UE LU3IN003 - Algorithmique.
Licence d'informatique.

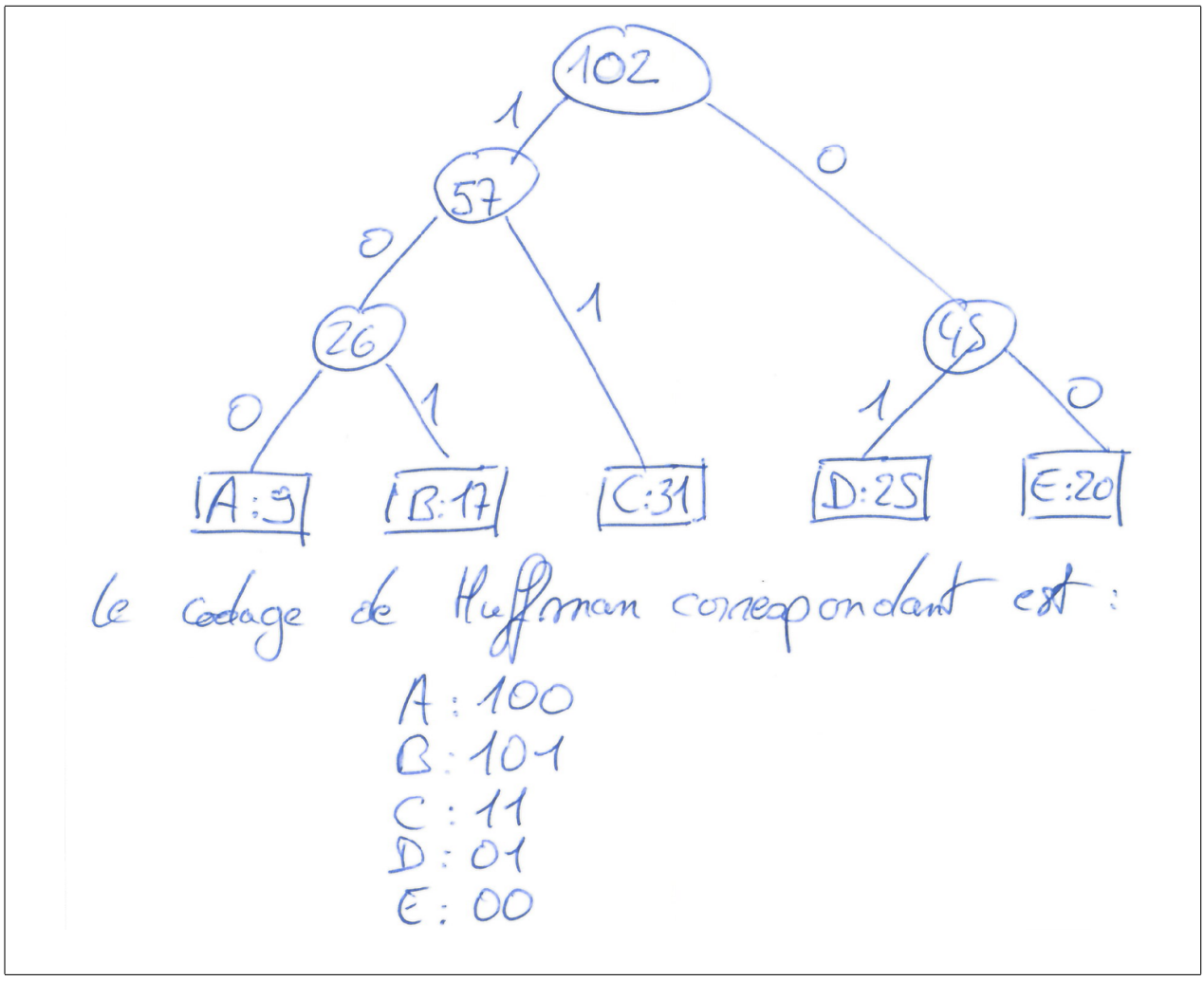
Examen du 12 janvier 2021. Durée : 1h30

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

Exercice 1 (3 points)

Question 1 (2/3) — 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



Question 2 (0.5/3) — Coder la chaîne DAC avec le code de la première question.

