Algorithmique et Structures de données 1

L2 2021-2022

Travaux Pratiques 9

Site du cours : `https://defelice.up8.site/algo-struct.html`
Les exercices marqués de (@) sont à faire dans un second temps.
Un fichier écrit en langage C se termine conventionnellement par `.c`.
Une commande de compilation est `gcc fichier_source1.c fichier_source2.c fichier_source3.c`.
Voici des options de cette commande.

- `-o nom_sortie` pour donner un nom au fichier de sortie (par défaut `a.out`).
- `-Wall -Wextra` pour demander au compilateur d'afficher plus de Warnings
- `-std=c11` pour compiler selon la norme C11
- `-g -fsanitize=address` pour compiler avec information de débogage et en interdisant la plupart des accès a une zone mémoire non reservée.

Exemple : `gcc -Wall fichier1.c -o monprogramme`

Bien qu'elle soit très gourmande en espace, on utilisera la définition de type suivante pour représenter des graphes. La structure `liste` est un tableau de tableau utilisé pour stocker le graphe par liste d'adjacence. La fin d'une liste sera codée par -1.

Exemple :

Le tableau de tableau suivant :

1	3	4	-1			
2	-1					
-1						
0	1	-1				
4	-1					

représente les listes d'adjacence suivantes :

$0 \to (1\ 3\ 4)$ $1 \to (2)$ $2 \to ()$ $3 \to (0\ 1)$ $4 \to (4)$

On rappel que les arcs de nos graphes ne sont pas étiquetés.

```
# define MAX_SOMMET 40  // Le nombre maximum de sommets d'un graphe
typedef struct gra
{
   int liste[MAX_SOMMET][MAX_SOMMET];
   int n; // nombre de sommets
} graphe_t;
```

Exercice 1. *Création de graphe*

Écrire la fonction `void creerGraphe(graphe_t* g,FILE* grDesc)` qui construit un graphe à partir du fichier `grDesc`.

Exemple de contenu du fichier `grDesc`.

```
7
0->6
3->3
6->2
1->0
2->0
```

le graphe a 7 sommets 0 1 2 3 4 5 6

Pour la lecture on peut utiliser `fscanf(grDesc,"%d->%d",&d,&a)`.

Exercice 2. *Accessible*

Créer la fonction `void accessible(graphe_t* g,int s,int* acces)` qui indique les sommets accessibles à partir de `s`. Après appel, la case `i` du tableau `access` (qui a été alloué à l'extérieur de la fonction) doit contenir `0` si il n'existe pas de chemin de `s` à `i` dans le graphe `g`. Dans le cas contraire la case `i` doit contenir une valeur non nulle.

Par exemple, après avoir appelé `acessible(g,2,acess)` sur le graphe de l'exercice précédent, le tableau `access` peut contenir

0	1	2	3	4	5	6
*	0	*	0	0	0	*

(* signifie non nulle)

Exercice 3. *Circuit*

Créer la fonction `int sansCircuit(graphe_t* g)` qui renvoie 1 si `g` ne contient pas de circuit, 0 sinon.

Un graphe est qualifié d'arbre s'il ressemble à un arbre (pas forcément binaire) Entre autres :

— tous les sommets sont accessibles à partir de la racine.

— Il est sans circuit

Exercice 4. *Forêt*

On dit qu'un graphe est une forêt si c'est un ensemble d'arbres. Créer la fonction `int estForet(graphe_t* g)` qui renvoie 1 si `g` est une forêt 0 sinon.

Exercice 5. *@Arbre*

Créer la fonction `int estArbre(graphe_t* g)` qui renvoie 1 si `g` est un arbre, sinon renvoie 0.

Exercice 6. *Distance*

Écrire une fonction `void distance(graphe_t* g,int s,int* dist)` qui remplit la case `i` du tableau `dist` (alloué à l'extérieur de la fonction) par la distance du sommet `s` au sommet `i` (La distance du sommet `s` à `i` est la longueur du plus court chemin qui va de `s` à `i`). Vous pouvez utiliser une file d'entiers. On convient que la distance d'un sommet inaccessible à `s` est `-1`.