CNAM

UTC 501

PROJET

SOLVEUR DE GRAPHE MPM

(METHODE DES POTENTIELS METRA)

SEBASTIEN PINET

Cnam UTC 501 – projet 2019 - 2020 semestre 2

Code : licence MIT

Document : GNU Free Documentation License

Index

Cnam UTC 501 – projet 2019 - 2020 semestre 2

Présentation

Ce projet a pour but de résoudre et afficher des graphes MPM, utilisés pour la gestion des tâches.

Définition :

La méthode MPM consiste en un graphe orienté, dont les sommets représentent les tâches et les arcs représentent les contraintes d'antériorité sont orientés vers la ou les tâches postérieures.
Les sommets sont représentés en général par des rectangles comprenant les dates de début et de fin au plus tôt et au plus tard ainsi que la référence et la durée de la tâche.

Objectifs du projet :

Un utilisateur doit pouvoir insérer dans un formulaire les tâches à effectuer, leurs durées et leurs antécédents.
Le résultat final doit parvenir à contrôler les erreurs que l'utilisateur aurait pu inscrire et afficher le graphe de manière claire, en indiquant les temps minimum et maximum pour chaque tâche ainsi que le chemin critique.

Ce projet se veut simple et ne couvre pas l'entièreté de la complexité de résolution des graphes MPM. Il n'a pas pour vocation à créer un logiciel professionnel mais seulement de manipuler un type de graphe en cherchant un algorithme de résolution.

Pour la résolution des algorithmes, je n'ai pas cherché de solution auprès d'une tierce partie (ni consultation d'internet ni interrogation de personnes compétentes). L'idée était que moi-même trouve une solution. Plusieurs solutions sont certainement applicables, j'ai essayé tant que possible d'en proposer une simple. Je vais maintenant présenter comment j'ai perçu le problème et ma démarche pour arriver au but recherché.

Cnam UTC 501 – projet 2019 - 2020 semestre 2

Plan

Pour réaliser ce projet j’ai dû répondre à trois questions :

- Récupérer les données de l’utilisateur et insertion dans une structure de données
- Chercher un algorithme de résolution, modéliser un graphe dans une structure de données
- Dessin du graphe à l’écran

Ainsi je vais tenter d’expliquer ma démarche en présentant une partie relative à la récupération des données utilisateur, une partie concernant la transformation des données en graphe pour arriver à la dernière partie concernant l’affichage du graphe.

Cnam UTC 501 – projet 2019 - 2020 semestre 2

Partie 1 : Récupération des données

Afin de réaliser ce projet je devais construire un formulaire, mon choix a été de se tourner vers le langage Javascript, pour plusieurs raisons. L'exécution du code se faisant dans un navigateur, il n'y a rien à installer pour l'utilisateur, à part avoir un navigateur assez récent (Chrome, Firefox, Safari, Opera, IE etc...) Il est facile de construire des formulaires en HTML et de manipuler l'affichage de la page grâce à Javascript. Pour finir, la totalité du code se trouve du côté client, il n'y a pas besoin d'appel à un serveur, une simple exécution d'un fichier HTML permettra de gérer les affichages et résoudre la partie algorithmie.

a. contrôle des erreurs

Le contrôle des erreurs utilisateur n'est pas extrêmement poussé, cependant j'ai mis en place quelques mécanismes qui peuvent apparaître fréquemment lors du remplissage des entrées.

Lorsque que l'on tentera d'afficher le graphe, l'algorithme vérifiera si les cases qui doivent être remplies le sont. C'est le cas pour le nom des tâches et les durées.

De plus les durées doivent être strictement supérieures à zéro. Ce projet se veut simple et ne gère pas les cas de durées égales à zéro (que l'on peut trouver dans les graphes de type PERT)

L'algorithme détecte si un antécédent n'a pas de tâche et si une tâche est entrée en double.

Lors d'une erreur, le script s'arrêtera, un message d'erreur s'affichera. Une fois les corrections effectuées on pourra redemander l'affichage du graphe sans recharger la page au préalable.

b. Transformation en structure de données

Un premier choix a été de trouver une structure de données appropriée pour transformer les données du formulaire en données exploitables.

