Structures de données et algorithmes

Examen écrit, 23 mai 2014

Livres fermés. Durée : 3h30.

Remarques

— Répondez à chaque question sur une feuille **séparée**, sur laquelle figurent votre nom et votre section.
— Soyez bref et concis, mais précis.
— Sauf mention explicite, toutes les complexités sont à décrire par rapport au temps d'exécution des opérations concernées. Soyez toujours le plus précis possible dans le choix de votre notation (Ω, O ou Θ).

Question 1

(a) Décrivez l'algorithme de tri rapide dans le formalisme de votre choix.

(b) Implémentez une fonction `Select(A, k)` permettant de trouver la k-ième plus petite valeur d'un tableau d'entiers `A`, sans qu'il soit nécessaire de trier `A`. Cette fonction peut modifier l'ordre des éléments du tableau `A`.
Suggestion : Inspirez vous de l'algorithme du tri rapide.

(c) Analysez la complexité de votre algorithme dans le pire et le meilleur cas. Justifiez.

(d) Que pensez-vous de la complexité du cas moyen ?

Question 2

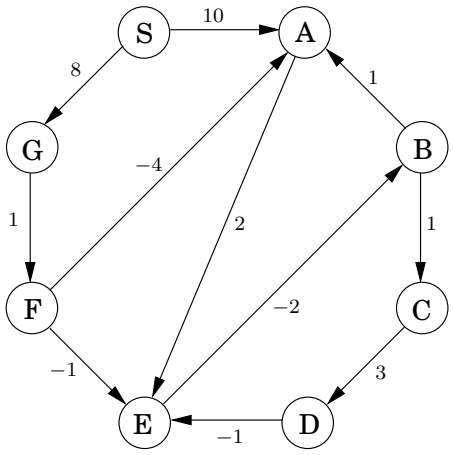
(a) Qu'est-ce qu'un tas binaire Min et Max ? Comment implémenter un tas dans un vecteur ?

(b) Soit le tableau `A = [-5, 0, 3, 8, 2, 1, 9, -2]`.
— Représentez graphiquement le tas généré par `BuildMaxHeap(A)` (telle que vue au cours).
— Comparez ce tas avec celui obtenu par l'insertion successive des éléments `A[i]` (pour `i=1, ..., 8`, dans cet ordre) dans une file a priorité `S` initialement vide (i.e., avec la fonction `Insert(S, x)`).

(c) Implémentez un conteneur `C` de valeurs numériques muni des fonctions suivantes :
— `Create()` : Retourne un conteneur `C` vide.
— `Insert(C, x)` : Insère la valeur `x` dans `C`.
— `Median(C)` : Retourne la valeur médiane des valeurs de `C`.
La complexité de `Insert(C, x)` doit être en $O(\log(n))$ et la complexité de `Median(C)` en $O(1)$, où n est le nombre de valeurs de `C`.
Remarque : Si n est impair, la médiane est définie comme la valeur à la position $\frac{n+1}{2}$ de la séquence triée des valeurs. Si n est pair, la valeur médiane est définie comme la moyenne des valeurs aux positions $\frac{n}{2}$ et $\frac{n}{2}+1$ de la séquence triée des valeurs.
Suggestion : Utiliser un tas MIN et un tas MAX.

Question 3

Soit le graphe `G` suivant :



(a) Implémentez un algorithme `ShortestPath(G, u, v)` permettant de calculer le plus court chemin d'un noeud `u` vers un noeud `v` dans un graphe `G`, en veillant à ce que cet algorithme soit adapté au graphe ci-dessus.

(b) Illustrez le fonctionnement de cet algorithme sur `G` pour la paire de nœuds `u=S` et `v=D`. Montrez dans une table l'évolution des distances calculées pour chaque nœud au cours des itérations de l'algorithme (seulement la boucle la plus externe s'il y en a plusieurs) et donnez le plus court chemin calculé.

(c) Quelle implémentation de graphe utiliseriez-vous pour obtenir des performances optimales ? Justifiez.

(d) Expliquez comment modifier votre algorithme pour qu'il renvoie parmi tous les chemins les plus courts entre deux noeuds, celui qui contient le moins d'arêtes.

Question 4

Soit une chaîne de caractères `s[1..n]` qui correspond à un texte corrompu dont on aurait enlevé tous les signes de ponctuation. Par exemple, en anglais :

"wheninthecourseofhumaneventsitbecomesnecessary"

On souhaite écrire une fonction permettant de séparer la chaîne de caractères en ses différents mots successifs. Le problème est rendu difficile notamment par le fait qu'une même chaîne peut être séparée de plusieurs manières possibles en une suite de mots corrects de la langue sans pour autant avoir du sens (La sous-chaîne "eventsitbecomes" pourrait par exemple correspondre à "event sit be comes" ou à "events it becomes"). Pour lever une partie de l'ambiguité, on supposera qu'on dispose d'une structure de données contenant pour chaque mot de la langue ciblée un poids strictement positif proportionnel à sa fréquence d'apparition dans des textes de cette langue. Cette structure de données sera accessible par une fonction `Frequency(w)` prenant en entrée un mot `w` (une chaîne de caractère) et renvoyant soit 0 si le mot `w` ne se trouve pas dans le dictionnaire, soit le poids f du mot s'il s'y trouve. La procédure de séparation des mots d'une chaîne cherchera alors à trouver la séparation maximisant le produit des poids des mots résultants.

(a) En supposant que la structure de données utilisée pour stocker les mots et leurs poids ne devra pas être mise à jour, quelle structure de données vue au cours vous semble la plus appropriée pour implémenter `Frequency(w)` ? Justifiez votre réponse et précisez la complexité de `Frequency(w)` en fonction du nombre de mots dans le dictionnaire.

(b) Proposez une approche gloutonne pour résoudre le problème (sans donner de pseudo-code) et montrez par un contre-exemple qu'elle ne fonctionnera dans le cas général où on ne fait pas d'hypothèse sur le contenu de la structure de données.

(c) Proposez une solution par programmation dynamique en passant par les étapes suivantes :

i Soit $M[n]$ le produit des poids des mots d'une séparation optimale de la chaîne `s[1..n]`. Formulez $M[n]$ de façon récursive. Précisez le cas de base.

ii Implémentez une procédure permettant de calculer efficacement la valeur de $M[n]$.

iii Adaptez votre solution pour reconstruire la séparation optimale identifiée.

iv Analysez la complexité de votre solution en fonction de la taille n de `s`, dans le pire et le meilleur cas.

Remarque : Vous pouvez supposer que la longueur l_{max} du mot le plus long dans la structure est connue.