POLYTECHNIQUE DE MONTRÉAL

LOG2810
STRUCTURE DISCRÈTES

TP 1 : Graphes

Mejdi Ghannem (1679027)
Gabriel-Andrew Pollo-Guilbert (1837776)

Remis à
Juliette TIBAYRENC

5 mars 2018

Table des matières

1 Introduction

Dans ce laboratoire, nous allons aider Joe, un étudiant qui se prépare a réaliser une série de braquages à travers le Canada afin de payer ses études. Pour ce faire, nous devons concevoir un logiciel lui permettant de déterminer le chemin le plus sécuritaire entre deux villes afin d'échapper à la police. Ce chemin doit être optimal afin de minimiser le temps passé sur la route afin d'échapper plus facilement a la police, tout en respectant certaines contraintes.

Le laboratoire a donc pour objectifs de nous familiariser avec les notions théoriques sur les graphes vus dans le cadre du cours LOG2810, sachant que dans notre logiciel, le Canada est représenté par un graphes avec des sommets (villes) et des arcs (distance entre chaque ville).

2 Solution

Notation : Soit un n-uplet $\mathcal{T} = (\mathcal{X}_1, \mathcal{X}_2, \ldots, \mathcal{X}_n)$. On peut dénoter l'object $\mathcal{X}_i$ comme $\mathcal{T}_{\mathcal{X}_i}$.

Afin de déterminer le meilleur chemin à utiliser, on utilise une version modifiée de l'algorithme de Dijkstra. Soit un graphe $\mathcal{G} = (\mathcal{S}, \mathcal{E})$ un 2-uplet contenant un semble $\mathcal{S}$ de sommets et un ensemble $\mathcal{E}$ d'arcs. L'algorithme cherche le chemin le plus court allant d'un sommet $\mathcal{S}_{\text{début}}$ à un sommet $\mathcal{S}_{\text{fin}}$, où $\mathcal{S}_{\text{début}}, \mathcal{S}_{\text{fin}} \in \mathcal{G}_{\mathcal{S}}$.

Algorithme 1 Procédure pour avancer un chemin

```
1:  procedure AVANCER(C, S)
2:      D ← OBTENIRDISTANCE(C, S)
3:      E ← OBTENIRESSENCE(C, S)
4:      if E < 12 then
5:          if MARQUESUPER(C_T) then return ∅
6:
7:          T ← OBTENIRMARQUESUPER(C_T)
8:          C ← RECALCULERCHEMIN(C, T)
9:          if C = ∅ then return ∅
10:
11:         E ← GETESSENCE(S)
12:         if E < 12 then return ∅
13:     end if
14:
15:     if STATIONSERVICE(S) then
16:         E ← 100
17:         D ← D + 15
18:     end if
19:
20:     return (S, C_T, C, E, D)
21: end procedure
```

On définit un chemin $\mathcal{C}$ se rendant au sommet $\mathcal{S}$ comme étant le 5-uplet $(\mathcal{S}, \mathcal{P}, \mathcal{T}, \mathcal{D}, \mathcal{E})$, où $\mathcal{P}$ est le chemin parent, $\mathcal{T}$ est le type de transport utilisé, $\mathcal{D}$ la distance parcourue et $\mathcal{E}$

l'essence restant. En effet, chaque chemin est en fait une liste liée de sorte que l'algorithme génère un graphe orienté acyclique avec racine le chemin vers $\mathcal{S}_{\text{début}}$ et comme feuilles les chemins vers les autres sommets possibles. On définit $\mathbb{C}$ comme l'ensemble des chemins $\mathcal{C}_{\mathcal{S}}$ allant de $\mathcal{S}_{\text{début}}$ jusqu'à un sommet $\mathcal{S}$.

Soit un chemin $\mathcal{C}$, la procédure décrite à l'algorithme 1 permet d'avancer ce chemin à un sommet $\mathcal{S} \in \mathcal{G}_{\mathcal{S}}$ si possible. On assume que $\mathcal{S}$ est adjacent à $\mathcal{C}_{\mathcal{S}}$.

