Licence d'informatique – Algorithmique (LU3IN003)
Fascicule de TD (deuxième partie)

Année 2023–2024

Manuel Amoussou
François Clément
Fanny Pascual
Olivier Spanjaard

Algorithmes de Dijkstra et Prim

Exercice 12 – Parcours en largeur et algorithme de Dijkstra

Q 12.1 Dérouler l'algorithme de Dijkstra pour déterminer l'arborescence des plus courts chemins depuis le sommet 1 dans le graphe $G = (S, A, c)$ orienté de la figure 1 (on représentera l'arborescence partielle des plus courts chemins à chaque étape de l'algorithme). En cas d'égalité, on examinera en priorité le sommet de plus petit indice.

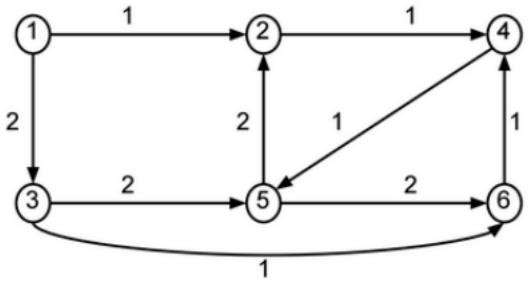


FIG. 1– Un graphe orienté valué.

On définit un nouveau graphe $G' = (S \cup S', A'_1 \cup A'_2, c')$ à partir de G, où :

- S' comporte un sommet x_a pour chaque arc $a = (x, y)$ de A pour lequel $c(a) = 2$,
- $A'_1 = \{a \in A : c(a) = 1\}$,
- A'_2 comporte deux arcs (x, x_a) et (x_a, y) pour chaque arc $a = (x, y)$ de A pour lequel $c(a) = 2$,
- c' est une fonction de coût unitaire (qui associe un coût de 1 à chaque arc de $A'_1 \cup A'_2$).

Q 12.2 Représenter G' ainsi que l'arborescence couvrante associée à un parcours en largeur de G' depuis le sommet 1.
A quoi correspond cette arborescence si on la réinterprète dans G ? (on ne demande pas ici de justification)

Q 12.3 On suppose ici que le graphe est représenté sous forme de tableau de listes de successeurs. Pour un graphe G avec des coûts 1 et 2, quelle est la complexité de l'algorithme consistant à construire un nouveau graphe G' à partir de G puis à réaliser un parcours en largeur de G' ? Rappeler la complexité de l'algorithme de Dijkstra. Conclusion ?

Exercice 13 – Algorithme de Dijkstra et analyse de sensibilité

On considère le graphe valué G de la figure 2.

Q 13.1 Appliquer l'algorithme de Dijkstra pour déterminer l'arborescence des chemins de coût minimum à partir du sommet a dans G.

Q 13.2 On suppose maintenant que le coût de l'arc (d, e) est égal à $3 - \epsilon$ où ϵ est un paramètre réel positif. Pour quelles valeurs de ϵ l'arborescence des chemins de coût minimum est-elle inchangée ?

Q 13.3 En supposant toujours que le coût de l'arc (d, e) est égal à $3 - \epsilon$, tracer la courbe donnant la valeur du plus court chemin de a à f en fonction de ϵ, pour $\epsilon \geq 0$.

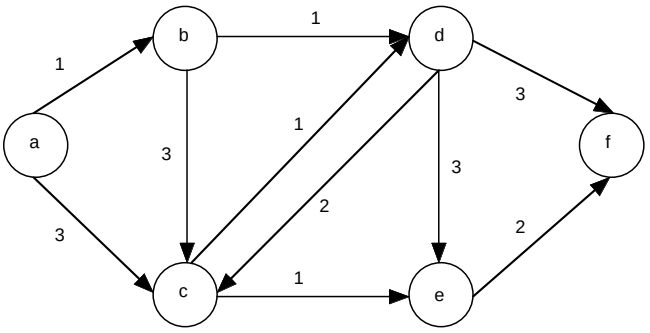


FIGURE 2 – Un graphe valué

Exercice 14 – Algorithme de Prim

Appliquer l'algorithme de Prim pour obtenir un arbre couvrant de poids minimum pour le graphe $G = (S, A, c)$ non orienté valué connexe représenté dans la figure 3.

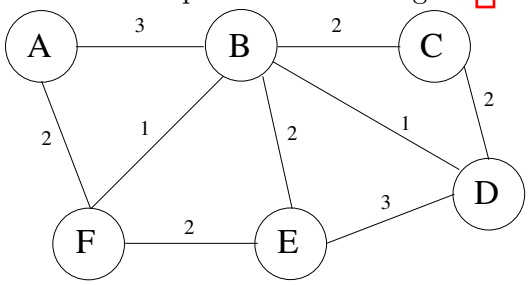


FIG. 3– Un graphe connexe valué.

Exercice 15 – Arbre couvrant et arborescence de chemins

Soit $G = (S, A, v)$ un graphe non orienté valué connexe, tel que pour tout $e \in A, v(e) \geq 0$. On suppose que l'on a construit un arbre de poids minimum de G, ainsi qu'une arborescence des plus courts chemins d'un sommet $s \in S$ à tous les autres sommets du graphe. On suppose maintenant que le coût de chaque arête augmente de 1 : les nouveaux coûts sont $v'(e) = v(e) + 1$.

Q 15.1 Est-ce que l'arbre couvrant de poids minimum change ? Donner un exemple dans lequel il change, ou bien prouver qu'il ne peut pas changer.

Q 15.2 Est-ce que l'arborescence des plus courts chemins change ? Donner un exemple ou elle change, ou bien prouver que cela n'est jamais le cas.

Exercice 16 – Vrai ou faux ?

Indiquer si chacune des propositions suivantes est vraie (dans ce cas là, le prouver), ou fausse (donner un contre-exemple). On supposera que $G = (S, A, v)$ est un graphe non-orienté connexe, avec $n = |S|$ et $m = |A|$.

a) Si le graphe G a plus de $n - 1$ arêtes, et s'il y a une seule arête de coût maximum, alors cette arête ne peut pas faire partie d'un arbre couvrant de poids minimum.

b) Si G contient un cycle avec une unique arête e de coût maximum, alors e ne peut pas faire partie d'un arbre couvrant de poids minimum.

c) Soit $e \in A$ une arête de coût minimum. Alors e appartient à un arbre couvrant de poids minimum de G.

d) Si l'arête de coût minimum de G est unique, alors elle fait nécessairement partie de tout arbre couvrant de poids minimum.

e) Si G contient un cycle avec une unique arête e de coût minimum, alors e fait partie de tout arbre couvrant de poids minimum.

f) La plus courte chaîne entre deux sommets fait nécessairement partie d'un arbre couvrant de poids minimum.

g) L'algorithme de Prim est valide même quand les coûts des arêtes sont négatifs.

Exercice 17 – Arbre de niveau de congestion minimal

Soit $G = (S, A)$ un graphe non orienté connexe valué à n sommets possédant m arêtes et soit $c(a)$ le poids de l'arête a.
Etant donné $T = (S, A_T)$ un arbre couvrant de G, le *niveau de congestion* de T est défini par le poids de l'arête de poids maximal dans A_T.
On s'intéresse à la recherche d'un *arbre couvrant de niveau de congestion minimal* dans G.

Q 17.1 L'arbre couvrant défini par les arêtes en gras pour le graphe de la Figure 6 est-il un arbre couvrant de niveau de congestion minimal ?

