info404 : algorithmes sur les graphes

TP 1 : représentation des graphes, parcours en largeur et profondeur

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).*
Vous pouvez récupérer le fichier "`hyvernat.c`*" sur ma page internet et vous en inspirer. (Ce fichier contient quelques détails supplémentaires.)*

Exercice 0 : prise en main de Unix, compilation en C

Cette partie ne doit pas prendre plus qu'une quinzaine de minutes. Elle ne donne pas lieu à un compte rendu et ne sera pas notée.

Question 1 : redémarrez votre ordinateur sous Linux. Si vous ne connaissez pas Unix, prenez quelques minutes pour explorer le système...

Question 2 : les deux applications dont nous aurons besoin sont
- un terminal (pour compiler et exécuter nos programmes)
- un éditeur de texte (pour écrire nos programmes)

Cherchez dans les menus ces deux applications et lancez les.
Créez un répertoire "`info404`" dans votre répertoire de travail.

Question 3 : pour survivre dans le terminal, il faut connaître les commandes suivantes
- "`> ls`" pour obtenir la liste des fichiers dans le répertoire courant
- "`> cd` *nom-répertoire*" pour changer de répertoire courant
- "`> man` *prog*" pour obtenir l'aide en ligne du programme *prog*

À quoi servent les commandes `mkdir` et `tar` ? Regardez le manuel de `gcc` (le compilateur C de Linux) ; qu'en pensez-vous ?

Question 4 : récupérez les fichiers `graphes.h` et `graphes.c` sur mon site internet ; sauvez ces fichiers dans votre répertoire `info404`. Le fichier `graphes.h` contient les définitions des types et les prototypes de fonctions ; le fichier `graphes.c` contient la définition des fonctions.

Pour compiler un programme qui utilise les types et fonctions définis dans `graphes.c` / `graphes.h`, il faut suivre les étapes suivantes :
- compiler un fichier contenant le code des fonctions définies dans `graphes.c` avec la commande "`> gcc -c graphes.c`". Ceci doit produire un fichier binaire "`graphes.o`". (Cette étape n'est effectuée qu'une seule fois...)
- Rajouter "`#include "graphes.h"`" dans votre fichier principal.
- Compiler votre fichier principal avec "`> gcc -Wall -c nom.c`", ce qui doit produire un fichier "*fichier*`.o`".
- Terminer la compilation en reliant les deux fichiers objets avec la commande "`gcc -Wall -o test nom.o graphes.o`", ce qui doit produire un exécutable "`test`"
- Exécuter votre programme avec la commande "`> ./test`".

Vérifiez que toutes ces étapes fonctionnent en compilant le programme suivant :

```
#include "graphes.h"
```

```
int main (void) {
  printf("Ca marche !\n") ;
  return(1) ;
}
```

Exercice 1 : quelques fonctions préliminaires

Tout le code que vous produisez devra se trouver dans un fichier "`nom.c`". C'est ce fichier qui sert également de compte rendu. Utilisez les commentaires, et débrouillez-vous pour rendre un fichier lisible. (Allez à la ligne régulièrement, indentez votre code etc.)

Ce fichier devra comporter vos nom et prénoms en commentaire au début. Ensuite, viendront les fonctions définies, puis la fonction `main` qui vous permettra de tester vos fonctions. (Pour une alternative, réferrez-vous au ficher squelette "`hyvernat.c`" disponible sur ma page internet.)

Question 1 : le fichier `graphes.h` contient le prototype suivant

```
void afficheGrapheListe (GrapheListe) ;
```

Cette fonction n'est pas définie dans le fichier `graphes.c`. Écrivez cette fonction dans votre propre fichier `nom.c`. La fonction prend en argument un graphe de type `GrapheListe` et doit afficher tous les arcs à l'écran.

Question 2 : définissez également les deux fonctions de conversion de prototypes

```
GrapheMatr listeVersMatrice(GrapheListe) ;
GrapheListe matriceVersListe(GrapheMatr) ;
```

Question 3 : définissez la fonction qui calcule l'inverse d'un graphe. Elle change la direction de tous les arcs et son prototype est donné par

```
GrapheListe transpose(GrapheListe) ;
```

Exercice 2 : parcours en largeur

En utilisant l'algorithme du TD2 (exercice 3, question 2), écrivez une fonction de prototype

```
int* biparti (GrapheListe) ;
```

Cette fonction devra renvoyer le tableau `NULL` si le graphe n'est pas biparti, et sinon, elle renverra un tableau de taille `n` (nombre de sommets) où la valeur pour l'indice `i` sera `-1` ou `1` suivant que le sommet `i` est dans le premier ou le deuxième ensemble de sommets.
Pour déclarer un tableau de taille `n` (variable de type `int`), il faut utiliser

```
int *T = malloc(n*sizeof(int)) ;
```

On peut ensuite avoir accès à la valeur pour l'indice `i` avec `T[i]`. Pour supprimer un tel tableau de la mémoire, il faut faire

```
free(T) ; t=NULL ;
```

Exercice 3 : parcours en profondeur, composantes fortement connexes

En utilisant l'algorithme du cours, programmez le calcul des composantes fortement connexes. Le prototype de la fonction sera

```
int composantesFortemenentConnexes(GraphesListe,*int) ;
```

La fonction prend en entrée un graphe et un tableau d'entiers (qu'il faudra déclarer avant l'appel de fonction, *cf.* exercice précédent) et renvoie comme valeur de retour le nombre de composantes fortement connexes. Elle devra également mettre dans le tableau (passé en argument) les numéros de composantes connexe pour chaque sommet.