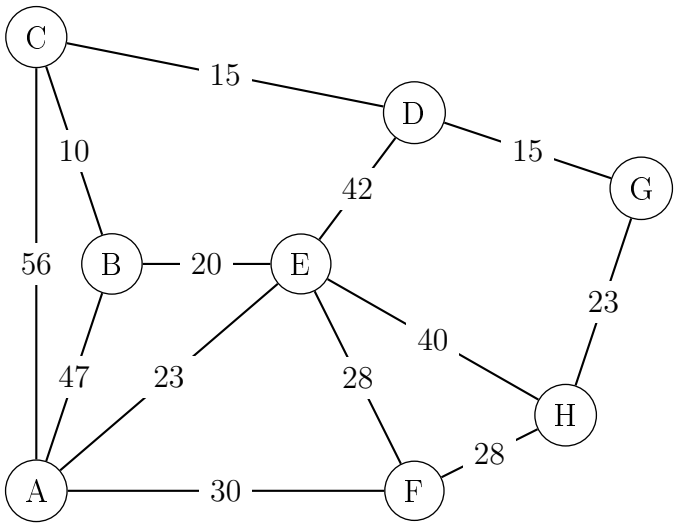
TD 6

# Algorithmes sur les graphes

**Correction**

On considère le graphe pondéré suivant :




Pour représenté un graphe pondérer ( orienté ou non ), il suffit de modifier la matrice d'adjacence en introduisant le poids de l'arête correspondante.

Pour l'exemple précédent, on obtient donc la matrice ( en prenant les sommets dans l'ordre alphabétique :

$$\begin{pmatrix} 0 & 47 & 56 & +\infty & 23 & 30 & +\infty & +\infty \\ 47 & 0 & 10 & +\infty & 20 & +\infty & +\infty & +\infty \\ 56 & 10 & 0 & 15 & +\infty & +\infty & +\infty & +\infty \\ +\infty & +\infty & 15 & 0 & +42 & +\infty & 15 & +\infty \\ 23 & 20 & +\infty & 42 & 0 & 28 & +\infty & 40 \\ 30 & +\infty & +\infty & +\infty & 28 & 0 & +\infty & 28 \\ +\infty & +\infty & +\infty & 15 & +\infty & +\infty & 0 & 23 \\ +\infty & +\infty & +\infty & +\infty & 40 & 28 & 23 & 0 \end{pmatrix}$$

Les types utilisés seront donc :

```
type poids = Inf | P of int;;
type graphe =poids array array ;;
```

Nous supposerons dans tout le TD que les poids sont des entiers positifs.

L'exemple devient alors :

```
let (exemple : graphe) =
[| [| P(0) ;  P(47) ;  P(56) ; Inf ; P(23) ;  P(30) ; Inf ; Inf |] ;
[| P(47) ;  P(0) ;  P(10) ; Inf ; P(20) ;  Inf ; Inf ; Inf |] ;
[| P(56) ;  P(10) ;  P(0) ; P(15) ; Inf ;  Inf ; Inf ; Inf |] ;
[| Inf ;  Inf ;  P(15) ; P(0) ; P(42) ;  Inf ; P(15) ; Inf |] ;
[| P(23) ;  P(20) ;  Inf ; P(42) ; P(0) ;  P(28) ; Inf ; P(40) |] ;
[| P(30) ; Inf ;  Inf ; Inf ; P(28) ;  P(0) ; Inf ; P(28) |] ;
[| Inf ;  Inf ;  Inf ; P(15) ; Inf ;  Inf ; P(0) ; P(23) |] ;
[| Inf ;  Inf ;  Inf ; Inf ; P(40) ;  P(28) ; P(23) ; P(0) |] ;
|];;
```

# 1 Fonctions utiles

1. Écrire la fonction `somme` qui prend en argument deux poids, et renvoie la somme des poids.

