UNIVERSITÉ DE MONTRÉAL
Département IRO

Été 2018
Neil Stewart

IFT 2015 E18

Devoir 2.

10/10, soit 10% de la note finale.

Les 10 points "Partie pratique" pour le cours E18 seront distribués de la façon suivante:
Devoir #1 (1 point), Devoir #2 (3 points), Devoir #3 (6 points).

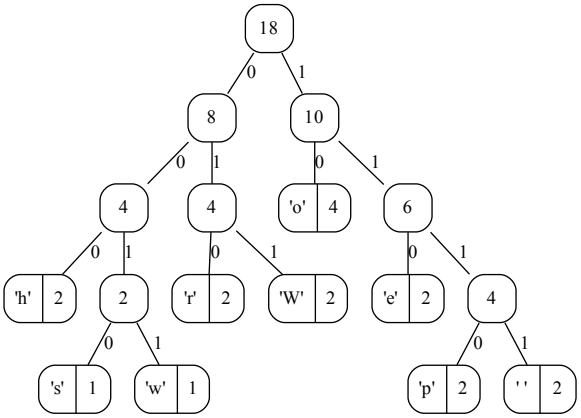
1 Partie Pratique (3 points)

Le codage de Huffman est un algorithme de compression de données utilisant un code préfixe, de longueur variable. Le code est déterminé à partir d'une estimation de la fréquence d'occurence pour chaque symbole dans une source.

Le principe du codage de Huffman repose sur la création d'un arbre binaire. Ses feuilles représentent chaque caractère et sa fréquence. Ses nœuds internes représentent la somme des fréquences de leurs sous-arbres.

Le but est de d'obtenir un code binaire, qui utilise le moins de bits pour les symboles les plus communs.

Soit "Who powers Whooper", la phrase à coder. Voici un exemple de l'arbre et de la table qui y correspond :



Char	Freq	Code
h	2	000
s	1	0010
w	1	0011
r	2	010
W	2	011
o	4	10
e	2	110
p	2	1110
	2	1111

Ils ont été générés comme suit :

```
echo Who powers Whooper | java HuffmanCode graph | dot -Tpdf -o graph.pdf
echo Who powers Whooper | java HuffmanCode table | pandoc -o table.pdf
```

Pour ce devoir, vous devez :

- Compléter la classe `Node`, qui représente un nœud (interne ou feuille) dans l'arbre de Huffman.
- Compléter la fonction `getCharacterFrequencies`, qui retourne un tableau avec la fréquence de tous les caractères représentables en ASCII sur *8 bits*, à partir d'une chaîne de caractère provenant de l'entrée standard (`stdin`).
- Compléter la fonction `getHuffmanTree` qui prend en paramètre le tableau de fréquences, génère un arbre de Huffman, et retourne le nœud racine de cet arbre.
- Compléter la fonction `printTable`, qui imprime un tableau sous le format *Markdown*, en effectuant une **recherche en profondeur** (*DFS*).
- Compléter la fonction `printGraph`, qui imprime un graphe sous le format *DOT*, en effectuant une **recherche en largeur** (*BFS*).

Pour ce devoir, vous pouvez :

- Utiliser n'importe quelle classe de la librairie standard de Java, qui implémente l'interface `Queue<E>`, y compris `java.util.PriorityQueue`.
- Référer au fichier `table.md`, qui contient l'exemple de tableau de l'énoncé pour avoir une idée du format à utiliser.
- Référer au fichier `graph.dot`, gracieusement fourni, qui contient l'exemple d'arbre de l'énoncé, ainsi que des commentaires expliquant le format à utiliser.

Il est aussi recommandé de vous référer à la section 10.1.2 du livre de Weiss. Le 3^{e} volume du livre *The Art of Computer Programming* peut être un bon supplément.

Veuillez remettre votre fichier `HuffmanCode.java` complété sur StudiUM avec les noms et matricules des auteurs en commentaire d'en-tête.

2 Partie Théorique (7 points)

1. (3 points)

   Le coût de recherche quand la clef n'est pas présente est une question importante pour les arbres de recherche binaire. Par exemple, dans la Question 6 du Devoir 1 vous avez sans doute suggéré l'insertion de clefs l'une après l'autre dans un arbre vide au début: à chaque insertion il fallait donc faire une recherche dans un arbre où la clef n'est pas déjà présente. Aussi, dans la démonstration du 30 mai il a été suggéré d'étendre l'arbre binaire en conçevant les références nulles comme des feuilles dans un arbre étendu. Ces nouveaux noeuds-feuilles sont appelés en anglais “failure

nodes", nous dirons "noeuds-d'échec". En utilisant le petit théorème déjà démontré dans le cours, il a été démontré dans la démonstration que le nombre de noeuds d'échec, dans un arbre de recherche avec N noeuds, est égal à $N + 1$.

(a) (1 point)

Dans le cours nous avons dit que I, la longueur de chemin intérieur ("Internal Path Length") est égale à la somme des profondeurs des noeuds. Si je prends par exemple l'arbre à gauche dans la Figure 4.15, page 113 de Weiss, j'obtiens $I = 1 + 2 + 2 + 3 + 1 = 9$. C'est intéressant parce que le coût moyen de recherche (clef présente) pour un arbre particulier avec N noeuds est $C(N) = I(N)/N$.

