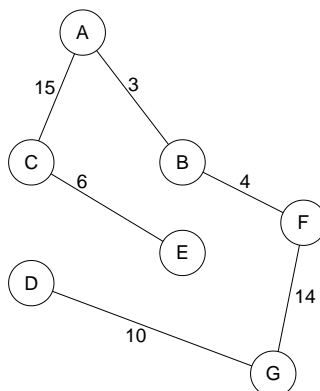
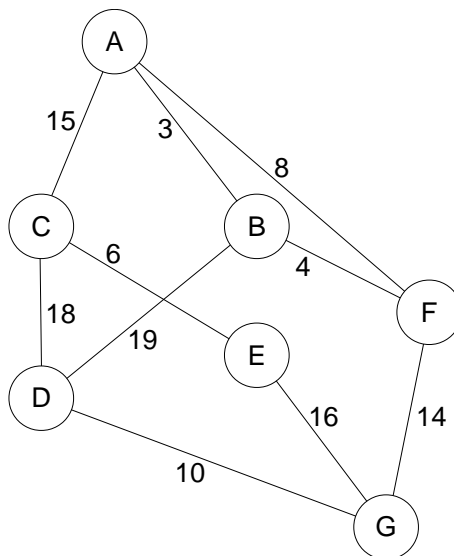


**Algoritmi e Strutture dati - ANNO ACCADEMICO 2016/17**  
**25 luglio 2017**

1	2	3	4	5	6

**Esercizio 1**

- a) Descrivere l'algoritmo di Kruskal: 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 al grafo in figura indicando il grafo risultante.
- c) Indicare un albero di copertura non minimo.



## Esercizio 2

Scrivere una funzione c++ che, dato un albero binario con etichette intere e un intero x, conti il numero di nodi la cui etichetta è uguale al numero di etichette uguali a x che compaiono nel suo sottoalbero.

```
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+(tree->label==quanti_x);
}
```

## Esercizio 3

Scrivere una funzione c++ che, dato un albero generico memorizzato figlio-fratello con etichette intere, aggiunga ad ogni nodo un ultimo figlio con etichetta uguale a quella del padre.

```
void aggiungi(Node* & tree, int x) {
    if (!tree)
        {tree=new Node;
         tree->label=x;
         tree->left=tree->right=0;
        }
    aggiungi(tree->right, x);
}

void f(Node* tree) {
    if (!tree) return;
    aggiungi(tree->left, tree->label);
    f(tree->left);
    f(tree->right);
}
```

## Esercizio 4

Calcolare la complessità del **for** in funzione di  $n > 0$ .

```
for (int i=0; i <= f(n); i++) cout << f(n)+g(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;  
    for (int i=0; i<=x;i++) a+= i;  
  
    int b = 2*g(x/2);  
  
    cout << a + g(x/2);  
  
    return 1 + 2*b; }  
}
```

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

### Funzione g

Calcolo for:

numero iterazioni:  $O(n)$

Complessità della singola iterazione:  $O(1)$

Complessità del for:  $O(n)$

$T_g(0) = d$

$T_g(n) = cn + 2T_g(n/2)$   $T_g(n)$  è  $O(n \log n)$

$R_g(n) = 1$   $R_g(n)$  è  $O(n^2)$

$R_g(n) = 1 + 4 R_g(n/2)$

### Funzione f

Calcolo for:

numero iterazioni:  $O(n^3)$

Complessità della singola iterazione:  $O(1)$

Complessità del for:  $O(n^3)$

$T_f(0) = d$   $T_f(n)$  è  $O(n^4)$

$T_f(n) = n^3 + T_f(n-1)$

$R_f(0) = 1$   $T_f(n)$  è  $O(n^3)$

$R_f(n) = n^2 + T_f(n-1)$

### Calcolo for del blocco:

numero iterazioni:  $O(n^3)$

Complessità della singola iterazione:  $T_f(n) + T_g(n) = O(n^4) + O(n \log n) = O(n^4)$

Complessità del for:  $O(n^7)$

## Esercizio 5

Descrivere la teoria della NP-completezza: indicare il significato degli insiemi P e NP, la relazione fra i due insiemi, il teorema di Cook, la riducibilità fra problemi, i problemi NP-completi.

## Esercizio 6

a) Indicare l'output del programma seguente

```
class alpha {
protected:
    int a;
public:
    alpha(){a=8;
        cout << "nuovo alpha " << endl;
    }
    void virtual f()=0;
    void g() {cout << a << endl; }
};

class beta: public alpha {
protected:
    int a;
public:
    beta() {a=5;
        cout << "nuovo beta " << endl;
    }
    void virtual f() { }
    void virtual g() {cout << a << endl;}
};

class delta: public beta {
protected:
    beta ob;
public:
    delta() {
        cout << "nuovo delta " << endl;
    }
    void f() { cout << a << endl;}
    void g(){cout << a+1 << endl;}
};

template<class T>
void funzione( T *obj){
    static int a;
    obj->f();
    obj->g();
    a++;
    cout << a << endl;
}

template<class T1, class T2>
void funzione1( T1 *obj1, T2 *obj2){
    funzione(obj1);
    cout << endl;
    funzione(obj2);
    cout << endl;
    funzione<alpha> (obj2);
    cout << endl;
}

void main(){
    delta *obj1= new delta;
    alpha *obj2=obj1;
    beta *obj3=obj1;
    funzione1(obj3, obj2);
}
```

nuovo alpha  
nuovo beta  
nuovo alpha  
nuovo beta  
nuovo delta  
5  
6  
1  
5  
8  
1  
5  
8  
2

b) Con le classi definite nell'esercizio 1, date le seguenti istruzioni:

delta \*p1=new delta;

delta obj1;

alpha \*p2=new delta;

dire quali sono giuste e quali errate fra le seguenti istruzioni, dandone la motivazione:

1. alpha \*p3=p1; ok
2. beta \*p3=p2; no: non c'è conversione implicita da puntatore a superclasse a puntatore a sottoclasse
3. beta \*p4=&obj1; ok
4. beta obj2=obj1; ok
5. alpha obj3; no: non si può instanziare una classe astratta
6. cout << obj1.ob.f(); no: obj1 e' protetto e f è void
7. funzione1<alpha,alpha>(p1,p2); ok
8. funzione1<beta,beta>(p1,p2); no: non c'è conversione implicita da puntatore a superclasse a puntatore a sottoclasse