Les graphes
Plus courts chemins
Shortest Paths

Le prince

Le prince est parti à la recherche du trésor ; il peut accomplir les actions suivantes :

— Du point de départ, aller à la ville du marché, en contournant la rivière par un gué : 4 jours.

— Du point de départ, traverser la forêt : 1 jour.

— Depuis la forêt, abattre des arbres pour traverser la rivière, et se rendre à la ville du marché : 2 jours.

— Depuis la forêt, se rendre à la capitale provinciale en traversant les marais : 7 jours.

— S'equiper chaudement au marché, et partir pour le col du nord : 5 jours.

— Trouver un bon cheval au marché, et se rendre à la capitale provinciale par la grand-route : 3 jours.

— Depuis le col du nord, se rendre au refuge du devin : 3 jours.

— Depuis la capitale provinciale, se rendre au refuge du devin : 4 jours.

— Se rendre de la capitale provinciale au palais du roi, en étant retardé par des contrôles : 10 jours.

— Au sortir du devin, partir directement chercher l'épée, et la trouver après s'être perdu par manque de carte : 20 jours.

— Au sortir de chez le devin, au mépris de ses avis, se rendre directement à la grotte et tuer le dragon avec un canif : 32 jours (il faut du temps pour le tuer avec un canif).

— Bien conseillé par le devin, prendre un raccourci pour le palais du roi : 5 jours.

— Un fois arrivé au palais du roi, séduire la bibliothécaire, puis trouver les cartes qui expliquent l'emplacement de l'épée et du trésor : 6 jours.

— En utilisant les cartes trouvées dans la bibliothèque, faire tout le tour de la montagne, et traverser un labyrinthe qui mène directement au trésor : 30 jours.

— En utilisant les cartes, aller chercher l'épée pour combattre le dragon : 7 jours.

— S'entraîner à l'épée, puis tuer le dragon : 8 jours.

— Une fois l'épée trouvée, au lieu d'affronter le dragon, utiliser l'épée pour creuser un tunnel par dessous, et déboucher directement dans la cachette du trésor : 18 jours.

— Une fois le dragon tué, résoudre l'énigme qui ouvre la cachette du trésor : 9 jours.

Comment doit-il faire pour récupérer le trésor le plus vite possible ? Quel temps lui faudra-t-il ?

1 La star des algos

Exercice 1.1 (Dijkstra)

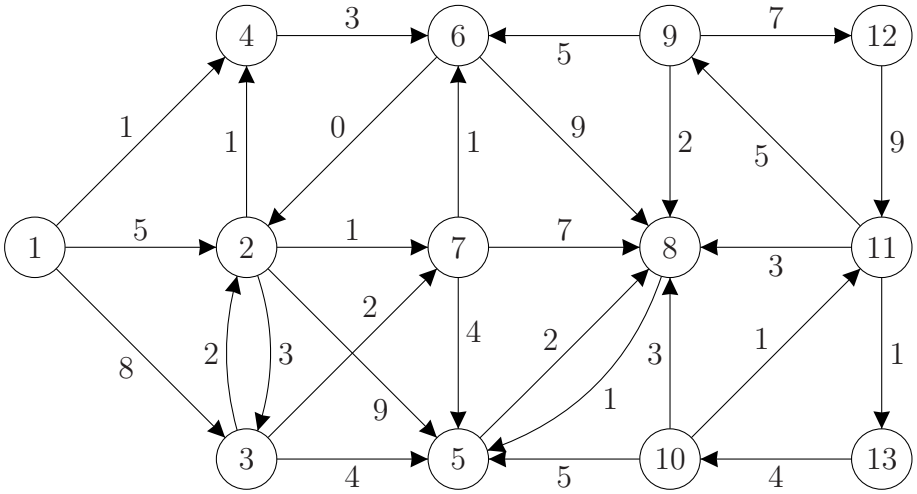


FIGURE 1 – Graphe G_1

Le but de l'exercice est de trouver le plus court chemin entre deux sommets donnés dans un graphe à coûts positifs (orienté ou non). Nous chercherons ici à trouver le plus court chemin entre les sommets 1 et 8 de la figure 1.

1. Au départ l'algorithme va remplir le tableau *dist* avec les valeurs suivantes :

	1	2	3	4	5	6	7	8	9	10	11	12	13
dist	0	5	8	1	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$	$+\infty$

   (a) Que représente ce tableau ? Que signifie la valeur $+\infty$ dans certaines cases ?

