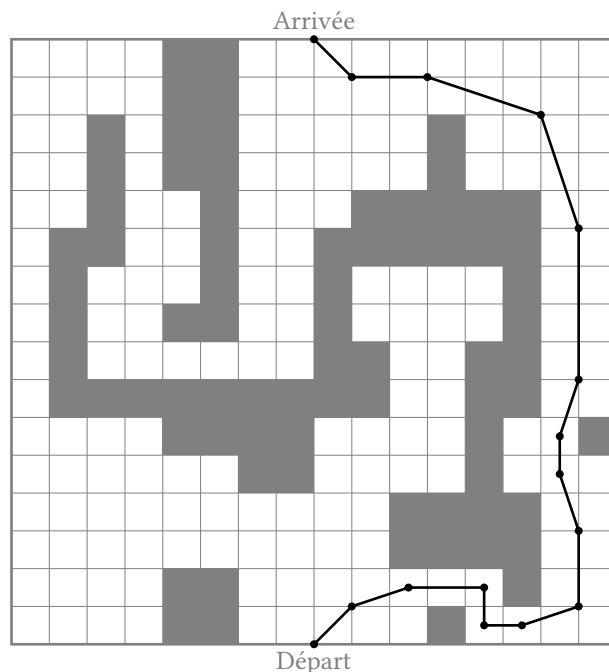


Outils logiques et algorithmiques – DM

On se place sur un terrain à deux dimensions, dont certaines zones sont intraversables. On cherche un itinéraire, si possible pas trop long, entre deux points donnés.



Le terrain est un carré de côté n , dont on retire r rectangles intraversables. Il est fourni sous la forme d'un fichier texte structuré ainsi :

1. la première ligne contient le nombre n ,
2. la deuxième ligne contient le nombre r ,
3. les r lignes suivantes décrivent les r rectangles intraversables.

Chaque rectangle est décrit par quatre nombres entiers positifs, donnés dans l'ordre : les coordonnées x et y du coin inférieur gauche, la largeur ∂x et la hauteur ∂y . En pratique, on pourra supposer que n est une puissance de deux. Le point de départ a les coordonnées $(n/2, 0)$, celui d'arrivée les coordonnées $(n/2, n)$. L'itinéraire à fournir en réponse est une séquence de paires de coordonnées, telle que l'on puisse aller d'un point au suivant en ligne droite sans croiser de zone intraversable.

L'objectif du DM est de réaliser un programme qui prend en entrée un fichier décrivant le terrain, et qui calcule un itinéraire entre le point de départ et le point d'arrivée. On s'intéressera à des terrains de dimensions variées :

	Mini	Petit	Grand	Très grand
n	$\leq 2^5$	$\leq 2^{10}$	$\leq 2^{15}$	$\leq 2^{20}$
r	$\leq 2^4$	$\leq 2^8$	$\leq 2^{12}$	$\leq 2^{16}$

Les différentes manières d'aborder le problème feront des compromis différents entre la simplicité de mise en œuvre et la capacité à résoudre des problèmes de grande taille. Le sujet comporte trois étapes.

1. Une approche simpliste du problème, pour se mettre en route.
Cette version sert à se familiariser avec le problème et doit permettre à tout le monde d'avoir une première version qui fonctionne. Cependant, elle n'est pas suffisante pour obtenir la moyenne.
2. Une approche plus efficace, qui est votre objectif principal.
Cette approche est décomposée en trois parties, qui peuvent être réalisées indépendamment et dans un ordre quelconque. Chaque partie réalisée sera prise en compte dans la note même si les autres sont absentes. Réaliser intégralement cette version promet une note bonne voire très bonne, selon la qualité du code.
3. Des suggestions d'optimisations pour aller jusqu'aux très grands terrains, voire au-delà.
Du bonus, avec encore quelques points supplémentaires.

