révision

Algorithmique

Ce document a pour objet de vous faire rédiger différents algorithmes classiques qui forment la base de l'algorithmique enseignée en première et seconde année. Afin de vous permettre de réaliser des tests sur les algorithmes que vous allez écrire, vous pouvez télécharger l'archive compressée `exemples.zip` qui, une fois décompressée, contient des données qui vous serviront à vérifier la correction de vos fonctions.

Recherche séquentielle dans un tableau uni-dimensionnel

Exercice 1

a. Rédiger une fonction `maximum(t)` qui prend pour argument un tableau uni-dimensionnel et renvoie la valeur du plus grand élément de ce tableau.

b. Rédiger une fonction `secondMax(t)` qui prend pour argument un tableau uni-dimensionnel et renvoie la valeur du deuxième plus grand élément de ce tableau.

c. Rédiger une fonction `distanceMin(t)` qui prend pour argument un tableau uni-dimensionnel et qui renvoie la plus petite différence séparant deux valeurs distinctes du tableau : $\min\left\{|x-y| \;\middle|\; (x,y)\in t^2,\ x\neq y\right\}$.

Exemple. Les deux plus grandes valeurs du tableau `t` présent dans le fichier `tableau.txt` sont 99 613 et 99 599. La différence minimale entre deux éléments distincts de ce tableau est égale à 5.

Exercice 2

a. Rédiger une fonction `comptage(txt)` qui prend pour argument une chaîne de caractères et renvoie un dictionnaire dont les clefs sont les caractères présents dans `txt` et les valeurs le nombre d'occurences de chacun de ces caractères dans ce texte.

b. Rédiger une fonction `chercheMot(txt, mot)` qui prend pour arguments deux chaines de caractères et qui renvoie le nombre d'occurrences de `mot` dans `txt`.

Exemple. Le nombre d'espaces `' '` contenus dans le fichier `hugo.txt` est égal à 101 459. Le mot `'REPUBLIQUE'` est présent 63 fois. (Consulter le premier chapitre du cours pour savoir comment manipuler un fichier texte).

Algorithmes de tri

Exercice 3

a. Rédiger une fonction `fusion(t1, t2)` qui prend pour arguments deux tableaux *triés* t_1 et t_2 et renvoie un tableau trié résultat de la fusion de ces deux tableaux.

b. En déduire une fonction *récursive* `triFusion(t)` qui prend pour argument un tableau et renvoie un nouveau tableau contenant les mêmes valeurs, triées par ordre croissant.

c. Tester votre fonction sur le tableau `t` présent dans `tableau.txt`.

Principe du tri fusion : le tableau t est séparé en deux parties égales (à un élément près) t_1 et t_2. Lorsque ces derniers comportent au moins deux éléments, ils sont triés par appel récursif, puis on fusionne les deux tableaux triés.

Exercice 4

Rédiger une fonction `dichotomie(x, t)` qui prend pour argument un entier x et un tableau *trié* d'éléments deux à deux distincts t et qui renvoie, s'il existe, un couple de valeurs (t_i, t_{i+1}) vérifiant $t_i \leqslant x < t_{i+1}$. Cet algorithme devra être de complexité logarithmique et non pas linéaire.

Exemple. Après avoir trié le tableau `t` du fichier `tableau.txt` votre fonction devra renvoyer le couple $(31\,410, 32\,075)$ pour $x = 31\,415$.

Théorie des graphes

Exercice 5

Dans cet exercice, on considère un graphe orienté G dont les sommets sont étiquetés par les entiers $[\![0, n-1]\!]$, représenté par *listes d'adjacence*. Autrement dit, G est représenté par un tableau g de longueur n, tel que pour tout $i \in [\![0, n-1]\!]$, `g[i]` est la liste des sommets `j` pour lesquels il existe un arc reliant le sommet i au sommet j.

a. Rédiger une fonction `degreEntrant(g, i)` qui prend pour arguments un graphe G et un sommet i et qui renvoie le degré entrant de i (c'est-à-dire le nombre d'arcs qui aboutissent à i).

b. Rédiger une fonction `transpose(g)` qui prend pour argument un graphe G et renvoie le graphe transposé, c'est-à-dire celui dont tous les arcs ont changé d'orientation.

Exemple. Dans le graphe orienté `g1` défini dans le fichier `graphe.txt`, le sommet 0 a un degré entrant égal à 4. Dans le graphe transposé de `g1`, le sommet 42 a un degré entrant égal à 8.