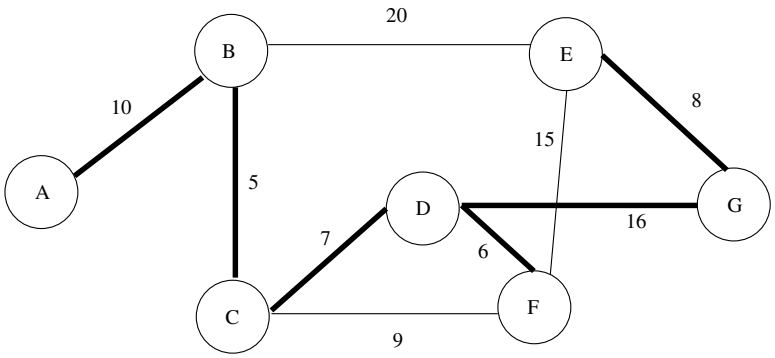


FIGURE 6 – Graphe non orienté valué et arbre couvrant

Q 17.2 Calculer un arbre couvrant de poids minimum du graphe représenté à la Figure 7. Vous préciserez le sous-graphe partiel obtenu à chaque itération.

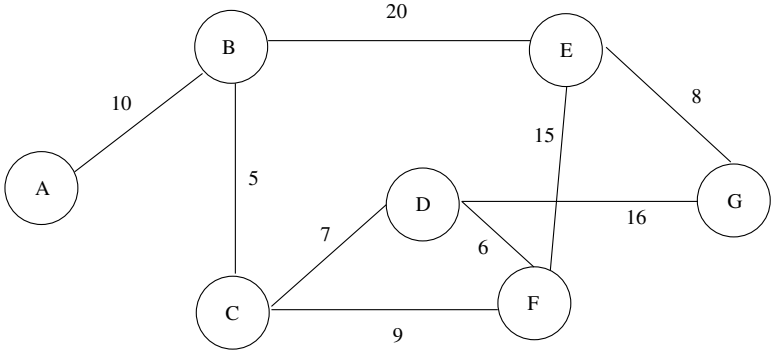


FIGURE 7 – Un graphe non orienté valué

Q 17.3 Dans le cas général, tout arbre couvrant de niveau de congestion minimal est-il un arbre couvrant de poids minimum ? Si oui, fournir une preuve. Si non, donner un contre-exemple.

Q 17.4 Réciproquement, tout arbre couvrant de poids minimum est-il un arbre couvrant de niveau de congestion minimal ? Si oui, fournir une preuve. Si non, donner un contre-exemple.

Q 17.5 En déduire un algorithme pour calculer un arbre couvrant de niveau de congestion minimal. Préciser sa complexité temporelle.

Algorithmique – LU3IN003

Algorithmes gloutons

Conception et analyse d'algorithmes gloutons

Exercice 1 – Algorithme de Kruskal

Soit $G = (S, A)$ un graphe connexe. Soit T l'ensemble des arêtes retournées par l'algorithme de Kruskal ayant comme entrée le graphe G. Le but de cet exercice est de montrer que T est un arbre couvrant de coût minimum.

Q 1.1 Montrer que T est un arbre couvrant. On pourra pour cela montrer que l'ensemble des arêtes T est une forêt, est couvrant, et tel qu'il y ait une seule composante connexe.

Soit T^* un arbre couvrant de coût minimum. Si $T = T^*$ alors T est un arbre couvrant de poids minimum. Sinon, soit e l'arête de plus petit coût dans $T^* \backslash T$. Soit $c(e)$ le coût de e.

Q 1.2 Montrer que $T \cup \{e\}$ contient un cycle $\mathcal{C}$ tel que :

a) Chaque arête de $\mathcal{C}$ a un coût inférieur ou égal à $c(e)$.

b) Il existe dans $\mathcal{C}$ une arête f qui n'appartient pas à T^*.

Q 1.3 Soit T_1 l'arbre $T \backslash \{f\} \cup \{e\}$. Montrer que :

— T_1 est un arbre couvrant

— T_1 a plus d'arêtes en commun avec T^* que T n'en a.

— Le coût de T_1 est supérieur ou égal au coût de T.

Q 1.4 Conclure.

Exercice 2 – Deuxième meilleur arbre couvrant de poids minimum

Soient $G = (S, A)$ un graphe connexe non orienté avec $n = |S|$ et $m = |A|$. On suppose que $m \geq n$. On considère une fonction de pondération c définie sur A et on suppose que tous les poids des arêtes sont distincts.

Soit $\mathcal{T}$ l'ensemble de tous les arbres couvrants de G et soit T^* un arbre couvrant minimum de G. On appelle deuxième meilleur arbre couvrant minimum, un arbre couvrant T tel que $c(T) = \min_{T' \in \mathcal{T}, T' \neq T^*} \{c(T')\}$.

Q 2.1 Montrer que l'arbre couvrant de poids minimum est unique.

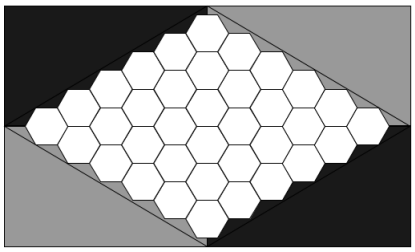
Q 2.2 Montrer que le deuxième meilleur arbre couvrant de poids minimum de G n'est pas forcément unique.

Exercice 3 – Jeu de Hex

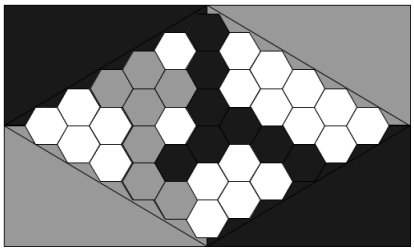
Le jeu de Hex est un jeu pour deux joueurs. Il se joue sur un plateau en forme de losange dont les cases sont hexagonales. Au début de la partie, le plateau est vide (voir figure de gauche). Chaque joueur est représenté par une couleur, noir et gris. Les joueurs possèdent des pions à leur couleur qu'ils disposent tour à tour sur une case de leur choix et un par un. Le plateau se remplit ainsi progressivement. L'objectif d'un joueur, par exemple Noir, est de relier les deux côtés du losange symbolisés par sa couleur (la couleur noire pour Noir). Si la configuration des pions noirs permet la création d'une

chaîne continue de pions noirs reliant un côté noir à l'autre (figure de droite), Noir a gagné et le jeu s'arrête (condition de victoire symétrique pour Gris).

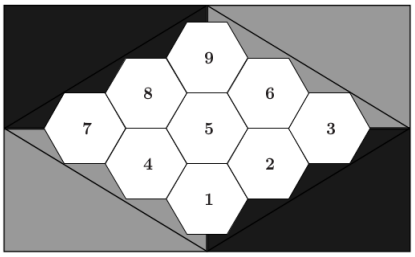
Un plateau vide au départ



Une configuration gagnante pour Noir



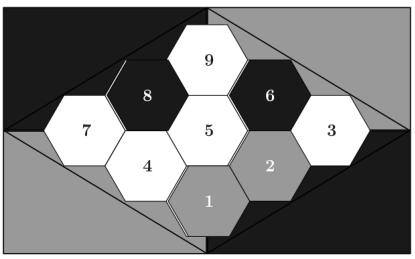
L'objet de cet exercice est de montrer comment la structure de données *union-find* peut être utilisée pour réaliser un programme de jeu de Hex. Plus précisément, on utilise cette structure de données pour détecter si un joueur a gagné à l'issue de son tour. Pour simplifier, on va considérer le jeu de Hex de format réduit 3×3 ci-dessous, dont les cases sont numérotées de 1 à 9. De plus, les quatre bords du plateau sont numérotés 10 (bord noir en haut à gauche), 11 (bord noir en bas à droite), 12 (bord gris en bas à gauche) et 13 (bord gris en haut à droite).



Q 3.1 Afin d'utiliser la structure de données union-find pour détecter si un joueur a gagné, indiquer comment définir l'ensemble des éléments, et à quoi correspondront les classes d'équivalence en cours de partie.

Q 3.2 Quelle est la partition initiale pour le jeu de format 3×3 donné plus haut ? (lorsque le plateau est vide)

Q 3.3 Si on place un pion noir sur la case 5 dans la configuration ci-dessous, quelle(s) opération(s) d'union(s) faut-il réaliser pour mettre à jour la structure de données ?



