MINI-PROJET

Conditions de travail

Le travail s'organise sur : 2 séances de TP, du travail personnel, et une soutenance.
A réaliser par équipes. Les équipes sont constituées à la base de 3 étudiants. Selon le nombre d'étudiants dans chaque groupe TD/TP, les enseignants vous communiquerons si des équipes de 2 ou 4 sont autorisées, et combien.
Tout changement de constitution des équipes après le 1er TP est interdit, sauf accord de l'enseignant. Lui demander avant, et ne pas le mettre devant le fait accompli (qu'il pourra refuser).

Outils génériques de manipulation de graphe

Une première étape du travail consiste en la représentation de graphes en mémoire, et en la mise en œuvre de quelques algorithmes généraux.

1. Graphes à prendre en compte

Votre programme doit être capable de prendre en compte n'importe quel graphe orienté valué, sachant que :

- Les sommets sont des numéros entiers de '0' à 'n-1' pour un graphe contenant 'n' sommets ;
- Les valeurs associées aux graphes sont des nombres entiers quelconques ;
- Soit x et y deux sommets du graphe : il y a au plus un arc de x vers y. ;
- Il n'y a pas de boucle ;
- Il peut y avoir des sommets isolés (sans prédécesseur ni successeur).

2. Mise en œuvre d'un graphe en mémoire / Lecture du graphe sur fichier

Vous commencerez par choisir une mise en œuvre de graphes en mémoire.
Lorsque ce choix a été fait, et que les définitions appropriées sont disponibles dans votre programme, vous développerez le code nécessaire à la lecture d'un graphe sur fichier.

Remarque : ici, il n'y a pas de ligne signifiant le nombre d'arcs, comme dans le document décrivant le TP1. C'est logique : le nombre d'arcs se calcule en lisant les arcs. Vous avez le choix d'utiliser comme sources des fichiers .txt contenant le nombre d'arcs à la deuxième ligne, comme dans TP1, ou ne le contenant pas, comme ici. (De toute façon, votre envoi de fichiers projet contiendra vos sources .txt, donc nous pourrons voir quel variante vous utilisez). Le code proposé en annexe est écrit en supposant que la ligne « nombre d'arcs » est absente.

Ex .

3	Nombre de sommets
1 2 0	arc (1,2) à valeur '0'
2 0 -1	arc (2,0) à valeur '-1'
0 1 1	arc (0,1) à valeur '1'
1 0 2	arc (1,0) à valeur '2'
-1	fin de fichier

Le code fourni en annexe peut être utilisé et modifié à votre gré.

Remarque : Une fois cette étape mise en œuvre, votre programme ne doit utiliser que le contenu de vos structures de données. Il ne doit pas accéder une nouvelle fois au fichier d'entrée.

3. Affichage du graphe

Après avoir sauvegardé un graphe dans votre structure de données, votre programme doit afficher le graphe sous forme textuelle, par exemple :

- une matrice d'adjacence dans laquelle les valeurs 'vrai' sont remplacées par la valeur associée à l'arc ; les valeurs 'faux' sont remplacées par un symbole de votre choix ;
- une liste des arcs sous la forme '(extrémité initiale, extrémité terminale, valeur)' ;
- …

Cette étape doit être mise en œuvre en parcourant votre structure de données, et non pas en parcourant le fichier d'entrée ! Elle permettra de vérifier que votre structure de données est correctement initialisée.

Remarque : Arrivé à ce stade, vous avez la certitude que votre structure de donnée contient bien la représentation de votre graphe. N'hésitez pas à effectuer plusieurs tests en utilisant des fichiers d'entrée différents.

4. Fermeture Transitive

Il s'agit tout simplement de calculer le graphe correspondant à la fermeture transitive de celui chargé par votre programme à l'étape 2.
Le résultat doit être stocké dans une structure de données similaire à celle utilisée pour le premier graphe.

En fin de traitement, vous devez reprendre votre code fait pour l'étape 3 pour afficher le résultat de la fermeture transitive.

Votre programme doit indiquer clairement les résultats intermédiaires de votre algorithme, c'est-à-dire le graphe intermédiaire calculé à chaque itération.

5. Détection de circuit

Sur la base de la structure de données que vous avez choisie, vous devez écrire une fonction permettant de déterminer si le graphe contient ou non au moins un circuit.

Vous pouvez utiliser l'une ou l'autre des deux méthodes vues en cours :

- par l'élimination successive des points d'entrée (et/ou des points de sortie),
- par le calcul de la fermeture transitive du graphe.