   (b) Comment a-t-on obtenu ces valeurs ?

   (c) Que faut-il faire ensuite pour mettre à jour les informations ?

2. (a) Que se passe-t'il si on choisit d'abord le sommet 2 pour mettre à jour les distances ?

   (b) Comment trouver un ordre de sommets tel que lorsqu'un sommet x est choisi, on est sûr que $dist[x]$ ne pourra plus être minimisée ?

3. Comment représenter le chemin obtenu (l'ordre des sommets) ?

4. (a) En déduire un principe de recherche des plus courts chemins d'un sommet donné vers tous les autres sommets du graphe.

   (b) Appliquer ce principe au graphe de la figure 1 pour trouver le plus court chemin de 1 à 8.

   (c) Écrire l'algorithme correspondant.

Exercice 1.2 (L'aller, puis le retour . . .)

L'objectif de cet exercice est de construire un plus court chemin *aller/retour* d'un sommet vers lui même en passant par un sommet particulier dans un graphe non orienté. On désire que le chemin de retour **ne passe pas** par les mêmes sommets qu'à l'aller.

On cherche à adapter l'algorithme de Dijkstra pour ce problème. En première approximation, on pourrait considérer qu'il suffit de répéter l'opération de la source à la destination, puis de la destination à la source. Malheureusement, les chemins ainsi trouvés ne garantissent pas que les sommets de l'aller n'apparaissent pas au retour.

L'algorithme que nous allons écrire cherchera donc à construire un plus court chemin pour l'aller, puis un plus court chemin pour le retour qui ne passe pas par les sommets du plus court chemin de l'aller. Dans la suite, nous supposerons qu'un tel chemin existe toujours.

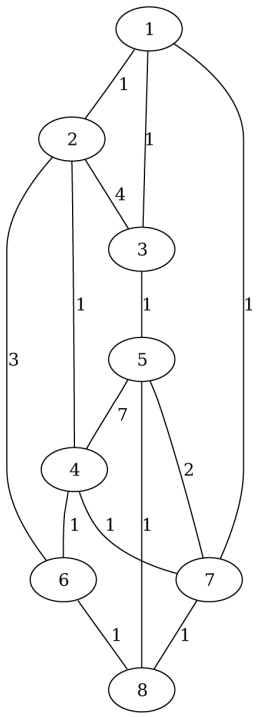


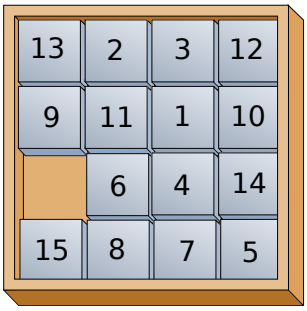
FIGURE 2 – Graphe G_2

1. **Méthode ?**
   (a) Quelle propriété d'un graphe non-orienté nous assure que nous pouvons trouver un tel chemin aller/retour ?
   (b) Comment peut-on adapter l'algorithme de Dijkstra pour résoudre notre problème ?
   (c) Donner un plus court chemin aller/retour (selon la définition précédente) pour le graphe de la figure 2 partant de 1 à 8 et revenant à 1.

2. **Optimisations ?**
   Nous allons en profiter pour optimiser l'algorithme de Dijkstra !
   (a) Quelle structure de données (vue en sup !) nous permet de récupérer la valeur minimum d'un ensemble en temps constant et d'ajouter un nouvel élément en temps logarithmique ?
   (b) Que faut-il ajouter aux opérations de base de cette structure dans le cas de l'implémentation de Dijkstra sur un graphe en représentation dynamique ?

3. Écrire tous les algorithmes nécessaires à la résolution de notre problème : trouver le plus court chemin aller/retour (satisfaisant la définition précédente) en partant d'un sommet *source* et passant par un sommet *dest*.

Exercice 1.3 (Jouons un peu !)



Le taquin (*Fifteen Puzzle*) est un puzzle qui se présente sous la forme d'un damier où peuvent coulisser des pièces carrées. Le but est de positionner les pièces de manière à ce qu'elles reforment le dessin d'origine (où comme dans la version d'origine, de manière à ce qu'elles se retrouvent dans l'ordre). Les pièces ne peuvent se déplacer que s'il l'une des cases adjacentes (haut, bas, gauche ou droite) est vide. Il n'y a qu'une case libre dans tout le taquin.

