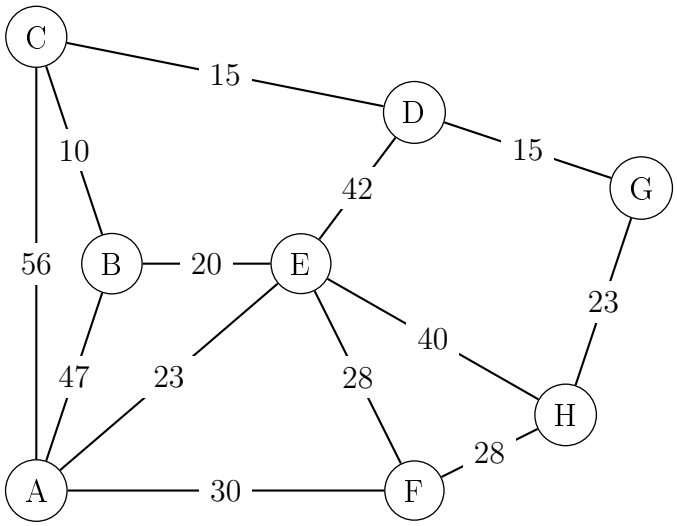
TD 6

Algorithmes sur les graphes

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
   ```

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

   ```
   inferieur : poids -> poids -> bool
   ```

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

   ```
   mini_poids  : poids -> poids -> poids
   ```

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 $\{1;2;...;k\}$.

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
   ```

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
   ```

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 :

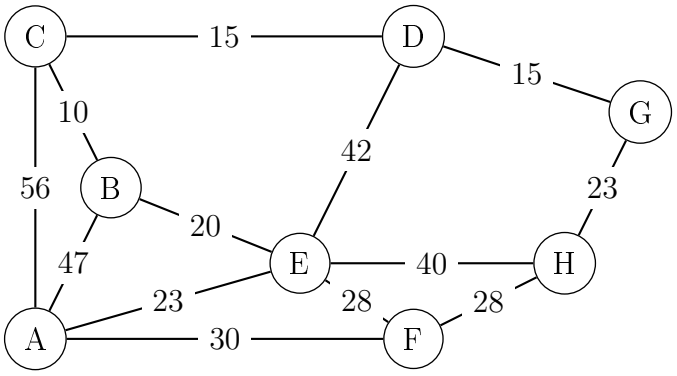
Si le plus court chemin reliant le sommet de départ s_0 à un autre sommet s_n passe par les sommets $s_1, s_2, ..., s_n$ alors, les différentes étapes sont aussi les plus courts chemins reliant s_0 aux différents sommets $s_1, s_2, ..., s_n$.

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
$\boxed{0(A)}$	∞	∞	∞	∞	∞	∞	∞

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
   ```

   (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
   ```

   (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
   ```

   (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
   ```