Vous trouverez un squelette de code sur la page du cours, qui contient un certain nombre d'éléments « annexes » comme une fonction de lecture du fichier d'entrée, ou des fonctions permettant d'afficher certaines structures de données (servez-vous en pour tester et déboguer !). En utilisant ce squelette vous pourrez vous concentrer sur les parties algorithmiques centrales.

1 Version simpliste

On représente le terrain par un tableau t à deux dimensions, dont chaque case correspond à un carré de côté 1. Plus précisément, la case $t_{i,j}$ contient *true* si le carré dont le coin inférieur gauche a les coordonnées (i,j) est traversable, et *false* sinon. Ce tableau décrit implicitement un graphe :

- chaque case contenant *true* est un sommet,
- les voisins d'un sommet sont les cases libres adjacentes (il y en a au maximum 4).

Questions.

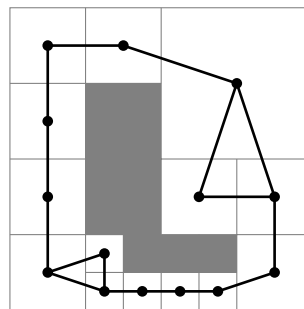
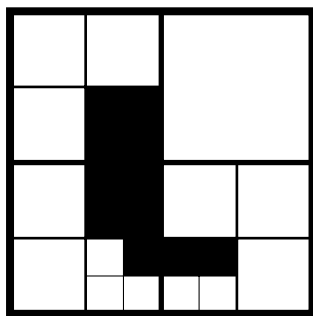
1. Combien de cases contient le tableau t , en fonction de la taille n du terrain ? À partir de quelle valeur de n environ risquez-vous de saturer la mémoire vive de votre ordinateur ? Vous pourriez supposer que l'espace mémoire utilisé par un tableau de taille s est environ $8s$ octets.
2. Écrire une fonction qui, partant de n et de la liste des rectangles intraversables crée un tableau t représentant le terrain.
3. Écrire une fonction d'exploration du terrain. Cette fonction doit prendre en entrée le tableau t , les coordonnées d'un point de départ et d'un point d'arrivée, et renvoyer un itinéraire valide sous la forme d'une liste de paires de coordonnées. *Indice : pour produire un chemin le plus court, vaut-il mieux un parcours en largeur ou un parcours en profondeur ?*
4. Compléter le programme pour qu'il prenne en entrée un fichier décrivant un terrain et affiche un itinéraire.

2 Version efficace

On résume le terrain par un quadtree (voir TP2). Ce quadtree définit un graphe pondéré de la manière suivante :

- chaque feuille libre du quadtree est un sommet,
- les voisins d'un sommet sont les régions libres adjacentes,
- la distance entre deux sommets adjacents est la distance euclidienne entre les centres des deux régions libres correspondantes.

Voici à gauche le découpage d'un terrain en quadtree pris comme exemple en TP, et à droite le graphe correspondant.



Questions préliminaires.

1. Dessiner le graphe correspondant au terrain montré en exemple sur la première page du sujet.
2. Considérons la situation suivante : un terrain de côté $n = 8$, dans lequel on a $r = 4$ rectangles intraversables disjoints, chacun de ces rectangles ayant une largeur et une hauteur $\partial x = \partial y = 2$. Donner deux configurations possibles sous ces contraintes : une générant un quadtree le plus simple possible, et une donnant un quadtree le plus complexe possible.

Partie A. Construction d'un quadtree. L'objectif de cette première étape est, à partir de la taille n du terrain et de la liste des rectangles intraversables, de construire un quadtree représentant l'ensemble des régions libres. Cette première étape est essentiellement résolue par les questions du deuxième TP. Il faut les éléments suivants.

1. Une fonction qui, étant donné un rectangle intraversable, construit un quadtree représentant le terrain laissé libre par ce rectangle.
2. Une fonction d'intersection qui, étant donnés deux quadtrees qt_1 et qt_2 , calcule un nouveau quadtree faisant l'intersection des régions libres de qt_1 et qt_2 .