Premièrement, on obtient la distance et l'essence dans le cas où la voiture emprunterait le chemin vers $\mathcal{S}$. On s'assure ensuite que son carburant reste au-dessus de 12%. Dans le cas échéant, on recalcule le chemin en utilisant une voiture de marque super si possible. Si possible, on remplit l'essence de la voiture, ce qui ajoute 15 minutes au trajet. Finalement, on retourne le nouveau chemin.

Afin d'optimiser les coûts de location du transport, il est toujours préférable de choisir la version *cheap* du véhicule. Par conséquent, lorsqu'on tente de déterminer le chemin le plus sécuritaire, il faut donner priorité à la marque la moins chère.

Algorithme 2 Procédure pour déterminer le chemin le plus sécuritaire entre deux chemins

```
1: procedure MINIMISER(C_1, C_2)
2:     if C_1 = ∅ then return C_2 else return C_1
3:
4:     if OBTENIRMARQUE(C_{1,T}) = OBTENIRMARQUE(C_{2,T}) then
5:         if C_{1,D} > C_{2,D} then return C_2 else return C_1
6:     end if
7:
8:     if MARQUECHEAP(C_{1,T}) then return C_1 else return C_2
9: end procedure
```

La procédure décrite en 2 montre comment le choix entre deux chemins $\mathcal{C}_1$ et $\mathcal{C}_2$ s'effectue. Si la marque utilisée est la même entre les deux chemins, alors on minimise en fonction de la distance parcourue. Sinon, on tente toujours de prendre la marque la moins coûteuse. Cette procédure assume qu'au plus $\mathcal{C}_1$ ou $\mathcal{C}_2$ est nul.

Il est important de noter que l'implémentation en Python fait appel à la surcharge de l'opérateur *plus grand que* ($>$), c'est-à-dire `__gt__(self, other)`, de la classe `Chemin`.

Algorithme 3 Procédure pour actualiser un chemin vers un sommet

```
1: procedure ACTUALISER(C, S)
2:     if C = ∅ then return Faux
3:
4:     if C_S = ∅ or C_S > C then
5:         C_S ← C
6:         return Vrai
7:     end if
8:
9:     return Faux
10: end procedure
```

Puisqu'on conserve en tout temps l'ensemble $\mathbb{C}$ de chemins les plus efficaces actuels,

il faut une procédure pour actualiser cette liste dans le cas où un chemin plus efficace est trouvé. La procédure décrite en 3 effectue ce travail. Si le nouveau chemin $\mathcal{C}$ allant vers le sommet $\mathcal{S}$ est sécuritaire que l'ancien $\mathcal{C}_{\mathcal{S}} \in \mathbb{C}$, alors il faut actualiser l'ensemble $\mathbb{C}$. La comparaison entre $\mathcal{C}_{\mathcal{S}}$ et $\mathcal{C}$ est effectuée de la même manière que la procédure 2. De plus, la procédure retourne une valeur booléenne décrivant si le chemin fut bel et bien actualisé.

Algorithme 4 Procédure d'initialisation de l'algorithme principal

1: **procedure** INITIALISATION($\mathcal{S}_{\text{début}}, \mathcal{S}_{\text{fin}}, \mathcal{T}, \mathcal{G}$)
2: $\quad\mathbb{C} \leftarrow \{\emptyset \mid \forall \mathcal{C}_{\mathcal{S}}, \mathcal{S} \in \mathcal{G}_{\mathcal{S}}\}$
3: $\quad\mathcal{C}_{\mathcal{S}_{\text{début}}} \leftarrow (\mathcal{S}_{\text{début}}, \emptyset, \mathcal{T}, 0, 100)$
4: $\quad\mathcal{O} \leftarrow \mathcal{C}_{\mathcal{S}_{\text{début}}}$
5: $\quad\mathcal{V} \leftarrow \{\mathcal{S}_{\text{début}}\}$
6: **end procedure**

Avec les dernières procédures décrites, il est maintenant possible de montrer l'algorithme 5 de Dijstrak utilisée pour trouver le chemin le plus sécuritaire. On commence par l'initialisation de l'algorithme comme montré par la procédure 4. Celle-ci se charge d'initialiser $\mathbb{C}$, le chemin optimal $\mathcal{O}$ à $\mathcal{S}_{\text{début}}$ et l'ensemble $\mathcal{V}$ des sommets visités par l'algorithme.

