Algorithmique (LU3IN003).
L3 Informatique.

Examen du 3 janvier 2023. Durée : 2 heures

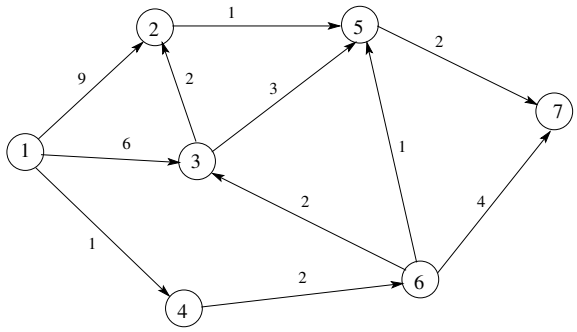
Documents non autorisés. Seule une feuille A4 portant sur les cours et les TD est autorisée.
Téléphones portables éteints et rangés dans vos sacs.

Le barème est indicatif et est susceptible d'être modifié.

Exercice 1 (3 points)

Répondez au questionnaire à choix multiples suivant, en cochant la case correspondant à la bonne réponse pour chaque question. Chaque bonne réponse apportera 0.5 point. Chaque mauvaise réponse coûte une pénalité de 0.25 point, une absence de réponse n'étant pas comptabilisée comme une mauvaise réponse.

Questions	**Réponses**
1. Quelle est la complexité d'un algorithme du type "diviser pour régner" qui divise un problème de taille n en quatre sous-problèmes de taille $n/2$ et dont l'opération de fusion est en $O(n)$?	☐ $\Theta(\log n)$ ☐ $\Theta(n)$ ☐ $\Theta(n \log n)$ ■ $\Theta(n^2)$ ☐ $\Theta(n^3)$
2. Quel algorithme peut-on utiliser pour détecter la présence d'un circuit absorbant dans un graphe ?	☐ parcours en prodondeur ■ algorithme de Bellman-Ford ☐ algorithme de Dijsktra ☐ algorithme de Bellman
3. Quel algorithme n'est pas de type "diviser pour régner" ?	☐ l'algorithme de tri fusion ☐ recherche dichotomique ■ algorithme de Prim ☐ algorithme de Karatsuba
4. Exécuter l'algorithme de Dijkstra sur le graphe de la question 4b, afin de trouver une arborescence des plus courts chemins de racine 1. Dans quel ordre les sommets sont-ils examinés ?	☐ (1,4,3,2,6,5,7) ☐ (1,4,6,3,5,7,2) ■ (1,4,6,5,3,7,2) ☐ (1,4,6,3,2,5,7) ☐ (1,2,3,4,5,6,7)
	suite sur la page suivante...

Questions	Réponses
4b. Représenter sur le graphe *l'arborescence des plus courts chemins* retournée par l'algorithme de Dijkstra (vous pouvez surligner les arcs sélectionnés) : 	
5. Cocher la permutation des sommets qui peut correspondre à un parcours en profondeur de racine 1 dans le graphe représenté dans la question 4b.	☐ (1,2,3,4,5,6,7) ☐ (1,3,5,2,4,7,6) ☐ (1,2,5,3,4,6,7 ■ (1,4,6,7,5,3,2) ☐ (1,7,2,6,4,3,5)

Exercice 2 (3 points)

On considérera dans cet exercice un graphe $G = (S, A)$ orienté et valué. On notera $c(e)$ le coût de l'arc $e \in A$. On supposera que $s \in S$ est une racine et qu'il n'y a pas de circuit absorbant dans G. Soit $\mathcal{A}$ une arborescence des plus courts chemins d'origine s. On notera $d(x)$ le coût du chemin de s à x dans $\mathcal{A}$. Le but de cet exercice est de voir dans quelle mesure des perturbations sur un graphe changent une arborescence des chemins de coût minimum.

Question 1 (1/3) — Supposons que l'on multiplie par 10 le coût de chaque arc. Soit G' le graphe obtenu (i.e. on a $G' = (S, A)$ et pour tout arc $e \in A$, le coût de e dans G' est $10 \times c(e)$). L'arborescence $\mathcal{A}$ est-elle nécessairement une arborescence des plus courts chemins d'origine s dans G' ? Justifiez votre réponse.