Le taquin est représenté par une matrice d'entiers $N \times N$: les pièces sont numérotées de 1 à $N^2 - 1$ et le trou aura la valeur 0.

1. On définit la *distance* d'une pièce à sa position finale comme la somme des déplacements horizontaux et verticaux à effectuer.
   La *distance globale* est la somme des *distances* de chaque pièce entre un taquin mélangé et un taquin résolu (où les cases sont triées dans l'ordre croissant de gauche à droite et de haut en bas).
   Comment calculer la distance globale entre un taquin mélangé t de taille $N \times N$ et le taquin résolu ?

2. *Résoudre le taquin ?*
   (a) Comment représenter le problème sous forme d'un graphe ?
   (b) Quel(s) algorithme(s) permettrait de résoudre le taquin ?
   (c) A quoi sert le calcul de la *distance globale* ?

3. *Quel problème pose l'implémentation d'un algorithme qui cherche une solution ?*

Exercice 1.4 (Et l'Astar dans tout ça ?)

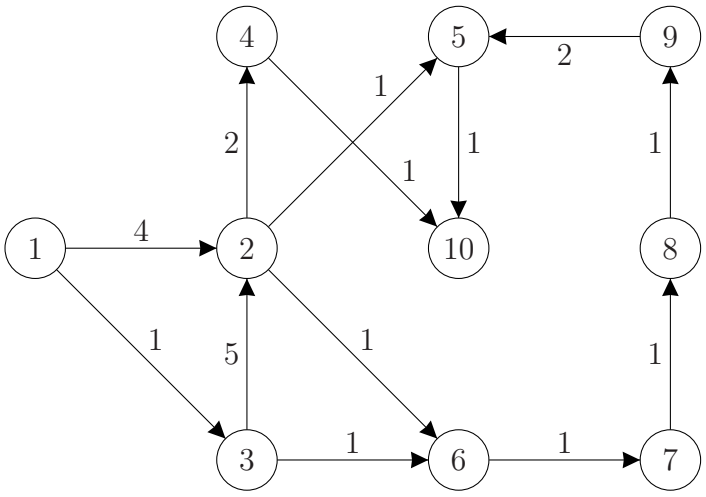


FIGURE 3 – Graphe pour A^*

1. Donner le principe de l'algorithme A^*.

2. Appliquer A^* au graphe de la figure 3, avec la fonction H donnée ci-dessous.

s	1	2	3	4	5	6	7	8	9	10
$H(s)$	3	2	3	1	3	6	4	3	2	0

   Comparer avec le résultat donné par Dijkstra.

2 C'est négatif

Exercice 2.1 (Bellman)

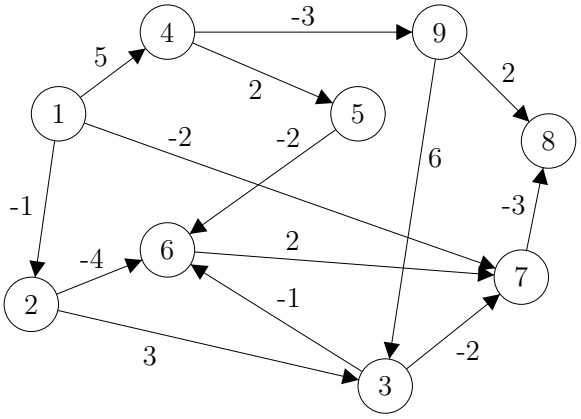
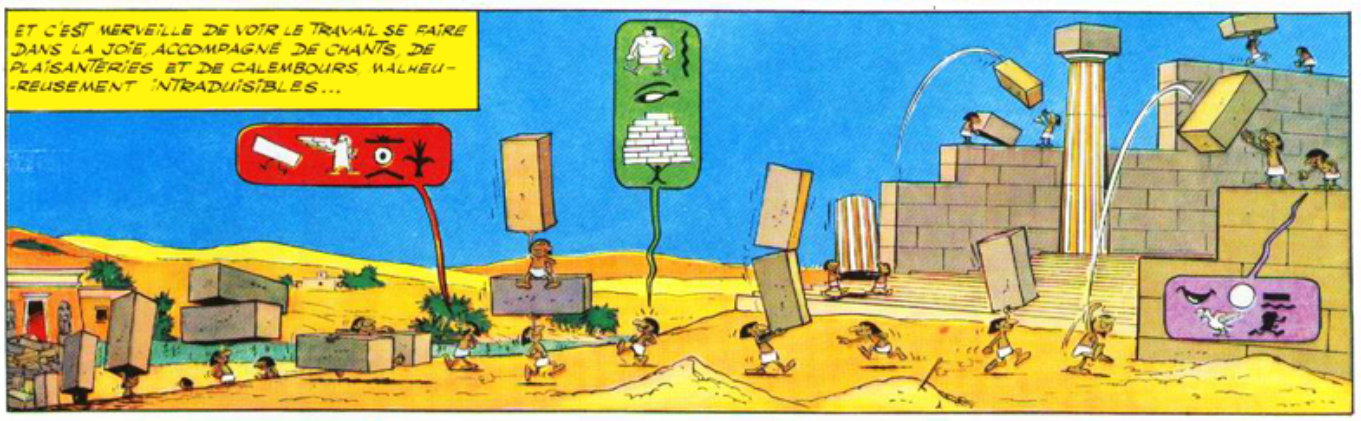


