Examen Terminal UE Programmation et Algorithmes

L1 MPCI

29 mai 2024 - Durée: 3h

Lorsque l'on vous demande d'écrire de décrire ou de donner un algorithme cela signifiera toujours en donner un pseudo-code, justifier de son exactitude et de sa complexité

On rappelle qu'aucun document, ni équipement électrique ou électronique n'est autorisé.

Les exercices :
— sont au nombre de 3 ;
— valent chacun 7 points (donc optimisez votre temps) ;
— sont de difficulté équivalente ;
— sont indépendants ;
— leur début est plus facile que leur fin (qui vaudra donc plus de point).

Rendez des copies séparées pour chaque exercice, ceci vous permettra de reprendre les exercices au cours de l'examen sans perdre le correcteur.

Exercice 1 – Champ de mines

Un champ de mines est représenté par une grille M de $n \times n$ cases (n lignes et n colonnes) telle que $M[l][c]$ représente la probabilité de survie d'une personne sur la case (l, c) qui se trouve à la ligne l et à la colonne c (la probabilité d'existence d'une mine à la case (l, c) vaut $1 - M[l][c]$). On supposera qu'il n'y a jamais de mines en $(0, 0)$ (la position initiale) et en $(n - 1, n - 1)$ (la sortie du champ de mines). La figure 1 montre un champ de mines de taille $n = 5$.

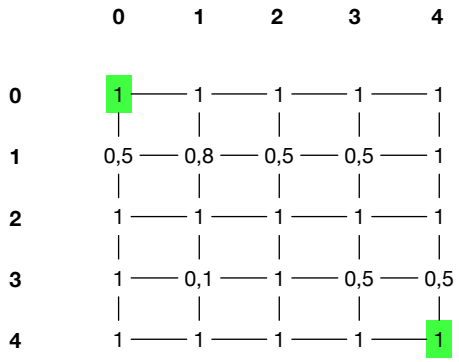

Figure 1 – Un champ de mines de taille 5

Le but de l'exercice est de trouver un chemin $P = p_0 \dots p_k$ allant de $p_0 = (0, 0)$ à $p_k = (n - 1, n - 1)$ permettant de sortir du champ de mines en maximisant ses chances de survie.

Les contraintes de déplacement sont :
- on doit se déplacer à chaque étape,
- on ne peut se déplacer que d'une case à chaque étape,
- on ne peut se déplacer que horizontalement ou verticalement : si $p_i = (l_i, c_i)$ et $p_{i+1} = (l_{i+1}, c_{i+1})$, alors $|l_i - l_{i+1}| + |c_i - c_{i+1}| = 1$ pour tout $0 \leq i < k$.

La figure 2 montre un chemin de $k = 8$ déplacements dont la probabilité d'arriver sain et sauf est de $\Pi_{0 \leq i < k} M[l_i][c_i] = 0.5 \times 0.1 \times 0.5 = 2.5\%$.

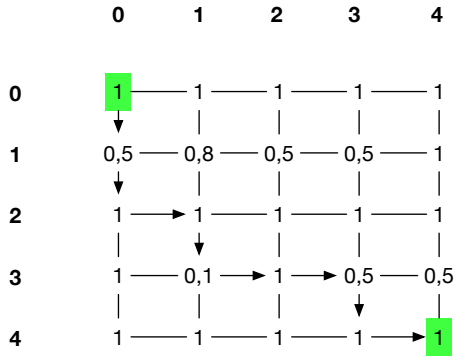

Figure 2 – Chemin $(0, 0) - (1, 0) - (2, 0) - (2, 1) - (3, 1) - (3, 2) - (3, 3) - (4, 3) - (4, 4)$

1.1 Exemple

Question 1.1.1 Montrer sur l'exemple de la figure 2 qu'il existe un chemin permettant d'atteindre la sortie du champ de mines avec une probabilité de 1.

Question 1.1.2 Montrer sur l'exemple de la figure 2 que si l'on ne peut pas revenir en arrière (à chaque déplacement soit la ligne soit la colonne augmente strictement), la probabilité de succès maximale n'est plus que de 80%.

1.2 Validation des chemins

Un chemin est dit *valide* s'il respecte les contraintes de déplacement (on se déplace d'une ligne ou d'une colonne à chaque fois, en avant ou en arrière), commence au départ du champ et se termine à sa sortie.

Question 1.2.1 Montrer que l'on peut supposer sans perte de généralité qu'un chemin ne repasse jamais deux fois par la même case. Quelles sont les longueurs minimales et maximales d'un chemin permettant de sortir du champ de mines ?

