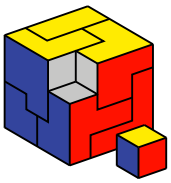
Prologin 2024



Concours national d'informatique

Épreuve écrite d'algorithmique
Angers, Lyon I, Rennes, Strasbourg

04 Février 2024

Labyrinthes

1 Préambule

Bienvenue à **Prologin**. Ce sujet est l'épreuve écrite d'algorithmique et constitue la première des trois parties de votre épreuve régionale, sa durée est de 3 heures. Pendant cette épreuve, vous passerez un entretien (10 minutes) avec un organisateur. Ensuite, vous aurez une épreuve de programmation sur machine (3 heures et 30 minutes) l'après-midi.

Conseils

— Lisez bien tout le sujet avant de commencer.

— **Soignez la présentation** de votre copie.

— N'hésitez pas à poser des questions.

— Si vous avez fini en avance, relisez bien.

— N'oubliez pas de passer une bonne journée.

Remarques

— Le barème est donné à titre indicatif uniquement.

— Indiquez lisiblement vos nom et prénom, la ville où vous passez l'épreuve et la date en haut de votre copie.

— Lorsqu'un algorithme est demandé, vous pouvez le décrire avec suffisamment de précision, le pseudo-coder ou l'implémenter dans le langage de votre choix. Dans le dernier cas, veuillez néanmoins préciser le langage que vous utilisez.

— Ce sont des humains qui lisent vos copies : laissez une marge, aérez votre code, ajoutez des commentaires (**seulement** lorsqu'ils sont nécessaires) et évitez au maximum les fautes d'orthographe.

— Le barème récompense les algorithmes les plus efficaces : écrivez des fonctions qui trouvent la solution le plus rapidement possible.

— Si vous trouvez le sujet trop simple, relisez-le, réfléchissez bien, puis dites-le-nous, nous pouvons ajouter des questions plus difficiles.

2 À propos du sujet

Ce sujet est composé de quatre parties indépendantes :

— Générer des labyrinthes simplement (23 points)

— Résoudre des labyrinthes avec la stratégie du parcours main-droite (28 points)

— Utiliser l'algorithme de Kruskal pour générer des labyrinthes (28 pts)

— Analyse amortie de la complexité d'Union-Find avec la fonction d'Ackermann (21 points)

2.1 Introduction

Bienvenue dans l'antique Crète, sous le règne de Minos. Suite à une étrange union entre un taureau et sa femme, naquit le Minotaure.
Pour emprisonner cette créature redoutable, Minos fit appel à Dédale, un artisan de renom dans le but qu'il crée un labyrinthe tel que le Minotaure ne pourra en sortir.

3 Générer des labyrinthes

3.1 Contexte

Après la demande de Minos, Dédale fut chargé de concevoir un labyrinthe destiné à emprisonner le redoutable Minotaure. Heureusement pour nous, une équipe d'espions de Prologin est parvenue à mettre la main sur les précieux plans du labyrinthe.
Cependant, la tâche qui nous attend est complexe, car ses plans sont incomplets. Au cœur de cette première partie, notre objectif sera de décortiquer le fonctionnement du labyrinthe et de combler les lacunes laissées par ces plans fragmentés.

3.2 Les matrices

Ce sujet parle de matrices : ce sont des tableaux à 2 dimensions, des tableaux de tableaux.
Pour T une matrice, on notera $T[y][x]$ l'élément à la $(y+1)$-ème ligne en partant du haut et la $(x+1)$-ième colonne en partant de la gauche.
On dit qu'une matrice est de taille $N \times M$ si elle a N lignes et M colonnes.

On note (x, y) la position de l'élément à la $(x+1)$-ème colonne et la $(y+1)$-ème ligne. On note donc (0,0) la position de l'élément à la première colonne et à la première ligne.

3.3 Le cadre du sujet

Question 1 (1 point)

Combien de cases une matrice à N lignes et M colonnes a-t-elle ?