Exercice 6

Dans cet exercice, on considère un graphe non orienté G dont les sommets sont étiquetés par les entiers $[\![0, n-1]\!]$, représenté par *listes d'adjacence*.

a. À l'aide d'un *parcours en largeur*, rédiger une fonction `distance(g, i, j)` qui renvoie la distance du sommet i au sommet j dans G, autrement dit la longueur du plus petit chemin reliant i à j. Si un tel chemin n'existe pas, cette fonction renvoie `None`.

b. À l'aide d'un *parcours en profondeur*, rédiger une fonction `composantes(g)` qui renvoie le nombre de composantes connexes de G.

Exemple. Dans le graphe orienté `g2` défini dans le fichier `graphe.txt`, la distance entre les sommets 7 et 91 est égale à 5, et `g2` possède 4 composantes connexes.

Exercice 7

Dans cet exercice, on considère un graphe pondéré non orienté G dont les sommets sont étiquetés par les entiers $[\![0, n-1]\!]$, représenté par *matrice d'adjacence*. Autrement dit, G est représenté par un tableau bi-dimensionnel g de taille $n \times n$ tel que `g[i][j]` est égal au poids de l'arc reliant le sommet i au sommet j. On suppose que `g[i][j]` est un réel positif, égal à $+\infty$ s'il n'existe pas d'arc reliant le sommet i au sommet j.
Rappel. La commande `float('inf')` permet de définir un nombre flottant égal à $+\infty$.

a. Rédiger une fonction `chercheMin(dist, dejaVu)` qui prend pour arguments un tableau `dist` de nombres positifs ou infinis et un tableau `dejaVu` de booléens de même taille, et qui renvoie l'indice i pour lequel `dist[i]` est minimal parmi les valeurs pour lesquelles `dejaVu[i]` est égal à `False`.

b. En déduire, à l'aide de l'algorithme de Dijkstra [a], une fonction `distance(g, i)` qui calcule la distance du plus court chemin reliant le sommet i au sommet j (cette fonction renvoie $+\infty$ s'il n'existe pas de tel chemin).

Exemple. Dans le graphe `g3` défini dans le fichier `graphe.txt`, la distance entre les sommets 2 et 17 est égale à 10.

a. Relire votre cours de première année à ce sujet.

Algorithme de Dijkstra

Cet algorithme utilise un tableau `dejaVu` initialisé à `False` et un tableau des distances `dist` entre le sommet i et les autres sommets, initialisé à 0 pour le sommet i et à $+\infty$ pour les autres.
Tant que les sommets n'ont pas tous été vus :

- on détermine, parmi les sommets non vus, le sommet s dont la valeur `dist[s]` est minimale ;
- le sommet s est marqué comme vu et le tableau `dist` est mis à jour en remplaçant `dist[k]` par `min(dist[k], dist[s] + g[s][k])` pour tout $k \in [\![0, n-1]\!]$.

Programmation dynamique

Exercice 8

On considère un tableau bi-dimensionnel t à n lignes et n colonnes, contenant des entiers positifs. On cherche à déterminer un chemin reliant la case $(0,0)$ (située en haut à gauche) à la case $(n-1,n-1)$ (située en bas à droite) en n'utilisant que des déplacements d'un cran vers le bas ou vers la droite, et dont la somme des valeurs des cases rencontrées (le *poids* du chemin) soit minimale.

a. On note $p(i,j)$ le poids minimal d'un chemin reliant la cas $(0,0)$ à la case (i,j). Exprimer $p(i,j)$ en fonction de $p(i-1,j)$ et de $p(i,j-1)$.

b. En déduire une fonction `poidsMinimal(t)` qui renvoie le poids minimal d'un chemin reliant la case $(0,0)$ à la case $(n-1,n-1)$.

c. Rédiger enfin une fonction `cheminDePoidsMinimal(t)` qui renvoie un chemin de poids minimal.

Exemple. Pour le tableau `t` défini dans le fichier `dynamique.txt`, un chemin de poids minimal est

$$(0,0) \to (1,0) \to (1,1) \to (1,2) \to (2,2) \to (2,3) \to (2,4) \to (3,4) \to (3,5) \to (3,6) \to$$
$$(3,7) \to (3,8) \to (4,8) \to (4,9) \to (4,10) \to (4,11) \to (5,11) \to (6,11) \to (7,11) \to (7,12) \to$$
$$(7,13) \to (8,13) \to (8,14) \to (9,14) \to (9,15) \to (9,16) \to (9,17) \to (10,17) \to (11,17) \to (12,17) \to$$
$$(13,17) \to (13,18) \to (14,18) \to (15,18) \to (15,19) \to (16,19) \to (17,19) \to (18,19) \to (19,19)$$

il a pour poids 245.