Oui.

En effet, le coût d'un chemin dans G' sera égal à 10 fois le coût de ce même chemin dans G. Un plus court chemin dans G restera donc unplus court chemin dans G' (et donc une arborescence des plus court chemins de G est une arborescence des plus court chemins de G').

Question 2 (1/3) — Supposons que l'on retire 1 au coût de chaque arc. Soit G'' le graphe obtenu (i.e. on a $G'' = (S, A)$ et pour tout arc $e \in A$, le coût de e dans G'' est $c(e) - 1$). L'arborescence $\mathcal{A}$ est-elle nécessairement une arborescence des plus courts chemins d'origine s dans G'' ? Justifiez votre réponse.

Non.

Contre-exemple : un graphe à 3 sommets A,B,C,D avec A racine et des arcs (A,B),(B,C),et (C,D) de coût 1, et un arc (A,D) de coût 2. Dans G l'unique chemin de coût min pour aller de A à D est le chemin (A,D), alors que dans G'', l'unique chemin de coût min pour aller de A à D est le chemin (A,B,C,D).

Question 3 (1/3) — Soit a et b deux sommets de S tels que $(a, b) \notin A$. Soit G''' le graphe G dans lequel on ajoute cet arc : $G''' = (S, A \cup \{(a, b)\})$. Pour tout arc $e \in A$, le coût de e dans G''' est $c(e)$, et le coût de l'arc (a, b) est de 5. À quelle condition $\mathcal{A}$ est-elle une arborescence des plus courts chemins d'origine s dans G''' ? Justifiez votre réponse.

$\mathcal{A}$ est-elle une arborescence des plus courts chemins d'origine s dans G''' si et seulement si $d(b) \leq d(a) + 5$, i.e. si cet arc n'est pas incompatible. En effet, comme vu en cours, une arborescence est une arborescence des plus courts chemins si et seulement il n'y a pas d'arc incompatible (et $\mathcal{A}$ étant une arborescence des plus courts chemins avant ajout de (a, b), on en déduit que tous les arcs de $\mathcal{A}$ sont compatibles).

Exercice 3 (7 points)

Une commune possède une parcelle de forêt rectangulaire de $n \times m$ kilomètres (ce qui représente un rectangle de n kilomètres sur m kilomètres). La commune souhaite vendre cette parcelle et en tirer le plus d'argent possible. Pour cela, elle peut diviser son terrain en plusieurs terrains plus petits à l'aide de coupes horizontales et verticales.

Définition 1. *On appelle* coupe horizontale *une séparation d'un terrain T de taille $i \times j$ en deux terrains T_1 de taille $i_1 \times j$ et T_2 de taille $i_2 \times j$ avec $i_1 + i_2 = i$ et i_1 et i_2 des entiers strictement positifs. Une telle coupe sera notée $Coupe\_Horizontale(T, i_1, T_1, T_2)$.*

Définition 2. *On appelle* coupe verticale *une séparation d'un terrain T de taille $i \times j$ en deux terrains T_1 et T_2 de taille $i \times j_1$ et $i \times j_2$ avec $j_1 + j_2 = j$ et j_1 et j_2 des entiers strictement positifs. Une telle coupe sera notée $Coupe\_Verticale(T, j_1, T_1, T_2)$.*

La commune peut effectuer plusieurs coupes à la suite. On appelle *plan de coupe* une liste P de coupes successives et réalisables. Cette liste peut être vide. Certains plans de coupe différents peuvent aboutir à un même ensemble de terrains. On illustre ces définitions par l'exemple suivant.

