Théorie des graphes TD/TP 1
Construire des graphes – Modéliser avec des graphes
Graphes d'états – Automates
Coloration de graphes
Matrice d'adjacence – Listes d'adjacence
Implémentation (C)

Question 1.1 : automates

Un automate fini déterministe est un quintuplé (Q, Σ, δ, q_0, F) constitué des éléments suivants :

- un alphabet fini (Σ)
- un ensemble fini d'états (Q)
- une fonction de transition (δ : $Q * \Sigma \rightarrow Q$)
- un état initial ($q_0 \in Q$)
- un ensemble d'états finaux (ou acceptant) $F \subseteq Q$

L'automate prend en entrée un mot et l'accepte (le reconnait) ou le rejette. Le langage associé à un automate est constitué de l'ensemble des mots qu'il reconnaît.

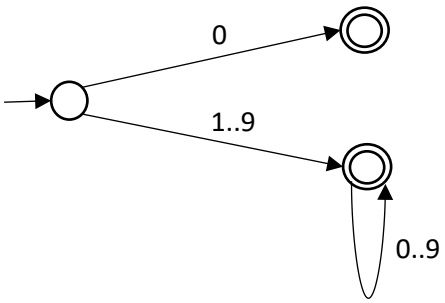
Fonctionnement de l'automate :

- Le processus commence à l'état de départ q_0.
- Les symboles du mot sont lus les uns après les autres.
- À la lecture de chaque symbole, on utilise la fonction de transition δ pour se déplacer vers le prochain état.
- Le mot est reconnu si et seulement si le dernier état atteint est un état final.

On peut schématiser un automate par un graphe dont les sommets sont les états et dont les arcs représentent les transitions. Sur chaque arc est indiqué le symbole lu provoquant le changement d'état.

– Etat initial 
– Etats finaux 

Exemple : automate permettant de valider un nombre entier normalisé



Exercice :

Faire un automate qui reconnaît les nombres écrits en notation scientifique. En notation scientifique, les nombres sont écrits sous la forme d'une mantisse (nombre décimal a tel que $1 \leq |a] < 10$) et d'un exposant entier précédé d'un E. (exemples : 2, 7.3, 2E15, -3.8E-5)

Question 1.2 : Le jeu des allumettes avec un graphe d'états

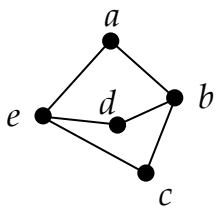
Ce jeu se joue à deux, avec *n* tas de *l* allumettes. A tour de rôle chaque joueur retire un certain nombre d'allumettes de l'un des tas. Celui qui retire la dernière allumette perd la partie.

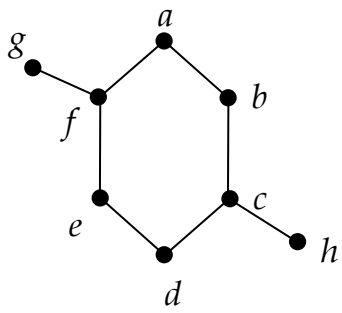
1. Modéliser ce jeu à l'aide d'un graphe d'états pour *n*=2 et *l*=3, dans le cas où un joueur peut retirer 1 ou 2 allumettes à chaque fois.
2. Que doit jouer le premier joueur pour être sûr de gagner la partie ?

Question 1.3 : le graphe en couleurs

L'algorithme de Welsh et Powell est un algorithme « glouton » qui permet de trouver un certain nombre de couleurs pouvant colorer un graphe. Les chercheurs cherchent encore un algorithme efficace permettant de minimiser ce nombre de couleurs (le « nombre chromatique » du graphe). Si vous trouvez cet algorithme, à vous la vie multicolore !

a) Appliquer l'algorithme de Welsh et Powell aux deux graphes représentés ci-dessous.

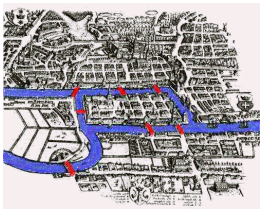


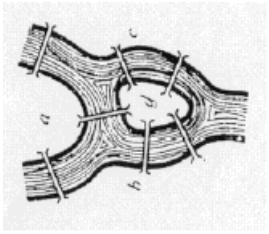


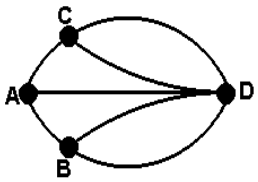
b) Comparer, pour chaque graphe, le nombre de couleurs obtenues par l'algorithme avec son nombre chromatique.

Question 1.4 : Königsberg (1736) et le théorème d'Euler

Sept ponts enjambent la Pregel, reliant quatre quartiers de la ville. Les habitants se demandent s'il existe un trajet leur permettant d'emprunter une seule fois tous les ponts. Les quartiers sont les sommets du graphe, les ponts les arêtes. Il y a quatre sommets (l'ordre du graphe est 4), au sommet A arrivent trois arêtes (le degré de A est 3) :







Euler modélise le problème et ouvre ainsi une nouvelle théorie. Le théorème d'Euler énonce qu'un graphe non orienté admet une chaîne eulérienne si et seulement si il est connexe et admet zéros ou deux sommets impairs. Si tous les sommets sont pairs, il s'agit de cycle eulérien.

a) D'après le théorème d'Euler, existe-t-il un trajet partant d'un point donné, passant par toutes les arêtes une et une seule fois (chaîne eulérienne) ? Ou bien existe-t-il une chaîne eulérienne revenant au point de départ (cycle eulérien) ?

b) À Königsberg, rebaptisée depuis Kaliningrad, il y a deux nouveaux ponts, l'un entre B et C et l'autre entre B et A.
Y a-t-il une chaîne eulérienne ? Où faudrait-il construire un autre pont pour obtenir un cycle eulérien ?