Q 3.4 Pour le plateau réduit 3×3 donné plus haut, quel test (exploitant la structure de données union-find) faut-il réaliser pour détecter si Noir a gagné ?

Exercice 4 – Le coût de la panne sèche

Le professeur Midas conduit une voiture entre Amsterdam et Lisbonne sur l'européenne E10. Son réservoir d'essence plein lui permet de parcourir n kilomètres, et sa carte lui donne les distances entre les stations-service sur la route. Le professeur souhaite faire le moins d'arrêts possible pendant le voyage.

Q 4.1 Donnez une méthode efficace permettant au professeur Midas de déterminer les stations-service où il peut s'arrêter.

Q 4.2 Montrez que votre stratégie aboutit à une solution optimale.

Exercice 5 – Coloration d'un graphe

Considérons un graphe $G = (S, A)$ non orienté, de degré maximal Δ. Le nombre chromatique de G, noté $\chi(G)$, est le nombre minimal de couleurs nécessaire pour colorier le graphe, c'est-à-dire pour colorier chaque sommet de telle façon que deux sommets distincts et adjacents aient toujours des couleurs différentes.

Q 5.1 Notons $\{1, 2, \ldots, n\}$ les couleurs. Donnez un algorithme glouton, qui parcourt tous les sommets du graphe et attribue une couleur à chaque sommet.

Q 5.2 Montrez que cette approche permet de majorer le nombre chromatique par $\chi(G) \leq \Delta + 1$.

Q 5.3 Donnez un exemple de graphe pour lequel l'algorithme précédent nécessite $\Delta + 1$ couleurs, mais pour lequel 2 couleurs seulement sont nécessaires pour colorier le graphe.

Exercice 6 – Policiers et voleurs

Soit un tableau T indicé de 1 à n, contenant dans chaque case soit le caractère $'P'$ (pour policier), soit $'V'$ (pour voleur). Le problème étudié est le suivant : un policier peut capturer au plus un voleur, un policier ne peut capturer un voleur que si celui-ci se trouve à une distance inférieure ou égale au paramètre k dans le tableau. Une *capture* est un couple (i, j) d'indices tels que $T[i] = 'P'$, $T[j] = 'V'$ et $|i - j| \leq k$ (autrement dit, un policier est en case i, un voleur en case j, et le policier en case i est à une distance qui lui permet de capturer le voleur en case j).

On appelle *appariement* un ensemble de captures où chaque policier et chaque voleur apparaît au plus une fois. Par exemple :

- pour $T = ['P', 'V', 'V', 'P', 'V']$ et $k = 1$, l'appariement $\{(1, 2), (4, 5)\}$ conduit à 2 captures.
- pour $T = ['P', 'V', 'P', 'V', 'V', 'P']$ et $k = 2$, l'appariement $\{(1, 2), (3, 5), (6, 4)\}$ conduit à 3 captures.

On cherche un algorithme qui, pour T et k fixés, permet d'obtenir un appariement avec un nombre maximum c_{max} de captures. Par exemple, dans les deux exemples précédents, on a respectivement $c_{max} = 2$ et $c_{max} = 3$ car les deux appariements sont optimaux.

Q 6.1 Soit n_P le nombre de caractères $'P'$ dans T, et n_V le nombre de caractères $'V'$ dans T. Donnez un exemple de T et k pour lesquels on vérifie à la fois $c_{max} \neq n_P$ et $c_{max} \neq n_V$.

Q 6.2 Considérons l'algorithme naïf qui énumère tous les appariements réalisables et retourne un appariement qui maximise le nombre de captures. Prenons le cas où n est pair, $n_P = n_V = \frac{n}{2}$ et $k = n-1$. Quel est le nombre d'appariements réalisables en fonction de n dans ce cas ? Qu'en déduisez-vous sur la complexité pire cas de cet algorithme ?

On considère maintenant l'algorithme suivant : on parcourt le tableau dans l'ordre des indices croissants, et, pour chaque policier rencontré, ce dernier capture le voleur le plus proche possible n'ayant pas déjà été capturé par un autre policier (si un tel voleur existe).
Précision : Si deux voleurs non encore capturés sont à même plus proche distance, à gauche et à droite du policier, alors le policier capture le voleur à sa gauche.

Q 6.3 À quelle famille d'algorithme cet algorithme appartient-il ?

Q 6.4 Quelle est sa complexité temporelle en fonction de n ? Justifiez brièvement.

Q 6.5 Montrez que cet algorithme n'est pas optimal.

Q 6.6 On appelle Suivant_P(T,i) la fonction retournant le premier indice $x > i$ tel que $T[x] = 'P'$ s'il existe, et $n + 1$ sinon. On appelle Suivant_V(T,j) la fonction retournant le premier indice $y > j$ tel que $T[y] = 'V'$ s'il existe, et $n + 1$ sinon.
Complétez l'algorithme Capture(T,k) pour qu'il retourne une solution optimale au problème.

```
i ← Suivant_P(T,0)
j ← Suivant_V(T,0)
c ← 0
tant que ..... et ..... faire
    si ..... alors
        c ← c + 1
        i ← Suivant_P(T,i)
        j ← Suivant_V(T,j)
    sinon
        si ..... alors
            i ← Suivant_P(T,i)
        sinon
            j ← Suivant_V(T,j)
retourner c
```

Algorithme 1 : Capture(T,k)

Exercice 7 – Playlist

Votre lecteur MP3 a une capacité de M megabits. Vous souhaitez télécharger de la musique sur votre lecteur. Vous sélectionnez n titres $t_1, \cdots, t_n$ de vos artistes préférés. Chaque titre t_i occupe m_i megabits. On suppose que $\sum_{i=1}^{n} m_i > M$: la capacité de votre lecteur ne vous permet pas de télécharger l'ensemble des n titres.

Q 7.1 Vous souhaitez maximiser *le nombre de titres téléchargés* sur votre lecteur MP3. On considère l'algorithme glouton qui sélectionne les titres dans l'ordre de taille m_i croissante jusqu'à ce que la capacité de stockage maximale soit atteinte. Cet algorithme fournit-il une solution optimale ? Le prouver ou fournir un contre-exemple.

Q 7.2 Supposons maintenant que vous souhaitez maximiser *la mémoire occupée dans votre lecteur MP3 par les titres que vous aurez téléchargés*. On considère l'algorithme glouton qui sélectionne les titres dans l'ordre de taille m_i décroissante jusqu'à ce que la capacité de stockage maximale soit atteinte. Cet algorithme fournit-il une solution optimale ? Le prouver ou fournir un contre-exemple.

Exercice 8 – Ordonnancement de tâches sur une machine

On s'intéresse au problème d'ordonnancement de tâches sur une machine qui consiste à définir les dates de début d'exécution d'un ensemble de n tâches J_i, $i = 1, \cdots, n$ caractérisées par une durée opératoire p_i et une date d'échéance d_i à laquelle la tâche devrait idéalement être terminée. La machine ne peut exécuter qu'une tâche à la fois. Si l'on note s_i la date de début d'exécution de la tâche J_i, le retard de J_i sera défini par $l_i = \max\{0, s_i + p_i - d_i\}$. Si $l_i > 0$, la tâche J_i est en retard.

Le but de l'exercice est de déterminer un ordonnancement minimisant la valeur du retard maximal $L = \max_i l_i$ pour l'ensemble des tâches.

Q 8.1 Déterminer la valeur de L pour un ensemble de $n = 6$ tâches pour les données suivantes : $p = (3, 2, 1, 4, 3, 2)$, $d = (6, 8, 9, 9, 14, 15)$ et $s = (5, 1, 0, 11, 8, 3)$.

Q 8.2 Différents algorithmes gloutons peuvent être décrits selon le choix de l'ordre fixé sur les tâches pour leur exécution. Proposer un contre-exemple pour montrer que l'ordre des durées opératoires croissantes n'est pas optimal.