Exemple 1. *Si la commune dispose d'un terrain T de 4×4 kilomètres carré, elle peut le découper en deux dans le sens vertical à un kilomètre du bord du terrain. Elle obtient alors un terrain T_1*

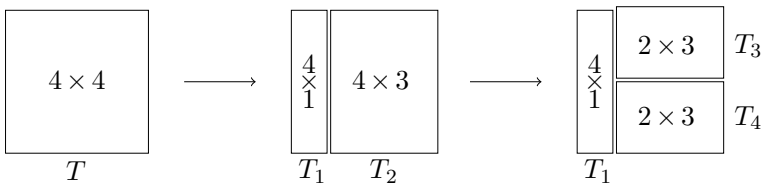


FIGURE 1 – Plan de coupe P

de taille 4×1 *et un autre* T_2 *de taille* 4×3*. Elle peut ensuite découper* T_2 *au milieu dans le sens horizontal : elle aura alors le terrain* T_1 *de taille* 4×1 *et deux terrains* T_3 *et* T_4 *de taille* 2×3*. Ce plan de coupe sera noté* $P = (Coupe\_Verticale(T, 1, T_1, T_2), Coupe\_Horizontale(T_2, 2, T_3, T_4))$*.*

Question 1 (0.5/7) — Indiquer les 5 plans de coupe possibles si la commune dispose d'un terrain T de taille 3×1.

On a 5 plans de coupes :
- $P_0 = ()$
- $P_1 = (Coupe\_Horizontale(T, 1, T_1, T_2))$
- $P_2 = (Coupe\_Horizontale(T, 2, T_1, T_2))$
- $P_3 = (Coupe\_Horizontale(T, 1, T_1, T_2), Coupe\_Horizontale(T_2, 1, T_3, T_4))$
- $P_4 = (Coupe\_Horizontale(T, 2, T_1, T_2), Coupe\_Horizontale(T_1, 1, T_3, T_4))$

Pour chaque terrain T_k de $i \times j$ kilomètres carré, la commune peut vendre T_k au prix $M(i, j)$ où M est une matrice de prix. On définit une fonction *eval* qui prend en paramètres les dimensions d'un terrain T et un plan de coupe P sur T et retourne l'argent obtenu en vendant les terrains obtenus à partir de T en suivant le plan de coupe P. On note ainsi $eval(i, j, P)$ la somme obtenu en vendant les terrains issus des coupes P à partir d'un terrain de taille $i \times j$ (cette fonction fait la somme des prix des terrains obtenus à l'issue du plan P).

Exemple 1. ***(suite)*** *On considère deux plans de coupe :*
- $P_1 = ()$
- $P_2 = (Coupe\_Verticale(T, 1, T_1, T_2), Coupe\_Horizontale(T_2, 2, T_3, T_4))$.

P_1 est un plan sans coupe, et P_2 est le plan de coupe obtenu sur la figure 1.
On considère la matrice de prix suivante :

M	$i = 1$	$i = 2$	$i = 3$	$i = 4$
$j = 1$	1	2	4	5
$j = 2$	2	3	8	9
$j = 3$	4	8	10	11
$j = 4$	5	9	11	13

TABLE 2 – Matrice de prix M

A l'issue de P_1*, la commune a un terrain de dimensions* 4×4 *: on a donc* $eval(4, 4, P_1) =$

$M(4,4) = 13$. *A l'issue de* P_2, *la commune a un terrain de taille* 4×1 *et deux terrains de taille* 2×3, *on a alors* $eval(4, 4, P_2) = M(4,1) + M(2,3) + M(2,3) = 5 + 8 + 8 = 21$.

Le but de cet exercice est de déterminer la somme d'argent maximale que la commune peut obtenir à partir en faisant des coupes de son terrain de taille $n \times m$ (on ne cherche pour l'instant pas à savoir quelles coupes il faut faire pour obtenir cette somme maximale). On va pour cela concevoir un algorithme de programmation dynamique.

On appelle *eval_opt* la fonction prenant en paramètres les dimensions d'un terrain T et retournant la somme d'argent maximale que l'on peut obtenir en suivant un plan de coupe pour le terrain T. Ainsi, $eval\_opt(i, j)$ sera la somme maximale que l'on peut obtenir en coupant un terrain de taille $i \times j$ via une succession de coupes.

Question 2 (0.5/7) — Soit T un terrain de taille $i \times j$ et soit $C = Coupe\_Horizontale(T, i_1, T_1, T_2)$ une coupe de ce terrain. Exprimer, en fonction des dimensions de T_1 et T_2, la plus grande valeur que l'on peut obtenir à partir du terrain T avec un plan de coupes P commençant par la coupe C.

