Projet Modélisation

Par Dylan Koby & Quentin Vecchio

Table des matières

Analyse du problème

L'algorithme dynamique à correction d'étiquettes est un algorithme bien connu en théorie des graphes, il est très ressemblant d'autres algorithmes de calcul de plus court chemin tel que Dijsktra car il s'agit de mettre à jour une ou plusieurs étiquettes d'un sommet selon son ou ses prédécesseurs. Cependant, une nouvelle notion entre en scène ici, celle de fenêtre de temps, car la structure d'une étiquette est définie comme suit :

Etiquette [Prédécesseur, Coût, Temps]

Il faut bien sûr chercher à minimiser le coût du chemin tout en respectant cette fenêtre de temps.

Comme nous venons de l'expliquer, il peut y avoir quelques conflits puisque un sommet peut avoir plusieurs prédécesseurs, c'est pour cela qu'une relation de dominance doit être vérifiée afin d'attribuer la bonne ou les bonnes étiquettes.

Algorithme de recherche du plus court chemin

Nous utilisons, comme demandé dans le cahier des charges, l'algorithme ci-dessous :

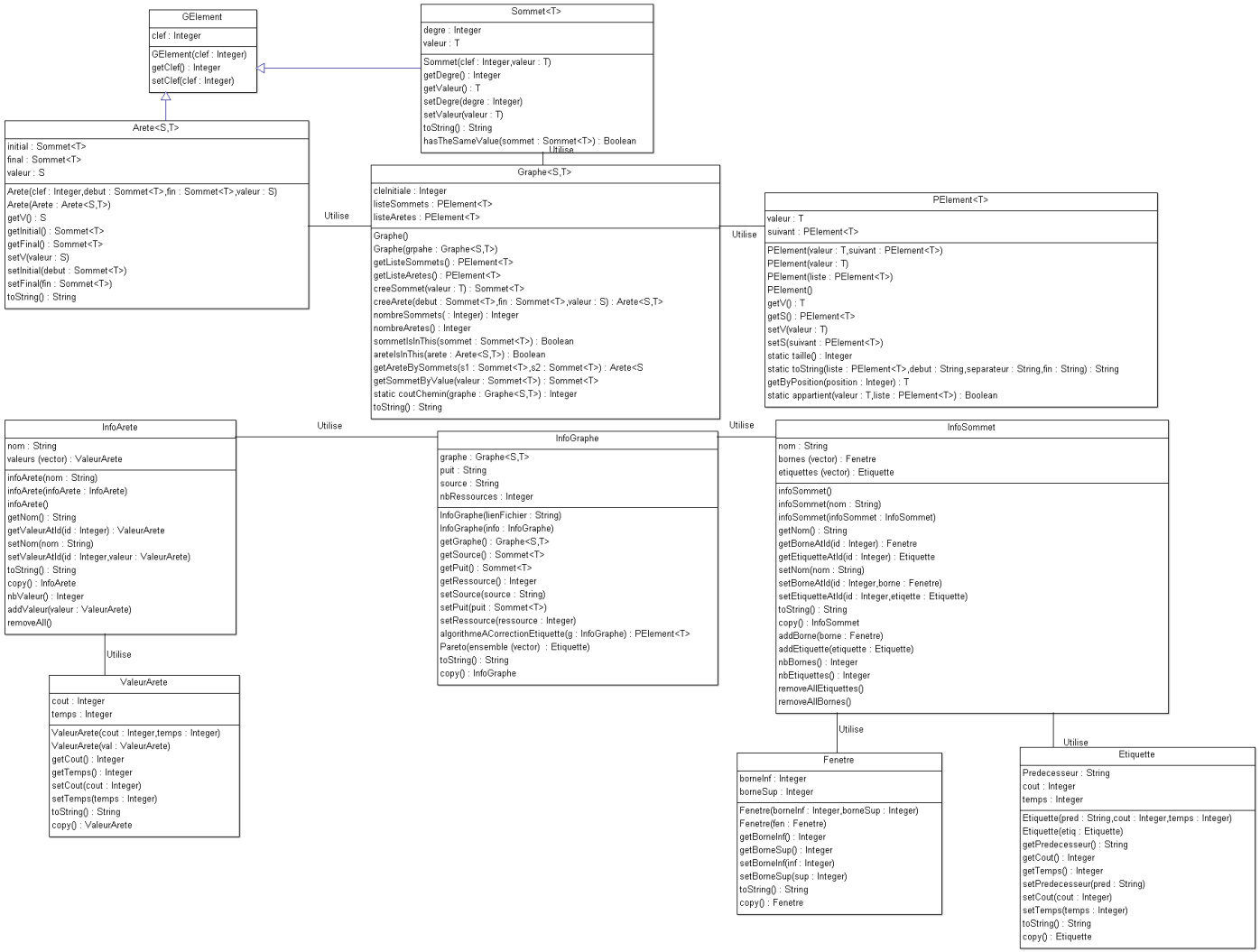
```
1  pour tous les x_i ∈ X faire ETIQ(i) ← ∅;
2  LIST ← {x_o};
3  ETIQ(o) ← {0}
4  tant que LIST ≠ ∅ faire
5  |  Choisir x_i ∈ LIST; LIST ← LIST − {x_i};
6  |  pour tous les x_j ∈ successeurs(x_i) faire
7  |  |  pour tous les E ∈ ETIQ(i) faire
8  |  |  |  si E_1 + w_ij ≤ b_j alors
9  |  |  |  |  E' ← (E_0 + v_ij, E_1 + w_ij);
10 |  |  |  |  ETIQ(j) ← Pareto(ETIQ(j) ∪ {E'});
11 |  |  |  |  si E' ∈ ETIQ(j) alors LIST ← LIST ∪ {j};
12 |  |  |  fin
13 |  |  fin
14 |  fin
15 |  Retourner le chemin ayant le plus petit coût en x_d;
16 fin
```