Compte tenu du traitement que vous devez effectuer au point 4, le choix est assez simple…

Attention : vous devrez utiliser à nouveau votre graphe après cette détection, donc n'hésitez pas à travailler sur une copie de votre structure de données si vous devez la modifier durant la vérification…

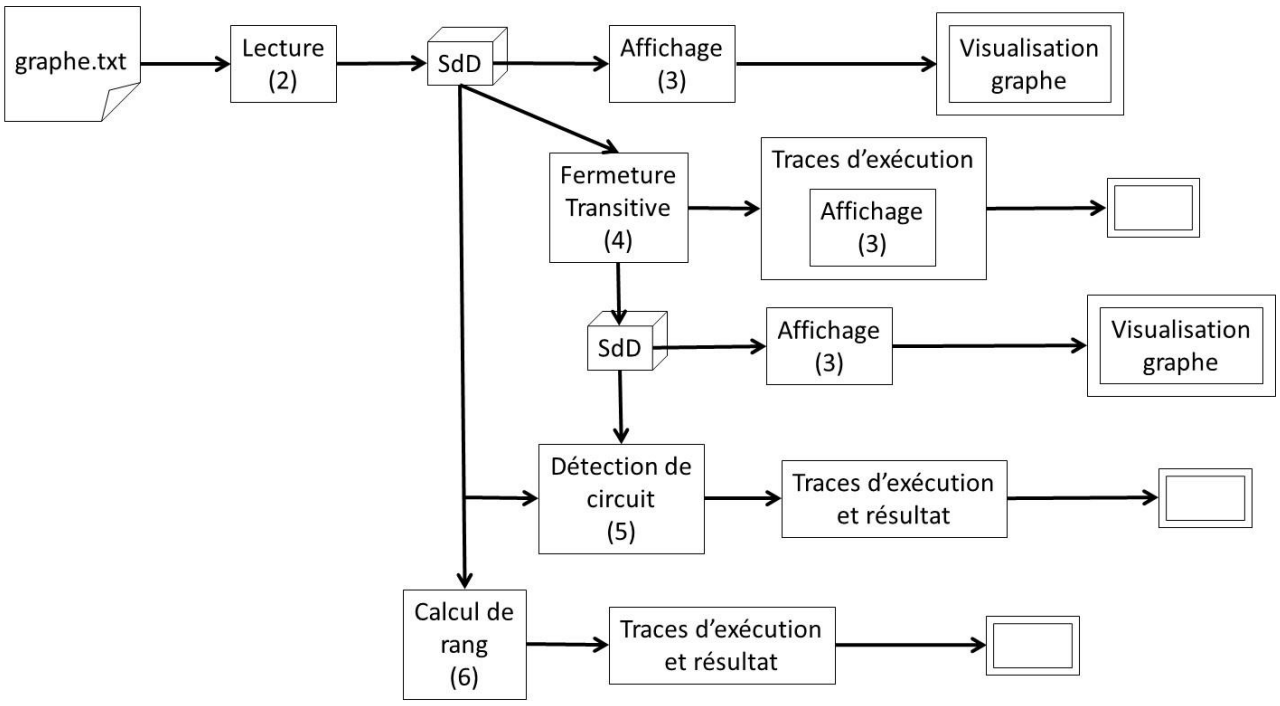
Traces d'exécution :
Votre programme doit indiquer clairement les résultats intermédiaires de votre algorithme, par exemple :

- pour l'élimination des points d'entrée et/ou des points de sortie : indication du ou des sommets supprimés à chaque étape ;
- pour le calcul de la fermeture transitive : valeur (matrice) disponible à chaque itération.

6. Calcul de rang

Si le graphe ne contient pas de circuit, vous devez calculer un rang pour chaque sommet.

En plus des résultats intermédiaires (valeurs de rang données à chaque sommet et ordre d'affectation), votre code doit se terminer par l'affichage synthétique des valeurs pour l'ensemble des sommets.



Résolution de problème d'ordonnancement

A réaliser après avoir mis en œuvre les traitements généraux précédents.

7. Lecture d'un tableau de contraintes

Il est question ici de lire en entrée un fichier contenant des contraintes d'ordonnancement telles que vues en cours ou TD. La structure de fichier définissant le graphe sur lequel vous devrez travailler sera différente de celle définie dans (2) (entre autres, le fait que tous les arcs sortant d'un sommet ont la même valeur (durée de la tâche) sera explicitement pris en compte). Cela veut dire qu'il faut commencer par développer une seconde fonction de lecture, adaptée à un fichier contenant un tableau de contraintes.