FIGURE 4 – Graphe G_3

1. Appliquer le principe de l'algorithme de Dijkstra (exercice 1.1) au graphe G_3 pour trouver le plus court chemin entre les sommets 1 et 8.
2. Quel est le problème ? Peut-on modifier le graphe afin d'appliquer Dijkstra ?
3. Comment choisir les sommets dans ce cas particulier afin de ne les traiter qu'une seule fois ?
4. En déduire un principe détaillé de recherche de plus courts chemins dans un graphe à coûts négatifs lorsque celui-ci ne contient pas de circuits.
5. Appliquer ce principe au graphe G_3.



Exercice 2.2 (Plus court chemin et parcours profondeur)

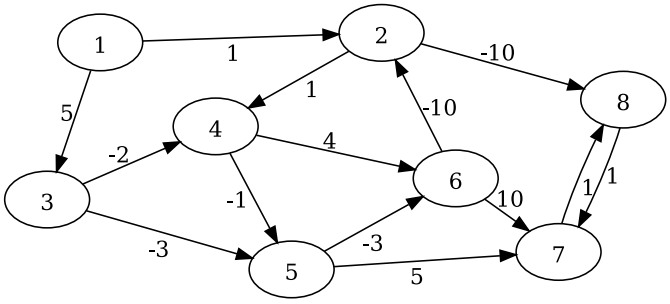


FIGURE 5 – Graphe G_4

Nous allons nous intéresser à la recherche de plus courts chemins dans des graphes orientés avec coûts quelconques. Notre problème ici est qu'il n'y a aucune garantie sur l'absence de circuit dans les graphes considérés. Plutôt que de chercher un algorithme résistant aux circuits, nous préférerons, pour le problème considéré, *éliminer* les circuits. Éliminer arbitrairement les circuits ne garantit pas forcément les meilleurs chemins vers tous les sommets du graphe, mais assure l'existence d'une solution satisfaisante dans tous les cas.

1. **Tri topologique et élimination des circuits :**

   En raison des coûts négatifs, l'algorithme retenu sera l'algorithme de Bellman.

   (a) Il existe plusieurs stratégies pour mettre en oeuvre cet algorithme : nous utiliserons ici un algorithme de parcours profondeur afin de récupérer les sommets dans l'ordre pour Bellman. La présence de circuits pose-t-elle un problème ?

   (b) L'algorithme de Bellman utilisé ici devra ignorer certains arcs du graphe, afin de ne pas emprunter de circuits. Quelles informations le parcours profondeur peut-il fournir afin de repérer ensuite ces arcs à éliminer ?

   (c) Écrire l'algorithme du parcours profondeur du graphe qui fournit une solution de tri topologique exploitable par l'algorithme de Bellman (qui recherche les plus courts chemins depuis une source donnée), ainsi que les informations nécessaires à l'"élimination" des circuits.

   (d) Appliquer au graphe de la figure 5 en prenant 1 comme source. Les sommets seront choisis en ordre croissant.

2. **Plus courts chemins :**

   (a) Appliquer l'algorithme de Bellman (qui ignore les circuits) sur le graphe de la figure 5 en utilisant les résultats de la question 1d pour calculer les plus courts chemins sans circuits à partir du sommet 1.