J'ai décidé de transformer les données en tableau de sommets, un sommet étant un objet composé d'une lettre, d'une durée et d'un tableau d'antécédents. Chaque antécédent du tableau d'antécédent pointe sur un sommet de la liste. Ainsi des parcours en récursion seront faciles à faire, à l'image des listes chaînées.

Quelques précisions sur le code.

Javascript ne fonctionne pas comme un pur langage objet, ainsi bien qu'ayant créé des objets, ceux là sont très éloignés du modèle de ceux de Java par exemple.

Ces objets ressemblent dans notre cas plus à des struct en C.

Cnam UTC 501 – projet 2019 - 2020 semestre 2

J’ai voulu m’essayer au paradigme fonctionnel mais je tiens à dire de suite que le résultat n’est pas qualifiable de fonctionnel, mon code ne respectant pas ces principes. Cependant pour les parties 1 et 2, le code est composé uniquement de fonctions. Une fois les données du formulaires récupérées celles-ci sont passées de fonction en fonction et transformées jusque au résultat voulu. Il n’y a pas de variable globale, seulement création d’une liste de sommets puis grâce à cette liste, création d’un graphe organisé par colonne. Il y aura des ajouts d’attributs aux objets sommets au fur et à mesure de la résolution du graphe.

A ce stade nous obtenons une liste en Javascript contenant chaque sommet (tâche) :

Exemple de l’objet sommet C de la liste, ayant A et B comme antécédents :

{ lettre : ‘C’,

duree : 8,

pred : [objet (A), objet(B)]

}

Cnam UTC 501 – projet 2019 - 2020 semestre 2

Partie 2 : Construction du graphe

La liste de tâches que nous obtenons n’est pas ordonnée. J’ ai choisi d’ordonner le résultat comme suit :

a. classement en colonnes

Un numéro de colonne contient les tâches de ce niveau, ainsi les tâches sans antécédents se trouvent sur la colonne 1 (la colonne zéro sera réservée pour ajouter la tâche alpha). Les tâches qui suivent les tâches de la colonne 1 viennent en colonne 2 et ainsi de suite pour toutes les tâches. A la fin il suffira d’ajouter une colonne pour la tâche finale Oméga.

Algorithme :

Création d’une structure de données Map() (collection de clé : valeur). Ici les clés correspondent aux numéros de colonne. Les valeurs pour chaque colonne sont un tableau d’objet (objet correspondant à une tâche, ceux récupérés dans la partie 1).

Pour chaque sommet de la liste de sommets :

Le code est simple et se fait par récursion :

- si le sommet n’a pas d’antécédent je retourne 1. (il viendra se placer dans la colonne 1)
- sinon je crée une liste. Pour chaque antécédent j’ajoute à cette liste le numéro de colonne de l’antécédent. J’obtiens une liste qui donne les colonnes où se situent les antécédents.
- Je retourne le max de cette liste + 1. (cette tâche vient en se mettant après la tâche se trouvant dans la colonne la plus élevée).

Il suffit de rajouter l’alpha et l’oméga. Alpha dans la colonne 0 et Oméga dans la colonne (taille de de Map() + 1).

Nous arrivons avec un graphe complet ordonné par colonnes.

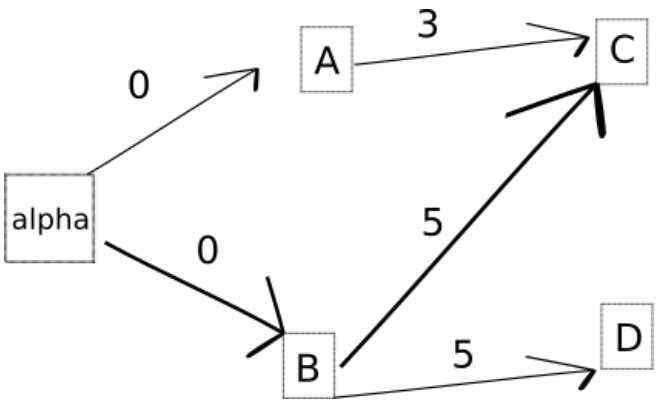
Ex : Map (0 : [Alpha], 1:[A, B], 2:[C,D,E], 3[F], 4 : [Omega])

b. ajout du temps minimum

Définition : Il s’agit de la date à laquelle la tâche peut commencer au plus tôt (tant que ses antécédents ne sont pas finis la tâche ne peut pas démarrer)

L’ajout des temps au minimum est très simple, il reprend l’algorithme de calcul des colonnes sauf qu’au lieu d’ajouter 1, on ajoute la durée de la tâche.