Q 8.3 Les dates d'échéance doivent donc être prises en compte pour définir un ordre sur les tâches. On définit l'urgence de la tâche J_i comme étant la valeur $d_i - p_i$ et on s'intéresse à l'algorithme glouton qui ordonnance les tâches selon l'ordre des urgences croissantes.
Cet algorithme est-il optimal ?

Q 8.4 Montrer que les ordonnancements ne contenant aucune période d'inactivité pour la machine sont dominants.

On considère l'algorithme glouton suivant que l'on appellera EDD :

```
Ré-indexer les tâches de façon à ce que d_1 ≤ d_2 ≤ ··· d_n;
t ← 0;
pour i variant de 1 à n faire
    Affecter la tâche J_i à l'intervalle [t, t + p_i];
    s_i ← t;
    C_i ← s_i + p_i;
    t ← C_i;
pour i variant de 1 à n faire
    Afficher les intervalles [s_i, C_i];
```

Q 8.5 Dérouler l'algorithme EDD sur l'exemple (cf. question 1) et calculer la valeur du retard maximal.

On suppose dans la suite que les tâches sont indexées dans l'ordre de leurs dates d'échéance croissantes. Etant donné un ordonnancement, une *inversion* est une paire de tâches J_i et J_j pour laquelle $d_i < d_j$ et J_j est ordonnancée avant J_i.

Q 8.6 Montrer que la permutation d'une paire de tâches inversées consécutives dans un ordonnancement n'entraîne pas l'augmentation de son retard maximal.

Q 8.7 Montrer qu'il existe un ordonnancement optimal ne possédant ni inversion, ni temps d'inactivité.

Q 8.8 Que peut-on dire de la valeur de retard maximal pour les ordonnancements ne possédant pas

d'inversions ni de temps d'inactivité ?

Q 8.9 Montrer que l'ordonnancement renvoyé par l'algorithme EDD est optimal.

Exercice 9 – Couplage de poids maximum

Soit $G = (S, A)$ un graphe non orienté pondéré avec des poids strictement positifs. On appelle *couplage* de G un ensemble d'arêtes de A qui n'ont aucun sommet en commun. Par exemple, $M_1 = \{\{A, B\}, \{E, F\}\}$ et $M_2 = \{\{B, C\}, \{D, E\}, \{F, G\}\}$ sont des couplages du graphe G_2.

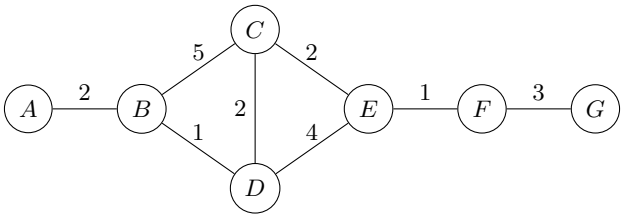
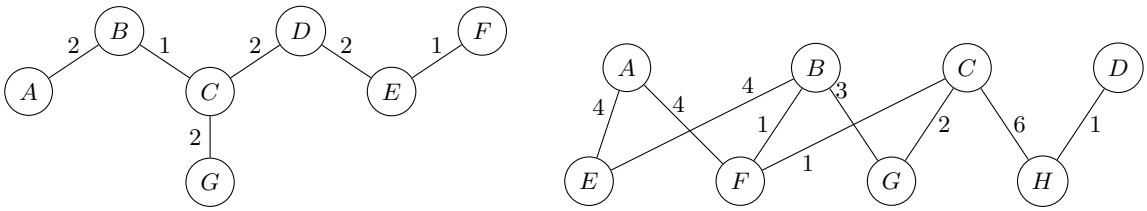


FIGURE 8 – Graphe G_2.

Un couplage M d'un graphe pondéré $G = (S, A)$ est dit *de poids maximum* s'il n'existe pas de couplage dans G dont la somme des poids (ou coûts) des arêtes est supérieure à la somme des poids des arêtes de M. Par exemple, le couplage M_2, de poids $5 + 4 + 3 = 12$, est de poids maximum pour G_2.

Q 9.1 Donner un couplage de poids maximum pour les graphes suivants :



Q 9.2 On considère l'algorithme 2 pour construire un couplage. Quelle est la complexité de cet algorithme ? Justifiez votre réponse.

$M \leftarrow \emptyset$
pour *toutes les arêtes* $\{u, v\} \in A$ *examinées par poids décroissant* **faire**
  si $\nexists\ \{u, v'\} \in M$ *et* $\nexists\ \{u', v\} \in M$ **alors**
    $M \leftarrow M \cup \{\{u, v\}\}$
retourner M

Algorithme 2 : COUPLAGEGLOUTON$(G = (S, A))$

On note M_g le couplage produit par l'algorithme glouton sur un graphe G et $c(M_g) = \sum_{e \in M_g} c(e)$ le poids de ce couplage, où $c(e)$ est le poids de l'arête e. On note M^* un couplage de poids maximum, et $c(M^*)$ son poids. Nous allons maintenant chercher à montrer que $c(M^*) \leq 2c(M_g)$ (autrement dit, le poids du couplage retourné par l'algorithme glouton est au moins 50% du poids maximum).

Q 9.3 Donnez un exemple de graphe et de couplage M_g pour lequel cette borne est effectivement atteinte, c'est-à-dire $c(M^*) = 2c(M_g)$.

Q 9.4 Soit $e \in M^*$ et $M_e \subseteq M_g$ l'ensemble des arêtes de M_g ayant au moins un sommet en commun avec e.

a) Montrer qu'il existe au moins une arête $f_e \in M_e$ telle que $c(f_e) \geq c(e)$.

b) Soit h une fonction qui associe à chaque arête $e \in M^*$ une arête $f_e \in M_e$ telle $c(f_e) \geq c(e)$. Soit $f \in M_g$. Montrer qu'il ne peut pas exister plus de deux arêtes $e \in M^*$ telles que $h(e) = f$.

c) Déduire de a) et b) que $c(M^*) \leq 2c(M_g)$.

Exercice 10 – Codage de Huffman

Q 10.1 Donner le codage de Huffman des caractères A, B, C, D, E pour les fréquences ci-dessous. En chaque branchement de l'arbre, on assignera 0 (resp. 1) à la branche vers le fils de plus petite (resp. grande) étiquette.

A	B	C	D	E
9	17	31	25	20

Q 10.2 Coder la chaîne DAC avec le code de la première question.

Q 10.3 Décoder 1011001110001100 avec le code de la première question.

Exercice 11 – Deviner la valeur d'une carte

Le but de cet exercice est d'élaborer une stratégie pour minimiser l'espérance du nombre de questions posées dans le jeu suivant. On dispose d'un paquet de 45 cartes composées d'une carte de valeur 1, de deux cartes de valeur 2, de trois cartes de valeur 3, et ainsi de suite jusqu'à neuf cartes de valeur 9. Quelqu'un tire une carte au hasard dans le mélangé, et on cherche à identifier la valeur de la carte en posant des questions dont la réponse est oui ou non.

Q 11.1 Proposer une méthode optimale, et indiquer l'espérance du nombre de questions posées.

Algorithmique – LU3IN003

Programmation dynamique

1. Programmation dynamique

Exercice 1 – Formation de binômes et monômes de projet

Une classe est constituée de $n \geq 1$ élèves. Les élèves doivent réaliser un projet, et pour cela chaque élève doit soit rester seul, soit se mettre en binôme avec un autre élève de la classe. Chaque élève est dans exactement un groupe (monôme ou binôme). On cherche à déterminer le nombre de configurations possibles, i.e. le nombre de façons de former des groupes différents en fonction de n.

Par exemple, si $n=3$ (il y a 3 élèves, que l'on appellera 1, 2, et 3), il y a 4 configurations possibles. En effet, soit chacun est seul $(\{1\},\{2\},\{3\})$; soit les élèves 1 et 2 sont en binôme et l'élève 3 est seul $(\{1,2\},\{3\})$; soit 2 et 3 sont en binôme et 1 est seul $(\{1\},\{2,3\})$; ou bien 1 et 3 sont en binôme et 2 est seul $(\{1,3\},\{2\})$. On note $nb(n)$ le nombre de configurations possibles pour une classe à n élèves. Ainsi, $nb(3)=4$.