Nous avons légèrement modifié sa structure pour que l'algorithme fonctionne avec nos différentes structures de données. Nous avons également ajouté certaines contraintes afin de gérer la borne inférieure de la fenêtre de temps d'un sommet. L'algorithme prend en entrée une structure de données de type InfoGraphe. Elle retourne une structure de données de type vector et qui contient des listes de sommets. Ces listes correspondent aux différents chemins que l'on peut prendre en partant de la source pour aller jusqu'au puit.

Plusieurs chemins peuvent exister. S'il tel est le cas, c'est que l'algorithme n'a pas su faire le choix, et qu'il faut faire un compromis entre le temps et le coût.

L'algorithme peut modifier à la demande la gestion de la liste de visite des sommets. En effet, on peut choisir une fonction qui dépilera la liste de visite. On pourra alors dépiler la liste comme si c'était une liste FIFO, on encore en choisissant l'ordre alphabétique des sommets, dans l'ordre croissant des clefs de création (voir parties des variantes).

Diagramme des classes UML



Choix de programmation

Nous avions réalisés plutôt dans ce semestre des TPs ainsi qu'un projet manipulant des graphes, c'est pour cela que nous avons décidés de reprendre nos structures de données pour réaliser ce projet. Elle est très simple d'utilisation et très peu gourmande en ressources puisque nous utilisons nos propres structures telles que des listes.

Nous avons également mis en place la notion de template (généricité) dans ce projet car nous utilisons nos propres structures de donnés concernant un graphe, ses sommets et ses arêtes.

Un graphe est donc défini par une source et un puit qui sont des chaînes de caractères que nous transformerons bien sûr en sommets, et un nombre de ressources.

Un sommet est défini par un nom sous une chaîne de caractères, un objet vector de bornes représentant la fenêtre de temps à respecter et un autre vector d'étiquettes.

Une arête est représentée par un sommet initial et un terminal ainsi que par une information « v » c'est-à-dire le nom de l'arête et la structure [coût, temps] correspondante.

Une étiquette est définie par une chaîne de caractère représentant le prédécesseur (convertie ensuite en un sommet), et de deux entiers représentant le coût et la durée.