Le fichier peut avoir la structure suivante :

`4`	Nombre de tâches
`1 10 2 -1`	La tâche 1 a pour durée d'exécution '10'. Elle ne peut s'exécuter que lorsque la tâche 2 est terminée
`2 20 3 -1`	Tâche 2 de durée '20 ; la tâche 3 doit être terminée
`3 30 -1`	Tâche 3 de durée '30 ' ; aucune contrainte
`4 40 1 2 3 -1`	Tâche 4 de durée '40' ; contraintes 1, 2 et 3
`-1`	

Les tâches sont numérotées à partir de 1.
Lors de la lecture des contraintes, vous devez construire le graphe d'ordonnancement correspondant. Vous noterez dans l'exemple que les tâches sont numérotées à partir de 1, sans rupture de séquence. Le sommet « début des travaux » peut donc être le sommet '0', et dans l'exemple ci-dessus la « fin des travaux » sera représentée par le sommet '5'. Ceci est en ligne avec les graphes que vous devez prendre en compte (voir point 1).
Au fur et à mesure de la lecture du fichier de contraintes, vous les afficherez à l'écran (pour vérifier que les bonnes informations sont lues.
En fin de lecture, vous devez afficher le graphe construit en reprenant votre code du point 3).

Remarque : Après cette étape, le fichier d'entrée n'est plus utilisé. Les traitements suivants sont exécutés exclusivement à partir de vos SdD.

8. Calcul du calendrier au plus tôt et du calendrier au plus tard

Sur la base du graphe construit en 7, vous devez calculer le calendrier au plus tôt et le calendrier au plus tard.

9. Validation du graphe

Imaginez maintenant lire directement un graphe (points 1 et 2) supposé être un graphe d'ordonnancement. Avec les contraintes de type « X ne peut commencer que lorsque Y est terminé », le graphe doit avoir les propriétés suivantes :

a) 1 seul point d'entrée ;
b) 1 seul point de sortie ;
c) Pas de circuit ;
d) Il existe un chemin du point d'entrée à tout autre sommet ;
e) Il existe un chemin de n'importe quel sommet au point de sortie.

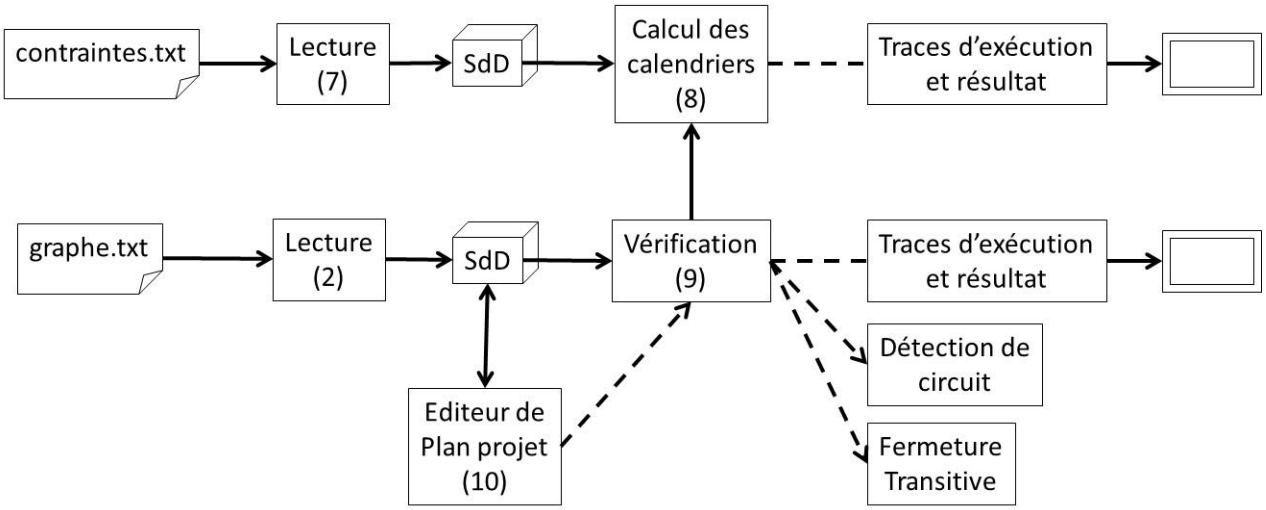
Il vous est demandé de vérifier ces propriétés. Les points a) et b) sont à vérifier depuis votre structure de données « graphe » ; le point c) utilise ce que vous avez fait en 5 ; les points d) et e) peuvent utiliser le travail fait en 4.

10. Editeur de plan projet

Dernière étape de votre projet, construire un « éditeur de plan projet ».
L'idée est d'offrir une interface à l'utilisateur lui permettant de modifier le plan projet chargé à l'étape 7).
Pour cela vous lui offrez, au travers d'un menu dans la console, les fonctions suivantes :

- ajout / suppression de tâche ;
- ajout / suppression de contrainte ;
- modification de la durée d'exécution.

