

TP 6&7 : Graphes – partie 1

Nous allons nous intéresser dans le TP à la représentation de graphes (orientés ou non) en langage C.

Un graphe peut être représenté par une liste de sommets **X** (chacun étant caractérisé par son nom, une étiquette -indice de sommet- éventuelle), et la liste des arcs définis par une relation **R** entre les sommets.

Par exemple, nous souhaitons manipuler un graphe $G = (X, R)$ défini par :

$X = \{2, 3, 4, 5, \dots, 15\}$

$R(i, j) = \begin{cases} 1 & \text{ssi "i est un diviseur de j"} \\ 0 & \text{sinon} \end{cases}$

Voici une **définition partielle** du fichier Graphe.h

...

```
typedef struct s_sommet *Sommet;
typedef struct s_arc *Arc;
typedef struct s_graphe *Graphe;
```

```
Graphe initialiseGraphe(void); // permet d'initialiser un graphe
void ajouteSommet(Sommet a, Graphe G) // ajoute le somme au graphe
```

```
void ajouteArc(Arc a, Graphe G); // ajoute l'arc a au graphe
void supprimeArc(Arc a, Graphe G); // supprime l'arc a du graphe
```

```
int existeSommet(Sommet s, Graphe G); // retourne 1 si le sommet s existe
// dans le graphe
int existeArc(Arc a, Graphe G); // retourne 1 si l'arc a existe dans le graphe
```

1. Représentation par matrice d'adjacence

Un graphe simple non étiqueté à n sommets numérotés peut être représenté par **une matrice carrée** $M(n, n)$ de 0 et 1 où $M_{i,j} = 1$ s'il existe un arc allant du sommet i au sommet j et $M_{i,j} = 0$ sinon.

Nota : cette représentation ne convient qu'aux graphes simples avec une forte redondance des informations pour les graphes non orientés et du stockage inutile des cas inintéressants.

Implantez la représentation en C dans le fichier Graphe.c et codez une fonction `main` dans un fichier séparé et codez un programme qui vérifie si le graphe G est **symétrique** ou **antisymétrique**

2. Représentation par liste contigüe de successeurs

Nous allons maintenant réimplanter ce graphe en utilisant une **liste contigüe de successeurs**. Dans ce cadre, la liste des arcs est stockée dans une liste.

Si l'avantage principal est la facilité de parcours, les inconvénients restent nombreux comme la redondance de la représentation pour les graphes non orientés ou encore le temps d'accès aux données (adressage)

Créez une structure en C supportant cette représentation interne et recodez l'implantation en C des fonctions.

3. Représentation par liste chaînée de successeurs

Une **liste chaînée** est une structure de données dans laquelle les objets sont arrangés linéairement, l'ordre étant déterminé par des pointeurs sur les différents éléments. Chaque élément de la liste contient :

1. Une donnée appelée **CLE**
2. Un champ successeur qui est un pointeur sur l'élément suivant dans la liste chaînée

Si le champ successeur d'un élément vaut `NULL`, cet élément n'a pas de successeur et est donc le dernier (ou queue) de la liste. Le premier élément de la liste est appelé la tête de la liste chaînée.

Créez une structure en C supportant cette représentation interne et recodez l'implantation en C.

Ecrire un fichier `Makefile` permettant de compiler votre programme suivant la représentation interne souhaitée.