IN104 - Projet informatique

Générateur de Labyrinthes

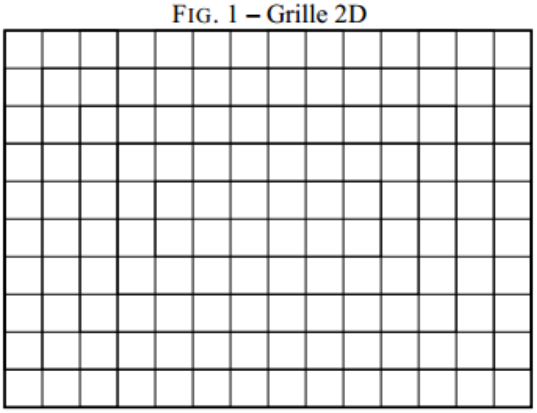
1 Description du projet

Ce projet a pour objet l'écriture d'un programme permettant de générer automatiquement et aléatoirement des labyrinthes parfaits.

1.1 Labyrinthe parfait

Considérons une grille 2D de cellules, comme dans la figure 1.1.

FIG. 1 – Grille 2D



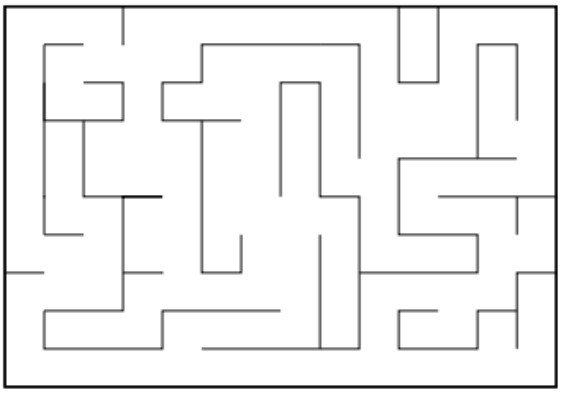
Les traits verticaux et horizontaux de la grille représentent des murs. Un labyrinthe sera vu comme une grille dont certains murs internes ont été détruits. Les murs externes seront vus comme des bordures infranchissables.

Un labyrinthe est dit *parfait* lorsque tout point du labyrinthe a un chemin simple (sans retour en arrière) et un seul vers tout autre point. Un labyrinthe parfait ne comporte donc ni cellule inaccessible, ni chemin circulaire, ni aire ouverte. La figure 2 illustre ce concept.

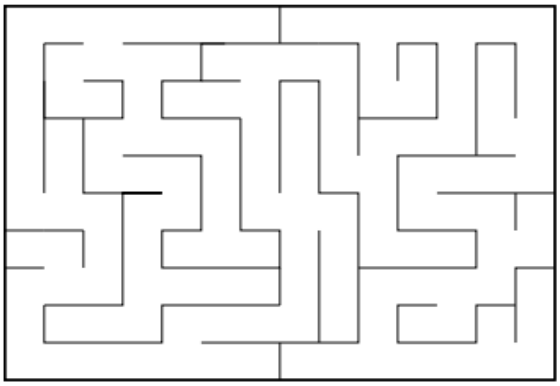
Une technique naïve de génération de labyrinthe est décrite par l'algorithme 1. Elle a pour principal mérite d'illustrer le fait que l'on va construire un labyrinthe en partant d'une grille complète et en abattant certains murs. Son défaut majeur est que le choix purement aléatoire du prochain mur à détruire aboutit presque à coup sûr à un labyrinthe imparfait. La question à résoudre est de parvenir à ne choisir un mur que parmi ceux dont la destruction n'invalide pas la possibilité que le labyrinthe finalement obtenu soit parfait.

Algorithme 1 naïf

partir d'un labyrinthe sous la forme d'une grille 2D.
tant que le labyrinthe n'a pas un aspect satisfaisant **faire**
  choisir un mur interne de façon aléatoire et le détruire
fin tant que



(a) Imparfait



(b) Parfait

FIG. 2 – Différents types de labyrinthes

1.2 Equivalence avec un problème de théorie des graphes

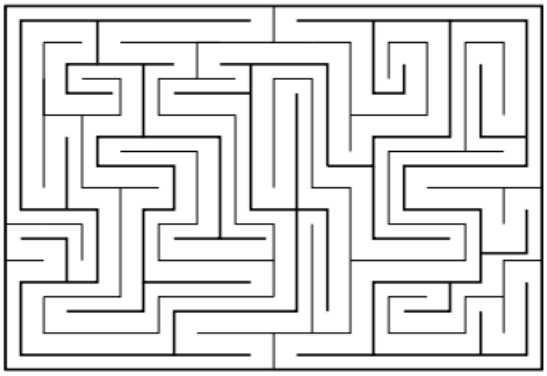
Il est souvent utile de ramener un problème algorithmique comme celui qui nous occupe à un problème connu, par exemple en théorie des graphes. En l'occurrence, on peut associer un graphe non-orienté G à tout labyrinthe de la façon suivante :

- chaque cellule du labyrinthe correspond à un noeud de G,
- deux noeuds sont reliés par une arête si les cellules correspondantes sont contiguës et non séparées par un mur.

Si le labyrinthe est parfait, on peut vérifier que le graphe G est à la fois connexe et sans cycle : autrement dit, G est un arbre. La figure 3 illustre cette propriété.

On cherche à exploiter cette caractérisation des labyrinthes parfaits dans le cadre d'un algorithme construisant un labyrinte en abattant des murs internes. On part d'un graphe sans arête correspondant à une grille 2D. Le fait d'abattre un mur interne revient à connecter deux noeuds de ce graphe. Il s'agit alors de connecter les noeuds du graphe jusqu'à ce que le graphe obtenu soit un arbre couvrant tous les noeuds. Autrement dit, on vient de ramener le problème de la génération d'un labyrinthe parfait à celui de *la construction d'un arbre de couverture* d'un graphe non-orienté.

