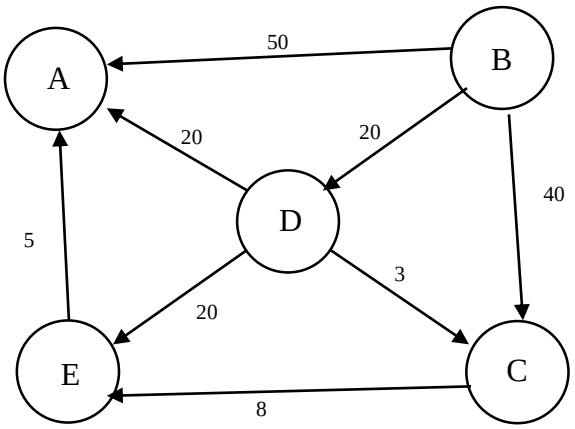
ANNO ACCADEMICO 20016/2017- 6 febbraio 2018
Algoritmi e Strutture Dati

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

Esercizio 1

a) Descrivere l'algoritmo di Dijkstra: 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 seguente a partire dal nodo B.



	A	B	C	D	E
A, B, C, D, E	Inf | -	0 | -	inf | -	inf | -	inf | -
A, C, D, E	50 | B	0 | -	40 | B	20 | B	inf | -
A, C, E	40 | D	0 | -	23 | D	20 | B	40 | D
A, E	40 | D	0 | -	23 | D	20 | B	31 | C
A	36 | E	0 | -	23 | D	20 | B	31 | C

Esercizio 2

a) Descrivere il metodo di ricerca hash a indirizzamento aperto e con concatenazione: come funziona, cosa sono le collisioni, gli agglomerati, le leggi di scansione, da quali parametri dipende la velocità di ricerca.
b) Sia data la seguente tabella hash a indirizzamento aperto con funzione hash modulare e scansione lineare. Si supponga che la tabella sia inizialmente vuota. Mostrare il contenuto della tabella dopo le operazioni indicate sulle colonne:

	contenuto iniziale	inserimento 356	inserimento 41	inserimento 287	eliminazione 41	inserimento 69
0	-1	-1	41	41	-2	69
1	-1	-1	-1	287	287	287
2	-1	-1	-1	-1	-1	-1
3	-1	-1	-1	-1	-1	-1
4	-1	-1	-1	-1	-1	-1
5	-1	-1	-1	-1	-1	-1
6	-1	356	356	356	356	356

Esercizio 3

Calcolare la complessità in funzione di **n**>0 dell'istruzione

```
y=g(f(n));
```

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

```
int f(int x) {
   if (x<=1) return 1;
   int b=0, i;
   for (i=1; i<=x; i++) b+=i;
   cout << b*b*b;
   return f(x-1)+ 4 + b;
}
```

```
int g(int x) {
  if (x<=1) return 10;
  int a=0;
  for (int i=0; i<f(x)*f(x); i++)
      a++;
   return 10+g(x/2)+g(x/2);
}
```

Indicare le eventuali relazioni di ricorrenza e spiegare brevemente il calcolo della complessità dei cicli.

Stima del tempo di f

numero iterazioni del for = O(n)
complessità di un'iterazione = costante
tempo del for = O(n)

$T_f(1)= d$
$T_f(n)= c\cdot n + T_f(n-1)$

T_f è $O(n^2)$

$R_f(n) = cn\text{^}2 + R_f(n-1)$
$R_f(n)$è O(n^3)

Stima del tempo di g:
numero iterazioni del for: $R_f(m)*R_f(m)= O(m^6)$
complessità di un'iterazione: $T_f(m) = O(m^2)$
tempo del for: $O(m^8)$

tempo di g
$T_g(1)= cost$
$T_g(m)= c\cdot m^8 + 2T_g(m/2)$
T_g è $O(m^8)$

stima del risultato di g
$R_g(1)= cost$
$R_g(m)= cost +2\ R_g(m/2)$
$R_g(m) = O(m)$

Complessità dell'istruzione:
$C[f(n)]+ C[g(R_f(n))] = O(n^2) + C[g(n\text{^}3)] = O(n^2) + O(n\text{^}24) = O(n\text{^}24)$

Esercizio 4
Scrivere una funzione boolena che, dato un albero generico non vuoto ad etichette di tipo int, memorizzato figlio-fratello, restituisce true se nell'albero c'è almeno un nodo tale che il suo primo e il suo ultimo figlio hanno la stessa etichetta. N.B: un nodo con un solo figlio verifica la condizione.

```
bool p_u(Node* t) {
  if (!t) return 0;

  if (!t-> left) return p_u (t->right);
    if (t-> left->label== ultimo (t->left)) return true;

   return (p_u(t->left)|| p_u(t->right));

  }

int ultimo (Node* t) {
  if (!t->right) return t->label;
  ultimo(t->right);
}
```

Esercizio 5
Siano dati due alberi binari ad etichette intere, con puntatori alla radice t1 e t2. Supponendo che gli alberi siano identici in quanto a struttura (ma non necessariamente rispetto al contenuto delle etichette), scrivere una funzione `Node* sottrai(Node* t1, Node* t2)` che prende in ingresso t1 e t2 e restituisce un puntatore alla radice di un terzo albero, anch'esso identico per struttura a t1 e t2, ma dove ogni nodo ha come etichetta la differenza delle etichette dei nodi nella stessa posizione in t1 e t2.

```
Node* sottrai(Node* t1, Node* t2) {
  if (!t1)
    return NULL;
  Node* n = new Node();
  n->label = t1->label - t2->label;
  n->left = sottrai(t1->left, t2->left);
  n->right = sottrai(t1->right, t2->right);
  return n;
}
```

Esercizio 6

Indicare l'uscita del seguente programma

a) Così come è scritto

b) Sostituendo l'istruzione indicata con ** con l'istruzione: `alpha *o=new alpha(80);`

c) Spiegare la differenza

```
template<class tipo>
void funzione( tipo * obj){
      obj->g();
};

class alpha {
protected:
      int a;
public:
      alpha(){a=10; cout << a << "  nuovo alpha  " <<  endl; a++;};
      alpha(int x){a=x; cout << a << "  nuovo alpha  " <<  endl; a++;};
      void g() {cout <<  a+1 << endl; }
};

class beta: public alpha {
int a;
alpha *o=new alpha();    **
public:
      beta() {a=24; cout << a << "  nuovo beta  " << endl;}
      void    g() {cout << a+2  << endl;}
    void h() {funzione<alpha>(o); cout << a+2  << endl;}
};


int main(){
      beta *obj1= new beta;
      alpha *obj2=obj1;
      funzione(obj1);
  funzione(obj2);
  obj1->h();
}
```

a)

```
10  nuovo alpha
10  nuovo alpha
24  nuovo beta
26
12
12
26
```

b)

```
10  nuovo alpha
80  nuovo alpha
24  nuovo beta
26
12
82
26
```

c) Vengono chiamati due diversi costruttori della classe alpha.