Q 1.1 Indiquer les valeurs de $nb(1)$ et de $nb(2)$.

Q 1.2 On numérote les élèves de 1 à n. Montrer que, si $n \geq 3$, il y a $nb(n-1)$ configurations dans lesquelles l'élève n est en monôme, et $(n-1) \times nb(n-2)$ configurations dans lesquelles l'élève n est en binôme.

Q 1.3 En déduire une formule de récurrence donnant $nb(n)$ en fonction de valeurs $nb(i)$ avec $i < n$.

Q 1.4 Écrire un algorithme de programmation dynamique prenant en argument un entier $n \geq 1$ et calculant $nb(n)$. Quelle est la complexité de cet algorithme ?

Exercice 2 – Découpe d'une corde

Le magasin "A la vieille campeuse" achète des cordes d'escalade de longueur n et les découpe (soigneusement) en cordes plus petites pour les vendre à ses clients. On souhaite déterminer un découpage optimal pour maximiser le revenu, sachant que les prix de ventes p_i d'une corde de i mètres sont donnés. Par exemple, supposons qu'on dispose d'une corde de $n = 10$ mètres, avec les prix de ventes indiqués dans le tableau suivant :

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	26

Q 2.1 On suppose que chaque découpe est gratuite. On définit la *densité* d'une corde de longueur i comme étant le rapport p_i/i, c'est-à-dire son prix au mètre. Une stratégie gloutonne naturelle consisterait à découper une première corde de longueur i dont la densité soit maximale. On continuerait ensuite en appliquant la même stratégie sur la portion de corde restante de longueur $n-i$. Montrer que cette stratégie ne conduit pas toujours à un découpage optimal.

Q 2.2 On cherche à établir un algorithme de programmation dynamique pour déterminer le revenu de ventes maximal que l'on peut obtenir pour une corde, en supposant toujours que chaque découpe est gratuite.

a. Quelle est la sous-structure optimale dont on a besoin pour résoudre ce problème ?

b. Caractériser (par une équation) cette sous-structure optimale.

c. En déduire le revenu de ventes maximal qu'on peut tirer d'une corde.

d. Ecrire un algorithme de programmation dynamique pour calculer ce revenu.

e. Quelle est la complexité de cet algorithme ?

f. Appliquer l'algorithme à l'exemple.

Q 2.3 Modifier l'algorithme précédent afin qu'il renvoie également la découpe optimale. Appliquer à l'exemple.

Exercice 3 – Location de canoës

Dans cet exercice, on considère un cours d'eau le long duquel sont installés n sites de location de canoës, indicés de 1 à n. En chaque site i, on peut louer un canoë, et le rendre en n'importe quel site $j > i$ en aval. Le coût de location d'un canoë pour se rendre de i à j est noté c_{ij}. On suppose qu'on connaît ce coût pour tout $1 \leq i < j \leq n$. On suppose de plus que $c_{ii} = 0$, et qu'on ne peut remonter le cours d'eau ($c_{ij} = +\infty$ pour $i > j$). Le coût d'une séquence d'arrêts $(i_1, i_2, \ldots, i_k)$ (pour $i_1 < i_2 < \ldots < i_k$ et $k \leq n$) est $c_{i_1 i_2} + c_{i_2 i_3} + \ldots + c_{i_{k-1} i_k}$. Par exemple, pour les données de la table 1 (où $n = 8$), le coût de la séquence $(1, 5, 7, 8)$ est $c_{1,5} + c_{5,7} + c_{7,8} = 5 + 8 + 1 = 14$. On cherche à déterminer le coût minimum d'une séquence pour aller de 1 à n.

	2	**3**	**4**	**5**	**6**	**7**	**8**
1	4	4	7	5	8	11	10
	2	4	2	3	2	5	9
		3	3	1	4	8	12
			4	4	3	2	11
				5	4	8	6
					6	3	7
						7	1

TABLE 1 – Tableau des coûts.

Le but de cet exercice est de concevoir un algorithme de programmation dynamique pour résoudre ce problème. On note $opt(i)$ le coût minimum d'une séquence de i à n (avec $1 \leq i \leq n$).

Q 3.1 Donner la valeur du cas de base $opt(n)$, ainsi que la formule de récurrence permettant de déterminer $opt(i)$ (pour $i \in \{1, \ldots, n-1\}$).

Q 3.2 On suppose les coûts c_{ij} stockés en machine sous la forme d'une matrice c de taille $n \times n$ ($c_{ij} = c[i][j]$). Ecrire un algorithme de programmation dynamique permettant de déterminer le coût minimal de location de canoë pour aller du site 1 au site n. Cet algorithme pourra utiliser un tableau opt de taille n, où la valeur $opt(i)$ sera stockée dans la cellule $opt[i]$.
Indiquer la complexité de votre algorithme.

Q 3.3 Soit $s(i)$ l'arrêt suivant i dans une séquence de coût minimum de i à n. Les valeurs $s(i)$ seront stockées sous la forme d'un tableau $s[1 \ldots n]$. Modifier l'algorithme afin que soient calculées également ces valeurs $s(i)$.

L'objet des questions 4 et 5 est d'appliquer l'algorithme à l'instance numérique de la table 1.

Q 3.4 Appliquer l'algorithme pour déterminer $opt(i)$ et $s(i)$ pour $i \in \{1, \ldots, n\}$.

Q 3.5 En déduire une séquence de coût minimum pour aller de 1 à 8.

Exercice 4 – Sous-séquence de plus grande somme

On dispose d'un tableau d'entiers *relatifs* de taille n. On cherche à déterminer la suite d'entrées consécutives du tableau dont la somme est maximale. Par exemple, pour le tableau $T = [5, 15, -30, 10, -5, 40, 10]$, la somme maximale est 55 (somme des éléments $[10, -5, 40, 10]$).

Q 4.1 Dans un premier temps, on s'intéresse uniquement à la valeur de la somme de cette sous-séquence.

a. Quelle est la sous-structure optimale dont on a besoin pour résoudre ce problème ?

b. Caractériser (par une équation) cette sous-structure optimale.

c. En déduire la valeur de la somme maximale.

d. Ecrire un algorithme de programmation dynamique pour calculer cette somme.

e. Quelle est la complexité de cet algorithme ?

Q 4.2 Modifier l'algorithme pour qu'il renvoie également les indices de début et de fin de la somme. Sa complexité est-elle modifiée ?

Exercice 5 – Parenthésage

Votre journal préféré a commencé à publier des petites énigmes de la forme :

Parenthéser $7 + 1 \times 6$ de façon à maximiser le résultat.	Parenthéser $0.1 \times 0.1 + 0.1$ de façon à maximiser le résultat.
Mauvaise réponse : $7 + (1 \times 6) = 7 + 6 = 13$	Mauvaise réponse : $0.1 \times (0.1 + 0.1) = 0.1 \times 0.2 = 0.02$
Bonne réponse : $(7 + 1) \times 6 = 8 \times 6 = 48$	Bonne réponse : $(0.1 \times 0.1) + 0.1 = 0.01 + 0.1 = 0.11$

Ces énigmes sont assez faciles à résoudre à la main, mais votre journal en publie des bien plus difficiles, dont les solutions peuvent bien sûr contenir des parenthèses imbriquées, comme par exemple : $((3 \times 2) \times (0.5 + 2)) \times (3 + 1)$. Pour ne pas vous ennuyer à résoudre ces énigmes, mais tout de même impressionner vos ami(e)s, vous décidez d'implémenter un algorithme pour résoudre ces énigmes. L'entrée de l'algorithme est une liste $x_1, o_1, x_2, o_2, \ldots, x_{n-1}, o_{n-1}, x_n$ de n nombres réels *positifs* $x_1, \ldots, x_n$ et de $n-1$ opérateurs $o_1, \ldots, o_{n-1}$. Chaque opérateur o_i est soit une addition ($+$) soit une multiplication ($\times$).