Question 1.5 : modélisation d'un graphe et code en C

Le conseil d'administration d'une société secrète (toute situation avec des personnes existantes est purement fortuite) est composé de 7 personnes : François S, Fabienne C, Julienne P, Louis M, Boubaker D, Ilhem F et JPS. Chacune de ces personnes influence certains de ses collègues conformément au tableau ci-dessous.

Personne	influence…
François S	C, JPS
Fabienne C	JPS
Julienne P	JPS
Louis M	personne
Boubaker D	personne
Ilhem F	personne
JPS	P, M, D, F

a) Représentez, au moyen d'un graphe - en précisant ce que représentent ici les sommets et arcs - les jeux d'influence au sein de ce conseil.

b) Ecrivez la matrice d'adjacence et calculez les demi-degrés intérieur et extérieur des sommets. Interprétez.

c) Pour le TP1, la partie qui suit est à coder en C en respectant les consignes suivantes :

- Définir la structure *Sommet* dans un header .h contenant les propriétés suivantes :
  - Le *nom* du sommet (chaîne dynamique)
  - Le *numéro* du sommet (entier)

- Dans ce même header, définir le prototype de sous-programme suivant, à implémenter dans un fichier source .c :
  - Il reçoit en paramètre le nom et le numéro du sommet. Il initialise les propriétés de la structure *Sommet*, en allouant (ajustant la taille de) son *nom* sur la taille du nom en paramètre, sans oublier de copier les paramètres dans les propriétés du *Sommet* à retourner.

- Définir la structure *Graphe* dans un header contenant les propriétés suivantes :

  - La matrice d'adjacence dynamique d'entiers (ou de booléens si vous préférez) ;
  - L'*ordre* (nombre de sommets du graphe) ;
  - Un tableau dynamique de sommets de type *Sommet*;

- Dans ce même header, définir le prototype des sous-programmes suivants, à implémenter dans un fichier source .c :

  - Un sous-programme qui reçoit en paramètre l'ordre et un pointeur sur le *Graphe*. Il initialise les propriétés de la structure *Graphe* : copier l'ordre en paramètre dans l'ordre du *Graphe*, allouer la matrice d'adjacence du *Graphe* d'*ordre* X *ordre* sommets et allouer le tableau dynamique du *Graphe* d'*ordre* sommets.
  - Un sous-programme qui qui reçoit en paramètre un nom de fichier et un pointeur sur le *Graphe*. Il récupère dans ce fichier texte les données de ce conseil d'administration pour les mémoriser dans les variables : lire l'ordre dans le fichier, appeler le sous-programme précédent pour initialiser les propriétés du *Graphe*, remplir la matrice d'adjacence (avec des 0 et des 1) et le tableau dynamique des sommets du *Graphe*.

- Un sous-programme qui affiche les jeux d'influences des personnes du *Graphe* en paramètre sous la forme « qui influence qui ».

- Écrire le **main** qui effectue les traitements suivants :
  - Allouer un *Graphe* ;
  - Inviter l'utilisateur à saisir le nom du fichier qui contient les données du conseil d'administration ;
  - Appeler le sous-programme qui récupère les données du fichier nommé ;
  - Appeler le sous-programme qui affiche les jeux d'influence des personnes ;

Exercice supplémentaire (bonus)

Question 1.6 : Allocation de fréquences[1] - Implémentation C

On désire attribuer des fréquences à *n* stations émettrices. A cause des interférences, deux stations distantes de moins de d_{min} km ne peuvent émettre avec la même fréquence. Connaissant la position des stations, combien faut-il de fréquences distinctes et comment les attribuer aux différentes stations ?

On peut représenter ce problème sous forme d'un graphe dans lequel les sommets sont les stations et deux stations sont adjacentes si elles sont trop rapprochées pour émettre à la même fréquence. On appellera ce graphe « graphe d'interférence ».
Il sera représenté par listes d'adjacence : chaque sommet possède la liste de ses sommets adjacents (chaque station possède la liste des stations avec lesquelles elle interfère).

Un code de base vous est fourni dans le lien Code de base pour le TP1 :

– Etudiez-le.
– Complétez-le en implémentant les algorithmes de coloration vus en cours (algorithme naïf et algorithme de Welsh et Powell).

– Testez les algorithmes et comparez les résultats obtenus avec les exemples de réseaux donnés sous-forme de fichiers textes. (Dans chaque fichier sont indiqués, la distance minimale d_{min}, le nombre de stations, puis pour chaque station son numéro et ses coordonnées. Les codes de chargement du réseau et de la construction du graphe d'adjacence sont fournis

– Un algorithme de recherche exhaustive (brute force) vous est également donné. Il permet d'obtenir une coloration optimale mais il ne peut être utilisé que sur des réseaux de petite taille car il est de complexité exponentielle.

[1] Les fréquences disponibles sont une ressource rare. Il s'agit de minimiser le nombre de fréquences utilisées ou pour un nombre de fréquences donné, de minimiser les interférences. Les problèmes d'attribution de fréquences dans les réseaux hertziens (ou de longueurs d'onde dans les réseaux optiques) sont en réalité plus complexes. On rejoint des problèmes de multicoloration de graphes ou encore de coloration impropre.