Question 1.2.2 Proposez un algorithme dont vous donnerez les paramètres, le code et la complexité permettant savoir si un chemin passé en paramètre est valide ou non.

Question 1.2.3 Proposez un algorithme dont vous donnerez les paramètres, le code et la complexité permettant de connaître la probabilité de survie d'un chemin passé en paramètre respectant les contraintes de déplacement.

1.3 On avance

On rajoute une contrainte de déplacement supplémentaire : on ne peut pas reculer. C'est à dire que si $p_i = (l_i, c_i)$ et $p_{i+1} = (l_{i+1}, c_{i+1})$, alors $l_i \leq l_{i+1}$ et $c_i \leq c_{i+1}$ pour tout $0 \leq i < k$.

Question 1.3.1 Proposez un algorithme dont vous donnerez les paramètres, le code et la complexité permettant de savoir si un chemin passé en paramètre respecte les contraintes de déplacement.

On note S la matrice carrée à n lignes telle que $S[l][c]$ soit la probabilité de survie maximale d'un chemin allant de $(0, 0)$ à (l, c).

Question 1.3.2 En utilisant le principe de la programmation dynamique que vous explicitez, donnez une équation permettant de calculer $S[l][c]$ à partir des éléments $S[l'][c']$ avec $0 \leq l' < l < n$ et $0 \leq c' < c < n$.

Question 1.3.3 En déduire un algorithme dont vous expliciterez les paramètres, le code et la complexité permettant de trouver $S[n-1][n-1]$.

Question 1.3.4 Explicitez en quelques lignes comment faire pour retrouver un chemin maximisant la probabilité de sortie du champ de mines en connaissant la matrice S.

1.4 Cas général

La contrainte supplémentaire ajoutée dans la partie précédente est supprimée : on peut atteindre chaque case (l, c) du champ par ses 4 voisins ($(l-1, c)$, $(l+1, c)$, $(l, c-1)$ et $(l, c+1)$).

Question 1.4.1 Montrez que l'on ne peut plus utiliser l'algorithme de la partie précédente pour résoudre le problème.

On note S la matrice tri-dimensionnelle carrée à n lignes telle que $S[k][l][c]$ soit la probabilité de survie maximale d'un chemin d'au plus k déplacements allant de $(0, 0)$ à (l, c).

Question 1.4.2 Quelles sont les valeurs de $S[0][l][c]$ pour $0 \leq l, c < n$?

Question 1.4.3 Quelle case de la matrice tri-dimensionnelle S correspond à la probabilité maximale de survie pour sortir du champ de mines ?

Question 1.4.4 En utilisant le principe de la programmation dynamique, donnez une équation permettant de calculer $S[k][l][c]$ à partir des éléments $S[k-1][l'][c']$ avec $0 \leq l', l < n$ et $0 \leq c', c < n$.

Question 1.4.5 En déduire un algorithme dont vous expliciterez les paramètres, le code et la complexité permettant de trouver la probabilité maximale de sortie d'un champ de mines.

Question 1.4.6 Donner un algorithme dont vous expliciterez les paramètres, le code et la complexité permettant de renvoyer un chemin de probabilité maximale de sortie d'un champ de mines

Exercice 2 – Fichiers et dossiers d'un disque dur

On souhaite créer des classes pour manipuler les fichiers et dossiers présents sur un disque dur.

On rappelle qu'un disque dur est organisé selon une arborescence et contient des éléments pouvant être des *dossiers* ou des *fichiers*. L'élément de départ, nommé racine, est un dossier et par convention s'appelle "/". Les dossiers sont des éléments permettant de stocker d'autres éléments (dossiers ou fichiers) et les fichiers sont des éléments qui peuvent être lus (des textes ou des images par exemple) ou exécutés (le programme python par exemple).

