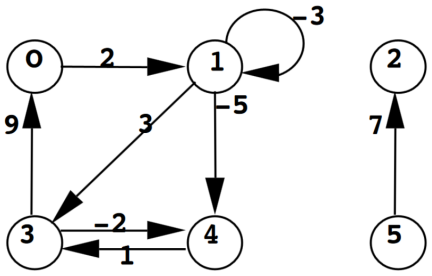
Rapport de Projet d'Informatique :

Graphe, Plus court chemin et Métro

Introduction :

Ce projet consiste en la réalisation d'un programme permettant de déterminer le plus court chemin entre deux stations de métro parisien. Ce programme se basera sur une structure de graphe (représentée ci-dessous). Chaque station représentera un nœud de ce graphe et chaque trajet possible entre deux stations consécutives sera représenté par les arcs de ce graphe.

Il nous faut donc réaliser pour ce projet cette structure de graphe accompagnée des fonctions la rendant utilisable, ainsi que la fonction de lecture d'un fichier regroupant les stations de métro et de la fonction représentant l'algorithme de plus court chemin de Belleman.



Exemple de graphe.
Les nombres sur les arcs représentent le poids des arcs.

Plan :

I.Les structures utilisées

La principale structure utilisée est celle des graphes :

```
#ifndef _Graphe
#define _Graphe
typedef struct {
        unsigned char nom_station[50];     //contient le nom de la station
        unsigned char nom_ligne[50];              //contient le nom de la ligne
        unsigned int num_station;          //contient le numéro de la station
        unsigned int pere;                 //utilize par l'algorithme de Belleman
        double poids_noeud;                //contient le poids du noeud
        Liste arc;              //Liste contenant les arcs partant du sommet considéré
        }* sommet;

typedef struct {
        sommet stations;
        unsigned int nX;
        unsigned int nA;
        }* Graphe;
#endif //_Graphe
```

Cette structure est composée de deux entiers non signés nX et nA qui représente respectivement le nombre de sommets du graphe et le nombre total d'arcs du graphe, ainsi que d'un pointeur sur un tableau qui est représenté ici par la structure sommet. Ce tableau donne pour chacune de ses lignes, le numéro, le nom du sommet, la ligne à laquelle il appartient, le poids du sommet ainsi que les arcs partant de ce sommet.
Les arcs sont représentés par une liste d'éléments comprenant le sommet d'arrivée et le poids de l'arc en question.

La structure ELEMENT, représentant l'élément d'un arc, est décrite ci-dessous :

```
#ifndef _ELEMENT
#define _ELEMENT
typedef struct{
unsigned int Xdest;          //Numero du sommet d'arrivée
double poids_arc;            //poids de l'arc ainsi réalisé
}  ELEMENT;

#endif // _ELEMENT
```

Ces arcs sont regroupés dans une liste donc voici la structure :

```
#ifndef _Liste
#define _Liste

struct cellule {
        ELEMENT val;
        struct cellule* suiv;};

typedef struct cellule* Liste;
#endif // _Liste
```

II. Les groupes de fichier

Nous allons ici voir les différents groupes de fichier dont nous disposons, c'est-à-dire les fichiers .h avec les fichiers .c, ainsi que le fichier du programme et le Makefile.

Il y a donc dans ce projet 4 groupes de fichier :

1. Les fichiers déjà réalisés en séance 9 et 10 portant sur les listes :

- liste.h
- liste.c
- element.h

2. Les fichiers des graphes :
   - graphe.c
   - graphe.h
3. Les fichiers tests : de tests1.c à tests6.c
4. Le Makefile et le main

III. Fonctions et explications

A) Les prototypes des fonctions

Différentes fonctions vont devoir être réalisé durant ce projet :

Premièrement, les fonctions liées à la structure de « graphe ».