   (b) Écrire la procédure `bellman_prof (`G`, `*src*`, `*pere*`, `*dist*`)` qui calcule l'ensemble des *plus courts* chemins (en ignorant les circuits) depuis le sommet *src* dans le graphe G. L'algorithme remplit le vecteur de pères (*pere*) et le vecteur de distances (*dist*) avec les chemins calculés.

Exercice 2.3 (Construction)

M. Christophe B. veut se faire construire une résidence secondaire à Villejuif-plage. Dans le tableau ci-dessous la liste des tâches à effectuer. Pour chaque tâche sont indiqués son libellé, sa durée et ses prédécesseurs (les tâches qui doivent être terminées avant la réalisation de la tâche courante). Les tâches peuvent être réalisées simultanément.

Le but est ici d'aider C.B. à réaliser son projet le plus rapidement possible.

Code tâche	Libellé	Durée (semaines)	Prédécesseurs
0	Obtention du permis de construire	2	-
1	Maçonnerie	7	-
2	Charpente de la toiture	3	0, 1
3	Toiture	1	2
4	Plomberie et électricité	8	0, 1
5	Façade	2	3, 4
6	Fenêtres	1	3, 4
7	Aménagement du jardin	1	3, 4
8	Plafonds	3	6
9	Peintures	2	8
10	Emménagement	1	5, 9

TABLE 1 – La maison de C.B.

1. Modéliser le projet sous forme de graphe.
2. Quelle est la durée minimale du projet donné en énoncé ?
   A quel problème de graphe s'apparente la recherche de la durée minimale du projet ?
3. Adapter le principe de Bellman (exercice 2.1) pour résoudre ce problème.
   Appliquer au graphe de la question 1.
4. Il est possible d'utiliser un parcours profondeur pour récupérer une solution de tri topologique (voir td 5) donnant ainsi l'ordre des sommets à traiter. Utiliser la deuxième méthode (voir exercice 2.1) pour écrire l'algorithme permettant de trouver la durée minimale du projet.
5. La *date au plus tôt* de la tâche i est la date à laquelle la tâche peut commencer au plus tôt. La *date au plus tard* de la tâche i est la date au plus tard où la tâche peut commencer sans entraîner de retard sur le projet. Si la *date au plus tôt* et la *date au plus tard* d'une tâche i sont identiques alors cette tâche est dite critique. Comment peut-on obtenir ces dates ?
6. Calculer les dates au plus tôt et au plus tard du projet de M. B. Quelles sont les tâches critiques ?

Exercice 2.4 (Floyd revisité – *partiel 2013*)

1. Expliquer simplement comment on peut modifier la procédure de Floyd (vue en cours), de sorte qu'elle s'arrête dès qu'elle détecte un circuit absorbant.
2. Dans certains problèmes de communication, on a besoin de connaître le point (ou les points) le moins éloigné de tous les autres points d'un réseau. On appelle **excentricité** d'un sommet x, dans un graphe orienté et valué par des coûts positifs, $G =< S, A, C >$, la quantité :

   $$\text{exc}(x) = \max_{y \in S}\{ppdistance(x, y)\}$$

   C'est la distance nécessaire pour atteindre x depuis chaque autre sommet. On appelle **centre** du graphe G, un sommet d'excentricité minimum.

   Expliquer simplement comment on peut utiliser la procédure de Floyd (vue en cours) pour trouver le centre d'un graphe.

Exercice 2.5 (Et le sourire...)

Soient un ensemble de pays : PB, B, A, S, I, E et F. Dans chacun de ces pays, le beurre a un prix différent : certains surproduisent, d'autres doivent importer. Afin de clarifier les échanges, chaque pays passe un accord commercial avec chacun de ses voisins fixant des aides pour l'exportation vers des pays déficitaires et des taxes pour l'exportation vers des pays surproducteurs.

Ceci peut se résumer en deux tableaux, les coûts étant donnés en milliers d'euro pour 20 tonnes transportées. Le tableau 2 est celui des taxes : $T_{i,j}$ représente la taxe payée pour passer du pays i vers le pays j ; le tableau 3 est celui des aides : le producteur reçoit $A_{i,j}$ s'il exporte de i vers j. S'il n'y a ni aide ni taxe, on supposera qu'il n'y a pas d'échange possible.