$v = eval\_opt(i_1, j) + eval\_opt(i - i_1, j)$

Question 3 (1.5/7) —

(a) Si la commune possède un terrain de taille $i \times j$, par quelles coupes peut-elle commencer son plan (en supposant que la commune décide de couper son terrain) ?

(b) En déduire une relation de récurrence permettant d'exprimer $eval\_opt(i, j)$.

(a) Si la commune décide de couper le terrain, elle peut effectuer une coupe horizontale pour toute valeur entre 1 et $i - 1$ inclus ou effectuer une coupe verticale pour toute valeur entre 1 et $j - 1$ inclus.
(b)

$$\begin{aligned} eval\_opt(i,j) = \max\{\ & M[i,j], \\ & \max_{k \in \{1 \dots i-1\}} eval\_opt(k,j) + eval\_opt(i-k,j), \\ & \max_{l \in \{1 \dots j-1\}} eval\_opt(i,l) + eval\_opt(i,j-l)\} \end{aligned}$$

Question 4 (0.5/7) — Pour quelles valeurs de i et j la fonction $eval\_opt(i, j)$ donne-t-elle le revenu maximal que la commune peut obtenir ?

$i = n, j = m$

Question 5 (2/7) — Écrire un algorithme de programmation dynamique pour calculer ce revenu maximal.

```
Entrées : M matrice de prix, n, m les dimensions du terrain.
V ← matrice de taille n × m
pour i ∈ {1 ··· n} faire
    pour j ∈ {1 ··· m} faire
        best ← M[i, j] //On initialise avec le gain sans coupe
        pour k ∈ {1 ··· i − 1} faire
            //Pour chaque coupe en largeur
            si V[k, j] + V[i − k, j] > best alors
                best ← V[k, j] + V[i − k, j]
            fin
        fin
        pour l ∈ {1 ··· j − 1} faire
            //Pour chaque coupe en longueur
            si V[i, l] + V[i, j − l] > best alors
                best ← V[i, l] + V[i, j − l]
            fin
        fin
        V[i, j] ← best
    fin
fin
retourner V[n, m]
```

Algorithme 1 : RevenuMax

Pour les boucles sur k et l, on peut s'arrêter au milieu, ça ne change pas l'ordre de grandeur pour la complexité.

Question 6 (1/7) — Quelle est la complexité de cet algorithme en fonction de n et m ? Justifiez votre réponse.

Le tableau V contient $n \times m$ cases, pour chaque case on compare $n-1+m-1+1$ plans de coupe différents (pas de coupe, $n-1$ coupes en largeur, $m-1$ coupes en longueur). Connaître les valeurs de ces coupes se fait en $O(1)$ (lecture d'une case dans une matrice). Chaque case se calcule donc en $O(n+m)$, soit une complexité en $O(nm(n+m))$.

Question 7 (1/7) — Décrire en quelques phrases comment modifier cet algorithme pour qu'il retourne également un plan de coupe menant à un revenu maximal. La complexité de l'algorithme est-elle modifiée ?

Plusieurs méthodes sont possibles :

1. On crée un deuxième tableau de taille nm dans lequelle on stocke la coupe à faire à chaque fois que *best* est mis à jour. Il faut alors repartir depuis n, m et reconstituer la suite de coupe pour la retourner.
2. Au lieu de retourner $V[n, m]$, on ajoute une étape de parcours arrière du tableau V. Étant donné une position i, j, on peut parcourir les différentes coupes possibles, jusqu'à trouver celle pour laquelle la somme des plans de coupes correspond bien à la valeur $V[i, j]$. Cette coupe correspond à la première que la commune doit effectuer. On peut répéter cette opération sur les deux terrains formés par cette coupe en trouvant les bonnes coordonnées dans le tableau des V (si on ne fait pas de coupe, alors la commune doit simplement vendre le terrain tel quel).

