clement.rouvroy@ens.psl.eu

Interview GAFAM

Consignes Le candidat respectera le langage imposé (C) et ne devra pas utiliser de librairie hors-programme. Il est invité à faire des schémas clairs et précis de ses algorithmes lors des appels au correcteur, c'est à la fois un gain de temps pour les deux et une manière de montrer qu'il a compris ce qu'il manipule. Les preuves écrites doivent être formelles, sauf si la consigne précise que ce n'est pas nécessaire, la rigueur sera évaluée. L'examinateur ne débugera pas votre code, en revanche en cas de doute ("ai-je le droit à telle ou telle librairie ?", "je suis sortie de la VM, comment y revenir ?", "ai-je le droit à une indication pour cette question ?") n'hésitez pas à poser votre question. Elle ne vous dévalorisera pas, si la réponse peut vous retirer des points, l'examinateur vous demandera avant de vous donner la réponse si vous l'acceptez (*si vous n'avez pas de chance, le jour de l'oral il vous retirera les points rien que pour avoir posé la question, par exemple si vous demandez "comment trier une liste en* $O(n\log(n))$*?" ou un autre résultat classique du programme, ça sera sûrement retenu contre vous*). Enfin, si l'examinateur n'est pas à votre portée quand vous avez besoin d'une aide / d'une question oral à donner, gardez le bras levé et lisez la suite, ne restez jamais passif.

Introduction du sujet

Ce sujet porte entièrement sur la programmation dynamique et va vous faire traiter différents exercices, majoritairement tirés de *Algorithmique* (par Cormen, Leiverson, Rivest et Stein) et du site *leetcode.com* sur lequel des prétendants à des grandes entreprises postent les exercices qu'ils ont eu durant leur entretien (un équivalent à votre *BEOS / prepas.org*). Vous êtes libres de traiter les exercices dans l'ordre que vous voulez.

Durant tout le TP vous avez accès à une fonction `eval_exercice_[NUMERO EXO]` qui évalue votre score, merci de ne pas toucher aux signatures des fonctions.

I Avant que ça commence

Question 1 à l'écrit

(Annulé car vous avez vu les graphes non-orientés comme des doubles pairs) 1. Est-ce que dans un graphe orienté il peut y avoir une arête d'un sommet à lui-même ? Dans un graphe non-orienté ?

1. Quel est l'avantage d'encoder un graphe par liste d'adjacences ? Par matrice d'adjacences ?
2. a) Ecrire un pseudo-code de parcours en profondeur pour un graphe non-orienté.

   b) Comment savoir grace à ce parcours si le graphe est cyclique ?

II Découpe de barres (5 points)

Etant donné une barre de longueur $n \in \mathbb{N}$ et un ensemble de couples $(\text{longueur}, \text{prix})$ on veut trouver un découpage de la barre qui *maximise le prix de la barre*

Question 2 code

Compléter la fonction `int maximise(int n, int nb_points, couple* materiaux)`

III Escalier à difficulté variable (7 points)

Vous êtes en direction de l'X pour vos oraux (même si vous préférez *évidemment* l'ENS) et vous tombez face à face aux escaliers de Lozères: vous êtes avec votre valise, vous êtes bien habillés pour vos oraux et vous devez faire une montée de 15 minutes en pleine chaleur parce que vos oraux sont évidemment début Juillet et qu'il fait +30 degrés sur le plateau (*oui c'est du vécu*). Vous ne voulez pas arriver plein de sueurs, heureusement vous êtes infoteux et vous pouvez donc utiliser la programmation dynamique:

A chaque marche, vous pouvez monter d'une ou deux marches d'un coup. Chaque marche a un coût (`cout[i]` est le coût de la $i$ème marche), le but est de minimiser le coût de la montée sachant qu'on doit toujours finir sur la dernière case. Par exemple:

- Entrée: [10,15,20]
- Sortie: 15

On décide de commencer par la deuxième marche et de ne payer que 15.

- Entrée: [1,5,5,10]
- Sortie: 6

Je vous laisse comprendre pourquoi (on peut sauter directement après le 10 depuis le deuxième 5).

Question 3 code

Compléter la fonction `int minimum_marches(int n, int* couts)` pour résoudre le problème.