   `somme : poids -> poids -> poids`

   Correction

Listing 1 – La fonction `somme`

```
let somme poids_1 poids_2 = match ( poids_1 , poids_2) with
  | P(a),P(b) -> P(a+b)
  | _ -> Inf;;
```

2. Écrire la fonction `inferieur` qui prend en argument deux poids, et renvoie le booléen traduisant la relation d'infériorité.

   `inferieur : poids -> poids -> bool`

   Correction

Listing 2 – La fonction `inferieur`

```
let inferieur  poids_1 poids_2 = match ( poids_1 , poids_2) with
  | P(a),P(b) -> a<b
  | Inf  , Inf -> false
  | P(a) , Inf -> true
  | Inf , P(a) -> false ;;
```

3. Écrire la fonction `mini_poids` qui prend en argument deux poids, et renvoie le minimum des deux poids

   `mini_poids  : poids -> poids -> poids`

   Correction

Listing 3 – La fonction `mini_poids`

```
let mini_poids  poids_1 poids_2 = match ( poids_1 , poids_2) with
  | P(a),P(b) -> P(min a b)
  | Inf  , Inf -> Inf
  | P(a) , Inf -> P(a)
  | Inf , P(a) -> P(a) ;;
```

# 2 Algorithme de Floyd-Warshall

On cherche à écrire un algorithme du chemin de poids minimal entre deux sommets $i$ et $j$. Notons $M$ la matrice d'adjacence et $n$ l'ordre du graphe $G$.
L'algorithme de Floyd-Warshall consiste à calculer la suite finie de matrices $M^{(k)}$, pour $0 \leq k \leq n$ avec $M^{(0)} = M$ et :

$$\forall k < n, \quad \forall (i,j) \in [\![1;n]\!]^2, \quad m_{ij}^{(k+1)} = \min\left(m_{ij}^{(k)}, m_{i,k+1}^{(k)} + m_{k+1,j}^{(k)}\right)$$

1. Démontrer le théorème suivant :

**Théorème 1 :**

Pour $G$ un graphe pondéré par des poids positifs, alors $m_{ij}^{(k)}$ est égal au poids du chemin minimal reliant le sommet $i$ au sommet $j$ et ne passant que par des sommets de l'ensemble $\{1;2;...;k\}$.

Correction

**Preuve :**
On raisonne par récurrence sur $k$.

- Si $k = 0$, $m_{ij}^{(k)} = mij$ est bien entendu le poids du chemin minimal reliant $i$ à $j$ sans passer par aucun autre sommet, ce qui correspond à l'arc $(i,j)$.
- Supposons le résultat vrai au rang $k$ : Pour tout $(i,j)$, $m_{ij}^{(k)}$ est le poids minimal ne passant que par des sommets de l'ensemble $\{1;2;...;k\}$, on cherche donc le poids minimal ne passant que par des sommets de l'ensemble $\{1;2;...;k;k+1\}$.
  - Si ce chemin ( de poids minimal) ne passe pas par $k+1$, il passe donc par des sommets de l'ensemble $\{1;2;...;k\}$, cela donne donc $m_{ij}^{(k)}$.
  - Si ce chemin ( de poids minimal) ne passe pas par le sommet $k+1$, il est donc la somme des poids des chemins de $i$ à $k+1$ et de $k+1$ à $i$ pour lesquels on a minimiser les poids et ne passant que par des sommets de l'ensemble $\{1;2;...;k\}$. Ce qui donne bien $m_{i,k+1}^{(k)} + m_{k+1,j}^{(k)}$.

  Le chemin étant celui de poids minimal, on a bien la propriété au rang $k+1$.

2. Écrire la fonction `construction` qui prend une matrice $m$ et renvoie la matrice $m^{(n)}$, où $n$ correspond à la taille de la matrice.

```
construction : graphe -> graphe
```

Correction

Listing 4 – La fonction `construction`

```
let construction (m:graphe)  =
  let n = Array.length m in
  for k = 0 to n-1 do
  for i = 0 to n-1 do
    for j = 0 to n-1 do
      m.(i).(j) <- mini_poids (m.(i).(j))  (somme m.(i).(k)  m.(k).(j))
    done;
```

```
  done;
  done;
  m;;
```

3. La fonction précédente renvoie la matrice des poids, mais ne retrace pas le chemin correspondant. Modifier l'algorithme précédent pour qu'il renvoie la liste des sommets, ainsi que le poids minimum.