En effet, ayant défini une nouvelle structure, nous devons définir les fonctions la caractérisant et nous permettant de remplir ce graphe. Nous avons donc les fonctions suivantes :

Graphe nouveau_graphe(unsigned int nX,unsigned int nA);

Fonction créant un graphe possédant nX sommets et nA arcs. Elle permet en particulier d'allouer de la mémoire à ce graphe.

void affiche_graphe(Graphe g);

Fonction d'affichage du graphe. Elle donne les informations contenues dans chaque sommets (nom de la station, numéro de ligne,...) ainsi que les arcs.

void detruit_graphe(Graphe g);

Fonction de destruction de graphe. Elle permet de libérer la mémoire allouée lors de la création du graphe.

void graphe_ecrit_nX(Graphe g, unsigned int nX);

Fonction permettant de changer le nombre de sommets du graphe.

void graphe_ecrit_nA(Graphe g, unsigned int nA);

Fonction permettant de modifier le nombre total d'arc du graphe.

unsigned int graphe_lit_nX(Graphe g);

Fonction permettant de lire le nombre de sommets du graphe

unsigned int graphe_lit_nA(Graphe g);

Fonction permettant de lire le nombre total d'arcs du graphe

double graphe_lit_poids(Graphe g, unsigned int u);

Fonction permettant de lire le poids du sommet u

void graphe_ecrit_poids(Graphe g, unsigned int u, double val);

Fonction permettant d'écrire le poids du sommet u et d'y mettre le réel val

void graphe_ecrit_poids_arc(Graphe g, unsigned int u, unsigned int v, double val);

Fonction permettant d'écrire le poids de l'arc ayant pour sommet de départ u et pour sommet de destination v et d'y joindre la valeur val.

double graphe_lit_poids_arc(Graphe g, unsigned int u, unsigned int v);

Foncrion permettant de lire le poids de l'arc partant du somemt u et arrivant au sommet v

void graphe_ajoute_arc(Graphe g, unsigned int u, unsigned int v, double val);

Fonction permettant d'ajouter un arc de sommet de départ u et de sommet d'arrivée v et ayant pour poids val.

Lors de ce projet, il faut aussi écrire des fonctions correspondant à notre objectif.

Graphe lit_graphe(char fichier);*

Fonction qui, à partir d'un fichier .csv, crée un graphe et le remplit avec les données du fichier. Cette fonction est complexe car elle doit utiliser les fonctions fscanf et fgets et remplir le graphe. Elle utilise donc des fonctions que nous avons décrite plus tôt et qui doivent donc fonctionner parfaitement pour que cette fonction fonctionne.

void Bellman (graphe g, unsigned int u) ;

Cette fonction est le retranscription en langage C de l'algorithme de Belleman qui ci-dessous. Elle permet de calculer les poids des autres sommets en fonction du sommet de départ de façon à ce qu'ils soient le plus petit possible. Elle fera par la suite l'objet d'une optimisation.

```
Recherche du plus court chemin dans le graphe G à partir du sommet  s
   Mettre les poids de tous les sommets à  +infini
   Mettre le poids du sommet initial à 0 ;
   pour i=1 à Nombre de sommets -1 ou stabilité faire
            /* Itération i*/
   |    pour chaque arc (u, v) du graphe faire
   |    |  si  poids(u) + poids(arc(u, v)) < poids(v) alors
   |    |          /*le chemin passant par l'arc u,v est plus court pour rejoindre v que les
   |    |            précédents chemins trouvés, mettre à jour le poids du sommet v car un
   |    |            chemin plus court  est trouvé */
   |    |    poids(v)= poids(u) + poids(arc(u, v))
   pour chaque arc (u, v) du graphe faire
   |  si poids(u) + poids(arc(u, v)) <poids(v) alors
   |     il y a un circuit négatif dans le graphe, solution impossible
   Ici, On a trouvé la solution
```

Algorithme de Belleman

void pcc(Graphe g, unsigned int u, int v)

Cette fonction utilise la fonction précédente pour calculer le plus court chemin entre v et u en partant de v et en remontant de père en père jusqu'à atteindre u. Elle affiche par la suite ce chemin sur le terminal avec le temps de trajets (poids).