	PB	B	A	S	I	F	E
PB							
B	15		8			22	
A	5			5		15	
S						5	
I				15		8	
F							
E						5	

TABLE 2 – Tableau des taxes (T)

	PB	B	A	S	I	F	E
PB		15	5				
B							
A		10					
S			5		15		
I							
F		10	15	5	8		5
E							

TABLE 3 – Tableau des aides (A)

Par ailleurs, le transport du beurre coûte lui-même un certain prix, précisé dans le tableau 4 et exprimé dans la même unité que pour les deux tableaux 2 et 3.

	PB	B	A	S	I	F	E
PB		2	4				
B	2		3			2	
A	4	3		3		4	
S			3		2	5	
I				2		6	
F		2	4	5	6		6
E						6	

TABLE 4 – Tableau des coûts (C)

1. Modéliser le problème sous forme d'un problème de plus court chemin dans un graphe orienté valué.
2. Résoudre le problème de l'exportation du beurre des PB vers E. Que remarque-t-on ?
3. Donner le principe algorithmique pour déterminer le moindre coût entre un sommet donné et tous les autres dans ce type de graphe.
4. De plus, si $A_{F,A} = 10$, que se passera-t-il ? Modifier le principe afin de l'optimiser dans ce type de cas.
5. Écrire l'algorithme correspondant.

3 Scotland Yard – *partiel 2013*

Les exercices qui suivent s'inspirent du jeu Scotland Yard : "Un des joueurs est Mister X, gangster en fuite dans Londres. Les autres joueurs, incarnant des détectives de Scotland Yard, doivent le capturer en coordonnant leurs mouvements."
À chaque tour Mister X et les détectives se déplacent d'une case sur le plateau. Les déplacements de Mister X ne sont connus de personne. Le jeu se déroule en un nombre maximum de tours.

Dans la suite nous supposerons qu'il est possible de connaître la probabilité qu'a Mister X de se déplacer d'une case à l'autre, et que les déplacements des détectives ont des coûts différents selon les cases.

Deux graphes représentent le problème. Ils ont en commun les propriétés suivantes :
— Chaque sommet représente une case du plateau.
— C'est un 1-graphe avec circuits.
— La représentation est dynamique (voir annexes).

Le premier graphe représente les déplacements probables de mister X (exercice 3.1), le deuxième les coûts des déplacements des joueurs (exercice 3.2).

Exercice 3.1 (Mister X)

Les déplacements de Mister X ne sont connus que de lui seul. Seules données : la case de départ et la probabilité qu'il a de se déplacer d'une case à une autre. Dans le graphe de Mister X la valeur de chaque arc (x, y) représente la probabilité (un réel p, $0 \leq p < 1$) qu'a Mister X d'aller en y depuis x.

Le but ici est donc de connaître la probabilité qu'à Mister X de se retrouver en chaque case, et le nombre de tours qu'il lui faudra pour y arriver.

1. Ici nous voulons calculer la probabilité maximum d'accéder à chaque sommet depuis la source. Comment ramener ce problème à une minimisation de sommes de coûts positifs ?

2. Plutôt que de connaître le chemin emprunté, nous voulons savoir le nombre d'arcs empruntés pour accéder à chaque sommet, ce sera le vecteur *pas* : comment l'obtenir ?

3. Quel algorithme faut-il utiliser ici ?

4. Comment représenter l'ensemble (voir ci-dessous) pour optimiser l'algorithme ?

5. Écrire l'algorithme : à partir du graphe (G dont les coûts sont des probabilités), du sommet source (*src* : la case de départ), du nombre maximum de tours (*maxpas*), il donne pour chaque sommet la probabilité maximum de l'atteindre depuis la source en *maxpas* arcs, ainsi que le nombre d'arcs nécessaires (les deux vecteurs *proba* et *pas*).

   L'algorithme utilisera (en plus des routines et types habituels donnés en annexe) la structure d'ensemble et les routines suivantes :

types
`t_ouvert` */* ensemble de couples (t_ listsom, reel) = (sommet, coût) */*

Routines :
- `ensemble_vide () : t_ouvert` Crée et renvoie un nouvel ensemble
- `est_vide (t_ouvert` *o*`) : booleen` Renvoie vrai si *o* est vide
- `maj (t_listsom` *ps*`, reel` *cout*`, t_ouvert` *o*`)` Ajoute le sommet *ps* avec la valeur *cout* dans *o* s'il n'est pas déjà présent, sinon met à jour *ps* dans *o* avec la valeur *cout*.
- `supp_min (t_ouvert` *o*`) : t_listsom` Renvoie et supprime le sommet ayant la valeur minimale dans *o*.

