EFREI 2016/2017 -- L3 Asyria – Théorie des Graphes -- TP

TP noté.
La note sera intégrée à la note finale, selon la formule : note finale = 0,8 DE + 0,2 TP
Travail à effectuer lors des 2 séances de TP prévues (08/11 et 14/11).
Envoi du résultat de votre travail par email, au plus tard le 20/11 à 23h59.
A réaliser par groupes de 5 ou 6.
Langage de programmation : C ou C++.

Les informations / idées de mise en œuvre ne sont données ici qu'à titre indicatif. Vous pouvez tout à fait prendre une autre approche, pourvu que votre programme fasse exactement les mêmes traitements.

Objectif : système de calcul des calendriers et marges pour un problème d'ordonnancement.

Ce « mini projet » est constitué de plusieurs étapes. Vous devez aller aussi loin que possible.

Etape 1 – Lecture de fichier de contraintes et stockage dans un graphe / Impression de la matrice d'adjacence et des valeurs

Le tableau de contraintes est représenté dans un fichier au format « texte ». Lors de sa lecture, le programme initialise une matrice d'adjacence et une matrice de valeurs.

Voir annexes pour la structure du fichier en entrée, une proposition de structures de données, et un exemple de code C++.

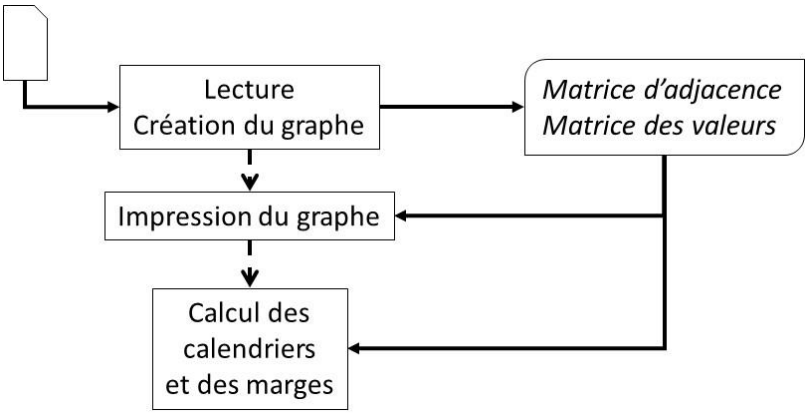
L'impression se fera ensuite sous forme de tableaux, par exemple :

Adj	0	1	2	3	4	5	6
0	0	1	0	0	0	1	0
1	0	0	1	0	1	0	0
2	0	0	0	0	1	0	0
3	0	0	0	0	0	0	1
4	0	0	0	0	0	0	1
5	0	0	1	1	0	0	0
6	0	0	0	0	0	0	0

Val	0	1	2	3	4	5	6
0		0				0	
1			10		10		
2					20		
3							30
4							40
5			50	50			
6							

On remarque qu'avec un minimum d'effort, on peut n'imprimer que les valeurs significatives de la matrice de valeurs.
L'impression de la seule matrice Val peut suffire, puisqu'une valeur imprimée correspond à 'vrai' dans la matrice Adj, alors que l'absence de valeur correspond à un 'faux'.

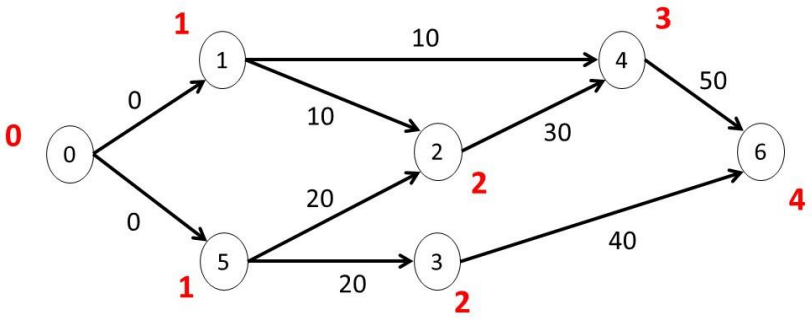
Etape 2 – Calcul des calendriers et marges



A ce stade, vous disposez d'un graphe sans circuit. Vous avez deux solutions pour calculer les calendriers (« au plus tôt » / « au plus tard ») :

Calcul sur la base du rang

Commencez par mettre en œuvre une procédure de calcul des rangs associés aux sommets. Vous obtiendrez par exemple les rangs illustrés sur le schéma ci-dessous :



Le calcul du calendrier au plus tôt se fait alors en prenant en compte les sommets dans l'ordre croissants du rang, et en appliquant la formule qui donne un rang en fonction du rang des prédécesseurs. Le rang du sommet « début » est fixé à 0.