B) Les fonctions

Graphe.c :

```
#include "graphe.h"
#include "liste.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

Graphe nouveau_graphe(unsigned int nX,unsigned int nA)
{
    Graphe g=calloc(1,sizeof(Graphe));
    sommet tableau=calloc(nX,sizeof(*sommet));
    unsigned int i=0;

    g->stations=tableau;
    g->nX=nX;
    g->nA=nA;
    for(i=0;i<nX;i++ )
    {
        (tableau+i)->arc=creer_liste();
    }
    return g;
}

void affiche_graphe(Graphe g)
{   unsigned int i;
    sommet tableau=g->stations;
    for (i=0; i<g->nX; i++)
    {
        printf("Nom de station : %s \n", tableau[i].nom_station);
        printf("Nom de ligne : %s \n", tableau[i].nom_ligne);
        printf("Numero de station : %u \n", tableau[i].num_station);
```

```
                printf("Poids noeud : %lf\n", tableau[i].poids_noeud);
                visualiser_liste(tableau[i].arc);
        }
}

void detruit_graphe(Graphe g)
{
        unsigned int i;
        sommet tableau=g->stations;
        for (i=0; i<g->nX; i++)
        {
                free(tableau[i].arc);
        }
        free(g->stations);
        free(g);
}

void graphe_ecrit_nX(Graphe g, unsigned int nX)
{
        g->nX=nX;
}

void graphe_ecrit_nA(Graphe g, unsigned int nA)
{
        g->nA=nA;
}

unsigned int graphe_lit_nX(Graphe g)
{
        return g->nX;
}

unsigned int graphe_lit_nA(Graphe g)
{
```

```
        return g->nA;
}

double graphe_lit_poids(Graphe g, unsigned int u)        //lit le poids du noeud u du graphe g
{       sommet p=(g->stations);
        double poids_noeud;
        if(g==NULL) {
                printf("graphe vide\n");
                exit(1)
        }
        if(u<1 || u>g->nX) {
                printf("u n'appartient pas au graphe");
                exit(1);
        }
        p=p+u;
        poids_noeud=p->poids_noeud;
        return poids_noeud;
}

void graphe_ecrit_poids(Graphe g, unsigned int u, double valeur)
{
        if(g==NULL) {
                printf("graphe vide\n");
                exit(1);
        }
        if(u<1 || u>g->nX) {
                printf("u n'appartient pas au graphe")
                exit(1);
        }
        (*((g->stations)+u)).poids_noeud=valeur;
}

void graphe_ecrit_poids_arc(Graphe g, unsigned int u, unsigned int v, double valeur)
{
```

```
        unsigned int i=0;
        sommet p=NULL;
        Liste l=creer_liste();
        if(g==NULL) {printf("graphe vide\n"); exit(1);}
        if(u<1 || u>g->nX) {
                printf("u n'appartient pas au graphe")
                exit(1);
        }
        p=(g->stations)+u;
        l=(p->arc);
        while(l->suiv!=NULL && l->val.Xdest!=v)
        {       l=l->suiv;
        }
        if(l->suiv==NULL){
                printf("v n'est pas dans le graphe"); exit(1);
        }
        else
        {       l->val.poids_arc=valeur;
        }
}

double graphe_lit_poids_arc(Graphe g, unsigned int u, unsigned int v)
{
        double valeur;
        unsigned int i=0;
        sommet p=NULL;
        Liste l=creer_liste();
        if(g==NULL) {printf("graphe vide\n"); exit(1);}
        p=(g->stations)+u;
        l=(p->arc);
        while(l->suiv!=NULL && l->val.Xdest!=v)
        {       l=l->suiv;
        }
        if(l->suiv==NULL){
```

```
                printf("v n'est pas dans le graphe"); exit(1);
        }
        else
        {       valeur=l->val.poids_arc;
                return valeur;
        }
}

Graphe lit_graphe(char* fichier)
{       FILE* fstation=NULL;
        Graphe g;
        sommet p;
        int nX;
        int nA;
        unsigned int nbstation;
        double latitude,longitude;
        char mot[512];
        unsigned char nmstation[50];
        unsigned char nomligne[50];
        int nbr_noeud=0;
        int nbr_arc=0;
        unsigned int station_depart;
        unsigned int station_arrivee;
        double poids_arc;

        if ((fstation=fopen(fichier,"r+"))==NULL)  {puts("Erreur ouverture fichier"); exit(1);}
        else
        {       fscanf(fstation,"%d %d",&nX,&nA);
                g=nouveau_graphe(nX,nA);
