**Algoritmi e basi di dati – modulo Algoritmi e Strutture dati – a.a. 2010/2011**

13 settembre 2011

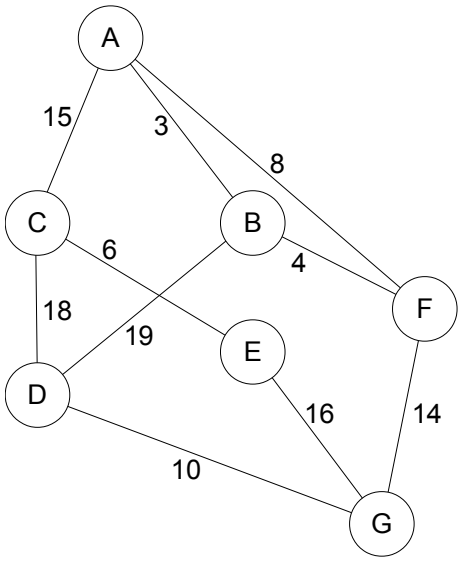
| Cognome | Nome | Matricola |
|---|---|---|
| | | |

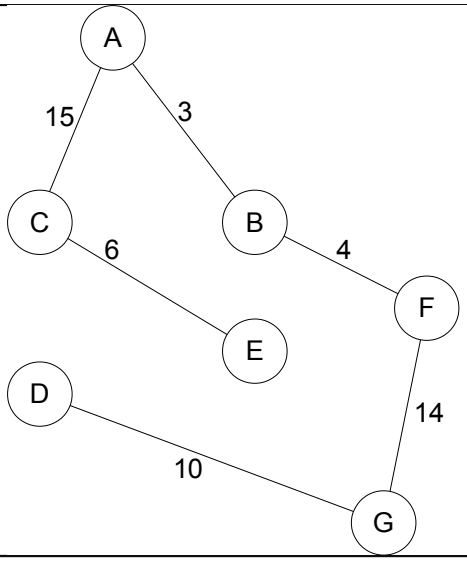
| 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|
| | | | | |

# Esercizio 1

Trovare l'albero di copertura di costo minimo del grafo in figura mediante l'algoritmo di Kruskal. Indicare l'ordine in cui vengono presi gli archi durante l'esecuzione dell'algoritmo.

| Archi | (AB) (BF) (CE) ~~(AF)~~ (DG) (GF) (CA) ~~(EG) (CD) (BD)~~ |
|---|---|



## Esercizio 2

Scrivere la funzione:
**`int binSearch(int* A, int x, int inf, int sup)`**
che esegue l'algoritmo di ricerca binaria su un array. Quindi cercare l'elemento **`20`** in:
**`A=[1 2 5 8 8 12 13 14 18 18 20 24 25 27 30]`**
Indicare tutte le chiamate a **`binSearch`** con relativi argomenti generate dalla chiamata **`binSearch(A,20,0,14)`**.

```
int binSearch(int* A, int x, int inf=0, int sup=n-1) {
   if (inf > sup) return 0;
   int k = (inf + sup)/2;
   if (x == A[k]) return 1;
   if (x < A[k]) return binSearch (A, x, inf, k-1);
   else return binSearch (A, x, k+1, sup);
}
```

| **inf**: | **sup**: |
|---|---|
| 0 | 14 |
| 8 | 14 |
| 8 | 10 |
| 10 | 10 |
| | |
| | |

## Esercizio 3

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) {
  int a = g(x);
  return a + g(a);
}
```

**Funzione g**

$T_g(0)$= d
$T_g(n)$= cn + $2T_g(n/2)$  $T_g(n)$ è O(nlogn)

$R_g(n)= 1$ $R_g(n)$ è $O(n^2)$
$R_g(n)= 1 +4\ R_g(n/2)$

**Funzione f**

$T_f(n)$ è $O(n^2 \log n)$

$R_f(n)=$ è $O(n^4)$

Calcolo for del blocco:
numero iterazioni: $O(n^4)$
Complessità della singola iterazione: $T_f(n) + T_g(n) = O(n^2 \log n) + O(n \log n) = O(n^2 \log n)$
Complessità del for: $=O(n^6 \log n)$

## Esercizio 4

Scrivere una funzione che, dato un albero binario **t** ad etichette intere, sommi all'etichetta di ogni nodo la differenza fra il numero di nodi con valore maggiore o uguale a **0** e quelli con valore minore di **0** appartenenti al suo sottoalbero.

```
void diff (Node* t, int& pos, int& neg) {
   if (!t) { pos = neg = 0; return; }
   int pos_l, neg_l, pos_r, neg_r;
   diff(t->left, pos_l, neg_l);
   diff(t->right, pos_r, neg_r);
   pos = pos_l + pos_r;
   neg = neg_l + neg_r;
   int a = pos - neg;
   pos += (t->label >= 0);
   neg += (t->label < 0);
   t->label += a;
}
```

## Esercizio 5

Scrivere una funzione che, dato un albero generico **t** ad etichette intere, conti il numero di nodi per cui il primo e l'ultimo figlio hanno la stessa etichetta.

```
int es5(Node* t) {
   if (!t) return 0;
   return controllo(t->left) + es5(t->left) + es5(t->right);
}


int controllo(Node* t) {
   if (!t || ! t->right) return 0;
   for (Node* t1 = t->right; t1->right; t1 = t1->right);
   return (t->label == t1->label);
}
```