Si je fais la même chose pour la recherche de clefs non-présentes, je dois définir E la longueur de chemin extérieur ("External Path Length") qui est égale à la somme des profondeurs des noeuds d'échec. Dans l'arbre de la Figure 4.15 nous avons $N = 6$ et donc $N + 1 = 7$ noeuds d'échec, et $E = 3 + 3 + 4 + 4 + 3 + 2 + 2 = 21$. Je le savais à l'avance parce qu'il y a un théorème qui dit que

$$E(N) = I(N) + 2N \tag{1}$$

Le coût moyen de recherche (clef absente) pour un arbre particulier avec N noeuds est $C'(N) = E(N)/(N + 1)$.

On trouve souvent dans les livres une preuve de (1) par induction sur le nombre N de noeuds dans l'arbre. C'est tout à fait correct. Par contre, ici je vous demande de démontrer (1) par induction forte ("strong induction", par exemple Johnsonbough 7ième édition, p. 102) sur le nombre de niveaux dans l'arbre.

(Commentaires généraux, pas nécessaires pour résoudre l'exercice: $I(N)$, $E(N)$,... pourraient faire référence aussi à une valeur moyenne pour un ensemble d'arbres avec N noeuds. Dans nos applications nous aurons typiquement $C(N) = \Theta(\log(N).)$

(b) (1 point)

Supposons que nous ayons deux fonctions *quelconques* $I = I(N)$ et $E = E(N)$ qui satisfont à (1) et telles que $C(N) = \Omega(1)$. Démontrez que

$$C'(N) = O(C(N)). \tag{2}$$

(c) (1 point)

Donnez un contre-exemple à (2) dans le cas où l'hypothèse $C(N) = \Omega(1)$ n'est pas satisfaite.

(Cela ne va jamais nous arriver dans la pratique. Mais les théorèmes ont des hypothèses. Mieux vaut savoir quelles sont les hypothèses minimales, pour être certain que cela ne va jamais nous déranger.)

2. ($1\frac{1}{2}$ points)

   On peut utiliser un monceau *min_heap* pour le problème de trouver le k'ième élément le plus grand (voir Weiss, page 1 et page 239). Il s'agit de garder une queue de priorité (implantée par monceau) avec les k éléments les plus grands.

   On commence par créer un monceau avec les k premiers éléments qui arrivent. Ensuite, chaque fois qu'un nouvel élément arrive, on le compare avec S_k = l'élément le plus petit dans le monceau (c'est le k'ième élément le plus grand pour les données vues jusqu'ici). Si le nouvel élément est plus grand que S_k, on remplace S_k par le nouvel élément. Maintenant le monceau a un nouvel élément S_k qui est le plus petit: cet élément est possiblement l'élément qu'on vient d'ajouter.

   Quand tous les éléments on été traités, on sort la valeur finale de S_k comme réponse.

   Démontrez que ce processus est $O(N \log k)$.

3. ($\frac{1}{2}$ point)

   Weiss, Exercice 4.8, page 161.

4. (1 point)

   Considérons le problème simple de chercher une clef dans un tableau linéaire, en supposant la clef présente. Disons que si nous trouvons la clef dans la i'ième case, cela prend i comparaisons, $i = 1, \ldots, N$, parce que nous devons passer à travers les premières i cases pour tomber sur la clef. Si nous pouvons mettre les clefs les plus probables plus au début du tableau, le coût moyen de recherche sera inférieur à $N/2$.

   Si la probabilité de la clef i est p_i, écrivez l'expression pour le coût moyen de trouver une clef, en nombre de comparaisons.

   Dans la vraie vie, il arrive assez souvent que les probabilités p_i, en ordre décroissant, sont proportionnelles à $1/i$, $i = 1, \ldots, N$ (loi de Zipf). Dans ce cas:

   (a) Donnez l'expression mathématique pour p_i en utilisant $H_N = \sum_{i=1}^{N} \frac{1}{i}$ le nombre harmonique.

   (b) Quel est le coût moyen de trouver une clef dans ce cas?

5. (1 point)

   Le nombre harmonique a été mentionné aussi dans le cours dans le contexte du coût de recherche dans un arbre binaire. En fait, il est possible de démontrer que H_N est approximativement égal à $\log_e N$, et dans la démonstration vous avez vu des bornes logarithmiques inférieures et supérieures dans le cas spécial où N ait la form $N = 2^{m+1}$. Ce qui veut dire que H_N tend vers l'infinie, mais pas très vite.

On pourrait même dire que la série H_N frôle la finitude, *i.e.*, elle échappe à peine de rester finie, parce que la série

$$H_N^{(r)} = \sum_{i=1}^{N} \frac{1}{i^r}$$

reste finie pour toute valeur de r strictement supérieure à 1.

Démontrez que $H_{2^m-1}^{(r)}$ est inférieure à $\sum_{k=0}^{m-1} \frac{2^k}{2^{kr}}$, ce qui est inférieure à $\frac{2^{r-1}}{2^{r-1}-1}$ (aussi à démontrer). De ce point de vue, H_N ne tend pas très vite vers l'infinie.

À réaliser en équipes de 1 ou 2. À remettre le 18 juin, 2018, avant `9:00`. Cela vous donne presque 3 semaines, mais attention, le 18 juin est un lundi, les règles habituelles s'appliquent pour la remise de la partie théorique, et l'examen Intra est le 20 juin.