Le but de cet exercice est de concevoir un algorithme de programmation dynamique retournant une manière de placer des parenthèses dans l'expression donnée de façon à maximiser la valeur du résultat.

On notera $P[i, j]$ la valeur maximale, après parenthésage, de l'expression $x_i, o_i, \ldots, o_{j-1}, x_j$.

Q 5.1 Donner les valeurs de i et j pour lesquelles $P[i, j]$ désigne la valeur maximale, après parenthésage, de l'expression $x_1, o_1, x_2, o_2, \ldots, x_{n-1}, o_{n-1}, x_n$.

Q 5.2 Un opérateur est dit *central* s'il ne se trouve pas entre deux parenthèses qui se correspondent. Par exemple, dans l'expression $(1 + 2) \times (4 + 8)$, l'opérateur $\times$ est un opérateur central mais pas les opérateurs $+$. On admet qu'il existe pour toute expression un parenthésage la maximisant et ayant un seul opérateur central. Soit o_k l'opérateur central de l'expression $x_i, o_i, \ldots, o_{j-1}, x_j$ après un parenthésage optimal : on a alors des parenthèses entre x_i et x_k puis des parenthèses entre x_{k+1} et x_n. Expliquez pourquoi on a :

$$P[i, j] = P[i, k] o_k P[k + 1, j].$$

Q 5.3 Combien d'opérateurs faut-il examiner afin de trouver l'opérateur central du parenthésage optimal de l'expression $x_i, o_i, \ldots, o_{j-1}, x_j$? En déduire une relation de récurrence pour $P[i, j]$.

Q 5.4 Écrire le programme dynamique associé à cette relation de récurrence. On s'intéressera uniquement à obtenir la valeur maximale de l'expression après parenthésage (et non le parenthésage proprement dit). Vous pouvez dans votre algorithme écrire directement les nombres sous leur forme x_i et les opérateurs sous leur forme o_k (inutile de transformer un caractère en un nombre ou en un opérateur).
Quelle est la complexité de votre algorithme ?

Q 5.5 Comment peut-on modifier le programme écrit à la question précédente de façon à obtenir le parenthésage optimal (on ne demande pas de réécrire le programme, mais d'expliquer en quelques phrases comment obtenir ce parenthésage optimal).

Q 5.6 Ce programme dynamique retourne-t-il toujours une solution optimale si les nombres x_i peuvent prendre une valeur négative ?

Exercice 6 – Jeu à deux joueurs

Considérons une rangée de n pièces de valeurs $v_1, \ldots, v_n$ et le jeu suivant à deux joueurs : à chaque tour, le joueur peut prendre soit la pièce la plus à gauche, soit la pièce la plus à droite. Le jeu s'arrête quand il n'y a plus de pièce. On cherche à calculer le gain optimal du joueur qui commence ; on considérera que les deux joueurs adoptent une stratégie qui maximise leur profit final. Par exemple, pour 5, 3, 7, 11 le gain optimal pour celui qui commence est 16(11 + 5) ; pour 8, 15, 3, 7 ce sera 22(7 + 15).

Q 6.1 Indiquer une rangée de pièces pour laquelle il n'est pas optimal pour le premier joueur de commencer par prendre la pièce de plus grande valeur. Autrement dit, cette stratégie gloutonne n'est pas optimale.

Q 6.2 Donner un algorithme en $O(n^2)$ pour calculer une stratégie optimale pour le premier joueur. Etant donnée la rangée initiale, l'algorithme calculera en $O(n^2)$ les informations nécessaires, et ensuite le premier joueur pourra déterminer en $O(1)$ le choix optimal à chaque tour en consultant les données précalculées.

Exercice 7 – Playoffs NBA

Les playoffs NBA (appelés séries éliminatoires au Canada) sont les séries éliminatoires de la National Basketball Association (NBA). Une série éliminatoire est une série de matchs deux équipes, au terme de laquelle l'une des deux équipes est éliminées. Chaque série se joue au meilleur d'une série de sept matchs, c'est-à-dire que la première équipe qui gagne 4 matchs remporte la série.

Plus généralement, considérons deux équipes A et B qui s'affrontent dans une série de matchs jusqu'à ce que l'équipe A remporte i victoires ou que l'équipe B remporte j victoires. Supposons que la probabilité de victoire de A sur un match (contre l'équipe B) est p, et que la probabilité de défaite de A est donc $q = 1 - p$ (il n'y a pas de match nul). Soit $P(i, j)$ la probabilité que A remporte la série (i.e. A a remporté i victoires et B moins de j victoires.

Q 7.1 Etablir une relation de récurrence pour $P(i, j)$ qui puisse être utilisée dans un algorithme de programmation dynamique.

Q 7.2 Déterminer la probabilité que l'équipe A remporte une série au meilleur des sept matchs si

$p = 0.4$ (ceci signifie que A gagne si elle est la première équipe à gagner 4 matchs).

Q 7.3 Ecrire un algorithme de programmation dynamique pour calculer la probabilité de gagner une série au meilleur des n matchs (n impair). Quel est sa complexité ?

Exercice 8 – Déplacements d'un robot

On considère un graphe $G = (V, E)$ non-orienté, avec $V = \{0, \ldots, n-1\}$ et $|E| = m$. Un robot est positionné initialement (au pas de temps 0) sur le sommet 0. Puis, entre chaque pas de temps k et $k+1$ ($k \in \{0, \ldots, T-1\}$), il se déplace aléatoirement dans G en suivant la règle suivante : depuis le sommet $v \in V$ sur lequel il est positionné au pas de temps k, il se déplace vers un voisin v' de v, selon une loi de probabilité uniforme (il sera donc positionné sur le sommet v' au pas de temps $k+1$). Par exemple, sur le graphe de la figure 9, si le robot est positionné au sommet 4 au pas de temps k, la probabilité que le robot soit positionné au sommet 1 (resp. au sommet 3) au pas de temps $k+1$ est 1/2 (resp. 1/2). Précisons que rien n'empêche le robot de se replacer sur un sommet déjà visité (le robot se déplace toutefois toujours entre chaque pas de temps).

On vise à concevoir une procédure de programmation dynamique dont le but final est, pour chaque sommet $v \in V$, de calculer la probabilité que le robot soit positionné en v au pas de temps T.

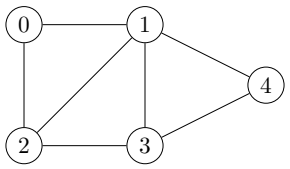


FIGURE 9 – Un exemple de graphe non-orienté.

Q 8.1 Soit $p_k(v)$ la probabilité que le robot soit positionné au sommet v au pas de temps k. A la manière de la relation de récurrence à l'origine de l'algorithme de Bellman-Ford, donner la relation de récurrence permettant de déterminer $p_k(v)$ connaissant les probabilités $p_{k-1}(u)$ ($u \in \{0, \ldots, n-1\}$), en notant $d(u)$ le degré d'un sommet $u \in V$. Comment initialiser la récurrence ?

Q 8.2 Analyser la complexité de l'algorithme qui met en œuvre cette récurrence (on ne demande pas d'écrire l'algorithme). On précisera la structure de données utilisée pour obtenir cette complexité.

Q 8.3 Appliquer cet algorithme au graphe de la figure 9 pour déterminer les différentes valeurs $p_k(v)$ ($v \in \{0, \ldots, 4\}$) pour $k \in \{0, 1, 2\}$.

Exercice 9 – Plus courts chemins entre toute paire de sommets

On vise dans cet exercice à concevoir un algorithme de programmation dynamique pour calculer les valeurs $d(i, j)$ des plus courts chemins pour toutes les paires de sommets i, j, dans un graphe orienté où les sommets sont indicés de 1 à n et où chaque arc (i, j) est muni d'une longueur ℓ_{ij}. Soit $d(i, j, k)$ la valeur d'un plus court chemin de i à j dont tous les sommets intermédiaires sont dans $\{1, \ldots, k\}$.