Initialisation

Rang	0	1	1	2	2	3	4
Tâche	0	1	5	2	3	4	6
	0						

Par ordre croissant des rangs

Rang	0	1	1	2	2	3	4
Tâche	0	1	5	2	3	4	6
	0	0	0	20	20	50	100

Pour le calcul du calendrier au plus tard, vous commencerez par initialiser la date au plus tard de fin de projet à la même valeur que sa date au plus tôt.
A vous de voir comment mettre en œuvre le calcul…

Méthode récursive

Sur la base de la définition récursive du calendrier au plus tôt, et en l'absence de circuit, il est possible d'utiliser une procédure récursive pour le calcul du calendrier. Cette procédure est lancée avec le sommet « fin de projet » en paramètre, et entraîne le calcul du calendrier pour tous les sommets.

Soit la définition par induction :

$$D^{+TOT}(t) = \max_{x \in \Gamma^{-1}(t)} [\, D^{+TOT}(x) + v(x,t) \,]$$

$$D^{+TOT}(\alpha) = 0 \qquad \text{avec } \alpha \text{ la tâche de début de projet}$$

Soit une fonction récursive mettant en œuvre cette définition de la date au plus tôt avec :
pour paramètre IN : une tâche,
pour paramètre IN/OUT (passage par référence) : un tableau des dates au plus tôt
pour résultat éventuel la date au plus tôt de la tâche en paramètre IN.

Le calcul du calendrier au plus tôt peut se faire en initialisant le tableau des dates au plus tôt à « vide » (par exemple « -1 » pour toutes les tâches), et en appelant cette fonction avec pour paramètre le sommet « fin de projet).

Voir l'exemple donné en cours pour le calcul récursif du rang : le principe est le même.

Après le calcul des calendriers, on calcule bien entendu les marges.

Calendriers et marges sont ensuite affichés sous forme de tableaux.

Etapes suivantes

Mettez en œuvre d'autres types de contraintes.

Commencez par celles ne créant pas de circuit. Le code de calcul des calendriers pourra alors être réutilisé.

Lorsque vous voudrez mettre en œuvre des contraintes menant à la création de cycle, il vous faudra disposer d'une méthode de calcul des chemins de valeur maximale. L'algorithme de « Warshall-Floyd », vu pour le calcul des chemins de valeur optimale, peut être utilisé dans la mesure où il n'y a pas de circuit à valaur strictement positive.
Cet algorithme a été vu pour le calcul des chemins de valeur la plus faible : en changeant les opérateurs utilisés, on peut le modifier pour une valeur la plus grande.
On note qu'il n'est absolument pas nécessaire ici de calculer les chemins eux-mêmes (matrice « P » de l'algorithme) ; seules les valeurs numériques comptent (matrice « L »).

Rendu du travail

Les graphes et tableaux de contraintes à utiliser pour tester votre programme vous seront fournis en séance. Vous devrez les saisir dans un fichier d'entrée pour qu'ils puissent être lus par votre programme.

IMPERATIF : le titre (sujet) de votre email devra être de la forme suivante :

« L3A-TG-TP-*nom1-nom2-…* »

> J'utilise des filtres automatiques à la réception des emails. Toute défaillance peut entraîner la perte de messages.

Vous devez joindre en pièces jointes :

- votre code source (fichiers de type .c / .cpp ou .h) ;
- vos fichiers contenant les graphes de test utilisés en entrée de votre programme (fichiers de type .txt) ;
- les traces d'exécution de votre programme (fichiers de type .txt).

Remarques :

- aucun fichier exécutable,
- aucun fichier « projet » lié à CodeBlocks, Visual Studio ou tout autre outil de développement (.cbp, .sln, …),
- pas de copie d'écran au format image (.jpeg, …) pour les traces d'exécution.
- pas de fichier archive zip, rar ou autre. Mettez autant de pièces jointes que nécessaire.

