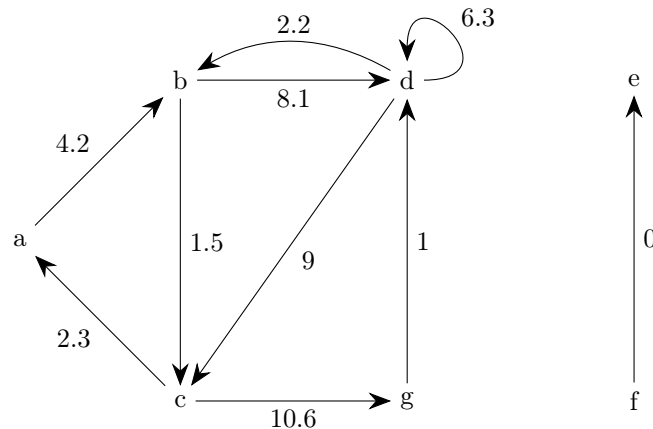


UE ALGO5 — TD2 — Séance 11 : Graphes

La figure ci-dessous représente un graphe $\mathcal{G}$. L'ensemble de ses sommets est $\{a, b, c, d, e, f, g\}$. Les arcs sont étiquetés par des réels, comme indiqué sur la figure.



Exercice 1.

Q 1. Illustrez chacun des termes suivants par un exemple pris dans ce graphe (s'il en existe un) : chemin, chemin de longueur 3, circuit, boucle, composante fortement connexe.

Corrigé —

- chemin : (a, b, c) , (a, b, c, a) , (a) , (a, b, c, a, b, d, c, g) , ...
- chemin de longueur 3 (la *longueur* est définie en terme du *nombre d'arcs du chemin*) : (a, b, c, g) , (a, b, c, a) , ...
- circuit : (a, b, c, a) , (b, d, b) , (d, d) , (b, d, b, d, d, b, d, b) , ...
- boucle : (d, d)
- composante fortement connexe : $\{a\}$ (toute composante réduite à un seul sommet), $\{a, b, c\}$, $\{a, b, c, d\}$, $\{a, b, c, d, g\}$

Exercice 2.

Q 2. Proposez une structure de donnée permettant de mémoriser un type **Graphe** représentant un graphe orienté avec arcs et sommets étiquetés. Dessinez le contenu de cette structure de données pour le graphe $\mathcal{G}$. On examinera les deux cas suivants :

- une structure de données basée sur des tableaux de taille fixe ;
- une structure de données basée sur des chaînages explicites par pointeurs.

Corrigé —

Rappels de notation : un graphe $\mathcal{G} = (X, R)$ est un couple composé d'un ensemble de *sommets* X et d'un ensemble d'*arcs* $R \subseteq X \times X$.

Avant de réfléchir à la structure de donnée à adopter, il faut se poser la question : de quelle(s) primitive(s) a-t-on besoin ? Cela dépend des algorithmes, mais en général on aura besoin de :

- tester l'existence d'un arc entre deux sommets donnés

—récupérer l'ensemble des successeurs d'un sommet donné (pour les algorithmes de parcours)
Selon les algos, on peut aussi vouloir récupérer l'ensemble des prédecesseurs...

NB : on peut distinguer deux étapes en laissant de côté l'aspect «sommets/arcs étiquetés» : quelle que soit la structure choisie on peut facilement ajouter les étiquettes après coup.

On peut considérer que l'ensemble des sommets est identifié par un entier : $X = \{n_1, \dots, n_n\}$; on se concentre donc sur l'implémentation de R (les arcs).

Implémentation à l'aide de tableaux de taille fixe

R est un ensemble de couples ; on peut envisager d'implémenter un ensemble par une liste dans un tableau (à une dimension). On a ainsi un tableau de couples. C'est l'implémentation la plus simple, mais très peu efficace pour les primitives mentionnées ci-dessus : rien que pour tester l'existence d'un arc entre deux sommets on peut être obligé de parcourir tout le tableau...

Pour retrouver la notion de *matrice d'adjacence*, on peut s'intéresser à cette primitive : tester l'existence d'un arc entre deux sommets est une fonction ($X \times X \rightarrow Bool$), ou un prédicat à deux paramètres. On peut implémenter cette fonction dans un tableau à deux dimensions : c'est la matrice d'adjacence.