Le code est 0110011.

Question 3 (0.5/3) — Décoder 1011001110001100 avec le code de la première question.

La chaîne est BACADA.

Exercice 2 (4 points)

On considère le graphe orienté et pondéré G_λ donné sur la figure 1, où λ est un réel positif ou nul.

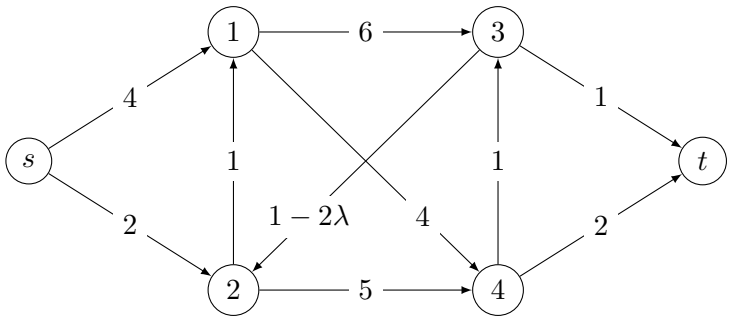


FIGURE 1 – Graphe G_λ

Question 1 (1/4) — 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.

Il y a trois circuits élémentaires dans le graphe, $1-3-2$, $2-4-3$ et $1-4-3-2$, les plus restrictifs étant les deux derniers de coût $7-2\lambda$. Ainsi il n'existe pas de circuit absorbant (et donc il existe un plus court chemin entre s et t) si $\lambda \in [0, 3.5]$.

Question 2 (0.5/4) — Pour quelles valeurs de λ peut-on utiliser l'algorithme de Dijkstra pour déterminer un plus court chemin entre s et t ?

$\lambda \in [0, 0.5]$.

Question 3 (0.5/4) — Quel(s) algorithme(s) vu(s) en cours peut-on utiliser pour déterminer un plus court chemin entre s et t lorsque $\lambda = 1$?

On ne peut utiliser que Bellman Ford puisque le graphe contient des circuits et que le poids de l'arc $(3, 2)$ est négatif.

Question 4 (0.5/4) — 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 ?

Parcours en largeur.

Question 5 (1.5/4) — 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 ?

On applique la récurrence $d_k(j) = \min_{i \ predecesseur \ de \ j} d_{k-1}(i) + c(i, j)$ car $d_k(j)$ représente le coût d'un plus court chemin de s à j qui utilise au plus k arcs (formule de récurrence de Bellman Ford). On s'arrête à l'issue de la première itération k où $d_k(t) < +\infty$. Complexité $O(p(n + m)$ ou $O(pm)$.

Exercice 3 (6 points)

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, $nb(3) = 4$.

Question 1 (1/6) — Indiquer les valeurs de $nb(1)$ et de $nb(2)$.

$nb(1) = 1$ et $nb(2) = 2$.

Question 2 (1.5/6) — On numérote les élèves de 1 à n. Montrer qu'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.

Si l'élève n est en monôme alors le nombre de configurations possible est le nombre de configurations possibles des $n-1$ autres élèves de la classes (c'est $nb(n-1)$). Il y a $n-1$ binômes possibles avec l'élève n (cet élève a le choix entre les élèves 1 à $n-1$). Etant donné un de ces binômes fixé, le nombre de configurations possibles est le nombre de configurations possibles pour les $n-2$ élèves restants (c'est $nb(n-2)$). Le nombre de configurations dans lesquelles l'élève n est en binôme est donc $(n-1) \times nb(n-2)$.

Question 3 (1/6) — En déduire une formule de récurrence indiquant la valeur de $nb(n)$ en fonction de valeurs $nb(i)$ avec $i < n$.

$$nb(n) = nb(n-1) + (n-1) \times nb(n-2).$$

Question 4 (2.5/6) — Écrire un programme dynamique prenant en argument un entier $n \geq 1$ et calculant $nb(n)$. Quelle est la complexité de cet algorithme ?

```
1 Calcule_NbConfig(entier n) {
2 Nb : tableau de n entiers. ;
3 Nb[1] = 1;
4 Nb[2] = 2;
5 pour i variant de 3 à n faire
6 |  Nb[i] = Nb[i − 1] + (i − 1) × Nb[i − 2];
7 fin
8 Retourner Nb[n];
9 }
```

Cet algorithme est en $\Theta(n)$.