IV Tribonacci (5 ou 10 points)

Soit la suite $(T_n)_{n\in\mathbb{N}}$ définie par:

$$T_0 = 0, T_1 = 1, T_2 = 1 \quad \text{et } T_{n+3} = T_{n+2} + T_{n+1} + T_n$$

Question 4 code

Compléter la fonction `int tribonacci(int n)` qui renvoie T_n pour $n \leq 37$. (5 points)

Pour (10 points) vous pouvez faire `tribonacci` *sans créer de tableaux de mémoïsation* (et toute autre structure de mémoïsation).

V Distance de Levenshtein modifiée (10 points)

Question 5 code

Compléter la fonction `int distance(char* m1, char* m2, int taille_m1, int taille_m2)` pour qu'elle renvoie la distance que vous trouvez (on considère que toutes les opérations valent 1, en particulier supprimer coûte autant qu'ajouter qui coûte autant que remplacer).

VI Eau Minecraft 2D (20 points)

Si vous jouez à Minecraft vous vous êtes forcément demandés (quoi, vous ne l'avez pas fait ?) comment le jeu faisait pour savoir si il doit prolonger ou non l'eau qui est sur un bloc (x, y, z) sur les blocs autour (pour ne pas que l'eau traverse des blocs durs / à l'inverse qu'elle soit bloquée par une barrière invisible).

Cet exercice vous demande d'implémenter l'algorithme en 2D (j'ai fait la version 3D, c'est la même en moins drôle).

Vous avez en entrée un tableau de n entiers tel que l'entier i corresponde à la hauteur du mur en coordonnée $x = i$. Vous devez *renvoyer le nombre de cases qui peuvent être remplies par de l'eau sans détruire un des trois principes fondamentaux de la physique.*

Par exemple pour l'entrée `hauteurs = [0,1,0,2,1,0,1,3,2,1,2,1]`, on a la solution suivante:

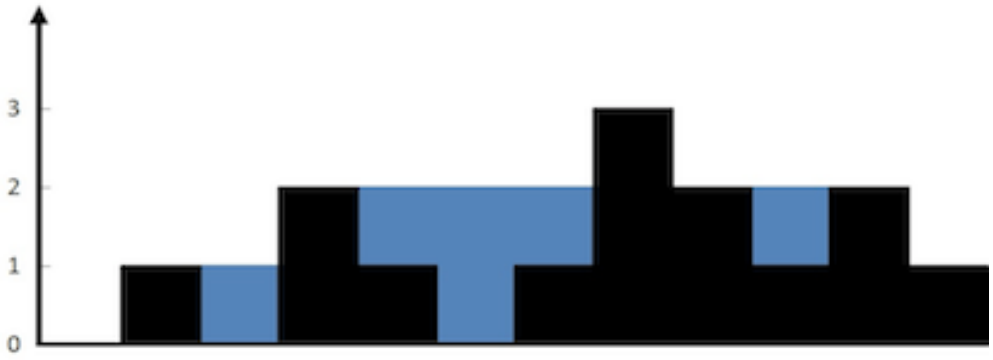

Figure 1: Solution renvoyée = 6

Question 6 code

Compléter la solution `int eau_minecraft(int n, int* hauteurs)`

VII Matching d'expressions régulières (30 points)

Vous verrez les expressions régulières l'année prochaine. Cependant vous pouvez tout de même coder des algorithmes dessus dès à présent. Voici un petit point de cours:

Definition 1 (fragment d')expression régulière

Une expression régulière e est soit:
- Le *mot vide*, noté ε.
- Une *lettre* ($a - z$ / $A - Z$).
- Un . qui signifie "*n'importe quelle lettre* " (il n'y a pas de S ici à "lettre").
- Une $*$ qui signifie "*répéter autant de fois que voulue le caractère qui précède*".

On peut donc voir une *expression régulière comme un ensemble de mots*. A savoir l'ensemble des mots que l'on peut obtenir à partir de l'expression. Voici quelques exemples:
- "coucou" est une expression régulière dont on ne peut obtenir *que* coucou.
- "." est une expression régulière dont on peut obtenir tous les mots *formés d'une seule lettre*
- "a*" peut-être vu comme $\bigcup_{n\in\mathbb{N}} a^n$ avec a^n le fait d'avoir n `a` collés. Par exemple $a^3 = aaa$
- ".*" représente *tous les mots* de l'alphabet car on répète autant de fois que l'on veut "." qui représente n'importe quelle lettre.
- "a**" peut dériver le même ensemble de mots que "a*".

Question 7 code

Compléter la fonction `bool isMatch(char* s, char* p)` qui renvoie vrai si et seulement si s peut-être obtenue depuis p.