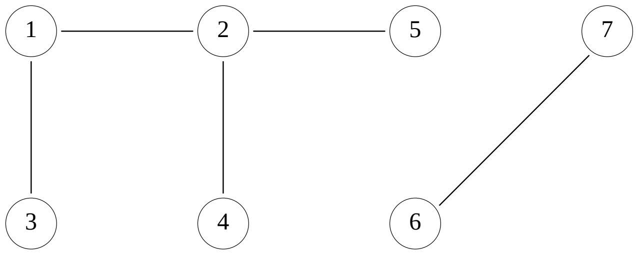
Quelques algorithmes classiques sur les graphes

Définitions. Un *graphe orienté* G est un couple (S, A) où S est un ensemble fini de *sommets* et $A \subseteq S \times S$ un ensemble d'*arcs*. Un *graphe non orienté* G est un couple (S, A) où S est un ensemble fini de *sommets* et A un ensemble fini de *paires* de sommets, appelées *arêtes*. Le graphe G est dit *valué* si l'on adjoint une fonction de A dans $\mathbb{R}$, appelée *fonction de coût*. On appelle *chemin* du sommet x au sommet y toute suite de sommet $x_0 = x, x_1, \dots, x_n = y$ avec $(x_i, x_{i+1}) \in A$ pour $0 \leq i < n$.

Exemple de graphe non orienté :



Les graphes permettent de modéliser de nombreuses structures relationnelles (réseau de transport, connections entre microprocesseurs, contraintes d'ordonnancement, etc) et de nombreux problèmes sur les graphes apparaissent naturellement, parmi lesquels on peut citer les plus fréquemment rencontrés :

- le graphe G est-il connexe et, dans le cas contraire, quelles sont ses composantes connexes ?
- y a-t-il un chemin de x à y et, si le graphe est valué, quel est le plus court de ces chemins ?
- le graphe G contient-il un cycle et, dans le cas contraire, peut-on exhiber un ensemble d'arbres *recouvrant* G ?

1 Représentation

Les deux manières les plus classiques de représenter un graphe sont les suivantes :

Matrice d'adjacence. On suppose que $S = \{1, \dots, n\}$ et on représente A par une matrice booléenne $n \times n$, M, telle que $M_{i,j}$ est Vrai si et seulement s'il y a un arc (resp. une arête) entre les sommets i et j.

Liste d'adjacence. On associe à chaque sommet i la liste de tous les sommets j tels qu'il y ait un arc (resp. une arête) entre i et j. Cette association est naturellement représentée en Caml par une liste d'association. Ainsi le graphe ci-dessus peut être représenté de la manière suivante :

```
type graphe == (int * int list) list ;;
let (g : graphe) = [ 1,[2;3] ; 2,[1;4;5] ; 3,[1] ; 4,[2] ;
                     5,[2] ; 6,[7] ; 7,[6] ];;
```

2 Parcours

Il y a essentiellement deux façons de parcourir tous les sommets d'un graphe, qui sont les suivantes :

2.1 Parcours en profondeur (*depth-first search*)

L'idée est de suivre un chemin le plus loin possible, puis de revenir en arrière jusqu'à trouver un arête menant à un sommet non encore exploré (Ce parcours correspond à la solution classique pour sortir d'un labyrinthe). Plus précisément, (`parcours` i) examine chaque arc (i, j) et, si j n'a pas encore été traversé, on rappelle récursivement (`parcours` j).

Ecrire une fonction `profondeur : graphe -> int list` qui parcourt un graphe en profondeur et rend la liste de ses sommets dans l'ordre où ils ont été rencontrés. Avec le graphe ci-dessus on obtient la liste 1, 2, 4, 5, 3, 6, 7.

2.2 Parcours en largeur (*breadth-first search*)

L'idée est ici au contraire de visiter tous les successeurs immédiats d'un sommet, puis ensuite de se rappeler récursivement sur ces sommets. Ainsi, on examine d'abord tous les sommets à la distance 1 du sommet de départ, puis ceux à la distance 2, etc. Plus précisément, (`parcours` i) visite tous les descendants directs de i non encore visités puis se rappelle récursivement sur chacun d'eux.

Ecrire une fonction `largeur : graphe -> int list` qui parcourt un graphe en largeur et rend la liste de ses sommets dans l'ordre où ils ont été rencontrés. Avec le graphe ci-dessus on obtient la liste 1, 2, 3, 4, 5, 6, 7.

3 Composantes connexes

Un *sous-graphe* G' de G est un graphe (S', A') tel que S' est un sous-ensemble de S et A' l'ensemble des arêtes de G ayant ses deux extrémités dans S'. Une *composante connexe* de G est un sous-graphe G' de G tels que pour tous sommets i et j de G' il existe un chemin de i a j dans G', G' étant maximal pour cette propriété. Exemple : dans le graphe ci-dessus il y a deux composantes connexes, correspondant aux ensembles de sommets $\{1, 2, 3, 4, 5\}$ et $\{6, 7\}$.

Adapter la fonction de parcours en profondeur pour écrire une fonction `composantes : graphe -> int list list` qui détermine les composantes connexes d'un graphe.

4 Recherche d'un cycle

On considère maintenant un graphe orienté. Un *cycle* est un chemin de i à i de longueur non nulle. On se propose de déterminer l'existence d'un cycle dans un graphe. Pour cela, on parcourt le graphe en profondeur en marquant les sommets visités, comme précédemment. Mais lorsque l'on descend à partir d'un sommet i on garde trace de tous les sommets visités *dans la descente*, c'est-à-dire l'orbite du sommet i et, au moment de rappeler la procédure sur les descendants directs, on teste leur appartenance à cette orbite, ce qui est synonyme, le cas échéant, de l'existence d'un cycle.