   `construction_avec_chemin : graphe -> (poids * int list) array array`

   Correction

Listing 5 – La fonction `construction_avec_chemin`

```
let construction_avec_chemin (m:graphe)  =
 let n = Array.length m in
 let chemin = Array.make_matrix n n (Inf, []) in
 for i = 0 to n-1 do
   for j = 0 to n-1 do
     chemin.(i).(j) <- ( m.(i).(j) , [] )
   done;
 done;

 for k = 0 to n-1 do
 for i = 0 to n-1 do
   for j = 0 to n-1 do
     let (p_i_j , c_i_j) =  chemin.(i).(j) in
     let (p_i_k , c_i_k) =  chemin.(i).(k) in
     let (p_k_j , c_k_j) =  chemin.(k).(j) in
     let som = somme p_i_k  p_k_j in
     if inferieur som p_i_j then
       chemin.(i).(j) <- (som , c_i_k @ [k] @ c_k_j)
   done;
 done;
 done;
 chemin;;
```

# 3 Algorithme de Dijkstra

Une deuxième méthode pour la recherche du chemin de poids minimal entre deux sommet $s_0$ et $s_n$ est proposé par E. W. Dijkstra (1930-2002) en 1959. L'algorithme dû à Dijkstra est basé sur le principe suivant :

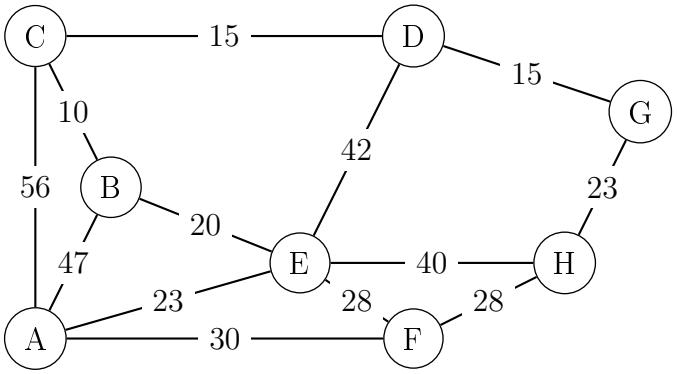
Si le plus court chemin reliant le sommet de départ $s_0$ à un autre sommet $s_n$ passe par les sommets $s_1, s_2, ..., s_n$ alors, les différentes étapes sont aussi les plus courts chemins reliant $s_0$ aux différents sommets $s_1, s_2, ..., s_n$.

On construit de proche en proche le chemin cherché en choisissant à chaque itération de l'algorithme, un sommet $s_i$ du graphe parmi ceux qui n'ont pas encore été traités, tel que la longueur connue provisoirement du plus court chemin allant de $s_0$ à $s_i$ soit la plus courte possible.

1. On utilise un tableau comportant trois information :

   - Le poids minimum du chemin issue de $s_0$.
   - L'indice du dernier sommet rencontré.
   - Le booléen lié au marquage du sommets

   Compléter le tableau suivant représentant le chemin de poids minimum en partant de $A$ pour aller vers $H$. Initialement, on attribut donc à $A$ le poids 0. Chaque sommet $s_i$ marqué sera encadré, le poids représentant alors le poids minimal de $s_0$ à $s_i$ :




| $A$ | $B$ | $C$ | $D$ | $E$ | $F$ | $G$ | $H$ |
|---|---|---|---|---|---|---|---|
| $\boxed{0(A)}$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |

