Projet tutoré de SD-Graphes

Ordonnancement de projets

1 Contexte - Exemple

Avant de se lancer dans la réalisation d'une enquête sur la consommation de glaces et sorbets, on veut estimer la durée du projet pour savoir si les résultats de l'enquête seront obtenus à temps pour la préparation de la période estivale.

On a mis en évidence 9 tâches à réaliser.

T1 - Réalisation des questionnaires

T2 - Enquête téléphonique avec saisie informatique simultanée

T3 - Enquête porte à porte
T4 - Saisie informatique des enquêtes porte à porte

T5 - Analyse informatique des résultats de l'enquête téléphonique
T6 - Analyse informatique des résultats de l'enquête porte à porte

T7 - Envoi de remerciements aux participants à l'enquête

T8 - Etude des résultats par un premier cabinet de conseil
T9 - Etude des résultats par un deuxième cabinet de conseil

Voici le tableau récapitulatif indiquant pour chaque tâche, ses contraintes de précédence et sa durée :

Tâche	T1	T2	T3	T4	T5	T6	T7	T8	T9
Précédence	-	T1	T1	T1, T3	T2	T3, T4	T4	T5, T6	T6, T8
Durée	5	5	10	8	15	5	20	5	10

De tels problèmes d'ordonnancement sont présents au quotidien. Les informations minimales communes à tous ces problèmes sont l'identification des tâches à réaliser, leurs contraintes de précédence et leur durée (connue avec plus ou moins de certitude d'ailleurs).
Nous nous proposons à travers ce projet de réaliser une application permettant de résoudre des problèmes d'ordonnancement simples.

2 Résoudre un problème d'ordonnancement

Résoudre un tel problème d'ordonnancement consiste à planifier l'exécution des tâches dans le temps. Pour cela différentes informations utiles sont à calculer :

- Date de début au plus tôt de chacune des tâches,
- Durée minimale du projet,
- Date de début au plus tard de chacune des tâches,
- Marge de chacune des tâches et mise en évidence des tâches critiques.

Pour ce faire, différents algorithmes doivent être mis en œuvre et le graphe relatif au problème doit alors préalablement être chargé en machine. Les résultats devront être stockés dans un fichier "results.txt"

3 Description d'un problème par fichier texte

Nous nous proposons d'utiliser un format usuel de fichiers pour décrire un problème d'ordonnancement (format inspiré de DIMACS - Center for Discrete Mathematics and Theoretical Computer Science - voir annexe A). Le format est le suivant :

- Des lignes commençant par c, suivi d'un espace, contenant des commentaires,
- une ligne commençant par p, suivi d'un espace, contenant la description du problème. Ici nous considérerons le nombre de sommets,
- n lignes commençant par v (vertex), suivi d'un espace, contenant les informations sur les noeuds,
- m lignes commençant par a (arc), suivi d'un espace, contenant les informations sur les contraintes de précédences.

Un fichier relatif à l'exemple du sondage pourrait être le suivant :

c	Sondage
p	9
v	1 5
v	2 5
v	3 10
v	4 8
v	5 15
v	6 5
v	7 20
v	8 5
v	9 10
	...

a	1 2
a	1 3
a	1 4
a	2 5
a	3 4
a	3 6
a	4 6
a	4 7
a	5 8
a	6 8
a	6 9
a	8 9

4 Exemples à traiter

Vous devrez tester votre programme sur différents exemples en indiquant, pour chacun dans le rapport final, le résultat et le temps CPU de l'exécution.

Les fichiers descriptifs de ces problèmes sont accessibles en ligne :

- Aller sur la page : `http://www.cristal.univ-lille.fr/~kessaci/`
- Cliquer sur *Teaching*
- Cliquer sur le lien du projet SD-Graphe

Les problèmes à résoudre sont de tailles différentes en nombre de sommets. Seule une implémentation propre et *optimisée* permettra la résolution des problèmes de grande taille.

5 Travail à rendre à votre tuteur

Le **mardi 29 mai avant 14h**, rendre à votre tuteur un rapport (*) d'analyse et de conception, de 5 à 7 pages, contenant :

- Précisions éventuelles du cahier des charges,
- Description de la méthode choisie pour répondre au cahier des charges (choix de l'algorithme et la façon dont il est utilisé),
- Choix et justification de la structure de données utilisée pour représenter un problème d'ordonnancement (comparaison entre 3 structures différentes avec leurs avantages et inconvénients respectifs par rapport à votre problème),
- Principaux algorithmes de résolution (chargement, niveau, tri, pcc) appliqués à votre structure de données.