Question 2 (2 points)

On considère l'algorithme suivant qui prend une matrice T de taille $N \times M$ et qui teste si T n'est composé que de 1 :

Entrée : T un tableau de taille $N \times M$
Sortie : `VRAI` si T n'est composé que de 1, `FAUX` sinon

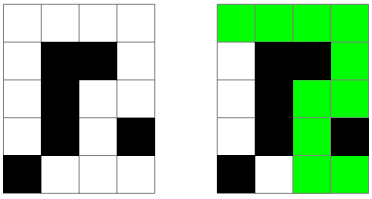
pour y allant de 0 inclus à M exclus **faire**
    pour x allant de 0 inclus à N exclus **faire**
        si $T[y][x] \neq 1$ **alors**
            renvoyer `FAUX`
        fin si
    fin pour
fin pour
renvoyer `VRAI`

Corriger l'erreur dans l'algorithme.

Ici notre but est de générer un labyrinthe dans un tableau 2D. On appelle labyrinthe une matrice de deux couleurs où l'on colorie en blanc les cases valant 0, et en noir les autres. Un labyrinthe ne peut jamais avoir la case $(0, 0)$ noire.

On dit qu'un labyrinthe est *résoluble* s'il existe un chemin (appelé *solution*) de la case $(0, 0)$ à la case $(M-1, N-1)$ composé uniquement de cases blanches et où on ne se déplace que dans des cases directement adjacentes (haut, bas, gauche ou droite). Ce chemin ne peut pas revenir sur lui-même.

Par exemple, le labyrinthe de gauche est résoluble (à droite l'unique solution, dessinée en vert) :



Question 3 (1 point)

Dessiner un labyrinthe non résoluble.

Question 4 (2 points)

On considère le labyrinthe où toutes les cases dont les coordonnées x et y sont impaires sont des murs. Dessiner ce labyrinthe pour $N = 7$ et $M = 5$.

Question 5 (4 points)

Donner une condition nécessaire et suffisante sur N et M pour qu'il existe une solution au labyrinthe vide (composé que de 0) de taille $N \times M$ passant par toutes les cases.

3.4 Génération de labyrinthes

On considère une table initialement remplie de tous les entiers de 0 à $N \times M - 1$ dans l'ordre de lecture (exemple ci-dessous pour une table de taille 6×5)

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14
15	16	17	18	19
20	21	22	23	24
25	26	27	28	29

Question 6 (3 points)

Proposer le code d'une fonction `initialisation` qui prend une matrice T remplie de 0 de taille $N \times M$ et qui renvoie la matrice remplie dans l'ordre de lecture comme dans l'exemple.

Question 7 (3 points)

On considère la fonction `distincts` suivante, qui prend une matrice T et renvoie un couple (a, b) tel que $a \in T$, et $b \in T$, et $a \neq b$ ou $(-1, -1)$ si T n'est composé que d'un seul nombre :

Entrée : T un tableau d'entiers de taille $N \times M$
Sortie : (a, b) s'il existe deux nombres différents dans T et $(-1, -1)$ sinon.

pour y allant de 0 à N exclus **faire**
  pour x allant de 0 à $M - 1$ exclus **faire**
    si $T[y][x] \neq T[y][x+1]$ **alors**
      renvoyer $(T[y][x], T[y][x+1])$
    fin si
  fin pour
fin pour
renvoyer $(-1, -1)$

Prouver que cet algorithme est incorrect en apportant un contre-exemple, puis proposer un correctif.

Question 8 (3 points)

Proposer le code d'une fonction `echange` qui prend en argument deux entiers a et b et une matrice T de taille $N \times M$ et qui renvoie la même matrice où tous les a ont été remplacés par des b.

Par exemple, `echange(2,6,T)` avec T la matrice de gauche devra donner la matrice de droite (en rouge les cases modifiées) :

0	1	2	3
1	2	3	4
2	3	4	5
3	4	5	6
4	5	6	7

0	1	6	3
1	6	3	4
6	3	4	5
3	4	5	6
4	5	6	7

Question 9 (4 points)

On considère l'algorithme suivant, en utilisant les fonctions `initialisation` telle que définie à la question 6 et `echange` et `distincts` définies dans les questions précédentes :

Entrée : Deux entiers N, M positifs
Sortie : Un labyrinthe T

Soit T un tableau de 0 de taille $N \times M$
`initialisation(`T`)`
tant que `VRAI` **faire**
  Soit k un nombre choisi uniformément au hasard entre 1 et $N \times M - 1$
  Soit $T' =$ `echange(`k`,0,`T`)`
  si `distincts(`T'`)` $= (-1, -1)$ **alors**
    renvoyer T
  sinon
    Mettre T à T'
  fin si
fin tant que

Est-ce que l'algorithme ne renvoie que des labyrinthes ?
Donner un labyrinthe résoluble et un labyrinthe non résoluble pouvant être généré par cet algorithme pour $N = 3$ et $M = 4$. Quelle est la probabilité que le labyrinthe renvoyé soit non résoluble ?