Q 9.1 On cherche à établir un algorithme programmation dynamique pour ce problème.

a. Indiquer la valeur de $d(i, j, k)$ pour $k = 0$.

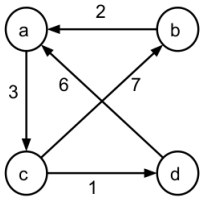
b. Donner l'équation de récurrence caractérisant $d(i, j, k)$.

c. En déduire la longueur $d(i, j)$ d'un plus court chemin entre i et j.

d. Ecrire un algorithme de programmation dynamique pour calculer les longueurs des plus courts chemins pour tous les couples de sommets.

e. Quelle est la complexité de cet algorithme ?

Q 9.2 Appliquer l'algorithme au graphe ci-dessous.



Exercice 10 – Sac-à-dos, avec répétitions

Lors du cambriolage d'une bijouterie, le voleur s'aperçoit qu'il ne peut pas tout emporter dans son sac-à-dos... Son sac peut supporter, au maximum, P kilos de marchandises (on supposera P entier). Or, dans cette bijouterie, se trouvent n objets différents, chacun en quantité illimitée. Chaque objet x_i a un poids p_i et une valeur v_i. Quelle combinaison d'objets, transportable dans le sac, sera la plus rentable pour le voleur ?
Exemple : pour $P = 10$ et les objets ci-dessous, le meilleur choix est de prendre un x_1 et deux x_4, ce qui fait un sac à 48 €.

objet	poids	valeur
x_1	6	30 €
x_2	3	14 €
x_3	4	16 €
x_4	2	9 €

Q 10.1 Dans un premier temps, on s'intéresse uniquement à la valeur maximale que pourra emporter le voleur.

a. Quelle est la sous-structure optimale dont on a besoin pour résoudre ce problème ?

b. Caractériser (par une équation) cette sous-structure optimale.

c. En déduire le montant maximal des objets volés.

d. Ecrire un algorithme de programmation dynamique pour calculer cette valeur maximale.

e. Quelle est la complexité de cet algorithme ?

Q 10.2 Modifier l'algorithme pour qu'il renvoie également la combinaison d'objets volés. Sa complexité est-elle modifiée ?

Exercice 11 – Voyageur de commerce

Dans le problème du voyageur de commerce, on dispose d'une matrice (ici supposée symétrique) d des distances entre n villes indicées de 1 à n (d_{ij} et d_{ji} représentant la distance entre la ville i et la ville j), et le but est, en partant de 1, de visiter chaque ville exactement une fois et de revenir en 1 en minimisant la distance totale parcourue. Par exemple, dans le graphe de la figure 10, la distance totale parcourue pour la tournée $(1, 2, 3, 4, 5, 1)$ est 15, alors que la distance totale parcourue pour la tournée $(1, 3, 5, 2, 4, 1)$ est seulement 10.

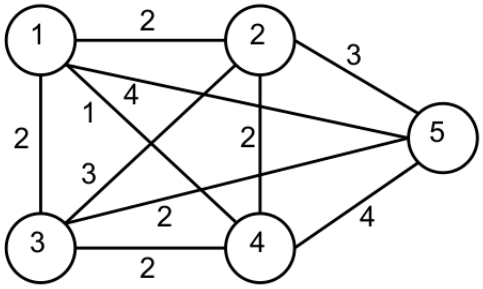


FIGURE 10 – Une instance du problème de voyageur de commerce.

Q 11.1 Donner la matrice d pour le graphe de l'exemple.

Q 11.2 Combien y a-t-il de tournées possibles pour n villes ?

Q 11.3 On cherche à établir un algorithme programmation dynamique pour déterminer la valeur d'une tournée optimale.

a. Quelle est la sous-structure optimale dont on a besoin pour résoudre ce problème ?

b. Caractériser (par une équation) cette sous-structure optimale.

c. En déduire la valeur d'une tournée optimale.

d. Ecrire un algorithme de programmation dynamique pour calculer cette valeur.

e. Quelle est la complexité de cet algorithme ?

Exercice 12 – Robot collecteur de pièces

Des pièces sont placées sur les cases d'un échiquier $\ell \times c$ (ℓ lignes et c colonnes). Il y a au plus une pièce par case. Un robot, positionné sur la case en haut à gauche de l'échiquier, va collecter le plus de pièces possibles et les placer sur la case en bas à droite. A chaque pas, le robot peut se déplacer d'une case vers la droite ou vers le bas. Quand le robot passe sur une case avec une pièce, il prend la pièce. Le but de cet exercice est de concevoir un algorithme pour collecter un nombre maximum de pièces.

Q 12.1 Dans un premier temps, on s'intéresse uniquement au nombre maximum de pièces qu'on peut collecter sans chercher à identifier la trajectoire correspondante.

a. Quelle est la sous-structure optimale dont on a besoin pour résoudre ce problème ?

b. Caractériser (par une équation) cette sous-structure optimale.

c. En déduire le nombre maximum de pièces qu'on peut collecter.

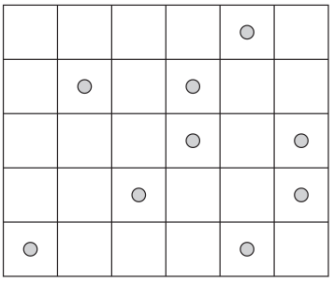
d. Ecrire un algorithme de programmation dynamique pour calculer ce nombre.

e. Quelle est la complexité de cet algorithme ?

Q 12.2 Modifier l'algorithme précédent afin qu'il renvoie également la trajectoire optimale.

Dans les questions qui suivent, on cherche à reformuler et résoudre le problème comme la recherche d'un plus long chemin dans un graphe.

Q 12.3 Pour l'échiquier ci-dessous, tracer un graphe valué avec un sommet source s et un sommet puits t, tel que le problème de collecte de pièces revient à déterminer un plus long chemin de s à t. Plus généralement, indiquer le nombre n de sommets et le nombre m d'arcs en fonction de ℓ et de c.



Q 12.4 En s'inspirant d'un algorithme vu en cours dont on rappelera le nom, donner le *principe* (sans entrer dans les détails) d'un algorithme de calcul d'un plus long chemin dans un graphe sans circuit. Appliquer cet algorithme sur le graphe de la question 1. Quel est le plus long chemin ?

Q 12.5 Quelle est la complexité de l'algorithme précédent en fonction de n et de m ? En déduire la complexité (en fonction de ℓ et de c) de l'algorithme qui, partant d'un échiquier sur lequel des pièces sont positionnées, détermine un itinéraire optimal pour collecter un nombre maximum de pièces.

Exercice 13 – Résolution d'un système d'inéquations

On considère un ensemble $\mathcal{X}$ de n variables réelles x_1, x_2, ..., x_n et m contraintes où chaque contrainte $k \in \{1, \cdots, m\}$ est de la forme suivante :

$$x_{j_k} - x_{i_k} \leq b_k$$

où x_{j_k} et x_{i_k} sont des variables de $\mathcal{X}$ et b_k est un réel.
On cherche à déterminer si ce système de contraintes possède une solution admissible, c'est-à-dire si l'on peut donner des valeurs aux variables de manière à ce que toutes les contraintes soient vérifiées.
Pour résoudre ce problème, on construit un graphe contenant :

- n sommets numérotés de 1 à n (un sommet par variable) ;
- un sommet s ;
- n arcs (s, l) $(l \in \{1, \cdots, n\})$;
- un arc (i_k, j_k) pour chaque contrainte $x_{j_k} - x_{i_k} \leq b_k$ $(k \in \{1, \cdots, m\})$

Il y a donc $n + 1$ sommets et $n + m$ arcs. Les arcs sont valués de la manière suivante : les arcs (s, l) ont une valeur 0, l'arc (i_k, j_k) associé à la contrainte $x_{j_k} - x_{i_k} \leq b_k$ a la valeur b_k.