Tous vos fichiers doivent être préfixés par vos noms (par exemple « dupont-durant-durand-tp.cpp », « dupon-durant-durand-traces.txt », « dupont-durant-durand-g01.txt »).
Tous les fichiers doivent pouvoir être installés dans le même répertoire (pas de sous-répertoire).

Dans le corps du message d'envoi, vous devrez indiquer ce qui est fait ou non.

Procédure de test :
Lorsque je teste vos programmes, je procède de la façon suivante :

- Toutes vos pièces jointes sont dans le même répertoire
  « …/Visual Studio 2015/Projects/L3A-1617-TP/L3A-1617-TP/ »
  Si vous les préfixez correctement, il n'y aura aucun conflit.
- J'ouvre le projet « L3A-1617-TP »
- J'ajoute les éléments du groupe que je corrige dans le projet
  (je les repère à l'aide du préfixe).

Si vous respectez les règles énoncées, il n'y aura aucun souci.

Annexe 1 – Syntaxe du fichier en entrée

Faites simple. Ne prenez pas de temps à mettre en œuvre un programme qui vérifie la structure du fichier ; ce n'est pas l'objectif du mini projet. Assurez-vous simplement que vous ne faites pas d'erreur lorsque vous le créez.

Par exemple :

`5`	Nombre de tâches
`5`	Nombre de contraintes
`1 10`	Durée de la tâche 1
`2 20`	Durée de la tâche 2
…	…
`5 50`	Durée de la tâche 5
`1 2 1`	Contrainte « 2 ne peut débuter que lorsque 1 est terminée »
`1 5 1`	Idem pour '5'
…	

Les lignes représentant les contraintes sont sous la forme :

<type de la contrainte> <sommet X> <sommet Y> <valeurs nécessaires>

Dans un premier temps, on ne s'intéressera qu'aux contraintes de type '1' : « X ne peut commencer que lorsque Y est terminée ».

On étendra ensuite cette liste de code :

1	Y ne peut commencer que lorsque X est terminée
2	Y doit commencer immédiatement lorsque X se termine
3	X et Y doivent commencer en même temps
4	…

A vous de voir jusqu'où vous pourrez aller.

Pour chaque type de contrainte, il y a 0, 1 ou plusieurs valeurs associées.

Annexe 2 – Proposition de structure de données

On peut par exemple utiliser les définitions suivantes :

```
typedef struct {
      int   code ;
      int   sommetX ;
      int   sommetY ;
      int   valeur ;
} t_contrainte ;
```

Le « code » correspond au type de la contrainte.
Pour les contraintes simples telles que « Y ne peut commencer que lorsque X est terminé », on utilisera par exemple le code '1'. A vous d'ajouter d'autres codes en fonction de ce que vous mettrez en œuvre.
Dans un premier temps, vous ne prendrez en compte que des contraintes de type '1'.
Attention : selon les types de contraintes que vous mettrez en œuvre, vous pouvez être amenés à modifier cette définition, par exemple en utilisant autre chose que 'int' pour le champ 'valeur', ou en ajoutant une seconde valeur.

```
typedef struct {
      int               nbTaches ;
      int               nbContraintes ;
      int*              durees ;
      t_contrainte*     contraintes ;
      int               nbSommets ;
      bool**            MAdj ;      // MAdj[x][y]=TRUE ==> il existe arc (x,y)
      int**             MVal ;      // MAdj[x][y]=TRUE ==> MVal[x][y]=valeur de
l'arc
} t_graphe ;
```

La sémantique des 4 premiers champs est évidente.
Pour un graphe MPM, on aura `nbSommets = nbTaches+2`.
`MAdj` et `MVal` représentent le graphe construit. Bien que la matrice des valeurs `MVal` est inutile dans un premier temps, on s'efforcera de la construire pour prévoir la mise en œuvre de contraintes autres que celles de type '1'.