FIG. 3 – Couverture des cellules par un arbre



1.3 Arbre de couverture

1.3.1 Définition et propriété principale

Un arbre de couverture d'un graphe G est un sous-graphe de G qui contient tous ses sommets et qui est un arbre. On suppose que G est connexe, ce qui assure l'existence d'au moins un arbre de couverture. On note qu'un graphe ayant N composantes connexes peut être couvert par une forêt de N arbres.

Dans un cadre général, on suppose que les arêtes du graphe sont étiquetées par des entiers représentant des poids ou des longueurs. Un tel graphe permet de modéliser à titre d'exemple un ensemble de villes (les noeuds du graphe) séparées par des distances (les arêtes étiquetées). Le problème de construire un ensemble de routes reliant l'ensemble des villes tout en minimisant la longueur totale des routes se ramène à la construction d'un arbre de couverture dont la somme des poids des arêtes soit minimale parmi l'ensemble des arbres de couverture. Ce problème de *construction d'un arbre de couverture minimal* est plus général que celui qui nous concerne, mais les algorithmes le résolvant peuvent être facilement adaptés au cadre aléatoire et fournissent en outre des labyrinthes parfaits "intéressants".

Les algorithmes présentés dans la suite reposent tous sur le lemme 1. Ce lemme n'est valide que pour des graphes ayant un unique arbre de couverture minimal, ce qui n'est pas le cas général. Cependant en pratique, on peut toujours sélectionner toute arête de poids minimal e entre X et $G-X$ (au sens du lemme) sans que l'arbitraire de ce choix n'influe sur le fait qu'on construise bien un arbre de couverture minimal.

Lemme 1 *On suppose que G est un graphe non-connexe dont les arcs sont étiquetés par des poids. On suppose en outre que G a un arbre de couverture minimal unique T. Soit X un sous-ensemble de sommets de G, et soit e l'arête de poids minimal connectant X à $G-X$ (le sous-graphe de G limité aux sommets n'apparaissant pas dans X). Alors e fait partie de l'arbre de couverture minimal T.*

1.3.2 Algorithme de Prim

L'algorithme de Prim construit l'arbre de couverture minimal T en ajoutant un sommet à chaque itération. Le fait qu'il choisisse une arête connectant T à $G-T$ permet d'assurer comme invariant le fait que T soit un sous-arbre de G. L'arbre T finalement retourné est couvrant car la condition d'arrêt de l'itération de construction stipule que T contient tous les sommets de G. Le choix d'une arête de poids minimal assure la correction de l'algorithme en vertu du lemme 1.

Dans le cadre de la génération d'un labyrinte parfait, le choix de l'arête de poids minimal sera remplacé par un choix aléatoire parmi les arêtes connectant T à $G-T$.

1.3.3 Algorithme de Kruskal

L'algorithme de Kruskal est dit "glouton" car il ajoute à chaque étape l'arête de poids minimal permettant de connecter des sommets qui ne l'étaient pas encore, cela sans jamais remettre en cause ce choix. Il se

Algorithme 2 Prim

T comporte initialement un seul sommet de G
tant que T ne contient pas tous les sommets de G **faire**
  déterminer l'arête e de G de poids minimal connectant T à $G - T$
  ajouter e à T
fin tant que
retourner T

base donc sur la sélection d'un optimum local, et le fait qu'il retourne un optimum global repose à nouveau sur le lemme 1.

La version simplifiée à implanter pour la génération de labyrinthe parfait comportera une phase initiale de "tri" aléatoire sur les arêtes.

Algorithme 3 Kruskal

trier les arêtes de G par poids croissant
soit T le sous-graphe de G, comportant tous les sommets de G mais aucune arête
pour les arêtes e considérés dans l'ordre croissant **faire**
  si les extrémités de e sont déconnectées dans T **alors**
    ajouter e dans T
  fin si
fin pour
retourner T

1.4 Cahier des charges

On décrira et implantera les structures de données et algorithmes permettant de :
- représenter un labyrinthe 2D au cours des différentes étapes de sa construction,
- générer un labyrinthe parfait en se basant sur les algorithmes décrits en section 1.3,
- fournir le résultat de la génération sous forme de fichier, avec un codage qui sera à définir.

A noter qu'outre la partie concernant le développement proprement dit, les élèves devront remettre un rapport comprenant :
- une description détaillée des choix de conception effectués,
- les liens entre la conception et le code effectivement implanté,
- des éléments permettant de se convaincre de la validité du générateur de labyrinthes (conception et implantation),
- des éléments permettant de juger de l'efficacité du générateur de labyrinthes (temps, mémoire).

2 Organisation du travail

2.1 Séance 1

- choix de la représentation du labyrinthe,
- initialisation du labyrinthe sous forme de grille 2D,
- fonction de destruction d'un mur,
- affichage/sortie fichier d'un labyrinthe.

2.2 Séance 2

- choix d'un algorithme parmi Prim et Kruskal,
- adaptation de l'algorithme choisi au cadre aléatoire,
- mise en place des structures de données associées à l'algorithme.

2.3 Séance 3

- implantation et validation par le test de l'algorithme choisi en séance 2,
- évaluation des performances (temps, mémoire) de l'implantation obtenue.

2.4 Séance 4

- amélioration de l'implantation obtenue en séance 2,
- conception et implantation de l'autre algorithme pour les étudiants les plus avancés.

2.5 Séance 5

- finalisation du ou des algorithmes obtenus.