**Sorbonne Université**
M1 IA2D – MOGPL

## La balade du robot

Un robot est utilisé dans le dépôt d'un grand magasin pour le transport d'objets. On s'intéresse à la minimisation du temps de transport du robot. Le robot peut se déplacer uniquement sur des lignes droites (rails). Tous les rails forment une grille rectangulaire. La distance entre deux rails voisins est d'un mètre. Le dépôt est un rectangle de $N \times M$ mètres et il est entièrement couvert par la grille. Le robot a une forme circulaire de diamètre égale à 1.6 mètre. Le rail passe par le centre du robot. Le robot se dirige seulement en quatre directions : nord, sud, est, ouest. Les rails suivent les directions sud-nord et ouest-est. Le robot se déplace uniquement vers la direction vers laquelle il fait face. La direction vers laquelle le robot fait face peut-être changée au croisement de rails. Initialement, le robot se trouve à un croisement de rails. Des obstacles dans le dépôt sont formés par des pièces de taille 1 mètre $\times$ 1 mètre posées sur le sol. Chaque obstacle est posé sur un carré de taille 1 mètre $\times$ 1 mètre formé par les rails. Le mouvement du robot est contrôlé par deux commandes AVANCE et TOURNE. La commande AVANCE a un paramètre entier $n \in \{1, 2, 3\}$. Quand il reçoit cette commande, le robot avance de $n$ mètres dans la direction vers laquelle il fait face. La commande TOURNE a comme argument *gauche* ou *droite*. Quand il reçoit cette commande, le robot change son orientation de $90^o$ degrés dans la direction indiquée par le paramètre. L'exécution de chaque commande dure une seconde. On souhaite écrire un programme qui détermine le temps minimum pour le déplacement du robot d'un point de départ donné à un point d'arrivée donné. Pour ce faire, vous devrez générer des instances que vous stockerez dans un fichier d'entrée au format décrit ci-dessous. Ensuite, une fois écrit votre programme pour résoudre le problème, vous générerez un fichier de résultats au format décrit ci-dessous.

**Fichier d'entrée** Le fichier d'entrée consiste en des blocs de lignes. Chaque bloc à l'exception du dernier correspond à une instance du problème. La première ligne de chaque bloc contient des entiers $M \leq 50$ et $N \leq 50$ separés d'un espace. Chacune des $M$ lignes est constituée de $N$ nombres 1 ou 0 séparés d'un espace. Un 1 représente la présence d'obstacle et 0 représente un carré libre (les rails sont entre les carrés). Un bloc se termine par une ligne qui contient quatre entiers positifs $D_1, D_2, F_1, F_2$, chacun de ces quatre nombres est suivi par un espace, et de la chaîne de caractères ("nord", "sud", "est", "ouest") indiquant l'orientation du robot au point de départ. $D_1, D_2$ sont les coordonnées du coin nord-ouest du carré dans lequel le robot est placé au départ (point de départ). $F_1, F_2$ sont les coordonnées du coin nord-ouest du carré dans lequel le robot doit arriver (point de destination). L'orientation du robot quand il arrive au point de destination n'est pas décrite. On va utiliser des coordonnées de type (ligne,colonne). Plus précisément, les coordonnées du carré le plus en haut et à gauche (le plus au nord-ouest) du dépôt sont 0,0 et celles du carré qui se trouve le plus bas à droite (le plus au sud-est) sont $M-1, N-1$. Le dernier bloc contient uniquement une ligne 0 0 pour indiquer la fin du fichier.

**Fichier de résultats** Le fichier de résultats contient une ligne par bloc à l'exception du dernier bloc du fichier d'entrée. Les lignes sont dans le même ordre que les blocs du fichier d'entrée. Chaque ligne contient le nombre minimum de secondes pour que le robot puisse se déplacer du point de départ au point de destination ainsi que la liste des commandes correspondant. Pour simplifier on écrira D (resp. G) pour TOURNE(droite) (resp. TOURNE(gauche)) et a$n$ pour AVANCE(n) avec $n \in \{1, 2, 3\}$. S'il n'existe aucun chemin du point de départ au point de destination la ligne doit contenir -1.

**Exemple**

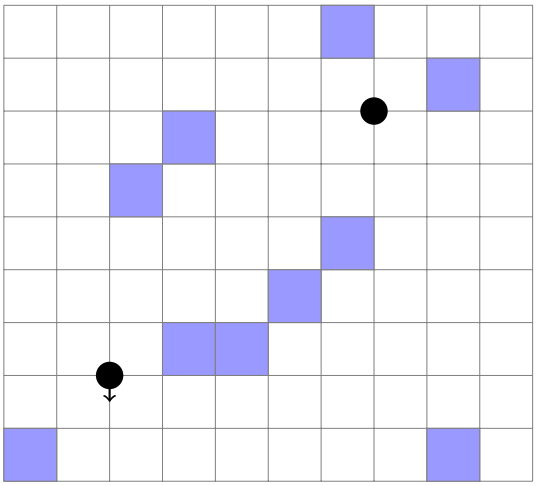

Figure 1: On considére l'instance ci-dessus.

