L3-INFO Algorithmique

Examen – 12 décembre 2022 – durée 2 h 30

Les documents et les téléphones (incluant smartphone, tablettes,... tout ce qui contient une interface réseau) interdits.
Seuls les dictionnaires pour les personnes de langue étrangère sont autorisés.

Comme d'habitude, vous êtes vivement encouragés à illustrer vos recherches et vos réponses par des schémas.

1 À propos de l'algorithme de Huffman *(4 points)*

On rappelle ci-dessous le principe de l'algorithme de Huffman :

F = FàPVide()
pour *chacun des K symboles s de l'alphabet* **faire**
    z = Nœud (`ArbreVide` (), s, `ArbreVide` ())
    `Insérer` dans F le nœud z avec pour priorité la fréquence de s
tant que *il reste au moins 2 nœuds dans F* **faire**
    x = `Extraire` (F)
    y = `Extraire` (F)
    z = Nœud (x, NULL, y) `// Rappel : pour les nœuds internes l'étiquette n'a aucune importance`
    `Insérer` dans F le nœud z avec pour priorité la somme des priorités de x et de y
renvoyer `Extraire` (F)

Question 1 : *(1 point)*

À l'aide de l'algorithme de Huffman, calculer un arbre de codage optimal pour la phrase à 34 symboles :

`UN HARMONIUM AJOUTA TON UT AU TRIO`

Attention à ne pas oublier les caractères « espace » entre les mots.

Question 2 : *(3 points)*

On veut démontrer la propriété suivante : « *Le nœud racine de l'arbre renvoyé par l'algorithme de Huffman possède une priorité égale à 1* ».

On utilisera pour cela l'invariant « *La somme des priorités des nœuds contenus dans la FàP F vaut 1* ».

(a) Justifier que cet invariant est vérifié avant d'entrer dans la boucle **tant que**.

(b) Justifier que l'invariant est maintenu par chaque itération de la boucle **tant que**.

(c) En déduire la propriété souhaitée.

2 Une omelette (d'après un exercice d'Yves Robert) *(9 points)*

Un producteur d'oeufs vient de construire un nouveau bâtiment pour loger ses poules pondeuses. Il s'agit d'un gigantesque immeuble de n étages (numérotés de 1 à n), où n est très grand.

Pour éviter de gâcher la marchandise, il voudrait déterminer à partir de quel étage la chute d'un œuf par la fenêtre est fatale pour l'œuf.

La seule opération possible, pour tester si la hauteur d'un étage est fatale ou pas, est de lancer un œuf par la fenêtre :

— s'il se casse, la chute est fatale et on ne peut pas réutiliser l'œuf ;

— sinon, la chute n'est pas fatale, et on peut réutiliser l'œuf pour un autre test.

L'enjeu de cet exercice est de proposer différents algorithmes pour trouver la hauteur précise à partir de laquelle la chute est fatale (on veut renvoyer un étage i tel qu'un œuf se casse depuis l'étage i mais pas depuis l'étage $i-1$).

Le choix d'utiliser un algorithme plutôt qu'un autre dépendra du nombre k d'œufs disponibles. **Dans tout cet exercice, les coûts des algorithmes seront exprimés en nombre de lancers effectués.** Le nombre d'œufs cassés n'est jamais pris en compte.

Question 3 : Avec 1 seul œuf *(1,5 points)*

Écrire un algorithme résolvant ce problème avec $k = 1$ seul œuf.
Déterminer le coût au pire de cet algorithme (en nombre de lancers donc).

Question 4 : Avec beaucoup d'œufs *(4 points)*

On suppose cette fois-ci qu'on peut utiliser un grand nombre k d'œufs (il n'y aura donc ici pas de contrainte sur le nombre d'œufs sacrifiés).

(a) Écrire un algorithme qui résout ce problème en $\lceil \log_2(n) \rceil$ lancers au pire (la notation $\lceil \ldots \rceil$ désigne l'arrondi à l'entier supérieur).

(b) Préciser combien d'œufs k auront été cassés au maximum, et décrire précisément le cas où cela se produit.

Question 5 : Un cas particulier *(2 points)*

On souhaite utiliser un algorithme similaire à celui que vous avez proposé à la question 4, mais il nous manque quelques œufs (k est légèrement inférieur à la valeur déterminée à la question 4.b).
Expliquer comment vous pouvez adapter cet algorithme (sans forcément le réécrire complètement).
Le coût en lancers de ce nouvel algorithme n'est pas demandé.

Question 6 : Avec 2 œufs *(1,5 points)*

On étudie enfin le cas où on dispose d'exactement 2 œufs pour résoudre le problème.
Proposer un algorithme qui résout ce problème en $2 \times \sqrt{n}$ lancers au pire.

3 Couverture de graphe par les sommets *(7 points)*

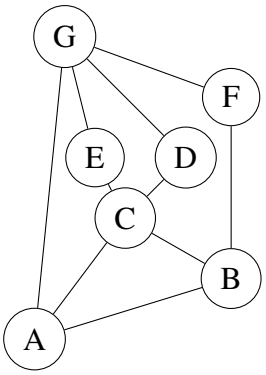
Étant donné un graphe G non orienté, on cherche à déterminer une *couverture de ce graphe par les sommets*, c'est-à-dire un sous-ensemble S de ses sommets tel que tout sommet de G :

— fait partie de S

— ou est voisin d'un des sommets de S

— ou les deux.

Par exemple, l'ensemble de sommets $\{A, D, E, F\}$ est une couverture du graphe ci-contre.



Le problème à résoudre est de déterminer une couverture *minimale*, c'est-à-dire contenant autant ou moins de sommets que toute autre couverture du même graphe.

Question 7 : Un exemple *(1 point)*

Déterminer une couverture **minimale** pour le graphe donné en exemple, et justifier qu'il n'y en a pas de plus petite.

Question 8 : Algorithme glouton *(4 points)*

On dit qu'un sommet est *couvert* si au moins un des ses voisins est dans S et on propose la stratégie de résolution suivante :

tant que S *n'est pas une couverture du graphe* G **faire**
    `Ajouter` à S un sommet de G qui a le plus grand nombre de sommets pas encore couverts parmi lui et ses voisins.

Écrire un algorithme en pseudo-code qui implémente cette stratégie. Il s'agira notamment de détailler :

— quelles structures de données et/ou marquages du graphe vous utilisez, par exemple pour représenter l'ensemble S ;

— comment vous testez si S est une couverture du graphe ;

— comment vous comptez les voisins non couverts d'un sommet ;

— comment vous sélectionnez le prochain sommet à ajouter à S...

Le plus important est que votre algorithme soit correct, mais une recherche d'efficacité sera appréciée par le correcteur. Le calcul de la complexité n'est pas demandé.

Question 9 : Un contre-exemple *(2 points)*

La stratégie proposée ci-dessus n'est pas optimale.
Donner un exemple de graphe et de trace d'exécution pour lesquels la couverture obtenue n'est pas minimale.

Indications : Un graphe à 5 sommets est suffisant, ou 7 si on veut éviter de départager les égalités dans le choix glouton. On s'intéressera plutôt à des graphes de forme arborescente.