- `log (reel` r`) : reel` retourne $-\infty$ si $r = 0$, $\log r$ sinon.

Exercice 3.2 (Les détectives)

Maintenant que l'on connaît les probabilités qu'à Mister X d'atteindre chaque case et le nombre de tours nécessaires, le joueur détective doit savoir quelles cases il pourra atteindre à moindre coût à chaque tour de jeu.

Nous allons donc écrire un algorithme de recherche de plus courts chemins légèrement différent de ce que l'on fait habituellement : il remplira une matrice indiquant à chaque ligne i les coûts des plus courts chemins de la source à tous les sommets **en exactement** $i-1$ **arcs** (la première ligne représentant l'état initial). L'algorithme sera ici un Bellman-Ford modifié.

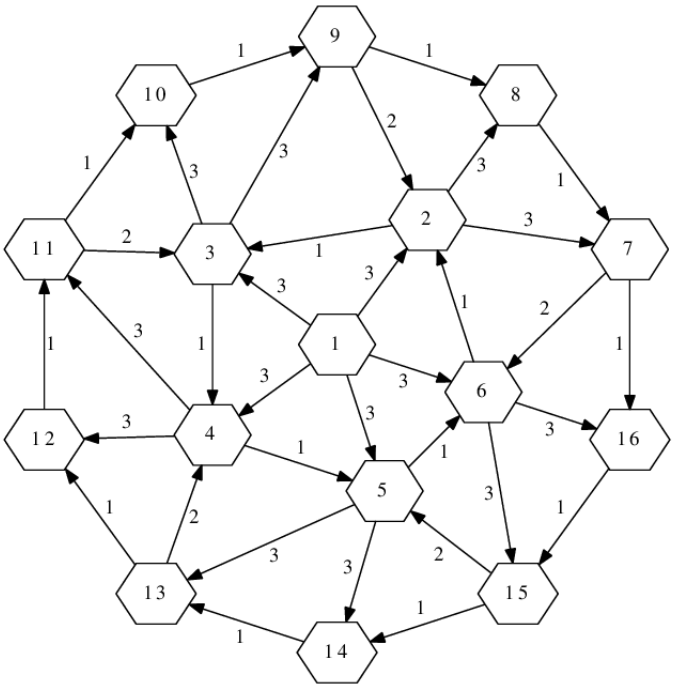


FIGURE 6 – Graphe du détective (avec les coûts des déplacements)

Par exemple, l'exécution de l'algorithme avec le graphe de la figure 6, à partir du sommet 1 et pour 5 tours donnera la matrice suivante :

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	
init.	0.	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	1
tour 1	∞	3.	3.	3.	3.	3.	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	2
tour 2	∞	4.	4.	4.	4.	4.	6.	6.	6.	6.	6.	6.	6.	6.	6.	6.	3
tour 3	∞	5.	5.	5.	5.	5.	7.	7.	7.	7.	7.	7.	7.	7.	7.	7.	4
tour 4	∞	6.	6.	6.	6.	6.	8.	8.	8.	8.	8.	8.	8.	8.	8.	8.	5
tour 5	∞	7.	7.	7.	7.	7.	9.	9.	9.	9.	9.	9.	9.	9.	9.	9.	6

1. Rappeler brièvement le principe de base (sans optimisation) de l'algorithme de Bellman-Ford (vu en td).

2. Dans ce cas particulier (matrice de distances en nombres d'arcs exacts), comment optimiser l'algorithme ?

3. Soit l'arc (x, y) de coût $cout(x, y)$, donner le code du relâchement de l'arc (x, y) au tour $i-1$ permettant de mettre à jour $dist[i, y]$ (représentant le coût du plus court chemin jusqu'à y en exactement $i-1$ arcs).

4. Écrire l'algorithme (avec ou sans optimisation) qui à partir du graphe G, du sommet source scr, du nombre maximum d'arcs $nbtours$ remplit la matrice $dist$ telle que $\forall i,\ 1 \leq i \leq nbtours + 1$, $dist[i, s]$ représente la distance minimum pour atteindre s depuis la source en exactement $i-1$ arcs.