A chaque commande de l'utilisateur, le graphe est modifié puis vérifié (point 9). Une commande non acceptée par le point 9 est refusée. Si tout se passe bien, chaque commande provoque le re-calcul des calendriers.
L'utilisateur pourra utiliser cet éditeur en cours de construction de graphe. Une autre hypothèse est que, une date « au plus tard » de fin de projet ayant été fixée (date de livraison du projet à un client par exemple), toute modification entraine une vérification de faisabilité (la date au plus tôt de fin de projet reste inférieure ou égale à la date de livraison fixée).



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.

Vous devez rendre votre travail à la date qui sera fixée par votre enseignant.
Le rendu s'effectue par email :

- adresse : sera fournie par l'enseignant (l'utilisation d'un autre adresse sera pénalisée),
- titre (sujet) : « [L3/L'3] – TG – *nom1 / nom2 / nom3* » (toute déviation sera pénalisées)
- corps (texte) du message : indiquez ce qui est fait ou non (reprenez les différents points ci-dessus et indiquez : ok, ok mais incomplet, bug, pas fait)
- 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.

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 »). **Cela concerne le fichier contenant la fonction main également**. L'expérience montre que si vous imposez ces consignes à la dernière minute,

elles ne seront pas vraiment respectées (par exemple, vos fichiers .txt seront nommés correctement, mais votre code demandera des fichiers dont le nom n'est pas conforme).

Pas de répertoire dans la structure de vos fichiers. Votre enseignant doit être capable de mettre toutes les pièces jointes dans un même répertoire, et compiler puis exécuter correctement votre programme.

Le déroulement des soutenances vous sera précisé ultérieurement.

Annexe – Lecture de graphe sur fichier

Le code suivant est une base pour la définition de structures de données permettant de représenter un graphe en mémoire, ainsi que pour la lecture d'un graphe sur fichier et son stockage en mémoire.

```
#include <fstream>

#define FICHIER_GRAPHE "mongraphe.txt"

typedef struct {
	int		nbSommets ;
	bool **		MAdj ; // MAdj[x][y] = TRUE  < == > il existe arc (x,y)
	int **		MVal ; // Si MAdj[x][y] = TRUE alors MVal[x][y] = valeur de l'arc (x,y)
} t_graphe ;

int main () {

	// Déclaration graphe

	t_graphe * G = new t_graphe ;

	// Lecture du graphe sur fichier

	ifstream fg ( FICHIER_GRAPHE ) ;

	// Lecture du nombre de sommets et allocation dynamique des SdD

	fg >> G->nbSommets ;
	G->MAdj = new bool * [ G->nbSommets ] ;
	for ( int s = 0 ; s < G->nbSommets ; s++ ) {
		G->MAdj[s] = new bool [ G->nbSommets ] ;
		for ( int extTerm = 0 ; extTerm < G->nbSommets ; extTerm++ ) {
			G->MAdj[s][extTerm] = false ;
		};
	};
	G->MVal = new int * [ G->nbSommets ] ;
	for ( int s = 0 ; s < G->nbSommets ; s++ ) {
		G->MVal[s] = new int [ G->nbSommets ] ;
		for ( int extTerm = 0 ; extTerm < G->nbSommets ; extTerm++ ) {
			G->MVal[s][extTerm] = 0 ;
		};
	};

	// Lecture des arcs

	int extInit ;
	int extTerm ;
	int valeur ;
	fg >> extInit ;
	while ( extInit != -1 ) {
		fg >> extTerm ;
		fg >> valeur ;
		G->MAdj[extInit][extTerm] = true ;
		G->MVal[extInit][extTerm] = valeur ;
		fg >> extInit ;
	} ;

	// Impression graphe
	for ( int x = 0 ; x < G->nbSommets ; x++ ) {
		cout << x << "\t" ;
		for ( int y = 0 ; y < G->nbSommets ; y ++ ) {
```

```
                        if ( G->MAdj[x][y] == true ) {
                                cout << "X\t" ;
                        } else {
                                cout << "\t" ;
                        };
                };
                cout << "\n" ;
        };
        for ( int x = 0 ; x < G->nbSommets ; x++ ) {
                cout << x << "\t" ;
                for ( int y = 0 ; y < G->nbSommets ; y ++ ) {
                        if ( G->MAdj[x][y] == true ) {
                                cout << G->MVal[x][y] << "\t" ;
                        } else {
                                cout << "\t" ;
                        };
                };
                cout << "\n" ;
        };

        system("PAUSE") ;
        return 1 ;
};
```