                printf("nX=%d, nA=%d\n",nX,nA);
                p=g->stations;
                fgets(mot,511,fstation);
                fgets(mot,511,fstation);
```

```
            while(fscanf(fstation,"%u %lf %lf %s",&nbstation,&latitude,&longitude,
nomligne)==4)
            {       fgets(nmstation,511,fstation);
                    strcpy(p->nom_station,nmstation); //rempli la section nom_station
                    strcpy(p->nom_ligne,nomligne);    //rempli la section nom_ligne
                    (p->num_station)=nbstation;
                    p++;
                    nbr_noeud++;          //permet de compter le nombre de noeud rempli
            }
            if(nbr_noeud!=nX) {printf("erreur lecture stations\n"); exit(1);}
            fgets(mot,511,fstation);
            while(fscanf(fstation,"%u %u %lf",&station_depart,&station_arrivee,&poids_arc)
==3)
            {       graphe_ajoute_arc(g,station_depart,station_arrivee,poids_arc);
                    p=(g->stations)+station_depart;
                    nbr_arc++;
            }
            if(nbr_arc!=nA) {printf("erreur lecture arc\n"); exit(1);}
        }
        return g;
}

void graphe_ajoute_arc(Graphe g, unsigned int u, unsigned int v, double val)
{       sommet p=g->stations;
        ELEMENT e;
        unsigned int nbrX;

        e.Xdest=v;              //creation de l'arc
        e.poids_arc=val;
        nbrX=g->nX;
        if(nbrX<u) {puts("Erreur station depart non presente\n"); exit(1);}
        p=p+u;
        p->arc=ajout_queue(e,p->arc);
}
```

```
void bellman(Graphe g, unsigned int s)
{
        unsigned int nonstab= 0, i;
        unsigned int u,v;
        double poids_inf = 10000;
        Sommet p

        for(i=0; i<graphe_lit_nX(g); i++){
                graphe_ecrit_poids(g, i, poids_inf);
        }
        graphe_ecrit_poids(g,d,0);
        while (!nonstab){
                nonstab = 1;
                for(i=0; i<graphe_lit_nA(g); i++){
                        u= g.station[i].pere;
                        v= g.station[i]->arc.Xdest;
                        if( graphe_lit_poids(g, u) + graphe_lit_poids_arc(g,u,v) < graphe_lit_poids
(g, v))
                        {
                                graphe_ecrit_poids(g,v) = graphe_lit_poids(g,u) +
graphe_lit_poids_arc(g,u,v));
                                g.station[v].pere=u;
                                nonstab = 0;
                         }
                }
        }
}

void pcc(Graphe g, unsigned int u, int v)
{
        unsigned int position = v, nbstations=0, nbchangements=0,i,;
        unsigned int itineraire[100];
```

```
        unsigned int changements[100];
        changements[0]=-10;
        itineraire[0] = v;
        bellman(g, u);
        if( graphe_lit_poids(g, v)== 10000){
                printf("Aucun itineraire trouve\n");
        }
        else{
                double temps = graphe_lit_poids(g, v);
                while(position != u)
                {
                        position = g.station[position].pere;
                        if (strcmp(g.station[position].nom_ligne, g.station[itineraire
[nbstations]].nom_ligne)){
                                printf("Changement entre %s et %s\n", g.station[itineraire
[nbstations]].nom_ligne, g.station[position].nom_ligne);
                                changements[nbchangements] = nbstations;
                                nbchangements++;
                        }
                        nbstations++;
                        itineraire[nbstation] = position;
                }
                printf("Trajet entre les stations %s et %s :\n\n", g.station[u].nom_ligne,
g.station[v].nom_ligne);
                printf("Durée du trajet : %d\n ", temps);
                printf("Nombre de stations : %u\n", nbstations);
                printf("Nombre de changements : %u\n\n", nbchangement);
                printf("Itineraire : \n");
                for(i=0;i<nbstations;i++)
                {       printf("Etape %u : %s\n", g.stations[itinéraire[i]].nom_ligne,g.stations
[itinéraire[i]].num_ligne);
                }
        }
}
```

