TD 6 – Graphes

Exercice 1. *Algorithme de Floyd-Warshall*

Soit $G = (V, A)$ un graphe orienté muni d'une pondération $w : A \to \mathbb{R}$ telle qu'il n'y a aucun circuit de longueur strictement négative. Le but de l'algorithme de Floyd-Warshall est de calculer la longueur du plus court chemin de i à j pour tous sommets i et j.
Pour ceci, on définit d_{ij}^k le poids d'un plus court chemin de i à j dont tous les sommets intermédiaires sont dans $\{1, \dots, k\}$, et π_{ij}^k le prédécesseur de j sur un tel chemin. En particulier pour $k = 0$, un tel chemin n'a pas de sommets intermédiaires. On utilise la convention $d_{ij}^k = \infty$ et $\pi_{ij}^k = NIL$ s'il n'existe pas de chemin de la forme souhaitée.

1. Donner une relation de récurrence pour d_{ij}^k et π_{ij}^k.
2. En déduire un algorithme pour calculer un plus court chemin de i à j dans G, pour tous sommets i, j.
3. Quelle est sa complexité ?

Exercice 2. *Algorithme de Dijkstra*

Soit $G = (V, A)$ un graphe orienté muni d'une pondération positive $w : A \to \mathbb{R}^+$ et un s un sommet du graphe. Le but est de trouver des chemins de poids minimaux depuis s vers tous les sommets du graphe, sous la forme d'une arborescence des plus court chemin depuis s, c'est-à-dire un sous-graphe (G', A') tel que :
– Les sommets de G' sont exactement les sommets accessibles depuis s.
– Le sous-graphe est une arboresence de racine s, c'est-à-dire un arbre de racine s dont toutes les arcs sont orientés en direction des feuilles.
– Pour tout sommet u, l'unique chemin de s à u dans l'arborescence correspond à un plus court chemin dans G.

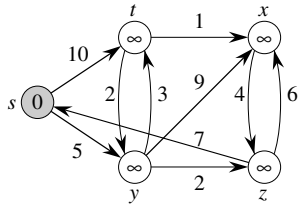
Voici l'algorithme de Dijkstra. Avec $\forall u \in V, adjacent[u]$ la liste des sommets adjacents à u.

Algorithm 1: Dijkstra(G, w, s)

```
début
    pour tous les sommet v ∈ V /* Initialisation                    */
    faire
        d[v] ← ∞, π[v] ← NIL;
    d[s] ← 0;
    E ← ∅;
    F ← V[G];
    /* ensemble des sommets de G                                    */
    tant que F ≠ ∅ faire
        u ← v|d[v] = min{d[x]|x ∈ F};
        /* on choisit le sommet avec la plus petite valeur de d     */
        F ← F − {u};
        E ← E ∪ {u};
        pour tous les sommet v ∈ adjacent[u] /* relâchement de l'arc (u,v) */
        faire
            si d[v] > d[u] + w(u,v) alors
                d[v] ← d[u] + w(u,v);
                π[v] ← u;
```

Dans la suite, on note $\delta(u, v)$ le poids du plus court chemin de u vers v (∞ s'il n'existe pas).

1. Faire tourner l'algorithme de Dijkstra sur l'exemple suivant



2. **Propriété du majorant :** Montrer que $d[v] \geq \delta(s, v)$ pour tout sommet v à tout instant de l'algorithme. Montrer que lorsque $d[v]$ a atteint sa borne inférieure $\delta(s, v)$, il n'est plus jamais modifié.
3. Montrer que pour chaque sommet u, $d[u] = \delta(s, u)$ au moment où u est ajouté à E.
4. Montrer si pour tout sommet u, $d[u] = \delta(s, u)$ alors G_Π nous fournit une arborescence de plus court chemin depuis s, où les sommets de G_Π sont $\{s\} \cup \{u | \Pi(u) \neq NIL\}$ et les arcs sont $\{uv | \Pi(v) = u\}$
5. En déduire la validité de l'algorithme de Dijkstra, c'est-à-dire qu'à la fin pour tout sommet u, $d[u] = \delta(s, u)$ et qu'il fournit une arborescence de plus court chemin.
6. Quelle est la complexité de l'algorithme ?

Exercice 3. *Dans les profondeurs de la topologie...*

On travaille sur des graphes *orientés*, que l'on suppose représentés par des *listes d'adjacence* : soit S l'ensemble des sommets et A l'ensemble des arcs, on associe à chaque sommet u dans S la liste $Adj[u]$ des éléments v tels qu'il existe un arc de u à v.