- S’il n’y a pas d’antécédent je retourne 0.
- Sinon je crée une liste, pour chaque antécédent j’ajoute à cette liste la durée de l’antécédent + le résultat de la récursion de l’antécédent. J’obtiens une liste des durées cumulées pour chaque antécédent.
- Je retourne le max de cette liste. (qui correspond au temps minimum auquel la tâche doit débuter).



calcul du min de C par récursion :
c : max de récursion de A et B
récursion de A : recursion de alpha :
alpha : pas d'antécédent , retourne O
A a 0 en min, retourne min + 3

pareil pour B : retourne 5

min de c = max [3,5] = 5

c. Ajout du temps maximum

Définition : Date à laquelle la tâche peut démarrer au plus tard sans rallonger la durée du projet.

Cette fois j’ai dû procéder différemment. Je suis parti de l’idée qu’Omega connaît ses antécédents et donc qu’une itération récursive à partir d’Omega passe par toutes les tâches.

Pour toutes les tâches je leur met comme valeur le temps minimum d’Omega, qui sera aussi son temps max.

Il suffit pour chaque antécédent d’Omega de calculer le temps max de son antécédent (la tâche en cours moins la durée de la tâche de l’antécédent).

En partant d’Omega et au cours de la récursion, si le calcul du temps max pour un antécédent est inférieur à la valeur qu’il contient, ce temps max est remplacé et on continue la récursion. Sinon on arrête la récursion à ce niveau, puisqu’on ne modifie pas la valeur du temps max, c’est donc que c’est déjà la bonne et les antécédents jusqu’à Alpha n’ont pas besoin de voir leur valeur modifiée.

d. chemin critique

On peut se passer de ce calcul et le faire de façon paresseuse, c'est à dire uniquement lors du traçage du graphe pour modifier la couleur de l'arc. Un arc connaît le nœud de départ et le nœud d'arrivée. Si ces deux nœuds sont critiques, c'est à dire que leur temps au minimum est égal à leur temps au maximum, alors l'arc sera dessiné en rouge.

La construction des arcs se fera dans la partie 3, lors du dessin du graphe.

Fin de la partie 2, à ce stade nous obtenons un graphe ordonné dans une structure de données de type clé-valeur.

Chaque objet contient maintenant en plus de ses attributs de base (lettre, duree, [antécédents]) les attributs min et max.

Exemple de l'objet sommet C de la liste, ayant A et B comme antécédents, min et max ont été ajoutés :

{ lettre : 'C',

duree : 8,

min : 6,

max : 8,

pred : [objet (A), objet(B)]

}

Exemple de structure de donnée « graphe », Map() :

Map (0 : [Alpha], 1:[A, B], 2:[C,D,E], 3[F], 4 : [Omega])

Dans cet exemple, la colonne 1 contient les sommets A et B, la colonne 2 les sommets C,D,E etc.

Les antécédents d'un sommet sont dans l'objet sommet, cependant cette structure n'indique pas directement où sont par exemple les antécédents de C, on sait que ce sont A et B, il faudra faire une recherche dans la Map() par colonne :

Exemple avec la recherche de C :

→ Pour chaque colonne :

→ pour chaque sommet de la liste :

→ Si la lettre du sommet est égale à C :

→ on retourne le sommet

Partie 3 : Traçage du graphe

a. Objectifs

L'objectif est de pouvoir tracer un graphe de manière claire, en affichant les informations suivantes :

- Nom de la tâche (ou sa lettre)

- Les temps au plus tôt et au plus tard

- Les arcs et la durée de la tâche

Enfin, l'utilisateur doit pouvoir déplacer une tâche du graphe (un noeud) avec la souris, au cas où le dessin initial trace des arcs en chevauchant des nœuds, ce qui altère la lisibilité. En permettant le déplacement des nœuds, l'utilisateur peut afficher le graphe comme il le souhaite.