Fichier main.c

```
int main()
{       Graphe g;
        unsigned  int depart;
        unsigned int arrivee;
        char cond;
        char* fichier="metro.csv";

        printf("Bienvenu dans l'interface d'itineraire\n");
        g=lit_graphe(fichier);
        while(cond!='0')
        {
                printf("1:nouveau trajet\n");
                printf("0: quitter\n");
                fflush(stdout);
                cond=getchar();
                switch(cond){
        case '1' :
                        printf("nouveau trajet\n");

                        printf("entrez la station de depart\n");
                        scanf("%u\n",&depart);
                        printf("entrez la station d'arrivee\n");
                        scanf("%u\n",&arrivee);

                        pcc(g, depart, arrivee)
                        break;
        case '0':
                        printf("quitter\n");
                        break;
                }
                getchar();
        }
```

```
        return 1;
}
```

Fichier Makefile :

```
CFLAGS=-c -g -Wall -Wextra
LDFLAGS=

test1 :test1.o graphe.o liste.o
        gcc -o $@ $^  $(LDFLAGS)
test2 :test2.o graphe.o liste.o
        gcc -o $@ $^  $(LDFLAGS)
test3: test3.o graphe.o liste.o
        gcc -o $@ $^  $(LDFLAGS)
test4:test4.o graphe.o liste.o
        gcc -o $@ $^  $(LDFLAGS)

main:main.o graphe.o liste.o
        gcc -o $@ $^  $(LDFLAGS)

%.o: %.c
        gcc $(CFLAGS) $<

all: test1 test2 test3 test4 main

clean:
        rm -f *.o test? *~
        rm -f *.o main *~
```

IV. Les Tests

Durant ce projet, different tests vont être réalisés. Ces tests sont de deux types :

-Les tests unaires permettront de tester une fonction indépendamment des autres. Ces tests porteront surtout sur les fonctions de la structure graphe et la fonction de calcul de plus court chemin. Quatre tests seront ainsi réalisés.

-Les tests globaux, qui testeront les fonctions globales et permettront de valider le fonctionnement du programme en entier.

A) Test1.c

Ce fichier permet de tester les fonctions nouveau_graphe, graphe_lit_nX, affiche_graphe, detruit_graphe.

```
int main()
{
        printf("Test 1\n");
        printf("On prends nX et nA = 2\n");
        unsigned int  nX,nA;
        unsigned char nom[50];
        nX=2;
        nA=2;
        Graphe g = nouveau_graphe(nX, nA);
        unsigned int i;
        sommet tableau=g->stations;
        for (i=0; i<graphe_lit_nX(g); i++)
        {
                printf("Donnez le nom de station %i\n", i);
                scanf("%s", &nom);
                strcpy(tableau[i].nom_station, nom);
                getchar();
        }
        printf("Affichage du graphe:\n" );
```

```
        affiche_graphe(g);
        detruit_graphe(g);
        printf("graphe detruit \n test affiche");
        affiche_graphe(g);
        return 0;
}
```