Algorithme 5 Chemin le plus sécuritaire pour voler une banque

1: **function** DIJKSTRA($\mathcal{G}, \mathcal{S}_{\text{début}}, \mathcal{S}_{\text{fin}}, \mathcal{T}$)
2: $\quad$INITIALISATION($\mathcal{S}_{\text{début}}, \mathcal{S}_{\text{fin}}, \mathcal{T}, \mathcal{G}$)
3:
4: $\quad$**while** $\mathcal{O}_{\mathcal{S}} \neq \mathcal{S}_{\text{fin}}$ **do**
5: $\quad\quad\mathcal{M} \leftarrow \emptyset$
6: $\quad\quad\mathcal{V} \leftarrow \mathcal{V} \cup \{\mathcal{O}_{\mathcal{S}}\}$
7:
8: $\quad\quad$**for all** $\mathcal{S} \in$ GETPOSSIBLES($\mathcal{O}$) **do**
9: $\quad\quad\quad\mathcal{N} \leftarrow$ AVANCER($\mathcal{O}, \mathcal{S}$)
10: $\quad\quad\quad$**if** ACTUALISERCHEMIN($\mathcal{N}, \mathcal{S}$) **then** $\mathcal{M} \leftarrow$ MINIMISER($\mathcal{M}, \mathcal{N}$)
11: $\quad\quad$**end for**
12:
13: $\quad\quad$**for all** $\mathcal{S} \in \mathcal{G}_{\mathcal{S}} \setminus \mathcal{V}$ **do**
14: $\quad\quad\quad$**if** $\mathcal{C}_{\mathcal{S}} \neq \emptyset$ **then** $\mathcal{M} \leftarrow$ MINIMISER($\mathcal{M}, \mathcal{C}_{\mathcal{S}}$)
15: $\quad\quad$**end for**
16:
17: $\quad\quad$**if** $\mathcal{M} \neq$ **then** $\mathcal{O} \leftarrow \mathcal{M}$ **else return** $\emptyset$
18: $\quad$**end while**
19:
20: $\quad$**return** $\mathcal{O}$
21: **end function**

Ensuite, l'algorithme boucle tant et aussi longtemps que le chemin optimal $\mathcal{O}$ n'est pas arrivé au sommet final $\mathcal{S}_{\text{fin}}$. Dans cette boucle, on *visite* le sommet actuel $\mathcal{A}$ au bout du chemin $\mathcal{O}$. Plus précisément, on détermine les chemins possibles que l'on peut emprunter sans revenir sur un sommet déjà visité par le chemin. Les chemins possibles sont ceux allant aux

sommets adjacents de $\mathcal{A}$ et qui ne sont pas encore dans le chemin.

Pour chacun de ces sommets $\mathcal{S}$, on calcule le nouveau chemin $\mathcal{N}$ y allant. On actualise le chemin $\mathcal{C}_\mathcal{S}$ si possible et on détemine s'il est plus sécuritaire que le chemin optimal actuel $\mathcal{O}$.

Afin que le chemin optimal $\mathcal{O}$ reste toujours le plus sécuritaire, on le compare avec tout les autres chemins $\mathcal{C}_\mathcal{S}$ n'ayant pas un sommet $\mathcal{S}$ visité par l'algorithme. On remarque que la minimisation est fait à l'aide d'un chemin $\mathcal{M}$. Un fois que celui-ci est déterminé comme étant le plus sécuritaire, on actualise $\mathcal{O}$. Puisque celui-ci commence à une valeur nulle dans la boucle, il est possible qu'il soit aussi nul après la minimisation. Dans ce cas, il en résulte que l'algorithme n'a pas trouvé de chemin sécuritaire.

