- TP 5. Plus courts chemins entre tous couples. Algorithme de Floyd-Warshall -

Le but de ce TP est de calculer l'ensemble des plus courts chemins entre tous les couples de sommets d'un graphe orienté $D = (V, A)$, puis d'appliquer ce calcul à la recherche de la fermeture transitive d'un graphe orienté.

Langage. Programme en C++. Votre programme pourra contenir (disponible sur le moodle du cours) :

```
#include <iostream>
#include <vector>
using namespace std;

const int N=5;
const int INF=9999;                         //La valeur infinie.

void floydWarshall(int longueur[][N],int dist[][N],int chemin[][N]);
void affichage(int dist[][N],int chemin[][N]);
void itineraire(int i,int j,int chemin[][N]);

int
main()
{

  int longueur[N][N]={{0,2,INF,4,INF},      //Les longueurs des arcs.
                      {INF,0,2,INF,INF},    //longueur[i][j]=INF si l'arc ij n'existe pas
                      {INF,INF,0,2,INF},
                      {INF,-3,INF,0,2},
                      {2,INF,INF,INF,0}};
  int dist[N][N];                           //Le tableau des distances.
  int chemin[N][N];                         //Le tableau de la premiere etape du chemin de i a j.
  floydWarshall(longueur,dist,chemin);
  affichage(dist,chemin);
  return EXIT_SUCCESS;
}
```

- Exercice 1 - À quoi ça sert les cours de graphes ?
On trouve à cette adresse : `http://about-france.com/france-rail-map.htm` un plan des principales lignes de train en France. Essayer de trouver un itinéraire nécessitant strictement plus de deux changements et trouver le trajet correspondant sur le site `https://www.oui.sncf/`. À titre d'exemple, on pourra essayer *Dinan-Mende* ou *Lons le Saunier-Guéret*. Faites la même requête sur le site `https://www.bahn.com/fr/view/index.shtml` ...

- Exercice 2 - Floyd-Warshall.
Initialiser le tableau **dist** à **longueur**.
Écrire une fonction `void floydWarshall(int longueur[][N], int dist[][N])` qui construit le tableau **dist** dont chaque entrée **dist**$[i][j]$ est la longueur minimale d'un chemin de i à j, et vaut *INF* si un tel chemin n'existe pas. Afficher ensuite le tableau **dist**, et vérifier la solution obtenue (à la main...).

- Exercice 3 - Calcul des chemins.
Initialiser le tableau **chemin** de telle sorte que :

— **chemin**$[i][j] = j$ lorsque ij est un arc.
— **chemin**$[i][j] = -1$ dans les autres cas.

Modifier ensuite la fonction `floydWarshall` de sorte que, lors du calcul du tableau **dist**, le tableau **chemin** soit aussi calculé, **chemin**$[i][j]$ devant contenir le voisin sortant de i le long d'un plus court chemin de i à j, ou -1 si un tel chemin n'existe pas. Afficher ensuite le tableau **chemin** et vérifier qu'il soit correct (à la main...).

- Exercice 4 - Calcul d'un itinéraire.

Écrire une fonction `void itineraire(int i, int j, int chemin[][N])` qui, prenant en entrée deux sommets du graphe, affiche un plus court chemin de i à j. L'appel à cette fonction affichera :

```
Entrer le depart : 2
Entrer la destination : 4
L'itineraire est : 2 3 4
```

Faites de même avec le réseau routier proposé sur le moodle du cours.

- Exercice 5 - Fermeture transitive.

La *fermeture transitive* d'un graphe orienté D est une matrice **fermeture** vérifiant **fermeture**$[i][j] = 1$ s'il existe un chemin orienté de i à j dans D, et **fermeture**$[i][j] = 0$ sinon.

En vous inspirant de la fonction `floydWarshall`, écrire une fonction `void fermetureTransitive(int arc[][N], int fermeture[][N])` qui calcule le tableau **fermeture**, le tableau **arc** étant la matrice d'adjacences d'un graphe orienté D. Tester votre fonction sur l'exemple suivant :

```
const int N=6;
int arc[N][N]={{0,0,0,1,0,1},//La matrice d'adjacences du graphe oriente D.
               {1,0,1,1,0,0},
               {0,0,0,1,0,0},
               {0,0,0,0,1,1},
               {0,0,1,0,0,1},
               {0,0,1,0,0,0}};
int fermeture[N][N];         // La matrice de la fermeture transitive de D.
```

- Exercice 6 - Composantes fortement connexes.

Écrire une fonction `void compFortConnexe(int n, int fermeture[][N])` qui affiche les composantes fortement connexes du graphe D. Le résultat pourra être de la forme :

```
Les composantes fortement connexes sont : {1,4}, {2}, {3,5,6}, {7}
```

- Exercice 7 - Pour aller plus loin.

Concernant le calcul de la fermeture transitive :

- Afficher les composantes fortement connexes de telle sorte que si une composante C apparaît avant C', il n'existe pas d'arc de C' vers C.