Annexe 3 – Lecture d'un fichier de contraintes et création des matrices du graphe

Considérons l'exemple suivant :

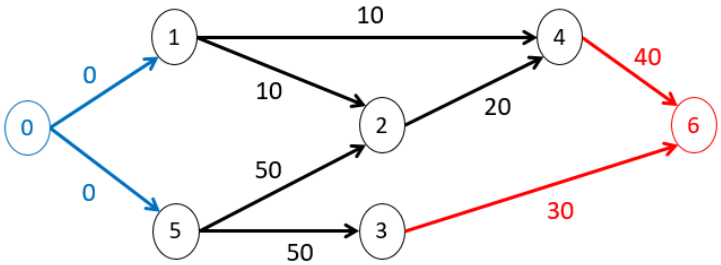
Tableau de contraintes :

Tâches	1	2	3	4	5
Durées d'exécution	10	20	30	40	50
Contraintes	Aucune	1 et 5	5	1 et 2	aucune

Ex. : '3' n'a aucune contrainte, tandis que '4' ne peut débuter que lorsque '1' et '2' sont terminés

Graphe associé :

Sommets :
1 à 5 : tâches
6 : fin de projet
0 : début de projet



Ex. : Entre le début de '2' et le début de '4', il doit s'écouler un temps au moins égal à 20
Entre le début du projet '0' et le début de '5', il n'y a pas de délai nécessaire.
Règle générale : Si, entre le début de 'x' et le début de 'y', il doit s'écouler un temps au moins égal à 't', alors on a un arc (x,y) de valeur t.

Le fichier en entrée, selon la structure évoquée précédemment, est :

```
5
5
1 10
2 20
3 30
4 40
5 50
1 2 1
1 2 5
1 3 5
1 4 1
1 4 2
```

Traitement du fichier :

Ligne 1	Le nombre de tâches permet de définir le nombre de sommets, et d'allouer les matrices `durees`, `Madj` et `Mval`. La matrice d'adjacence est initialisée à 'faux'.
Ligne 2	Le nombre de contraintes permet d'initialiser le tableau de `contraintes`.
Lignes 3 à nbTaches+2	La matrice `durees` est remplie.
Lignes nbTaches+3 à nbTaches+3+nbContraintes	Le tableau `contraintes` est initialisé.

En fin de lecture, les contraintes sont analysées afin de remplir correctement les tableaux Madj et Mval.

Le code ci-dessous vous donne un squelette de programme permettant d'effectuer tous ces traitements, en utilisant la définition du type `t_graphe` donnée précédemment.

```
…
#include <fstream>
…

#define FICHIER_GRAPHE ”C01.txt”

int main () {

      // Déclaration graphe

      t_graphe * G = new t_graphe ;

      // Lecture du graphe sur fichier

      ifstream fg ( FICHIER_GRAPHE ) ;
            // FICHIER_GRAPHE est une variables (ou autre)
            // donnant le nom du fichier

      fg >> G->nbTaches ;
      fg >> G->nbContraintes ;
      G->nbSommets = G->nbTaches +2 ;
      G->durees = new int [ G->nbTaches +1 ] ;
            // Tâches numérotées à partir de 1
            // G->durees[0] non utilisé
      G->contraintes = new t_contrainte [ G->nbTaches +1 ] ;
            // Contraintes numérotées à partir de 1
            // G->contraintes[0] non utilisé
      G->MAdj = new bool * [ G->nbSommets ] ;
      G->MVal = new int * [ G->nbSommets ] ;
            // Sommet 0 : début de projet
            // Sommets 1 à nbTaches : tâches
            // Sommet nbTaches+1(=nbSommets-1) : fin de projet
      for ( int s = 0 ; s < G->nbSommets ; s++ ) {
            G->MAdj[s] = new bool [ G->nbSommets ] ;
            G->MVal[s] = new int [ G->nbSommets ] ;
            for ( int extTerm = 0 ; extTerm < G->nbSommets ; extTerm++ ) {
                  G->MAdj[s][extTerm] = false ;
            };
      };
      int tacheCourante ;
      int dureeTache ;
      for ( int t = 1 ; t <= G->nbTaches ; t++ ) {
            // 1 ligne pour chaque tâche
            fg >> tacheCourante ;
            fg >> dureeTache ;
            G->durees[tacheCourante] = dureeTache ;
      } ;
      for ( int c = 1 ; c <= G->nbContraintes ; c++ ) {
            // 1 igne pour chaque contrainte
            fg >> G->contraintes[c]->code ;
            switch ( G->contraintes[c]->code ) {
            case 1 :    // la tâche sommetX
                        // ne peut commencer que si
                        // la tâche sommetY est terminée
                        fg >> G->contraintes[c]->sommetX ;
                        fg >> G->contraintes[c]->sommetY ;
                        break ;
            …
            } ;
      }
```

```
        … mise à jour matrices
        … impression du graphe

        …

        return 1 ;
};
```