4 Résoudre des labyrinthes

4.1 Contexte

Un jour, Dédale expliqua à Ariane comment sortir du labyrinthe en utilisant un fil de laine.
Malheureusement, cette information fut partagée avec Thésée, qui par la suite tua le redoutable Minotaure.
Apprenant la nouvelle, Minos décida de punir Dédale en l'enfermant dans le labyrinthe qu'il avait lui-même conçu sans fil afin qu'il ne puisse pas sortir. Dans cette deuxième partie, nous allons aider Dédale à s'échapper grace à une technique imparable (la technique de la main droite)

4.2 Objectif

Le but de cette partie est d'analyser les solutions à des labyrinthes. On se concentre d'abord sur créer une fonction capable de résoudre un labyrinthe en utilisant la technique de la main droite, avant d'essayer de réduire le nombre de solutions à un labyrinthe.
Dans cette partie, les matrices des labyrinthes ne seront composées que de 0 et de 1.

4.3 La méthode de la main droite

La méthode de la main droite consiste à suivre le mur dit « toujours du même côté », c'est-à-dire en gardant un mur à sa droite tout au long du parcours, ce qui permet de suivre un chemin continu sur le bord jusqu'à la sortie du labyrinthe. Elle nous permettra toujours de trouver une sortie : en suivant le mur du même côté, on suit le contour du labyrinthe, et donc s'il y a une ouverture, on la trouvera.

Dans cette partie, l'on considère que l'on crée 4 variables représentant 4 directions possibles :

1. la variable `bas` valant 0
2. la variable `droite` valant 1
3. la variable `haut` valant 2
4. la variable `gauche` valant 3

Une *direction* est donc un entier parmi $\{0, 1, 2, 3\}$.

Question 10 (2 points)

Donner le code d'un algorithme `rotation_droite` et d'un algorithme `rotation_gauche` respectivement qui prend en argument une direction et qui renvoie la direction obtenue après rotation de 90° dans le sens horaire (sens des aiguilles d'une montre), respectivement anti-horaire (sens inverse au sens des aiguilles d'une montre).

Question 11 (3 points)

Donner le code d'un algorithme `deplacement` qui prend en argument un labyrinthe T de taille $N \times M$, une position (x, y) et une direction d et qui renvoie un couple (x', y') de la nouvelle position obtenue après avoir avancé dans la direction d depuis (x, y) si le déplacement est valide. Si le déplacement est invalide on renverra $(-1, -1)$.
On fera bien attention à ce que le coup ne nous fasse pas sortir en dehors du labyrinthe.

Question 12 (4 points)

On considère maintenant l'algorithme du parcours de la main droite suivant, celui-ci peut revenir sur ses pas :

Entrée : Un labyrinthe T
Sortie : `VRAI` si le labyrinthe est résoluble et `FAUX` sinon

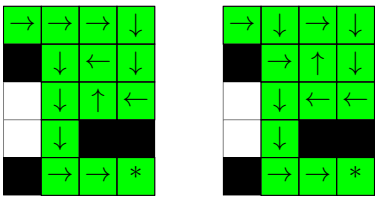
Soient $x = 0$, $y = 0$
Soit $d =$ `bas`
tant que `VRAI` **faire**
  si $(x, y) = (M - 1, N - 1)$ **alors**
    renvoyer `VRAI`
  fin si
  tant que `deplacement`$(T,\ x,\ y,\ d) = (-1, -1)$ **faire**
    $d \leftarrow$ `rotation_gauche`(d)
    si $(x, y) = (0, 0)$ et $d =$ `bas` **alors**
      renvoyer `FAUX`
    fin si
  fin tant que
  $(x, y) \leftarrow$ `deplacement`$(T,\ x,\ y,\ d)$
fin tant que

Prouver que cet algorithme est incorrect en apportant un contre-exemple, puis proposer un correctif.