Q 13.1 Dans cette question, on s'intéresse au système de 4 variables et 5 contraintes suivant :

$$\begin{aligned} x_4 - x_3 &\leq 5 \\ x_3 - x_1 &\leq 8 \\ x_1 - x_4 &\leq -10 \\ x_1 - x_2 &\leq -11 \\ x_2 - x_3 &\leq 2 \end{aligned}$$

a. Représentez le graphe G associé au système ci-dessus.

b. Appliquez un algorithme de recherche de plus courts chemins pour déterminer des plus courts chemin d'origine s aux sommets 1, 2, 3 et 4 dans G.
On justifiera le choix de l'algorithme et on détaillera les étapes du déroulement de l'algorithme (on fournira en particulier les changements dans l'arborescence courante et l'arborescence finale obtenue). Que constatez-vous ?

c. Pouvez-vous en déduire que ce système de contraintes possède une solution admissible ? Si oui, donnez un exemple de valeurs aux variables. Si non, donnez un ensemble de contraintes incompatibles.

Q 13.2 On s'intéresse maintenant à un système quelconque à n variables et m contraintes et à son graphe associé (on ne considère donc plus le système donné dans la question 1).

a. Enoncez une condition nécessaire sur le graphe associé pour que le système possède une solution admissible. Prouvez que cette condition est bien une condition nécessaire.

b. Montrez que cette condition est suffisante.

Exercice 14 – Coloration des sommets d'un graphe

Soit $G = (S, A)$ un graphe non-orienté à n sommets. Une coloration c de G consiste à affecter à chaque sommet s une couleur $c(s)$ de telle sorte que pour chaque arête $\{x, y\}$, les couleurs $c(x)$ et $c(y)$ soient distinctes.

L'exemple de la figure 11 montre une coloration d'un graphe G_1 en 3 couleurs blanc, gris, noir.

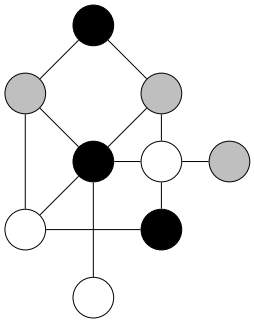


FIGURE 11 – Une coloration d'un graphe G_1.

Soit $\nu : S \mapsto \{1, \ldots, n\}$ une numérotation des sommets de G. On définit le graphe orienté $G(\nu) = (S, A(\nu))$ pour lequel chaque arête $\{x, y\}$ est transformée en l'arc (x, y) si $\nu(x) < \nu(y)$ ou en l'arc (y, x) si $\nu(x) > \nu(y)$. Par abus de langage, dans $G(\nu)$, on identifiera un sommet x et son numéro $\nu(x)$.

Q 14.1 Déterminer le graphe $G_1(\nu_1)$ pour le graphe G_1 et la numérotation ν_1 des sommets de la figure 12 où le numéro $\nu_1(x)$ est inscrit à l'intérieur du cercle associé à x.

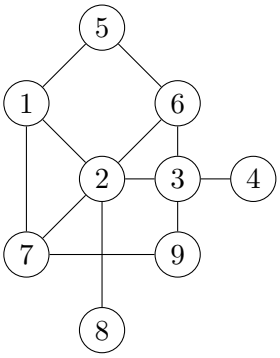


FIGURE 12 – Une numérotation ν_1 de G_1.

Q 14.2 Prouver que pour toute numérotation ν, le graphe $G(\nu)$ est sans circuit.

Q 14.3 Donner un tri topologique de $G(\nu)$, valide pour toute numérotation ν.

Pour chaque sommet x de $G(\nu)$, on note $d_\nu(x)$ la longueur en nombre d'arcs d'un plus long chemin d'extrêmité x dans $G(\nu)$ (c'est-à-dire que $d_\nu(x)$ est la longueur maximale d'un chemin dont le dernier sommet est x). On note également $D(\nu)$ la plus grande valeur des $d_\nu(x)$.

Q 14.4 A la manière de la relation de récurrence à l'origine de l'algorithme de Bellman, donner une relation de récurrence permettant de déterminer $d_\nu(x)$ pour tout x. Comment initialiser la récurrence ?

Q 14.5 A l'aide de la relation de récurrence précédente, indiquer pour chaque sommet x du graphe $G_1(\nu_1)$ (sommet identifié par son numéro $\nu_1(x)$) sa valeur $d_{\nu_1}(x)$. Quelle est la valeur de $D(\nu_1)$?

x	1	2	3	4	5	6	7	8	9
$d_{\nu_1}(x)$									

Q 14.6 En considérant un chemin de $G(\nu)$ de longueur $D(\nu)$, montrer que pour chaque valeur de l dans $\{0, \ldots, D(\nu)\}$, il existe au moins un sommet x tel que $d_\nu(x) = l$.

On note $E_l(\nu)$ l'ensemble des sommets x tel que $d_\nu(x) = l$.

Q 14.7 Déduire de ce qui précède que les ensembles $E_l(\nu)$ ($l \in \{0, \ldots, D(\nu)\}$) forment une partition de S. (On rappelle qu'une partition de S est famille des parties de S non vides, disjoints deux à deux, et dont l'union est égale à S.)

Q 14.8 Soit $l \in \{0, \ldots, D(\nu)\}$. Montrer que si x et y ($x \neq y$) appartiennent à $E_l(\nu)$, alors il ne peut exister ni un arc (x, y) ni un arc (y, x) dans $G(\nu)$.
En déduire que dans le graphe G, on peut colorier, quel que soit l dans $\{0, \ldots, D(\nu)\}$, tous les sommets de $E_l(\nu)$ avec une même couleur.

Q 14.9 Appliquer les résultats des questions précédentes pour déterminer la partition des sommets de G_1 en sous-ensembles de sommets de même couleur, partition obtenue pour la numérotation ν_1.

Exercice 15 – Plus court chemin depuis un sommet donné

On considère le graphe orienté et pondéré G_λ donné sur la figure 13, où λ est un réel positif ou nul.

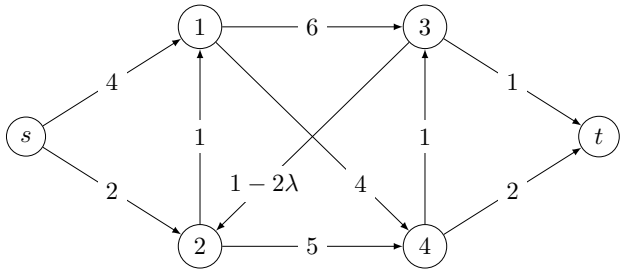


FIGURE 13 – Graphe G_λ

Q 15.1 Indiquer les valeurs du paramètre λ (sous la forme d'un intervalle) pour lesquelles il existe un plus court chemin entre le sommet de départ s et le sommet d'arrivée t. Justifiez votre réponse.

Q 15.2 Pour quelles valeurs de λ peut-on utiliser l'algorithme de Dijkstra pour déterminer un plus court chemin entre s et t ?

Q 15.3 Quel(s) algorithme(s) vu(s) en cours peut-on utiliser pour déterminer un plus court chemin

entre s et t lorsque $\lambda = 1$?

Q 15.4 On suppose maintenant que l'on cherche un chemin de s à t comportant *un nombre minimum d'arcs* (on ne s'intéresse pas au coût du chemin dans cette question). Quel algorithme vu en cours recommandez-vous d'appliquer ?

Q 15.5 On suppose dans cette question que l'on cherche un chemin de s à t de coût minimum *parmi les chemins comportant un nombre minimum d'arcs*. Quel algorithme vu en cours pouvez-vous adapter (et comment) pour déterminer un tel chemin ? Si l'on note p le nombre minimum d'arcs à emprunter pour aller de s à t, quelle est la complexité de l'algorithme que vous proposez ?