A la fin du projet (**lundi 11 juin 12h** au plus tard), rendre à votre tuteur un rapport final (*) contenant :

- Partie analyse et conception du premier rapport avec des compléments éventuels au rapport précédent,
- Mode d'emploi (commandes de compilation et de lancement, description des ficihiers d'entrée et sortie),
- Description des exemples traités et résultats obtenus,
- Conclusion (point sur ce qui a été fait / non fait), améliorations possibles,
- Bilan personnel sur le projet.

ainsi que les sources (uniquement les .c et .h) et fichiers exemples (personnels) dans une archive en tar exclusivement.

(*) voir avec votre tuteur s'il souhaite une version imprimée des deux rapports.

Vous utiliserez le serveur GitLab de Polytech Lille pour le rendu des rapports et codes :

`https://archives.plil.fr/`

Puis :

- Créer un nouveau projet (privé) sous le nom SD-GC_nombinome1_nombinome2
- Ajouter votre binôme aux membres de votre projet avec le niveau de droit **Master**
- Ajouter les intervenants de projet aux membres de votre projet avec le niveau de droit **Reporter** : mkessaci, fseynhae, bcarre, sjanot

A Détails sur le format de fichier pour un problème d'ordonnancement simple - PERT

[Extrait de l'explication sur le site de DIMACS...]

Files are assumed to be well-formed and internally consistent : node identifier values are valid, nodes are defined uniquely...

— **Comments.** Comment lines give human-readable information about the file and are ignored by programs. Comment lines can appear anywhere in the file. Each comment line begins with a lower-case character **c**.
`c This is an example of a comment line.`

— **Problem line.** There is one problem line per input file. The problem line must appear before any node or arc descriptor lines. For PERT instances, the problem line has the following format.
`p NODES`
The lower-case character `p` signifies that this is the problem line. The `NODES` field contains an integer value specifying n, the number of nodes in the network.

— **Node Descriptors.** All node descriptor lines must appear before all arc descriptor lines. The required format for node descriptor lines depends upon the intended problem type. For PERT problems instances, the node descriptor lines describe the processing time of the task. There is one node descriptor line for each node, with the following format.
`v ID TIME`
The lower-case character `v` signifies that this is a node descriptor line. The `ID` field gives a node identification number, an integer between 1 and n. The `TIME` field gives the processing time $p(v)$ of the node `v`.

— **Arc Descriptors.** There is one arc descriptor line for each arc in the network. The required format for arc descriptor lines depends upon the intended problem type. In a PERT problem, precedences are indicated.
For a PERT instance, arc descriptor lines are of the following form.
`a SRC DEST`
The lower-case character `a` signifies that this is an arc descriptor line. The `SRC` (source) and the `DST` (destination) field gives the identification number for the source vertex v, and the destination vertex w, indicating that w has for predecessor v. Identification numbers are integers between 1 and n.

B Proposition pour implémenter le tri des sommets par niveaux

Nous proposons de faire ce tri en deux phases :
- Recherche du niveau d'un sommet
- Tri selon les niveaux

Clairement, cette proposition est très coûteuse, et vous pouvez réfléchir à une implémentation plus optimale ! (notamment pour résoudre les problèmes de grande taille)

Procédure niveau

```
   Données : G : Graphe (n : nombre de sommets)
   Résultat : N[] : Tableau contenant le niveau des sommets
   Locales : change : Booléen indiquant s'il y a eu un changement
             i, j : Variables de boucle
Début
   Pour i de 1 à n faire
      N[i] ← 0
   change ← 1
   Tant que change
      change ← 0
      Pour i de 1 à n faire
         Pour j parcourant les prédécesseurs de i faire
            Si N[i] < N[j] + 1 Alors
               N[i] ← N[j] + 1
               change ← 1
            FSi
         FPour
      FPour
   FTantque
Fin
```

Procédure tri

```
   Données : N[] : Niveau des sommets, n : nombre de sommets
   Résultat : S[] : Tableau contenant les sommets triés par niveau
   Locales : p : Position courante dans le tableau S
             niv : Indique le niveau en cours d'examen
             i : Variable de boucle
Début
   p ← 0
   niv ← 0
   Tant que p < n faire
      Pour i de 1 à n faire
         Si N[i] = niv Alors
            S[p] ← i
            p ← p+1
         FSi
      FPour
      niv ← niv + 1
   FTant que
Fin
```