4.4 Compter des solutions

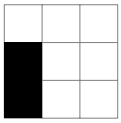
On sait maintenant résoudre des labyrinthes ! On s'intéresse donc à présent à compter le nombre de solutions différentes d'un labyrinthe, et à essayer de rendre notre labyrinthe plus difficile en réduisant le nombre de solutions qui existent.

Une solution est un ensemble de couples (x, y) par où passe le chemin, si bien que deux chemins différents passant par les même cases vont être considérés comme une seule même solution.
Par exemple, ces deux chemins allant de $(0, 0)$ à $(3, 4)$ seront considérés comme étant une même solution, car ils passent par les mêmes cases :



Question 13 (1 point)

Combien de solutions distinctes possède ce labyrinthe ?



Question 14 (4 points)

Donner un labyrinthe résoluble de taille 3×3 dont le nombre de solutions n'est pas une puissance de 2.
Notre but va être de créer un algorithme qui prend un labyrinthe résoluble en entrée et qui rajoute des murs jusqu'à ce qu'il n'y ait plus qu'une unique solution.

Question 15 (5 points)

En vous inspirant du code de la question 12 (résoudre un labyrinthe avec la technique de la main droite), proposer le code d'une fonction `main_droite(T)` qui prend en argument un labyrinthe T de taille $N \times M$ et qui renvoie la solution calculée avec la technique de la main droite. Il est à noter qu'une solution ne peut pas revenir sur ces pas, il faudra faire attention à retirer de la solution des cases sur lesquelles on est déjà passé si tel est le cas.
La solution sera renvoyée sous la forme d'un tableau de taille $N \times M$ contenant des 1 à chaque case empruntée par la solution.

Question 16 (4 points)

On suppose maintenant que l'on a aussi codé une fonction `main_gauche(T)` qui renvoie la solution calculée avec la technique de la main gauche (on suit toujours le mur de gauche)
Proposer une condition nécessaire et suffisante sur un labyrinthe pour qu'il y ait unicité de la solution.
Donner le code d'une fonction `est_unique(T)` qui prend en argument un labyrinthe T et renvoie `VRAI` ou `FAUX` selon si le labyrinthe ne possède qu'une unique solution.

Question 17 (3 points)

On considère le code suivant :

Entrée : Un labyrinthe T
Sortie : Un labyrinthe T' construit à partir de T où l'on a rajouté des cases pour n'avoir qu'une seule solution

tant que non `est_unique(T)` **faire**
    Soit S_1 = `main_droite(T)`
    Soit S_2 = `main_gauche(T)`
    Soit (x, y) une case de S_1 qui n'est pas dans S_2
    Mettre $T[y][x]$ à 1
fin tant que

Prouver que cet algorithme est incorrect en apportant un contre-exemple, puis proposer un correctif.

Question 18 (2 points)

Prouver que le programme corrigé termine.

5 Union-Find pour générer des labyrinthes

5.1 Contexte

Deux personnes ayant réussi à s'échapper du labyrinthe, la rumeur disant que le labyrinthe est un échec est remontée jusqu'aux oreilles des dieux nordiques.
En quête d'une solution plus robuste, ils font désormais appel à nous pour concevoir un labyrinthe dont nul ne pourra en sortir.

5.2 Union-Find

Union find est une structure qui permet de représenter des groupes d'objets que l'on veut rendre équivalents.
On aura 2 opérations possibles :

1. **Union :** qui permet de rassembler deux groupes
2. **Find :** qui permet de connaître le groupe d'un objet donné

Pour faire cela, on numérote chaque objet, et chaque objet aura un « parent ».
Les parents seront tous stockés dans un tableau `parent` tel que `parent[i]` est le parent de l'objet numéro `i`.
Certaines entrées de `parent` seront leur propre parent, on les appellera les « représentants ».
La fonction `find(i)` a pour but de renvoyer le représentant de l'objet `i`, c'est-à-dire de regarder la liste des parents de `i` jusqu'à trouver un représentant.
La fonction `union(x,y)` a pour but de mettre le représentant de 'x' comme un parent du représentant 'y' ou inversement.

5.3 Généralitées sur Union-Find

Question 19 (2 points)

On a le tableau de parents $[0, 2, 2, 4, 6, 1, 6]$. Pour chaque nombre, donner son représentant.