La figure 3 montre un exemple de disque dur :

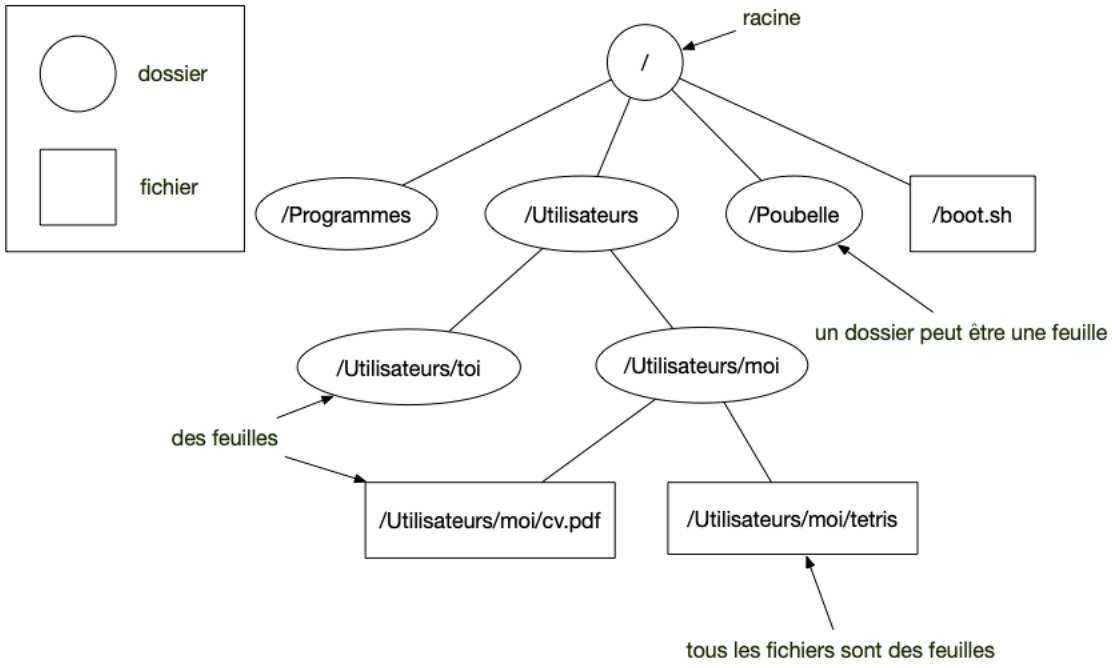

Figure 3 – Les dossiers et fichiers d'un disque dur.

On utilise souvent une métaphore familiale pour désigner les éléments de cette arborescence : les *enfants* d'un dossier sont les éléments qu'il contient (fichier ou dossier) et chaque élément différent de la racine contient un unique dossier *parent*. Par exemple pour l'arborescence de la figure 3 : les enfants de la racine sont les dossiers `Programmes`, `Utilisateurs` et `Poubelle`, ainsi que le fichier `boot.sh` ; le père du dossier `moi` est le dossier `Utilisateurs`.

On supposera ici que le nom d'un fichier ou d'un dossier est uniquement composé de lettres de l'alphabet, des caractères `"."` et `"_"` ; sauf le dossier racine dont le nom est `"/"`.

On peut représenter un endroit dans le disque dur par un *chemin* qui est une chaîne de caractères reprenant les noms des différents dossiers permettant d'aller de la racine à l'endroit voulu séparé par des `"/"`. Ainsi la chaîne `"/Utilisateurs/toi"` correspond à l'élément de nom `"toi"` et de chemin `"/Utilisateurs/"` (le fichier est stocké dans le dossier de nom `"Utilisateurs"` lui-même stocké dans la racine).

2.1 Modèle UML

On veut dans cette partie faire une modélisation UML de cette organisation.

Question 2.1.1 Proposez une modélisation UML des classes `Fichier` et `Dossier`. On doit pouvoir :

— créer un fichier ou un dossier grâce à son nom et à son dossier parent (la racine sera créée avec un dossier parent valant `None`).
— connaître le nom du fichier ou du dossier,
— connaître le chemin du dossier ou du fichier,
— connaître le dossier parent du fichier ou du dossier,
— connaître tous les éléments stockés dans un dossier,
— savoir si un fichier est un programme exécutable (comme le programme python) ou non (comme un fichier texte par exemple). Pour cela, un `Fichier` doit avoir une méthode `est_exécutable()`.

Question 2.1.2 En supposant que les classes `Fichier` et `Dossier` sont dans le fichier `"disque_dur.py"`, écrivez le code permettant de créer l'exemple de la figure 3 (on ne demande pas d'écrire le code des classes ici, mais seulement de les utiliser, en supposant qu'elles existent).

Question 2.1.3 Certaines fonctionnalités sont identiques entre les classes `Fichier` et `Dossier`. Comment faire pour ne pas avoir à re-écrire plusieurs fois le même code ?

2.2 Code des classes

Question 2.2.1 Écrivez le constructeur de la classe `Dossier`.