Pour la complexité, on effectue au plus $n-1$ coupes en largeur et sur chacun des n terrains obtenus on effectue $m-1$ coupes en longueurs, le nombre de coupes maximum est donc en $O(nm)$. A chacune de ces étapes, dans le premier cas, on fait une simple lecture $O(1)$, dans le deuxième cas on doit comparer les différents plans pour trouver celui qui a donné la meilleure valeur, soit $O(n+m)$ comparaisons. On a donc dans le premier cas un coût pour le backtrack en $O(nm)$ et dans le deuxième cas un coût en $O(nm(n+m))$, la complexité est donc inchangée en terme d'ordre de grandeur.

Exercice 4 (9 points)

On considère l'algorithme suivant, appelé CoupeCycles, que l'on applique sur un graphe connexe $G = (S, A)$:

Entrées : $G = (S, A)$: un graphe connexe non orienté pondéré.
Trier les arêtes de A par coûts décroissants : $c(e_1) \geq \cdots \geq c(e_m)$.
$R = \emptyset$.
pour *chacune des arêtes* e_i*, examinées par coûts décroissants,* **faire**
  si *le graphe est toujours connexe après suppression de l'arête* e_i **alors**
    Supprimer l'arête e_i : $A = A \setminus \{e_i\}$.
    $R = R \cup \{e_i\}$.
  fin
fin
Retourner (S, A).

Algorithme 2 : Algorithme CoupeCycles.

Question 1 (0.5/9) — Exécuter cet algorithme sur le graphe $G_1 = (S_1, A_1)$ de la figure 2. Vous indiquerez l'ordre dans lequel les arêtes sont examinées et vous surlignerez les arêtes qui restent dans A à l'issue de l'exécution de l'algorithme.

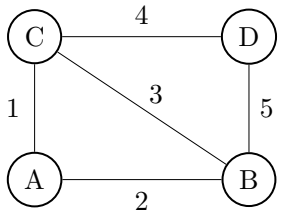


FIGURE 2 – Graphe G_1.

L'algorithme examine d'abord l'arête $\{B, D\}$, qu'il retire car le graphe $(S_1, A_1 \setminus \{\{B, D\}\}$ est toujours connexe. Dans ce nouveau graphe à 4 sommets et 4 arêtes, l'algorithme examine l'arête $\{C, D\}$ qu'il laisse car son retrait rendrait le graphe non connexe. L'algorithme examine ensuite l'arête $\{B, C\}$, qu'il retire, puis les deux dernières arêtes qu'il laisse. Il retourne donc l'arbre constitué des arêtes $\{\{B, A\}, \{A, C\}, \{C, D\}\}$.

Les questions suivantes concernent le cas général, i.e. **un graphe G connexe quelconque.**

Question 2 (1.5/9) — Comment peut-on vérifier qu'un graphe G est toujours connexe après suppression d'une de ses arêtes $e_i = \{a, b\}$? Vous indiquez en une phrase ou deux comment il est possible de faire cette vérification en utilisant un algorithme vu en cours. Vous préciserez également la complexité de cette vérification en fonction du nombre de sommets et/ou d'arêtes du graphe.

On fait un parcours à partir de a dans le graphe dans lequel on a retiré $\{a, b\}$ (il suffit en fait d'ignorer l'arête $\{a, b\}$ dans le tableau de listes d'adjacence), et on regarde si l'on peut atteindre b sans utiliser de point de regénération. Si c'est le cas alors G est toujours connexe après suppression de $\{a, b\}$, sinon G n'est plus connexe après suppression de $\{a, b\}$. Complexité en $O(|S| + |A|)$ et donc en $O(|A|)$ puisque le graphe est connexe.

Question 3 (1/9) — Montrer que l'algorithme CoupeCycles retourne un arbre couvrant de G.

Par construction, on obtient un graphe connexe. De plus, le graphe retourné ne contient pas de cycle. En effet, supposons par l'absurde que ce graphe contienne un cycle $\mathcal{C}$. Au moment de l'examen de l'arête de coût maximum de $\mathcal{C}$, la suppression de cette arête n'aurait pas déconnecté le graphe, et cette arête aurait été supprimée. On obtient ainsi un graphe connexe sans cycle, c'est à dire un arbre. Cet arbre contenant tous les sommets du graphe, c'est un arbre couvrant.