Question 20 (3 points)

Odin, pour rigoler, a demandé à Thor et Loki d'imiter le Minotaure dans le nouveau labyrinthe. Dessiner les deux dieux faisant ce qu'Odin leur a demandé.

Question 21 (4 points)

Donner une fonction en $O(n)$ **dans le pire des cas** qui prend en argument le tableau des parents et qui renvoie le nombre de représentants.

Question 22 (5 points)

Programmer la fonction `find` qui permet de trouver le représentant d'un nombre. On fera en sorte que chaque nœud par lequel la fonction est passée dans son parcours ait pour parent direct ce représentant après exécution. [1]

Question 23 (4 points)

On définit le rang d'un élément x comme la longueur du plus long chemin menant à x en suivant le même type de parcours que précédemment.
On stockera et on mettra à jour le rang des représentants dans un tableau `rang`.
Programmer la fonction `union` de façon à garder (si possible) le rang des représentants petit.

1. Il s'agit d'une optimisation appelée « Compression de chemin ». Sans cela, la complexité temporelle augmente.

5.4 Construction avec Kruskal

Nous allons maintenant construire un labyrinthe avec notre nouvelle structure, union find !
L'idée de l'algorithme va être de maintenir des groupe tel que si l'on peut atteindre la case y à partir de la case x, y et x sont dans le même groupe. A l'inverse, si on ne peut pas atteindre x à partir de y, ils ne sont pas dans le même groupe.
A chaque étape, on choisit un mur non-traité, s'il permet de relier 2 cases qui n'étaient pas reliées, on casse le mur. Sinon, le mur reste.

Question 24 (4 points)

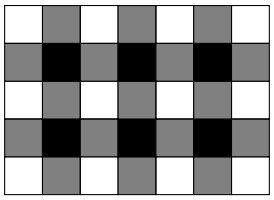
Montrer que la fonction suivantes permet bien de mélanger le tableau de manière uniforme.

procedure `MELANGER`(T, N)
  Entrée : Un tableau T et sa taille N
  Sortie : Le tableau T mélangé

  pour y `allant de` 0 `inclus à` N `exclus` **faire**
    Soit x un nombre aléatoire tiré uniformément entre 0 inclus et y inclus
    On inverse les cases y et x du tableau T
  fin pour
fin procedure

Question 25 (5 points)

Implémenter l'algorithme de Kruskal pour la construction de labyrinthe. On pourra partir d'un labyrinthe rempli de 1 sauf pour les cases (x,y) avec x,y pairs, et l'idée est que retirer certains murs séparant deux de ces cases revient à relier deux zones du labyrinthe, et donc d'effectuer une `Union`. Vous devez donc utiliser les fonctions `Union` et `Find` dans votre algorithme pour créer un labyrinthe ne possédant qu'une solution.
Le dessin suivant représente les labyrinthes qui pourront être générés. À la fin de votre algorithme, les cases blanches doivent être des cases libres, les cases noires doivent être des murs, et les cases grisent peuvent être les deux.



Question 26 (1 point)

Montrer qu'il ne peut exister aucun cycle dans un labyrinthe ainsi généré.

6 La complexité amortie d'Union-Find

On définit la fonction de Ackermann par récurrence :

$$A(m,n) = \begin{cases} n+1 & \text{si } m = 0 \\ A(m-1,1) & \text{si } m > 0 \text{ et } n = 0 \\ A(m-1, A(m,n-1)) & \text{sinon} \end{cases}$$

Voici un tableau décrivant les quelques premières valeurs[2] de $A(m,n)$:

m \ n	0	1	2	3
0	1	2	3	4
1	2	3	4	5
2	3	5	7	9
3	5	13	29	61

Question 27 (3 points)

Donner une expression explicite de $A(1,n)$, $A(2,n)$ et de $A(3,n)$ en fonction de $n \in \mathbb{N}$.