Question 2.2.2 Écrivez la méthode permettant de connaître le chemin d'un `Dossier` ou d'un `Fichier` en utilisant les attributs et méthodes que vous avez décrits à la question 2.1.1. Par exemple, pour le fichier de nom `"cv.pdf"` de la figure 3, la méthode rendra la chaîne de caractères `"/Utilisateurs/moi/cv.pdf"`.

Question 2.2.3 Donnez le code complet de la classe `Fichier`.

2.3 Énumération

Question 2.3.1 Écrivez une fonction permettant d'afficher un à un le nom de chaque élément (`Dossier` et `Fichier`) d'une arborescence. Cette fonction prendra un `Dossier` en paramètre et pourra être récursive.

Question 2.3.2 Explicitez la complexité de la fonction de la question précédente.

Question 2.3.3 Dans quel ordre sont parcourus les éléments de l'arborescence pour l'algorithme de la question 2.3.1 ? Explicitez ce parcours sur l'exemple de la figure 3.

Exercice 3 – Fenêtres bariolées

Soit K un entier strictement positif et $s = s_0 \dots s_{n-1}$ une liste de n entiers telle que $\{s_i | 0 \leq i < n\} = \{0, \dots, K-1\}$. Une *fenêtre bariolée de* s est une sous-liste $s[i : j+1] = s_i \dots s_j$, avec $0 \leq i \leq j < n$, contenant tous les entiers de 0 à $K-1$ ($\{s_l | 0 \leq i \leq l \leq j < n\} = \{0, \dots, K-1\}$). Notez que K est définie pour la liste initiale, pas la sous liste.

3.1 Sous-liste

Soit `tous(K, liste)` une fonction qui rend `True` si la liste `liste` contient tous les entiers entre 0 et K-1 (inclus).

Question 3.1.1 Proposez un algorithme qui implémente `tous(K, liste)`. Vous expliciterez sa complexité et donnerez le nombre de cases mémoires utilisées.

Question 3.1.2 Proposez un algorithme nommé `ok_K(l)` prenant en paramètre une liste $l = l_0 \dots l_{n-1}$ et rendant 'True' s'il existe K tel que $\{l_i | 0 \leq i < n\} = \{0, \dots, K-1\}$. Vous utiliserez `tous(K, liste)` pour créer cet algorithme. Vous expliciterez sa complexité.

Question 3.1.3 Soit `suite` une liste telle que `ok_K(suite) == True`. Donner le code et la complexité d'une fonction `fenêtre(i, j, suite)` rendant `True` si liste$[i : j+1]$ est une fenêtre bariolée de `suite`.

3.2 Glouton

Question 3.2.1 Proposez un algorithme glouton dont vous donnerez les paramètres, le code et la complexité permettant de trouver une fenêtre bariolée $s[0 : j+1]$ à partir d'une liste s telle que `tous(K, s) == True` telle que $s[0 : j]$ ne soit pas bariolée.

Vous expliciterez pourquoi cet algorithme est glouton, pourquoi il trouve bien une fenêtre bariolée et pourquoi elle est minimale.

Question 3.2.2 Proposez un algorithme glouton prenant une suite s et un entier $i^\star$ en paramètres et rendant l'entier j le plus petit possible pour que $s[i^\star : j+1]$ soit une fenêtre bariolée de s. Vous donnerez également la complexité de cet algorithme.

Question 3.2.3 En déduire un algorithme dont vous donnerez les paramètres, le code et la complexité pour déterminer la plus petite longueur d'une fenêtre bariolée d'une liste s telle que `tous(K, s) == True`.

3.3 Optimal

On cherche maintenant à trouver une fenêtre bariolée $s[i : j+1]$ de taille $(j+1-i)$ minimum le plus rapidement possible.

Question 3.3.1 Montrez que la complexité du problème de la recherche d'une fenêtre bariolée à partir de s et de K (K est donné et s est telle que $\{s_i | 0 \leq i < n\} = \{0, \dots, K-1\}$) est au moins en $\mathcal{O}(n)$.

Question 3.3.2 Donnez une condition nécessaire et suffisante impliquant $s[i]$ pour que $s[i+1 : j+1]$ soit bariolée si $s[i : j+1]$ l'est.

Question 3.3.3 En déduire un algorithme dont vous donnerez les paramètres et le code, de complexité $\mathcal{O}(n)$ et nécessitant $\mathcal{O}(K)$ cases mémoires pour résoudre le problème de trouver une fenêtre bariolée de taille minimum.