Question 4 (3.5/9) — Montrer qu'à l'issue de chaque itération de l'algorithme, il existe un arbre couvrant de coût minimum qui ne contient pas les arêtes de R. Vous pouvez pour cela faire une preuve par induction, en montrant que cette propriété est vérifiée avant la première itération, et que si elle est vérifiée à la fin de l'itération $(i-1)$ elle est également vérifiée à la fin de l'itération i.

Indication : on notera R_k les arêtes de l'ensemble R à la fin de l'itération k. On pourra raisonner en considérant, à la fin de la i-ème itération, un arbre couvrant de coût minimum $\mathcal{A}$ ne contenant pas les arêtes de R_{i-1}. En partant de cet arbre, on montrera qu'il existe un arbre couvrant de coût minimum ne contenant pas les arêtes de R_i.

On montre cette propriété, qui est un invariant de boucle, par induction sur le nombre d'itérations.

Avant la première itération, $R = \emptyset$: il existe un arbre couvrant de coût min qui ne contient pas les arêtes de R.

Supposons qu'à la fin de la $(i-1)$-ème itération la propriété est vérifiée, et montrons qu'elle le sera également à la fin de la i-ème itération.
Soit $\mathcal{A}$ un ACCM qui ne contient pas les arêtes de R_{i-1}. Si à l'issue de la i-ème itération, aucune arête n'est ajoutée à R, l'invariant est toujours vérifié (car $R_i = R_{i-1}$).
Supposons maintenant que l'arête $e_i = \{a, b\}$ est ajoutée à R lors de la i-ème itération.

- Si $e_i \notin \mathcal{A}$, la propriété est vérifiée.
- Si $e_i \in \mathcal{A}$: soit $\mathcal{A}'$ le graphe obtenu quand on retire e_i à $\mathcal{A}$. $\mathcal{A}'$ contient alors deux composantes connexes $\mathcal{A}'_1$ et $\mathcal{A}'_2$, une contenant a et l'autre b. Il existe dans le cocycle de $\mathcal{A}'_1$ au moins une arête $e' \notin R_{i-1}$ de coût inférieur au coût de e_i car, par construction, e_i est une arête de coût maximum d'un cycle d'arêtes n'appartenant pas à R_{i-1}. En remplaçant e_i par e' on obtient un arbre couvrant de coût minimum (on n'a pas augmenté le coût de l'arbre) ne contenant ni les arêtes de R_{i-1} ni e_i, c'est à dire aucune arête de R_i. Ainsi, l'invariant est vérifié à la fin de l'itération i.

Cette propriété est donc bien un invariant de boucle.

Question 5 (1/9) — Montrer que l'arbre retourné par l'algorithme CoupeCycles est un arbre couvrant de coût minimum de G.

A la fin de l'exécution de l'algorithme, on a mis $m - (n-1)$ arêtes dans R (puisque A est un arbre couvrant, comme montré dans la question précédente). L'invariant de boucle prouvé dans la question précédente indique qu'il existe un arbre couvrant de coût minimum ne contenant pas les arêtes de R. Le seul arbre couvrant ne contenant pas les arêtes de R est l'arbre retourné puisque R contient $m - (n-1)$ arêtes : l'arbre retourné est donc un arbre couvrant de coût minimum.

Question 6 (1.5/9) — Quelle est la complexité de cet algorithme ? Cette complexité est-elle meilleure que la complexité des deux autres algorithmes retournant des arbres couvrants de coût minimum que vous avez vus en cours ? Vous indiquerez le nom de ces algorithmes et justifierez votre réponse.

Complexité de l'algorithme : m itératios, et à chaque itération on fait un parcours en $O(m)$ (le graphe est connexe donc $O(n + m) = O(m)$. On a donc un algorithme en $O(m^2)$.

Algorithme de Kruskal : complexité en $O(m \log m)$.
Algorithme de Prim : complexité en $O(n^2)$ ou $O(m \log n)$ - suivant si l'on utilise des tas ou non.

Notre algorithme a donc une complexité pire cas moins bonne que les algorithmes de Prim et Kruskal.