Question 28 (3 points)

On définit les puissances itérées de Knuth pour a, n, b des entiers positifs :

$$a \uparrow^n b = \begin{cases} a^b & \text{si } n = 1 \\ 1 & \text{sinon si } b = 0 \\ a \uparrow^{n-1} (a \uparrow^n (b-1)) & \text{sinon} \end{cases}$$

Cela permet de généraliser la notion de puissances. Par exemple, on peut maintenant écrire[4] $2^{2^{2^{2^2}}} = 2 \uparrow^2 5$.
On remarque depuis cette définition, que $a \uparrow^n b = a \uparrow^{n-1} (a \uparrow^{n-1} (a \uparrow^{n-1} (\dots a \uparrow^{n-1} a) \dots))$, avec b occurrences du nombre a
Montrez que, pour tout a et pour tout $n > 0$:

- $a \uparrow^n 1 = a$;
- $2 \uparrow^n 2 = 4$.

Question 29 (4 points)

Montrer que $A(m,n) = 2 \uparrow^{m-2} (n+3) - 3$ pour $m \geq 3$

Question 30 (4 points)

Soit α la fonction qui, à un nombre n, associe le plus petit nombre m tel que $A(m,1) \geq n$. On veut montrer que Union-Find a une complexité amortie[5] en $O(\alpha(n))$, pour n éléments. Dans la réalité, on aura donc souvent $\alpha(n) \leq 4$. On considère donc souvent Union-Find comme une structure qui fait des opérations en temps constant.
Soient :

- $A^i(m,n) = A(m, A(m, \dots A(m,n) \dots))$, i fois, c'est à dire : $A^{i+1}(m,n) = A(m, A^i(m,n))$ et $A^0(m,n) = 1$;
- $\texttt{niveau}(x) = \max\{k \in \mathbb{N}, \texttt{rang}(\texttt{parent}[\texttt{x}]) \geq A(k, \texttt{rang}(x))\}$;
- $\texttt{iter}(x) = \max\{i \in \mathbb{N}, \texttt{rang}(\texttt{parent}[\texttt{x}]) \geq A^i(\texttt{niveau}(x), \texttt{rang}(x))\}$.

Montrer que :

- $0 \leq \texttt{niveau}(x) < \alpha(n)$;
- $1 \leq \texttt{iter}(x) \leq \texttt{rang}(x)$.

2. D'à partir de la 4e colonne, les valeurs deviennent très larges[3]. Par exemple, $A(4,4) = 2^{2^{2^{65536}}} - 3$
3. En revanche, il faut attendre la 6e colonne avant d'atteindre la quantité de paracétamol qu'il me faudra pour remédier au mal de crâne causé par cette section.
4. Aussi couramment écrit $2 \uparrow\uparrow 5$, ou "pourquoi t'aurais besoin d'écrire un nombre aussi absurdement grand de toute façon ?"
5. la moyenne des complexités pour une suite d'opérations

Question 31 (2 points)

Après avoir effectué q opérations, on définit le potentiel d'un nœud x par :

$$\Phi(q,x) = \begin{cases} \alpha(n) \cdot \texttt{rang}(x) & \text{si } x \text{ est un représentant ou si } \texttt{rang(x)} = 0 \\ (\alpha(n) - \texttt{niveau}(x)) \cdot \texttt{rang}(x) - \texttt{iter}(x) & \text{sinon} \end{cases}$$

A l'aide de la question précédente, justifier que l'on a cet encadrement de $\Phi(q,x)$:

$$0 \leq \Phi(q,x) < \alpha(n) \cdot \texttt{rang}(x)$$

Question 32 (5 points)

Soit $\Psi(q) = \sum_x \Phi(q,x)$

Utilisez Ψ pour prouver que la complexité amortie d'une opération d'Union-Find est $O(\alpha(n))$

7 Questions Bonus

N'abordez ces questions que si vous vous êtes relu 42 fois.

Question bonus 33 (1 point)

Évitez de ne pas ignorer cette question pour ne pas perdre -1 point. [6]

Question bonus 34 (8 points)

Écrivez deux chiffres décimaux distincts (entre 0 et 9, donc). Vous remportez le minimum des deux chiffres en nombre de points, seulement si personne n'a écrit les même deux chiffres.

Question bonus 35 (1 point)

Écrivez un algorithme pour déterminer si un nombre est pair ou impair. Si votre algorithme est identique à celui d'un autre dans la salle, vous n'avez pas le point

Question bonus 36 (1 point)

À l'aide de portes NON, NAND et NOR, concevoir une porte "Peut-être exclusif"

6. En revanche, il est conseillé d'ignorer cette note de bas de page pour éviter de perdre au Jeu.