Correction

| $A$ | $B$ | $C$ | $D$ | $E$ | $F$ | $G$ | $H$ |
|---|---|---|---|---|---|---|---|
| $\boxed{0(A)}$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| | $47(A)$ | $56(A)$ | $\infty$ | $\boxed{23(A)}$ | $30(A)$ | $\infty$ | $\infty$ |
| | $43(E)$ | $56(A)$ | $65(E)$ | | $\boxed{30(A)}$ | $\infty$ | $63(E)$ |
| | $\boxed{43(E)}$ | $56(A)$ | $65(E)$ | | | $\infty$ | $58(F)$ |
| | | $\boxed{53(B)}$ | $65(E)$ | | | $\infty$ | $58(F)$ |
| | | | $65(E)$ | | | $\infty$ | $\boxed{58(F)}$ |
| | | | | | | | |

2. Le tableau précédemment aura pour type (`poids` $*$ `int` $*$ `bool`)`array`, le poids représentant le poids minimal, l'entier représente l'indice du dernier sommet rencontré, et le booléen est lié au marquage du sommets.

   (a) Écrire la fonction `recherche_min` qui recherche le poids minimum dans le tableau parmi les sommets non marqués. La fonction prend en argument un tableau et l'entier désignant la taille de ce tableau et renvoie l'indice correspondant au poids minimum non déjà marqué.

   ```
   recherche_min : (poids * 'a * bool) array -> int -> int
   ```

   Correction

Listing 6 – La fonction `recherche_min`

```
let recherche_min tab n =
    let poids_min = ref Inf in
    let indice_min = ref (-1) in
    for i = 0 to n-1 do
      let (p,indice, marque) = tab.(i) in
      if not marque && inferieur p !poids_min then
        (poids_min := p ; indice_min := i )
    done;
    !indice_min ;;
```

   (b) Écrire la fonction `tableau_dijkstra` qui construit le tableau précédent. Les arguments seront le graphe, l'indice de départ et celui d'arrivé.

   ```
   tableau_dijkstra :
     poids array array -> int -> int -> (poids * int * bool) array
   ```

   Correction

Listing 7 – La fonction `tableau_dijkstra`

```
let tableau_dijkstra g init final =
  let n = Array.length g in
  (* Creation du tableau, avec
        * le poids minimum cumule,
        * l'indice de la provenance
        * le booleen de marquage *)
  let tableau = Array.make n (Inf,  0 , false) in
  tableau.(init) <-  ( P(0) , init , false ) ;
  (* la construction du tableau tant que l'indice final n'est pas
marque *)
  while
    ( let (p,indice, marque) =tableau.(final) in not marque) do
    let i = recherche_min tableau n in
    for j = 0 to (n-1) do
      let (p_i,indice_i,marque_i ) = tableau.(i) in
      let (p_j,indice_j,marque_j ) = tableau.(j) in
      let s = somme  g.(i).(j) p_i in
      if inferieur s p_j then
        begin
           tableau.(j) <- (s , i , marque_j) ;

        end ;
    tableau.(i) <- (p_i,indice_i, true) ;
    done ;
  done;
  tableau;;
```

(c) Une fois que le tableau est construit, il faut reconstituer le chemin. Écrire la fonction `construction_chemin` qui prend un tableau de type précédent, ainsi que l'indice de départ et celui d'arrivée, et renvoie la liste des sommets du chemin de poids minimal.

```
construction_chemin : ('a * int * 'b) array -> int -> int -> int
```

Correction

Listing 8 – La fonction `construction_chemin`

```
let construction_chemin tableau init final =
  let l = ref [final] in
  let indice_courant = ref final in
  while !indice_courant <> init do
    let (p,i,marque ) = tableau.(!indice_courant) in
    l:= i:: !l ;
    indice_courant := i ;
  done;
!l;;
```

(d) En déduire la fonction `poids_et_chemin` qui prend en argument un graphe, un sommet de départ et un sommet d'arrivée, et renvoie le couple liste , poids correspond au chemin de poids minimal répondant à la question.

```
poids_et_chemin : poids array array -> int -> int -> int list * poids
```

Correction

Listing 9 – La fonction `poids_et_chemin`

```
let poids_et_chemin g init final =
  let tableau = tableau_dijkstra g init final in
  let l = construction_chemin tableau init final in
  let (p,i,marque ) = tableau.(final) in
  (l,p);;
```