# UE Projet informatique

### Projet « Navigateur », feuille 1

Le but de ce projet est de réaliser un système de navigation qui, étant donnée le plan d'une ville, un point de départ $A$ et un point d'arrivée $B$, affiche les instructions nécessaires pour aller de $A$ à $B$ le plus rapidement possible.

Les objectifs pour la première séance sont d'implémenter une structure de données *file (d'attente)* basée sur le principe « premier entré, premier sorti », une structure de données *graphe* en utilisant une matrice d'adjacence, et de calculer le plus court chemin entre deux sommets du graphe en termes de nombre d'arcs (c'est-à-dire, en supposant que le temps nécessaire pour parcourir chaque arc soit le même).

## Structure de données *file*

Une file d'attente peut être réalisé en utilisant un tableau (appelé aussi *liste* en Python) ou on insère les nouveaux éléments d'un côté et on les extrait (en ordre d'arrivée) de l'autre coté. Écrire un module `file` (dans un fichier nommé `file.py`) contenant les fonctions suivantes :

- `nouvelle_file()`, qui renvoie une file vide ;
- `enfiler(f, x)`, qui modifie la file `f` en insérant l'élément `x` (cette fonction ne renvoie aucun résultat) ;
- `defiler(f)`, qui modifie la file `f` en éliminant le premier élément inséré qui reste et le renvoie comme résultat ;
- `file_vide(f)`, qui renvoie `True` si la file `f` est vide et `False` si elle contient des éléments ;
- `test_file()`, une fonction de test qui crée une file, y insère plusieurs éléments (par exemple, les entiers de 0 à 9 en ordre croissant) et défile les éléments un par un, pour vérifier qu'ils sortent dans le bon ordre.

## Structure de données *graphe*

Un graphe orienté $G$ de $n$ sommets peut être représenté par sa matrice d'adjacence, une matrice de taille $n \times n$ telle que $G_{i,j} = 1$ s'il y a un arc sortant du sommet $i$ et allant vers le sommet $j$, et $G_{i,j} = 0$ si cet arc n'existe pas. Écrire un module `graphe` (dans un fichier nommé `graphe.py`) contenant les fonctions suivantes :

- `nouveau_graphe(n)`, qui renvoie un graphe de `n` sommets sans aucun arc ;
- `taille_graphe(G)`, qui renvoie le nombre de sommets du graphe `G` ;
- `ajouter_arc(G, i, j)` qui ajoute un arc `i` $\rightarrow$ `j` au graphe `G` ;
- `existe_arc(G, i, j)` qui renvoie `True` s'il y a un arc `i` $\rightarrow$ `j` dans le graphe `G` ;
- `afficher_graphe(G)` qui affiche la matrice d'adjacence du graphe `G` sur l'écran ;
- `test_graphe()`, une fonction de test qui crée un graphe aléatoire et l'affiche sur l'écran ;

## Algorithme de parcours en largeur

Le parcours en largeur d'un graphe `G` à partir d'un sommet `s` est décrit par le pseudo-code suivant :

```
algorithme parcours-en-largeur(G, s)
    n := |G| (nombre de sommets)
    pour i := 0 à n - 1 faire
        couleur[i] := blanc
    H := graphe vide de n sommets
    F := file vide
    couleur[s] := rouge
    enfiler(F, s)
    tant que F n'est pas vide faire
        i := défiler(F)
        pour j := 0 à n - 1 faire
            si l'arc (i, j) appartient à G et couleur[j] = blanc alors
                couleur[j] := rouge
                ajouter l'arc (i, j) à H
                enfiler(F, j)
        couleur[i] := vert
    renvoyer H
fin
```

Cette fonction renvoie un graphe `H` qui contient les sommets de `G` et seulement les arcs parcourus en explorant le graphe à partir du sommet `s`.

Écrire, à l'aide des fonctions des modules `file` et `graphe` (utiliser `from file import *` et `from graphe import *`), un module `parcours` contenant les fonctions :

- `parcours_en_largeur(G, s)`, une implémentation en Python de l'algorithme en pseudo-code ;
- `test_parcours_en_largeur()`, une fonction de test qui crée un graphe `G` aléatoire, l'affiche, et affiche le résultat de la fonction `parcours_en_largeur(G, s)` à partir d'un sommet `s` de `G` ;
- `trouver_chemin(G, s, t)`, qui renvoie un tableau contenant les sommets traversés lors d'un chemin de `s` à `t` (utiliser la fonction `parcours_en_largeur`) ;
- `test_trouver_chemin()`, une fonction de test qui crée un graphe `G` aléatoire, l'affiche, et affiche le résultat de la fonction `trouver_chemin(G, s)` à partir d'un sommet `s` de `G`.

## Lecture et écriture du graphe d'un fichier

Ajouter au module `graphe` les fonctions suivantes :

- `lire_graphe(nom_fichier)` qui renvoie un graphe construit à partir de la matrice d'adjacence contenue dans le fichier appelé `nom_fichier`.
- `ecrire_graphe(G, nom_fichier)` qui crée un fichier contenant la matrice d'adjacence de `G`
- `test_lire_ecrire_graphe()`, une fonction de test qui crée un graphe aléatoire, l'affiche, en écrit la matrice d'adjacence dans un fichier, relit le graphe du fichier et l'affiche sur l'écran pour vérifier que l'écriture et la lecture aient ont réussi.