Université de Strasbourg
Structure de Données et Algorithmes Feuille de TP

Projet : labyrinthe

Rendez votre projet avant le dimanche 10/05/2015 minuit.
Le projet est réalisé en binôme. Après avoir terminé votre projet, faites un `make clean`, nommez votre répertoire par vos noms. Créez une archive du répertoire (`.tar.gz`) et envoyez-la sur moodle. Par exemple : `Nicolas_Et_Thomas.tar.gz`.

Documentez tous les fichiers source à l'aide de `Doxygen`. Vous indiquerez l'auteur de chaque fonction.

Exercice 1 Makefile

Créez un fichier `Makefile` permettant de compiler vos programmes. Vous indiquerez les dépendances entre fichiers objets (.o) et fichiers d'en-tête (.h) de manière à recompiler ce qui est nécessaire (et rien de plus) si un fichier d'en-tête est modifié. Créez deux règles `gendoc` et `exedoc` qui génère et crée la documentation Doxygen, et une règle `clean` qui nettoye le répertoire de tous les fichiers compilés et de la documentation.

Exercice 2 Ensembles

Écrivez une bibliothèque de manipulation d'ensembles de couples d'entiers (x, y), en utilisant la programmation par mutation. Les entiers x et y seront dans un intervalle de valeurs fixe (entre 0 et une valeur maximale donnée).

Cette bibliothèque contiendra au moins les fonctions suivantes :

- `EnsAlloc` : création d'un nouvel ensemble ;
- `EnsFree` : libération éventuelle de la mémoire utilisée ;
- `EnsEstVide` : teste si un ensemble est vide ;
- `EnsAjoute` : ajoute un couple à un ensemble ;
- `EnsSuppr` : retire un couple d'un ensemble ;
- `EnsEstDans` : teste si un couple appartient à un ensemble ;
- `EnsTaille` : nombre d'éléments dans l'ensemble ;
- `EnsTirage` : tire un couple aléatoirement dans un ensemble, et le retire de l'ensemble ;

Ces différentes fonctions seront appelées fréquemment, choisissez donc soigneusement le type d'implémentation que vous allez utiliser pour obtenir des performances raisonnables.

Exercice 3 Matrices

Écrivez une bibliothèque de manipulation de matrices h x l de booléens, en utilisant la programmation par mutation. Les entiers h et l (hauteur et largeur de la matrice) seront donnés en paramètre à la fonction de création de la matrice.

Cette bibliothèque contiendra au moins les fonctions suivantes :

- `MatAlloc` : création d'une nouvelle matrice h x l initialisée à `FAUX` ;
- `MatFree` : libération éventuelle de la mémoire utilisée ;

- `MatVal` : renvoie la valeur booléenne `matrice[x,y]` ;
- `MatSet` : stocke une valeur booléenne dans `matrice[x,y]` ;
- `MatSauve` : enregistre une matrice dans un fichier ;
- `MatLit` : lit une matrice depuis un fichier ;

Exercice 4 Bibliothèque graphique
Utilisez la bibliothèque graphique `graph` fournie en projet. Vous pouvez éventuellement ajouter vos propres fonctions dans une bibliothèque supplémentaire.

Exercice 5 Générateur de labyrinthe
Le générateur de labyrinthe fonctionnera de la manière suivante :

- initialisation de la matrice vide (`FAUX` dans la matrice représente une case vide, `VRAI` un mur) ;
- création des bords de la matrice ;
- création d'un certain nombre de graines initiales, sur les bords et au centre de la matrice : des murs peuvent être construits autour de ces graines ;
- initialisation de l'ensemble des cases constructibles
- boucle principale. Tant qu'il reste des cases constructibles :
  — choisir aléatoirement une case constructible ;
  — y construire un mur ;
  — mettre à jour les cases voisines dans l'ensemble des cases constructibles.

Vous écrirez une fonction `EstConstructible`, qui teste si une case de la matrice est constructible : elle doit posséder un mur sur un de ses 4-côtés, et, parmi les 8-voisins de cette case, les 5 voisins opposés à ce mur doivent être vides.

Ce programme aura la syntaxe suivante : `./genlab [-v] [-d] [-l <largeur>] [-h <hauteur>] <fichier>`

`-v` : visualise le labyrinthe pendant sa génération ;
`-d` : visualise le labyrinthe et les cases constructibles pendant la génération (implique `-v`) ;
`-l <largeur>` et `-h <hauteur>` : largeur et hauteur du labyrinthe, en nombre de cases. Vous ferez attention à la taille de la fenêtre (pas plus de 1000 x 800 pixels), et la taille des cases affichées devra être modifiée en conséquence ; valeurs par défaut : largeur = 300, hauteur = 200, taille d'une case affichée = 3x3 pixels ;
<fichier> : nom de fichier dans lequel le labyrinthe sera sauvegardé.

Exercice 6 Recherche de chemin
Écrivez un programme `cheminlab` qui lit un labyrinthe depuis un fichier, et recherche un chemin aléatoire depuis le coin supérieur gauche jusqu'au point inférieur droit du labyrinthe. Options :

`-v` : visualise le parcours du labyrinthe pendant la recherche ;
`<fichier>` : nom du fichier labyrinthe à lire.

Pour savoir si une case a déjà été visitée, vous stockerez chaque case parcourue dans une structure `Ensemble`.