Pour tracer le graphe j'ai utilisé la librairie Konvajs de javascript. c'est une librairie qui étend les possibilités du canevas Javascript, en ajoutant notamment l'interaction d'évènements, la manipulation à la souris des formes etc. Pour plus d'informations sur cette librairie :

https://konvajs.org/

Avant d'utiliser konvajs, j'ai obtenu un premier résultat en utilisant simplement une balise html canvas, dans laquelle j'ai pu dessiner le graphe et obtenir un premier résultat. Cependant l'utilisation simple d'une balise canvas limite les possibilités de dessin dans le sens où les formes ne sont pas des objets mis en mémoire que l'on peut modifier simplement, il s'agit plus d'un dessin comme on le ferait avec papier et crayons, c'est à dire que pour modifier une forme il faudrait gommer les parties modifiées et les redessiner. C'est pourquoi l'utilisation d'une librairie facilite amplement le travail si on veut modifier le dessin interactivement.

b. Algorithme

Dans cette partie je suis retourné à une programmation un peu plus conventionnelle, en recréant quelques classes. Ces classes serviront à créer des formes pour le dessin du graphe.

Au niveau des classes, il est à noter que je n'ai pas utilisé l'héritage mais la composition. Ainsi une classe qui hériterait d'une autre ne va pas avec cette méthode étendre une classe mère mais prendre

en attribut une instance de cette « classe mère ». Ainsi on conserve toutes les informations de la classe mère et on évite au compilateur un travail de résolution de liens.

Dans la partie 3, le fichier main.js possède une variable globale « stage », la scène.

Une fonction dessiner() contient tout ce qu'il y a à faire lors de cet appel : Récupération d'un graphe et dessin du graphe sur une couche, qui sera ajoutée à la scène. A chaque appel à dessiner(), la couche de la scène contenant le graphe est effacée, une nouvelle couche est redessinée.

Dessin du graphe :

Le graphe est récupéré sous cette forme :

Map (1 : [tacheA, tacheB, tacheC], 2 :[tacheC, tacheD], 0 : [alpha], 3 : [omega])

1. Positionnement des taches :

Le fait de positionner par colonne une liste de tâche permet de répartir les sommets suivant leur position dans le graphe.

Connaissant le numéro de colonnes des tâches et le nombre de tâches par colonne, on peut positionner les tâches dans le canevas et les espacer à équidistance les uns des autres.

Alpha va se positionner complètement à gauche et au milieu du canevas dans la hauteur

La colonne 1 va se positionner après Alpha, et chaque tâche de cette colonne va se répartir selon la hauteur du canevas.

Omega sera positionné à la fin et au milieu du canevas dans la hauteur

L'algorithme de positionnement est le suivant :

Pour les x :

Division de la longueur du canevas par le nombre de colonnes + 1 donne l'espace entre chaque colonne. Cet espace multiplié par le numéro de colonne donne son emplacement en x.

Pour les y :

Même calcul, cette fois il faut prendre la hauteur du canevas et le nombre de tâches de la colonne en cours. Une itération à partir de 0 et un incrément de 1 pour chaque tâche de la liste place les tâches les unes au dessous des autres.

Exemple du calcul pour le positionnement en X des nœuds, suivant la colonne dans laquelle ils sont :

canvas de 100 par 100 :

placement de 5 sommets à equidistance en X :

100/6 * numéro de colonne + 1 (car les colonnes sont numérotées à partir de 0

2. Positionnement des arcs :

Pour chaque tâche on obtient un objet Noeud avec sa position. Pour chaque Nœud, on va ajouter ses nœuds antécédents. Ainsi pour chaque nœud, si un Nœud a des antécédents, alors pour chaque antécédent il y a création d'un objet Arc() qui prendra dans son constructeur le nœud de départ et le nœud d'arrivée. Nous obtenons les positions des nœuds entre lesquels l'arc doit être dessiné. Le nœud de départ a l'information sur la durée de la tâche, qu'il suffira de dessiner au milieu de l'arc.