Le fichier d'entrée est :

```
9  10
0  0  0  0  0  0  1  0  0  0
0  0  0  0  0  0  0  0  1  0
0  0  0  1  0  0  0  0  0  0
0  0  1  0  0  0  0  0  0  0
0  0  0  0  0  0  1  0  0  0
0  0  0  0  0  1  0  0  0  0
0  0  0  1  1  0  0  0  0  0
0  0  0  0  0  0  0  0  0  0
1  0  0  0  0  0  0  0  1  0
7  2  2  7  sud
0  0
```

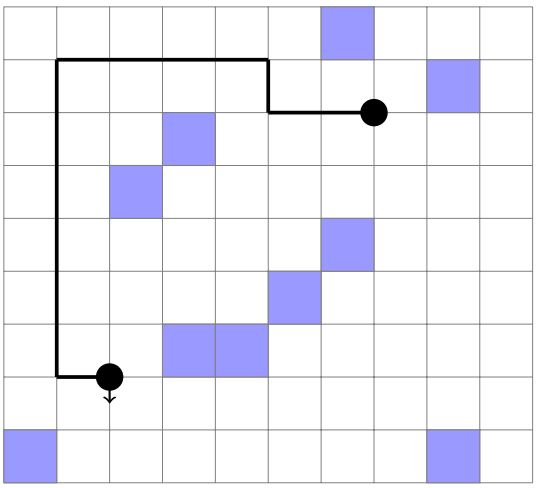

Figure 2: La solution optimale pour cet exemple.

Le fichier de sortie est :

```
12 D a1 D a3 a3 D a3 a1 D a1 G a2
```

(a) Donner une formulation du problème en le transformant en un problème dans un graphe orienté.

(b) Coder un algorithme pour la résolution du problème de complexité aussi faible que possible. Evaluer la complexité de votre méthode.

(c) Effectuer des essais numériques pour évaluer le temps de calcul de votre algorithme *en fonction de la taille de la grille*. Générer des instances avec $N = M = 10, 20, 30, 40, 50$ avec chaque fois un nombre d'obstacles fixé (par exemple 10 pour une grille $10 \times 10$, 20 pour une grille $20 \times 20$, etc.). Vous stockerez les instances dans un fichier d'entrée ainsi que les résultats dans un fichier résultats. Pour chaque valeur de $N$ on tirera 10 instances aléatoirement et on reportera dans un tableau ou sur une courbe les temps moyen d'exécution.

(d) Effectuer des essais numériques pour évaluer le temps de calcul de votre algorithme *en fonction du nombre d'obstacles présents*. Pour une grille de taille $20 \times 20$ générer des instances avec 10, 20, 30, 40, 50 obstacles. Vous stockerez les instances dans un fichier d'entrée ainsi que les résultats dans un fichier résultats. Pour chaque valeur de nombre d'obstacles on tirera 10 instances aléatoirement et on reportera dans un tableau ou sur une courbe les temps moyen d'exécution.

(e) Pour engendrer aléatoirement une grille de taille $(M, N)$ donnée comportant un nombre $P$ donné d'obstacles, on propose une procédure qui tire aléatoirement un poids entier entre 0 et 1000 pour chaque case et qui cherche à placer les $P$ obstacles sur $P$ cases dont la somme est minimale, en respectant les contraintes suivantes :

- chaque ligne de la grille ne peut contenir plus de $\frac{2P}{M}$ obstacles,
- chaque colonne de la grille ne peut contenir plus de $\frac{2P}{N}$ obstacles,
- aucune ligne ni aucune colonne ne peut contenir la séquence 101 (c'est-à-dire que les deux cases voisines d'une case libre ne peuvent simultanément contenir un obstacle).

Après avoir modélisé le problème d'optimisation ci-dessus par un programme linéaire, proposer une interface permettant à l'utilisateur de choisir la taille de la grille et le nombre d'obstacles, puis de générer (après tirage aléatoire) les positions des obstacles en résolvant avec Gurobi le programme linéaire formulé, et enfin, après avoir choisi un point de départ, une orientation initiale et un point de destination, d'afficher la solution obtenue par l'algorithme proposé à la question b.

**Organisation et dates**

Le travail est à effectuer en binôme constitué de deux personnes inscrites dans le même groupe de TD. Le binôme devra être déclaré par mail à votre chargé de TD

(objet du mail: binome MOGPL-25) au plus tard le vendredi 14 Novembre 2026. Les projets seront déposés au plus tard le **lundi 8 décembre 2025** à minuit sur le site moodle de MOGPL. Votre livraison sera constituée d'une archive zip (pas de .tar ou .targz etc) nommée GRnumgroupe_nom1_nom2.zip qui comportera les sources du programme, un fichier README détaillant comment exécuter le programme, et un rapport rédigé (un fichier au format pdf nommé GRnumgroupe_nom1_nom2.pdf) qui présentera le travail effectué et les réponses aux différentes questions. Le plan du rapport suivra le plan du sujet. Il est fortement recommandé de rédiger son rapport en LaTeX. Les projets rendus feront l'objet d'une soutenance sur machine en salle tme lors de la semaine du 15 décembre 2025.