Exercice 4 (7 points)

On dispose d'un groupe d'agents situés à différentes positions géographiques. Pour tout couple d'agent u et v, il est possible de connecter directement l'agent u et l'agent v, et ce avec un coût $c(\{u, v\})$. On s'intéresse au problème qui consiste à connecter l'ensemble des agents directement ou indirectement (en passant par d'autres agents) de façon à ce que le coût total de connection soit minimum. Ce problème consiste donc à identifier un arbre couvrant de coût minimum dans le graphe non orienté G où chaque sommet représente un agent, chaque arête $\{u, v\}$ représente une connection possible entre u et v, et le coût de l'arête est $c(\{u, v\})$.

On suppose que l'on dispose d'un arbre couvrant de coût minimum T pour le graphe G valué par la fonction c. L'objet de cet exercice est de voir si après diminution du coût de connection entre deux agents, l'arbre T est toujours un arbre couvrant de coût minimum.

On suppose que le coût de connection entre deux agents u et v est modifié et on note x la valeur de ce nouveau coût. On considère une nouvelle fonction de coût c' définie de la manière suivante :

$c'(\{u,v\}) = x < c(u,v)$ et pour toute arête e différente de $\{u,v\}$, $c'(e) = c(e)$.
Il s'agit donc de voir si l'arbre T est un arbre couvrant de coût minimum du graphe G valué par la fonction c', et si ce n'est pas le cas, de construire à partir de T un arbre couvrant de coût minimum T' pour le graphe G valué par la fonction de coût c'.

Question 1 (2.5/7) — On considère (dans cette question uniquement) le graphe G_1 de la figure 2.

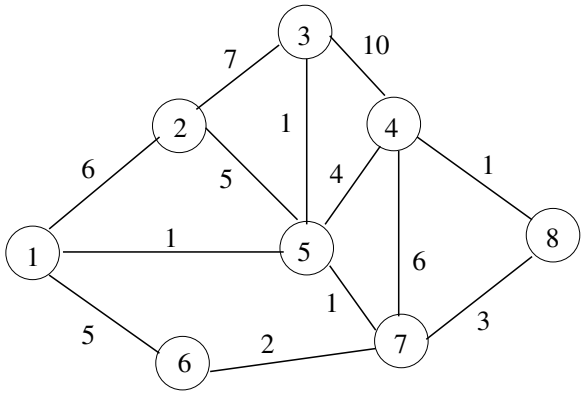


FIGURE 2 – Exemple de graphe de connection entre agents

(a) Calculer un arbre couvrant de coût minimum pour G_1. Vous pouvez surligner les arêtes de l'arbre couvrant de coût minimum sur le graphe G_1, ou bien indiquer la liste des arêtes qui ont été sélectionnées. Quel algorithme avez-vous utilisé ? Vous indiquerez dans quel ordre le arêtes ont été sélectionnées.

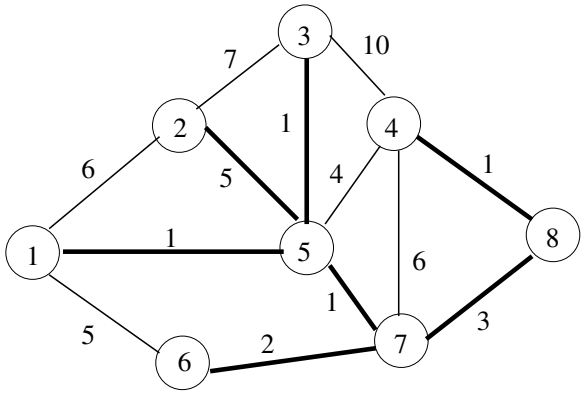


FIGURE 3 – Arbre couvrant de coût minimum de G_1

> On peut par exemple utiliser l'algorithme de Prim ou l'algorithme de Kruskal pour déterminer un arbre couvrant de coût minimum (cf. arbre représenté par des arêtes en gras sur le graphe de la figure 3 de coût 14).