Parcours en profondeur :

Le parcours en profondeur d'un graphe G fait usage des constructions suivantes :

- A chaque sommet du graphe est associée une *couleur* : au début de l'exécution de la procédure, tous les sommets sont blancs. Lorsqu'un sommet est rencontré pour la première fois, il devient gris. On noircit enfin un sommet lorsque l'on est en fin de traitement, et que sa liste d'adjacence a été complètement examinée.
- Chaque sommet est également *daté* par l'algorithme, et ceci deux fois : pour un sommet u, $d[u]$ représente le moment où le sommet a été rencontré pour la première fois, et $f[u]$ indique le moment où l'on a fini d'explorer la liste d'adjacence de u. On se servira par conséquent d'une variable entière "temps" comme compteur évènementiel pour la datation.
- La manière dont le graphe est parcouru déterminera aussi une relation de paternité entre les sommets, et l'on notera $\pi[v] = u$ pour dire que u est le père de v (selon l'exploration qui est faite, c'est à dire pendant le parcours v a été découvert à partir de u).

On définit alors le parcours en profondeur (**PP**) de la manière suivante :

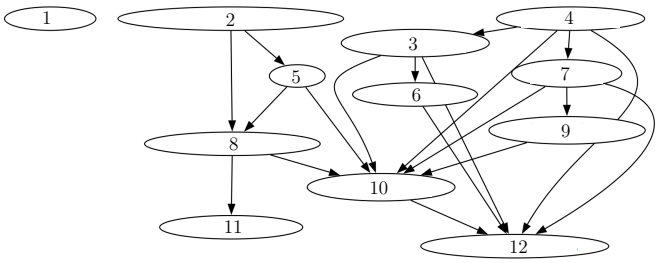
Algorithm 2: PP(S, $Adj[]$, $temps$)

```
début
    pour tous les sommet u ∈ S/* Initialisation                    */
    faire
        couleur[u] ← BLANC;
        π[u] ← NIL;
    temps ← 0;
    pour tous les sommets u ∈ S/* Exploration                      */
    faire
        si couleur[u] = BLANC alors
            Visiter_PP(u);
```

Algorithm 3: Visiter_PP(u)

```
début
    couleur[u] ← GRIS;
    d[u] ← temps;
    temps ← temps + 1;
    pour tous les v ∈ Adj[u] faire
        si couleur[v] = BLANC alors
            π[v] ← u;
            Visiter_PP(v);
    couleur[u] ← NOIR;
    f[u] ← temps;
    temps ← temps + 1;
```

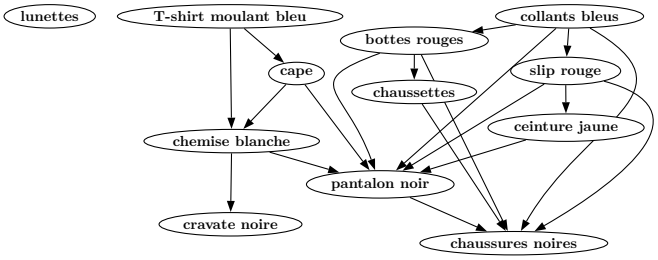
1. Faire tourner l'algorithme sur l'exemple suivant



2. Quel est la complexité de PP ? Que peut-on dire de $d[u]$, $f[u]$, $d[v]$ et $f[v]$, pour deux sommets u et v du graphe ?

3. Montrer que v est un descendant de u si et seulement si à l'instant $d[u]$, il existe un *chemin blanc* de u à v.

Tri topologique : n tâches sont données avec des contraintes de précédence $A < B$ signifie que la tâche A doit être effectuée avant la tâche B. L'objectif est de trouver un ordre des tâches qui respecte les contraintes de précédence. Pour cela, on modélise le problème par un graphe orienté :
– les sommets sont les tâches et
– il y a un arc $A \to B$ si et seulement si $A < B$, i.e. si A doit être exécuté avant A.
Voici par exemple le graphe de contrainte pour permettre à Clark Kent de s'habiller le matin avec son habit de Superman sous son costume :



Il s'agit alors de trouver un ordre des sommets tel que les sommets pointés par un sommet donné soient toujours à sa droite. On remarque que si le graphe admet un circuit, ce n'est pas possible.

4. Montrer qu'un graphe est sans circuit si et seulement si le parcours en profondeur ne produit aucun arc arrière, c'est-à-dire un arc uv tel que v est un ancêtre de u dans l'arboresence produite par le parcours en profondeur.

5. Utiliser le parcours en profondeur pour écrire un algorithme qui calcule le tri topologique de G. Donner sa complexité et prouver sa validité

6. En déduire un tri topologique pour habiller Superman grâce à l'exemple de la question 1.