La figure 1 montre l'arbre orienté acyclique généré par notre algorithme lors d'une recherche partant de la ville de Montréal (1) vers la ville de Sandy Lake (8) en voiture. On remarque que l'algorithme nous donne le trajet pour aller aux autres villes qu'il a découvert. Il est impossible de s'y rendre avec une voiture de marque *cheap*, car elle manquerait d'essence pour faire Thunder Bay (7) à Sandy Lake (8). Par conséquent, une voiture de marque *super* fut utilisée.

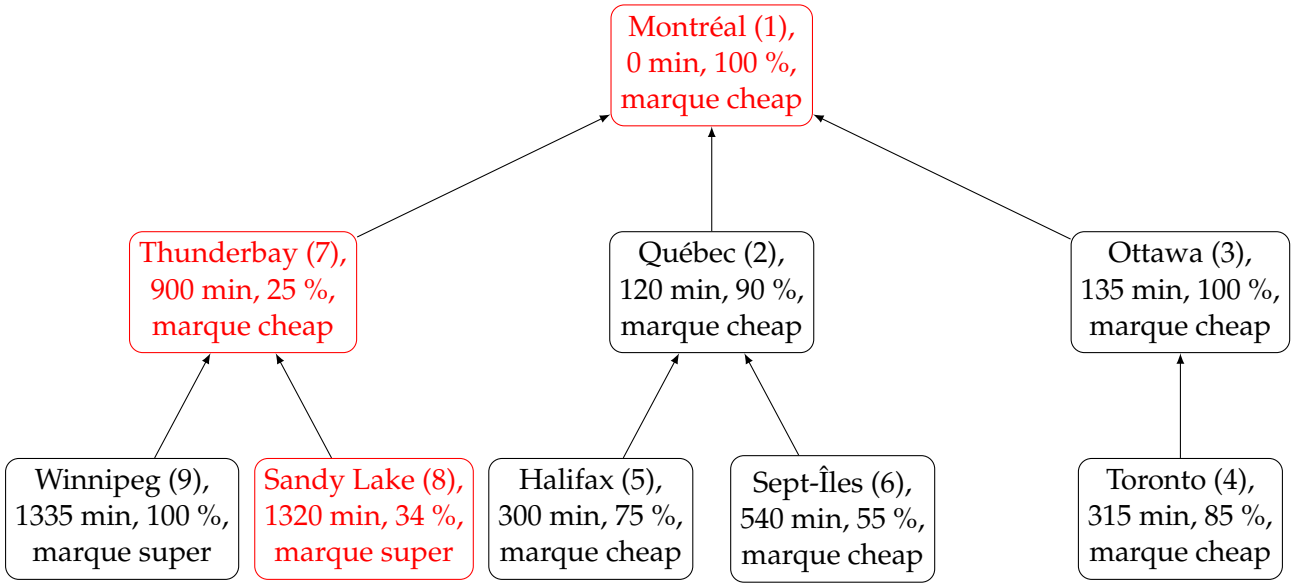

FIGURE 1 – Montréal (1) vers Sandy Lake (8) avec une voiture

Finalement, il est important de préciser que l'algorithme donne le chemin à l'envers à cause de l'utilisation des listes simplement liées. Par conséquent, il faut parcourir le trajet pour obtenir la liste dans le bon ordre. La figure 2 à la prochaine page montre le diagramme de classe utilisé dans notre programme Python. La classe Djikstra reflète l'algorithme expliqué dans les pages plus hauts.

