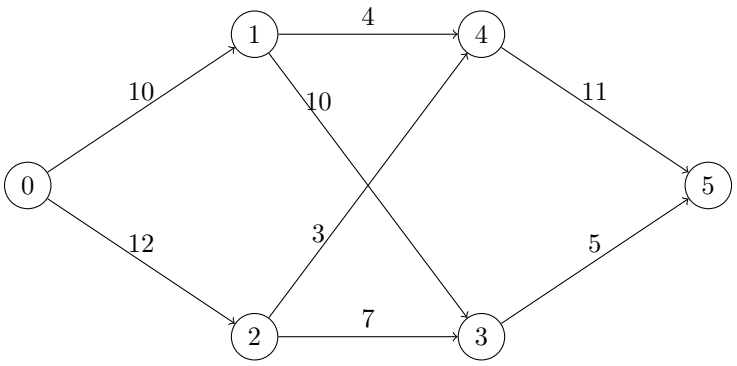
PLUS COURT CHEMIN DANS UN GRAPHE ACYCLIQUE

Un graphe est un couple (S, A) où S est un ensemble de sommets et A est un ensemble d'arêtes, c'est-à-dire de couples $(s, t) \in S \times S$.

Un chemin dans un graphe est une suite finie de sommets $(s_0, ..., s_{p-1})$ tels que pour tout $i \in \{0, ..., p-2\}$, le couple (s_i, s_{i+1}) est une arête. Un sommet t est dit accessible à partir d'un sommet s s'il existe un chemin d'origine s et d'extrémité t. Une source d'un graphe est un sommet s tel que tout autre sommet soit accessible à partir de s ; un but est un sommet t qui est accessible à partir de tout autre sommet.

Un graphe est dit valué si on dispose d'une application qui à chaque arête associe une valeur numérique. Dans un graphe valué, le score d'un chemin est la somme des valeurs de chaque arête.

Le schéma suivant représente graphiquement un graphe valué à 6 sommets et 8 arêtes :



Un graphe est dit acyclique s'il n'existe aucun chemin $(s_0, ..., s_{p-1})$ tel que $s_0 = s_{p-1}$. Dans un graphe acyclique, il ne peut y avoir qu'une seule source et un seul but.

Dans cette séance, il est sous-entendu que tous les graphes sont valués, acycliques et qu'ils possèdent une source et un but. L'objectif est de construire un algorithme qui donne le plus court chemin entre la source et le but (chemin de valeur minimale).

1 Représentation informatique

Nous représenterons un graphe par une liste de triplets d'entiers (o, ℓ, f), où o et f sont des numéros de sommets et ℓ est la valeur de l'arête (o, f) (ces triplets seront appelés arêtes valuées). Les $n+1$ sommets sont numérotés de 0 (pour la source) à n (pour le but).

Par exemple, le graphe représenté ci-dessus est codé par

```
let graphe =[0,10,1; 0,12,2; 1,4,4; 1,10,3; 2,3,4; 2,7,3; 3,5,5; 4,11,5];;
```

Pour définir un chemin, nous pourrions utiliser une liste d'entiers. Mais nous voulons pouvoir aussi connaître le score d'un chemin. Aussi nous allons utiliser une possibilité de CAML, qui est de pouvoir définir des types nouveaux :

```
type chemin = Vide | Extr of int * int * chemin;;
```

Cette définition crée un type défini récursivement (comme les listes). Un chemin vide sera symbolisé par la constante `Vide` et si on connaît un chemin, alors on peut le prolonger en lui ajoutant une extrémité grâce au constructeur `Extr(n,s,chem)` : cela représente le chemin `chem` auquel on a ajouté en position finale le sommet portant le numéro `n`, ce nouveau chemin ayant pour score `s`.

Par exemple, le chemin $(0, 1, 3)$ de score 20 dans le graphe ci-dessus est représenté par :

```
let chemin = Extr(3,20,Extr(1,10,Vide));;
```

On définit deux fonctions sur les chemins par reconnaissance de motifs :

```
let score chem =
    match chem with
    | Vide -> 0
    | Extr(n,s,ch) -> s;;

let extremite chem =
    match chem with
    | Vide -> 0
    | Extr(n,s,ch) -> n;;
```

Par convention, le chemin Vide (qui symbolise le chemin à zéro arête) a pour extrémité la source de numéro 0, puisque nous cherchons des chemins d'origine 0.

2 Description de l'algorithme

Nous allons construire une liste de chemins, partant de l'origine.

Au départ, cette liste contient le chemin Vide.

Pour chaque chemin appartenant à la liste, on calcule d'abord tous ses prolongements, c'est-à-dire tous les chemins construits à partir du chemin donné en lui ajoutant une arête : on peut prolonger un chemin si et seulement si son extrémité n'est pas le but. On range tous les nouveaux chemins dans une nouvelle liste. On réduit alors cette nouvelle liste : si deux chemins ont la même extrémité, alors on ne garde que celui de score minimal. Puis on remplace la liste par la nouvelle liste. On répète cette opération tant qu'il reste un chemin dans cette liste dont l'extrémité n'est pas le but.

2.1 Prolongements

Dans les types qui suivent, le mot `graphe` est un synonyme de `(int * int * int) list`, c'est-à-dire qu'il est le type d'une liste d'arêtes valuées.

a) Écrivez une fonction `tab_successeurs graphe n` de type `graphe -> int -> graphe vect`, qui calcule le tableau `t` de longueur `n` tel que `t.(i)` soit la liste des arêtes valuées qui ont pour origine le sommet `i`.

b) Écrivez une fonction `prolonge chem aretes` de type `chemin -> graphe -> chemin list`, qui construit la liste suivante :
   - si la liste d'arêtes valuées `aretes` est non vide, alors c'est la liste de tous les prolongements du chemin `chem` par ces arêtes ;
   - sinon c'est la liste `[chem]`.

   Vous n'oublierez pas de calculer le score de chacun des prolongements (c'est dans la définition d'un chemin...).

c) Écrivez une fonction `prolongements list_chem t_succ` de type `chemin list -> graphe vect -> chemin list`, qui construit la liste de tous les prolongements possibles de tous les chemins de la liste de chemins `list_chem`.

2.2 Réduction

a) On suppose que la liste `list_chem` est une liste de chemins, qui ne contient pas deux chemins de même extrémité. Écrivez une fonction `optimise chem list_chem` de type `chemin -> chemin list -> chemin list`, qui rend la liste suivante :
   - si la liste ne contient aucun chemin ayant même sommet extrémal que le chemin `chem`, on ajoute ce chemin à la liste ;
   - si la liste contient un chemin ayant même sommet extrémal que le chemin `chem`, on le garde si son score est meilleur que celui de `chem`, sinon on le remplace par `chem`.

b) Écrivez une fonction `reduire list_chem`, de type `chemin list`, qui réduit la liste : parmi tous les chemins de la liste paramètre ayant même sommet extrémal, on ne garde que celui de meilleur score. Vous devez donc obtenir une liste ne contenant pas deux chemins ayant la même extrémité.

2.3 Algorithme solution

Écrivez un algorithme `plus_court_chemin graphe n` de type `graphe -> int -> chemin`, qui calcule le plus court chemin de la source (numérotée 0) au but (numéroté n) dans le graphe.

Sur le réseau a été déposé un fichier contenant divers graphes. Vous pouvez tester votre fonction sur chacun d'eux.