numero di archi? Dimostrare la risposta. **Tutti gli alberi di copertura hanno (n-1) archi**



(A, F) (F, C) (E, D) (E, F) (G,F) (B,D)



Esercizio 3
Calcolare in funzione di n>=0 la complessità del blocco

```
for (int i=0; i <= f(n)*g(n); i++) a+= 1;
```

con le funzioni **f** e **g** definite come segue:

```
int f(int x) {
  if (x==0) return 1;
  int y=0;
  for (int i=1;i<= g(x) ;i++) y+=1;
  int b = 1 + 4*f(x/2);
  return y+b;
}
```

```
int g(int x) {
  if (x==0) return 1;
  int a=0;
  for (int i=1;i<=x*x;i++)
    { a++;
      cout << a;
    }
  return a/x + g(x-2) ;
  }
```

Indicare le relazioni di ricorrenza e il numero di iterazioni e la complessità di ogni iterazione per i comandi ripetitivi.

Funzione g

Numero iterazioni del for: O(n^2)
Complessità di una iterazione: O(1)
Complessità del for: O(n^2)

Tg(0)= a
Tg(n)= bn^2+Tg(n-2) O(n^3)

Rg(0)= 1
Rg(n)= bn+Rg(n-2) O(n^2)

Funzione f

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

Tf(0)= a
Tf(n)= bn^5+Tf(n/2) O(n^5)

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

blocco
Numero iterazioni del for: Rf(n)*Rg(n) = O(n^2 logn)* O(n^2) = O(n^4 logn)
Complessità di una iterazione: Tf(n)+Tg(n) = (n^5)+ O(n^3) = O(n^5)
Complessità del for: O(n^9 logn)

Esercizio 4

a) Dare la definizione di albero binario.
b) Scrivere una funzione in c++ con complessità O(n) che, dato un albero binario con etichette intere, conta il numero di nodi la cui etichetta è uguale al numero di foglie che ci sono nel suo sottoalbero.

```
int conta (Node* t, int & leaves) {
    if (!t) {leaves=0; return 0;}
    int leaves_l, leaves_r, conta_l,conta_r;
    conta_l=conta(t->left,leaves_l);
    conta_r=conta(t->right,leaves_r);
    leaves=leaves_l+leaves_r + (t->left==0 && t->right==0);
    return (t->label==leaves)+ cont_l + conta_r;
    }
```

Esercizio 5

a) Dare la definizione di albero generico.

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

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

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

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

Esercizio 6 Sia dato il seguente programma c++.

```
class A {
public:
       A(){cout << "nuovo A" <<  endl; };
       void virtual f()=0;
        void g() {cout << "g di A" <<
endl; }

};

class B: public A {
public:
       B(){cout << "nuovo B" <<  endl; } ;
        void f()  {cout << "f di B" <<
endl; }
        void g() {cout << "g di B" <<
endl; }
};

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

class D: public C {
public:
      D(){cout << "nuovo D" <<  endl; };
      void f(int x){cout << x <<  endl; }
};

int main(){

    A* vet[3];
    vet[0]= new B;
    vet[1]= new C;
    vet[2]= new D;
    for (int j=0; j<3; j++) {
           vet[j]->f();
           vet[j] -> g();
            }

}
```

a) Indicare la sua uscita
b) Indicare l'uscita aggiungendo "virtual" alla g() di A
c) Spiegare le eventuali differenze fra i casi sopra citati.
d) Spiegare perché aggiungendo una qualsiasi delle seguenti istruzioni alla fine del main il compilatore dà errore: vet[1]= new A; vet[2]->f(3); D* obj= vet[2];

a)
nuovo A
nuovo B
nuovo A
nuovo C
nuovo A
nuovo C
nuovo D
f di B
g di A
f di C
g di A
f di C
g di A

b)
nuovo A
nuovo B
nuovo A
nuovo C
nuovo A
nuovo C
nuovo D
f di B
g di B
f di C
g di C
f di C
g di C

d)

vet[1]= new A;	**non si può istanziare una classe astratta**
vet[2]->f(3);	**vet[2] è di tipo A che è una classe che non ha la funzione f(int)**
D* obj= vet[2]	**conversione da supertipo a sottotipo non ammessa**