Tous les objets Arc sont mémorisés dans une liste d'arcs.

3. Déplacement des nœuds :

Pour chaque nœud on ajoute une écouteur d'évènements à la souris « mousedrag » (déplacement de la souris avec bouton gauche enfoncé).

A cet écouteur on lui associe une fonction. Cette fonction va mettre à jour, pour chaque arc de la liste des arcs, la position de son nœud de départ et de son nœud d'arrivée.

Conséquences : Lorsque l'utilisateur déplace un nœud, tous les arcs du dessin voient leurs positions mises à jour et le graphe est automatiquement redessiné.

Il faut souligner que cet algorithme n'est pas optimal car toutes les positions des arcs sont recalculées, même celles des arcs qui n'ont pas bougé.

Cnam UTC 501 – projet 2019 - 2020 semestre 2

Installation :

Décompresser l’archive et ouvrir le fichier index.html dans un navigateur.

Une accès à internet ou un serveur n’est pas requis (le code Javascript s’exécute du côté client et modifie directement la page).

Instructions :

Les instructions principales sont affichées à l’écran.

Pour préremplir un graphe de 6 tâches, cliquer sur REMPLIR

Pour dessiner le graphe, cliquer sur DESSINER

Pour ajouter une tâche, cliquer sur AJOUTER

Pour remplir une tâche :

- une lettre (ou un mot) doit être inscrit
- une durée doit être renseignée, celle-ci doit être strictement supérieure à 0
- Si pas de prédécesseurs renseignés, la tâche est considérée comme tâche initiale (alpha est son antécédent).
- Sensible à la casse, si pour une tâche la lettre = A, la tâche ayant A comme prédécesseur doit inscrire A (et non pas a).
- Si une tâche a plusieurs prédécesseurs, ceux-ci doivent être séparés par des virgules ET sans espace.
- Le bouton X permet de supprimer la tâche
- Il n’y a pas de limites en nombre de tâches à ajouter, tant qu’il y a de l’espace...

A noter : En cliquant sur DESSINER, le graphe est modélisé, si des erreurs utilisateurs sont détectées, celles-ci sont affichées à l’écran et donnent les instructions pour corriger les informations entrées.

Si le graphe s’affiche, c’est qu’il n’y a pas eu d’erreurs utilisateurs.

Les nœuds (tâches) du dessin sont déplaçables à la souris en cliquant sur la tâche et en la déplaçant.

Le bouton FERMER permet de fermer le dessin, l’utilisateur peut à nouveau modifier des tâches.

Après modifications le dessin peut être redessiné sans recharger la page.

Conclusion :

Ce projet a pu me permettre de m'initier à la programmation de graphes, bien que les solutions trouvées soient des solutions personnelles. Elles m'ont permis de réfléchir à une implémentation tout en réfléchissant à d'autres solutions, plus ou moins pertinentes.

Les objectifs du projet sont atteints :

- Contrôle des erreurs utilisateur
- Modélisation d'un graphe
- Dessin du graphe avec nœuds déplaçables.

Sur le code, l'essai à la programmation fonctionnelle est un échec, tant au niveau de son implémentation que de son découpage. Une programmation orientée objet structurée est peut-être plus claire bien que comportant plus de lignes. Bien que la qualité du code ne me satisfasse pas, le programme fonctionne.

Enfin c'était une occasion de manipuler une librairie graphique pour le web et de tester différentes structures de données en javascript. Ce langage peut réserver quelques (mauvaises) surprises. Il est très simple pour s'initier à la programmation web, mais ne bénéficie pas (encore) d'un comportement robuste comme on peut le trouver en Java ou en Python par exemple.

Remerciements :

Je tenais à remercier Monsieur Bernard Couapel pour avoir suggéré la réalisation d'un projet, sans qui l'idée ne me serait jamais venue de m'essayer à la programmation de graphes.