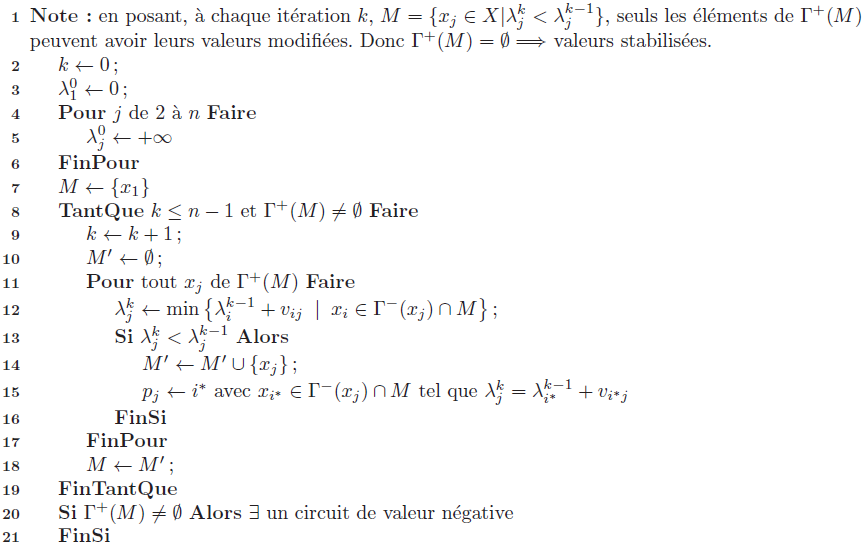
# Rapport de projet algorithme Bellman-Ford optimisé L3-info 2016

Auteurs : André ERBA, Olivier Mertz

## Présentation du problème :

Il s’agit d’implémenter en c++ l’algorithme Bellman-Ford optimisé de recherche du plus court chemin dans un graphe quelconque avec détection des circuits absorbants.



Nous avons essayé de suivre au plus cet algorithme :

**Pour cela, nous avons implémenté les classes Vertex et Edge (sommets et arêtes)  les plus simples possibles:**

/\*

\* Vertex class

\*/

class Vertex{

private:

string name;

public:

Vertex(string name) :name(name){

}

string getName(){

return name;

}

/\*

\* Edge class

\*/

class Edge{

private:

double weight;

Vertex \* source;

Vertex \* destination;

public:

Edge(Vertex \*s, Vertex \*d, double w) : source(s), destination(d), weight(w){

}

Vertex \* getVertexSource(){

return source;

}

Vertex \* getVertexDestination(){

return destination;

}

void setWeight(double w){

weight = w;

}

double getWeight(){

return weight;

}

};

## Objets utilisé

**La classe Graphe contient la méthode shortest\_path() qui déroule l’algorithme Bellman-Ford optimisé.**

**La classe Graphe contient également la méthode add\_edge() qui permet de saisir les arêtes du graphe. Les sommets sont automatiquement ajoutés s’ils n’existaient pas. Cette méthode met également à jour vertices\_list et edges\_list qui sont les listes contenant tous les sommets et toutes les arêtes.**

**Une donnée privée lambda de type** map<string, double> permet de connaitre la valuation d’un sommet. On y accède en O(1).

**Les ensembles** marked\_vertex et alternate\_marked\_vertex permettent de stocker les sommets marqués.

successors\_list de type map<string, Successors \*> permet de parcourir tous les successeurs d’un sommet.

Enfin predecessor de type map<string, string> permet de stocker le predecesseur de chaque sommet sur le plus court chemin en cours d’évaluation.

Une fois qu’on a exécuté ford-opti, on peut connaitre le plus court chemin pour chacun des sommets si la méthode renvoie 1, sinon elle indique qu’il y a un cycle.

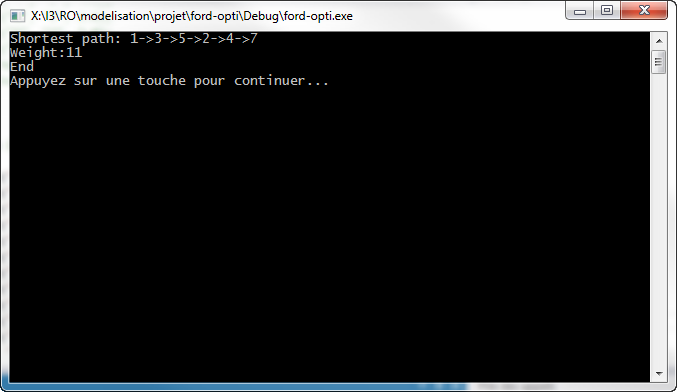
**Si un sommet « X » ne possède pas de plus court chemin, alors lambda(X) reste avec sa valeur maximale** DBL\_MAX **et la méthode** getShortestPathTo("X") **indique qu’il n’y a aucun chemin pour accéder à « X »**

## Tests :

Dans le fichier main.cpp, un certains nombre de graphes fait en cours ont été évalués par l’algorithme.

Tous ont donné un résultat exact pour le plus court chemin.

Exemple TD 4 Exo 1 (qui sera lancé par défaut) :



Un fichier qui représente un graphe plus conséquent peut également être testé en dé-commentant Les lignes comprises entre //TEST FICHIER et  //FIN TEST FICHIER