B) Test2.c

Ce code permet de tester les fonctions graphe_ecrit_poids, graphe_lit_poids, graphe_ecrit_poids_arc et graphe_lit_poids_arc

```
int main()
{       ELEMENT e;
        Graphe g;
        sommet p;
        Liste A=creer_liste();
        unsigned int nX,nA,i;
        double val1, val2,lecture;
        unsigned int u,v;

        nX=3;
        nA=6;
        val1=2.6;
        val2=8.4;
        e.poids_arc=0;
        u=1;
        v=2;

        //creation du graphe de parametre nX et nA permettant de tester les fonctions
```

```
g=nouveau_graphe(nX,nA);
p=g->stations;
for (i=0;i<nX;i++)
{       p->num_station=i;
        A=p->arc;
        e.Xdest=i+1;
        A=ajout_tete(e,A);
        e.Xdest=i;
        A=ajout_tete(e,A);
        p=p+1;
}
//Visualisation du graphe
affiche_graphe(g);
//test fonction ecrit poids
puts("ecriture poids sommet\n");
graphe_ecrit_poids(g,u,val1);
affiche_graphe(g);
//test fonction lit_poids
puts("lecture du poids du sommet u");
lecture=graphe_lit_poids(g,u);
printf("valeur voulue : %lf     valeur obtenue : %lf /n",val1,lecture);
//test fonction ecrit poids arc
puts("ecriture poids arc");
graphe_ecrit_poids_arc(g,u,v,val2);
affiche_graphe(g);
//test fonction lit poids arc
puts("lecture poids arc");
lecture=graphe_lit_poids_arc(g,u,v);
printf("valeur voulue : %lf     valeur obtenue : %lf /n",val2,lecture);
}
```

C) test3.c

Ce programme permet de tester la fonction lit_graphe.

```
int main()
{
	Graphe g;
	char* fichier="graphe1.csv";
	g=lit_graphe(fichier);
	affiche_graphe(g);
	return 1;
}
```

D) test4.c

Ce code permet de tester la fonction graphe_ajoute_arc.

```
int main()
{
	Graphe g;
	unsigned int depart=2, arrivee=3;
	double valeur=5.6;
	g->nX=3;
	g->nA=3;
	for(i=0;i<4;i++)
	{
		(*(g->stations)+i)->num_station=i;
	}
	puts("Graphe de depart");
	affiche_graphe(g);
	graphe_ajoute_arc(g,depart,arrivee,valeur);
	puts("Graphe apres ajout");
```

```
        affiche_graphe(g);
        return 1;
}
```

V. Répartition du travail et planning

Nous nous sommes répartis les taches comme suit :

Laure : fonctions du graphe :

-Graphe nouveau_graphe(unsigned int nX,unsigned int nA);

-void affiche_graphe(Graphe g);

-void detruit_graphe(Graphe g);

-void graphe_ecrit_nX(Graphe g, unsigned int nX);

-void graphe_ecrit_nA(Graphe g, unsigned int nA);

-unsigned int graphe_lit_nX(Graphe g);

-unsigned int graphe_lit_nA(Graphe g);

Fonctions du projet :

- void bellman(Graphe g, unsigned int s)

- void pcc(Graphe g, unsigned int u, int v)

Caroline : fonctions de graphe

-double graphe_lit_poids(Graphe g, unsigned int u);

-void graphe_ecrit_poids(Graphe g, unsigned int u, double val);

-void graphe_ecrit_poids_arc(Graphe g, unsigned int u, unsigned int v, double val);

-double graphe_lit_poids_arc(Graphe g, unsigned int u, unsigned int v);

Fonctions du projet :

-Graphe lit_graphe(char* fichier);

-void graphe_ajoute_arc(Graphe g, unsigned int u, unsigned int v, double val);

Ainsi, chacune d'entre nous peut se familiariser avec l'utilisation de la structure "graphe" en réalisant des fonctions caractérisant cette structure et participer aux fonctions du projet en ayant chacune une fonction cruciale à la réalisation du projet (lit_graphe et pcc).
Nous nous sommes imposés des delais pour la réalisation de chaque groupe de fonction et leurs tests.

Les fonctions sur la structure de Graphe et leurs tests doivent être réalisées avant le 10 Mai. Le reste des fonctions correspondant au programme du projet doivent être réalisés avant le 22 Mai.

Conclusion :

Durant ce projet, nous avons pu réaliser un programme permettant de calculer le plus court chemin entre deux stations du métro parisien. Nous avons réussi à réaliser ce projet en concevant des fonctions permettant d'utiliser la structure de graphe et des fonctions correspondant au projet.