3. Une fonction qui combine les deux précédentes pour faire l'intersection des quadrees correspondant à chaque région intraversable.

Petite différence par rapport au TP : à chaque feuille Libre on associe un nombre entier, qui permettra de l'identifier en tant que nœud du graphe. Pour les trois fonctions citées ci-dessus vous n'avez pas à vous soucier de ce numéro : vous pouvez affecter par défaut le numéro -1 à chaque feuille. On ne s'occupera de donner un numéro unique à chaque feuille qu'une fois l'arbre intégralement construit (et la fonction réalisant cette numérotation vous est fournie).

Partie B. Construction d'un graphe. L'objectif de cette deuxième étape est, à partir d'un quadtree comme le précédent, de construire le graphe de ses régions libres. On vous suggère la technique suivante.

1. Numéroté les régions libres du quadtree à l'aide de la fonction de numérotation fournie.
2. Construire un tableau associant à chaque numéro d'une région libre les coordonnées de son centre. *Attention : les régions les plus petites peuvent avoir un côté de longueur 1, et donc un centre avec des coordonnées non entières.*
3. Représenter le graphe par un tableau t de listes d'adjacence pondérées : la case t_i contient une liste paires, chaque paire donnant le numéro d'un voisin du sommet numéro i et la distance de ce voisin.
4. Initialiser ce tableau avec exclusivement des listes vides, puis parcourir récursivement le quadtree pour remplir les listes d'adjacence. *Ne pas oublier : il faut donner une longueur à chaque nouvelle arête ajoutée au graphe. En l'occurrence, on prend la distance euclidienne entre les centres des deux régions concernées.*

Voici les critères que vous pouvez suivre pour remplir les listes d'adjacence :

- un quadtree réduit à une feuille ( ou ) n'implique aucune arête,
- un quadtree représentant une division du terrain en quatre zones






NO	NE
SO	SE

 implique un certain nombre d'arêtes :
 - des arêtes internes aux quatres régions NO, NE, SO, SE,
 - des arêtes « horizontales » à l'interface entre les régions NO et NE, et entre les régions SO et SE,
 - des arêtes « verticales » à l'interface entre les régions SO et NO, et entre les régions SE et NE.

On vous recommande donc d'écrire trois fonctions récursives :

- 4.a. une fonction auxiliaire `interface_horizontale`, qui énumère les arêtes à l'interface entre deux quadrees adjacents horizontalement,
- 4.b. une fonction auxiliaire `interface_verticale`, qui énumère les arêtes à l'interface entre deux quadrees adjacents verticalement,
- 4.c. une fonction principale, qui énumère toutes les arêtes du graphe représenté par un quadtree.

Quelques questions à se poser pour se mettre sur la bonne voie :

- Quelle est l'interface horizontale entre  et  ?
- Quelle est l'interface horizontale entre  et  ?
- Quelle est l'interface horizontale entre  et

NO	NE
SO	SE

 ?
- Quelle est l'interface horizontale entre

NO	NE
SO	SE

 et

NO'	NE'
SO'	SE'

 ?

Partie C. Recherche de plus court chemin. Le graphe construit au terme de la partie précédente est un graphe pondéré dans lequel chaque arête a un poids strictement positif. Nous sommes dans un bon cas d'application pour l'algorithme de Dijkstra. Vous pouvez donc conclure la recherche d'un itinéraire en réalisant les éléments suivants.

1. Réaliser une structure de file de priorité, dotée des trois fonctions suivantes.
 - (a) Une fonction pour ajouter à la file un élément associé à une priorité. *La priorité sera donnée par un nombre flottant. Le nombre le plus petit sera le plus prioritaire.*
 - (b) Une fonction pour tester si une file est vide.
 - (c) Une fonction pour retirer (et récupérer) l'élément le plus prioritaire de la file.