Matrice d'adjacence pour le graphe de l'énoncé :

$$\begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

Implémentation à l'aide de structures chaînées

Comme pour les tableaux, on peut implémenter un graphe par une liste (chaînée) de couples de sommets. Mais ce n'est pas intéressant puisqu'on perd toute la structure sous-jacente.

L'idée de la *liste d'adjacence* est d'associer à un sommet la liste de ses successeurs. On a ainsi une liste (chaînée) de sommets, un sommet étant une liste (chaînée) d'arcs, un arc étant un pointeur vers le sommet d'arrivée.

Par exemple en C :

```
1 typedef struct cell_sommet *p_sommet;
2 typedef struct cell_arc *p_arc;
3
4 typedef p_arc liste_arcs;
5
6 typedef struct cell_sommet {
7     int etiq;
8     liste_arcs successeurs;
9     p_sommet suiv;
10 } sommet;
11
12 typedef struct cell_arc {
13     int etiq;
14     p_sommet orig, dest;
15     p_arc suiv;
16 } arc;
17
18 typedef p_sommet graphe;
```

Selon les besoins, on peut aussi associer à chaque sommet la liste de ses prédécesseurs.

Toutes les variantes intermédiaires sont possibles : tableau de listes, etc.

Exercice 3.

Pour chacune de ces structures :

Q 1. écrire un algorithme calculant le nombre d'arcs du graphe ;

Corrigé —

Avec une matrice d'adjacence

```
NbArcs ← 0
2 pour i de 1 à n
  pour j de 1 à n
4   si  $M_{ij} = 1$  alors NbArcs ← NbArcs + 1
```

Avec une liste d'adjacence

```
graphe g;
2 p_sommet sc;
  p_arc ac;
4 NbArcs = 0;
  sc = g;
6 while (sc != NULL) {
  ac = sc->successeurs;
8   while (ac != NULL) {
    NbArcs += 1;
10    ac = ac->suiv;
  }
12  sc = sc -> suiv;
}
```

Q 2. écrire l'algorithme de parcours générique d'un graphe.

Corrigé —

On rappelle l'algorithme de parcours générique d'un graphe :

```
1 Parcours_Graphe(g,origine)
  L ← ListeVide
3 Insérer(origine,L)
  tant que non EstVide(L)
5   x ← Premier(L)
   L ← ExtrairePremier(L)
7   si x non marqué alors
    Marquer(x)
9    Traiter(x)
    pour tout  $y \in \text{ensSuccesseurs}(x,g)$ 
11     Insérer(y,L)
```

On fait toujours abstraction de la structure dans laquelle on insère les sommets à traiter : on suppose disposer des primitives ListeVide, Insérer, EstVide, Premier, ExtrairePremier.

Les deux points de l'algorithme à raffiner sont :

- comment marquer un sommet ? Dans l'idéal, le «marquage» comme le test pour savoir si un sommet est marqué est de coût constant (pas de structure «ensemble des sommets marqués»...).
- comment réaliser le **pour tout** $y \in \text{ensSuccesseurs}(x,g)$...

Avec une matrice d'adjacence

On peut marquer chaque sommet dans un tableau de booléens, initialisé à faux.

```
Parcours_Graphe(g,origine)
2 B : tableau sur [1..n] de booléens
  { Initialisation de B }
4 pour i de 1 à n
    B(i) ← faux
6
L ← ListeVide
8 Insérer(L,origine)
tant que non EstVide(L)
10   x ← Premier(L)
    L ← ExtrairePremier(L)
12   si non B(x) alors
    B(x) ← vrai
    Traiter(x)
    pour i de 1 à n
16     si  $M_{xi} = 1$  alors
        Insérer(i,L)
```

Avec une liste d'adjacence

On peut ajouter un champ **marque** dans la structure sommet :

```
typedef struct cell_sommet {
2   int etiq;
   int marque;
4   liste_arcs successeurs;
   p_sommet suiv;
6 } sommet;

8 void Parcours_Graphe(graphe g, p_sommet origine) {
   p_sommet x;
10  Ensemble L;
   p_arc = a;
12
   L = ListeVide();
14  Insérer(origine,L);
   while(!EstVide(L)) {
16     x = Premier(L);
     L = ExtrairePremier(L);
18     if (! x->marque) {
        x->marque = 1;
20     Traiter(x);
        a = x->successeurs;
22     while (a != NULL) {
        Insérer(a->dest,L);
24     a = a->suiv;
    }
26 }
}
28 }
```