(b) On notera T_1 l'arbre couvrant de coût minimum obtenu dans la question précédente. Supposons que le coût de l'arête $\{6,7\}$ passe de 2 à 1. T_1 est-il toujours un arbre couvrant de poids minimum ?

Oui.

(c) Supposons maintenant que le coût de l'arête $\{1, 2\}$ passe de 6 à 5. T_1 est-il toujours un arbre couvrant de poids minimum ?

Oui.

(d) Supposons maintenant que le coût de l'arête $\{1, 2\}$ passe de 6 à 4. T_1 est-il toujours un arbre couvrant de poids minimum ?

Non. En remplaçant l'arête $\{2, 5\}$ par l'arête $\{1, 2\}$ dans T_1, on obtient un arbre couvrant de coût inférieur au coût de T_1 ($13 < 14$).

Question 2 (1.5/7) — On suppose que $\{u, v\} \in T$ dans le graphe G. Après diminution du coût de $\{u, v\}$, T est-il toujours un arbre couvrant de coût minimum ? Justifiez votre réponse.

Oui, T est toujours un arbre couvrant de coût minimum pour le graphe G valué par c', i.e. (G, c'). On le montre par l'absurde : supposons que T ne soit plus un arbre couvrant de coût minimum après diminution du coût de $\{u, v\}$, c.a.d dans (G, c') (le coût ayant diminué de δ). Soit T' un arbre couvrant de coût minimum dans (G, c'). Etant donné un arbre A on note $c(A)$ le coût de A dans (G, c) (avant changement du coût de $\{u, v\}$), et $c'(A)$ le coût de A dans (G, c') (après changement du coût de $\{u, v\}$). On a : $c'(T') < c'(T)$ (car T' est un arbre couvrant de coût minimum, ce qui n'est pas le cas de T dans (G, c')), On a $c'(T) = c(T) - \delta$, et $c'(T') \geq c(T') - \delta$ (seul le coût de $\{u, v\}$ a changé). Donc $c(T') \leq c'(T') + \delta < c'(T) + \delta \leq c(T) - \delta + \delta$. D'où $c(T') < c(T)$: T n'était donc pas un arbre couvrant de coût minimum dans (G, c). Contradiction.

Question 3 (1/7) — On suppose que $\{u, v\} \notin T$. Dans quel cas T est-il toujours un arbre couvrant de coût minimum après diminution du coût de $\{u, v\}$? On ne demande pas de justification.

Soit γ la chaîne de u à v dans T. T est toujours un arbre couvrant de coût minimum dans (G, c') si le coût $c(e)$ de chaque arête e de γ satisfait la condition $c(e) \leq x$.

Question 4 (2/7) — En déduire le principe d'un algorithme qui permet de construire un nouvel arbre couvrant de coût minimum T' à partir de T. Quelle est la complexité de cet algorithme ? Cet algorithme est-il plus efficace que l'algorithme utilisé à la question 1(a) ?

Pour obtenir un arbre de coût minimum de (G, c') à partir de T :

Si $\{u, v\}$ appartient à T, alors T est un arbre couvrant de coût minimum de (G, c').

Si $\{u, v\}$ n'appartient pas à T. Soit γ la chaîne de u à v dans T.

Si $c(e) \leq x$ pour toute arête e de γ, alors T est un arbre couvrant de coût minimum de (G, c').

Sinon l'arbre de coût minimum de (G, c') est obtenu à partir de T en supprimant l'arête de coût maximum de γ et en ajoutant l'arête $\{u, v\}$.

Si on ne tient pas compte du coût du calcul de l'arbre T, la complexité de l'algorithme de calcul de T' est $O(n)$ (ceci étant du à la recherche de la chaîne entre u et v dans T, en faisant un parcours de racine u si T est codé par une liste d'adjacence). Il est donc moins coûteux de calculer T' à partir de T plutôt que de recalculer un arbre couvrant de coût minium dans (G, c') (coût $O(m \log n)$ ou $O(m \log m)$).