Pour réaliser une telle structure de données efficacement, vous pouvez employer la structure de tas binaire vue en cours.

2. Explorer le graphe à l'aide de l'algorithme de Dijkstra, en partant d'une région libre contenant le point de départ. *Note : le point de départ de coordonnées $(n/2, 0)$ sera généralement au coin de deux régions libres. Vous pouvez prendre par défaut la région dont le point de départ est le coin inférieur gauche.*

Attention : vous ne devez pas vous contenter de calculer la longueur du chemin le plus court depuis la source vers chaque sommet du graphe, mais également donner les moyens de construire ces chemins les plus courts. Pour cela il faut renvoyer un tableau des prédécesseurs, donnant pour chaque sommet atteignable le sommet qui le précède sur un chemin le plus court depuis la source.

3. À l'aide du tableau des prédécesseurs, reconstruire un chemin allant du point de départ au point d'arrivée et utilisant comme points intermédiaires les centres des régions libres traversées. *Note : comme pour la région de départ, deux régions toucheront généralement le point d'arrivée. Vous pouvez prendre par défaut la région dont le point d'arrivée est le coin supérieur gauche.*

Les trois parties A, B et C réunies doivent vous permettre d'écrire un programme qui prend en entrée un fichier décrivant un terrain, et qui affiche un itinéraire valide du point de départ au point d'arrivée. Ce programme devrait être capable de traiter en temps raisonnables des terrains pris dans la catégorie « grand ».

3 Version optimisée

Voici quelques suggestions d'améliorations pour permettre à votre programme de traiter des terrains de taille et de complexité supérieures, ou améliorer la qualité du chemin produit. Vous pouvez également proposer d'autres améliorations que celles décrites ici.

Recherche de chemin A*. L'algorithme de recherche A* modifie l'algorithme de Dijkstra avec les trois éléments suivants :

- on ne cherche plus à calculer un plus court chemin depuis la source vers chaque sommet du graphe, mais uniquement vers un sommet cible choisi,
- dès que l'on a atteint le sommet cible choisi, l'algorithme s'arrête,
- lorsque l'on place un sommet s dans la file de priorité, sa priorité est calculée comme la somme de :
 - a. la distance du plus court chemin trouvé vers s (la valeur utilisée dans l'algorithme de Dijkstra),
 - b. la distance euclidienne entre s et le point d'arrivée choisi.

Ainsi, les sommets plus proches de l'arrivée auront une priorité renforcée et auront tendance à être explorés plus tôt. Ceci maximise les chances d'arriver au sommet cible (et donc d'arrêter l'exploration) après n'avoir exploré qu'une partie réduite du graphe.

Construction partielle du graphe. À ce stade, si vous mesurez le temps passé par votre programme dans ses différentes grandes étapes, la partie la plus coûteuse devrait être la construction du graphe. Or, un algorithme comme A* n'explore qu'une partie du graphe : la partie non explorée a été construite pour rien. Vous pouvez améliorer votre programme en ne construisant pas intégralement le graphe dès le début, mais plutôt en l'étendant petit à petit, à mesure que de nouvelles régions sont explorées par l'algorithme de recherche de chemin. Si de vastes pans du graphes ne sont pas explorés, il ne seront donc pas non plus construits et le temps ainsi gagné pourra être sensible. *L'approche la plus simple pour cette optimisation consiste à maintenir la création a priori de tous les sommets du graphe, mais à retarder la création des arêtes. Cette version suffit à obtenir un gain intéressant.*

Chemins améliorés. L'algorithme proposé ici impose un itinéraire passant par le centre de chaque région libre traversée. Cette contrainte peut engendrer de petits détours inutiles. Vous pouvez obtenir un chemin plus court de différentes manières :

- en retravaillant *a posteriori* l'itinéraire produit par votre algorithme, pour en gommer certains détours, ou
- en définissant différemment le graphe sur lequel la recherche est faite.