Une fenêtre est bien sur définie par deux entiers représentant la borne inférieure et la borne supérieure constituant la fenêtre de temps.

La classe GElement est la classe de base de la classe Arete et de la classe Sommet, elle ne contient qu'une clef (un index) et des méthodes virtuelles implémentées dans ses classes enfants.

La classe PElement constitue la structure de liste chaînée nous permettant de manipuler notre graphe, ses sommets et ses arêtes.

Enfin, la classe TestUnitaire nous permet, comme son nom l'indique, d'effectuer de multiples tests concernant ce projet, en créant tous les éléments détaillés jusqu'ici.

Création d'un graphe

La création d'un graphe peut se faire manuellement. Pour ce projet, il nous était demandé de générer un graphe à partir d'un fichier texte. Ce fichier texte doit suivre un nombre de règle syntaxique et de forme afin d'être manipulable par le programme. Vous trouverez dans l'archive des fichiers de test. Ces fichiers sont les fichiers test qui nous ont été donné. Cependant nous les avons remodifiées en nous inspirant de la structure ci-dessous. Cette structure nous a été donnée dans le sujet du projet :

#Instance graphe0 à 4 sommets et 6 arcs	commentaire
ressource 1	nombre de ressources avec fenêtre
sectionSommets	
s0 0 0	sommet ; bornes inférieure et supérieure de la fenêtre
i1 0 5	
i2 0 7	
i3 0 8	
p0 0 10	
source	
s0	nom-sommet-source
puits	
p0	nom-sommet-puits
sectionArcs	
arc01 s0 i1 4 3	nom-arc ; sommet-initial ; sommet-terminal ; coût ; temps
arc02 s0 i2 8 2	
arc03 i1 i2 4 3	
arc04 i1 i3 2 6	
arc05 i2 i3 3 4	
arc06 i2 p0 2 6	
arc07 i3 p0 3 4	
sectionGraphe	
graphe1 s1 p1	nom-graphe ; sommet-source ; sommet-puits

Variantes mises en place

Comme présentées dans l'énoncé, plusieurs variantes d'algorithme sont possibles, or nous en avons implémentées deux, l'une consiste à choisir les sommets à visiter selon leur clef (leur numéro) et l'autre à choisir les sommets selon l'ordre lexicographique portant sur le nom des sommets (ex : « s0 » < « s1 », « i1 » < « p0 »).

La mise en place de ces variantes a été effectuée dans le fichier TestUnitaire.h, en tête de ce header, il est présent notre méthode basique pour choisir les sommets « depilageFifo », et nos deux variantes « depilageClef » et « depilageNom ».

Pour utiliser ces variantes, il vous suffit simplement de passer la méthode désirée en paramètres de notre algorithme à corrections d'étiquettes appelé dans la méthode « TestUnitaireInfoGraphe ».

Voici un exemple pour utiliser le choix du sommet selon la clef :

```
vector<PElement<Sommet<InfoSommet> >*>* chemins =
InfoGraphe::algorithmeACorrectionEtiquette(g, depilageClef);
```

Première variante

Cette variante porte sur la clef des sommets, au lieu d'obtenir simplement les voisins dans un ordre aléatoire, nous les obtenons selon l'ordre croissant de leurs clefs.

Pour cela, nous effectuons un dépilage des sommets de la manière suivante : nous récupérons le premier sommet que nous plaçons en tant que minimum (peut-être temporairement), puis à chaque sommet susceptible d'être choisi, nous regardons si sa clef est inférieure au premier sommet choisi, si tel est le cas, ce sommet devient le minimum, et ainsi de suite pour les autres sommets susceptibles d'être choisis. Il ne s'agit en effet que d'un simple tri par ordre croissant via une variable de minima.

Seconde variante

Cette seconde variante porte sur le nom des sommets, pour cela, le principe de la première est réutilisé excepté la comparaison qui ne vérifie plus la clef des sommets mais leurs noms.

Aucun problème pour comparer deux chaînes de caractères étant donné que l'opérateur de comparaison « < » est correctement surchargé.