info404 : algorithmes sur les graphes

TP 2 : génération de labyrinthes...

Pierre Hyvernat
Laboratoire de mathématiques de l'université de Savoie
bâtiment Chablais, bureau 22
téléphone : 04 79 75 94 22
email : Pierre.Hyvernat@univ-savoie.fr
www : http://www.lama.univ-savoie.fr/~hyvernat/

> *Le compte-rendu de TP sera contenu en commentaire dans les sources de votre programme. Vous devrez donc me rendre un unique fichier appelé "*`nom.c`*" où* `nom` *est votre nom. Votre fichier devra être le plus lisible possible (indentation intelligente de votre code, présentation lisible de vos commentaires).*

Le but de ce TP est d'essayer de générer un labyrinthe intéressant sur une grille rectangulaire. On peut, en première approximation, représenter un labyrinthe sur une telle grille par un arbre : une pièce est reliée à une pièce voisine (graphe non-orienté) s'il y a une ouverture entre les deux. Le fait de prendre un arbre permet de garantir les deux propriétés suivantes :

- connexité : toute la grille est accessible (il n'y a pas de partie qui ne sert à rien)
- acyclicité : il n'y a qu'un seul chemin pour aller d'un endroit à un autre

Question 0 : discutez le bien fondé de ces propriétés.

Exercice 1 : parcours en largeur

Le fichier `labyrinthes.h` contient des définitions de types et des prototypes de fonctions. Les points importants sont les suivants

- définitions de constantes `LARGEUR` et `HAUTEUR` : c'est la taille de la grille
- un type de données `Tile` : un trucs entre les piles et les files. On peut insérer un élément en tête de structure et on peut récupérer soit le premier (comme dans une pile) soit le dernier (comme dans une file). On peut stocker des arcs (avec un `but`, une `origine` et un `poids`), mais on peut également s'en servir pour stocker juste une valeur entière. (C'est juste qu'on n'utilise pas tous les champs.)

Le fichier `labyrinthes.c` contient, entre autre, la définition de la fonction `grillePleine` : cela permet d'initialiser le graphe qui contient des passages entre toutes les pièces. Le graphe est non-orienté avec des poids aléatoires. Toutes les listes d'adjacences sont triées par ordre de poids croissant.

Question 1 : écrivez la fonction `afficheGrille`. Cette fonction doit prendre en argument une grille et l'afficher à l'écran.

Question 2 : écrivez la fonction `parcours` : pour ça, vous utiliserez un parcours en largeur (en utilisant le type `Tile` pour faire une file). Au lieu de générer un tableau `pere` contenant l'arbre du parcours en largeur, vous utiliserez un graphe (type GrapheListe) initialement vide. Toutes les arêtes de l'arbre du parcours seront ajoutées au graphe `pere` avec une ligne de la forme

```
pere->Adj[x] = cons(p,z,pere->Adj[x]) ;
```

Question 3 : affichez la grille correspondant à `pere` : c'est un labyrinthe. Qu'en pensez-vous, et comment expliquez-vous ce que vous voyez ?

Exercice 2 : un autre parcours

Question 1 : que se passe-t'il si, dans la fonction `parcours`, on remplace la file par une pile (en remplaçant juste la fonction `retireDernier` par la fonction `retirePremier`) ?

Question 2 : testez et constatez. Quel labyrinthe vous parait le plus intéressant ?

Question 3 : quel type de parcours avez-vous obtenu avec une pile ?

Exercice 3 : algorithme de Kruskal

Dans les deux parcours précédents, l'aléatoire venait du fait que l'on considérait les voisins dans un ordre arbitraire. Cependant, le parcours suivait quand même une structure assez rigide.

L'algorithme de Kruskal va permettre de rajouter dans le graphe `pere` des arêtes de manière complètement aléatoire : on les rajoute par ordre de poids croissant. (Les poids sont tous aléatoires...)

Question 1 : programmez l'algorithme de Kruskal. (Pensez à utiliser les fonctions existantes...)

Question 2 : testez et comparez.

Exercice bonus : ensuite...

Question 1 : comment peut-on résoudre un labyrinthe ? Donnez des détails, et programmez le si vous avez le temps.

Question 2 : les labyrinthes générés aléatoirement sont-ils vraiment difficiles ? Comment pensez-vous pouvoir améliorer la situation ?

Question 3 : programmez l'algorithme de Prim appliqué aux labyrinthes...