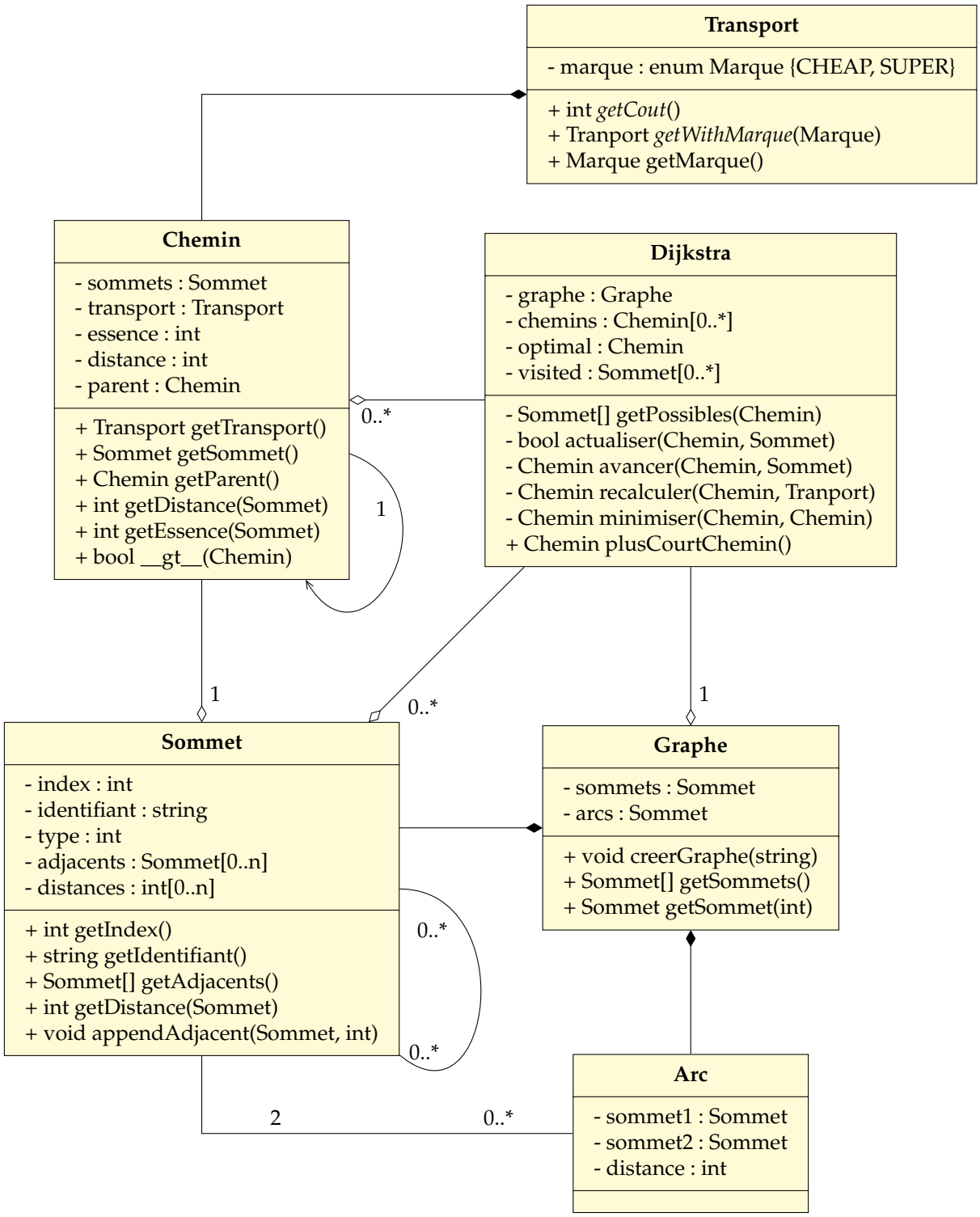

FIGURE 2 – Schéma UML des classes

3 Difficultés rencontrées

3.1 Conserver en mémoire le trajet d'un chemin

Dans l'algorithme, il est nécessaire de conserver en mémoire le trajet actuel d'un chemin. Pour ce faire, on peut soit conserver un tableau des sommets parcourus en ordre ou utiliser des listes liées comme dans notre cas. Le problème dans la première approche est qu'il faut effectuer une copie de cette liste à chaque fois qu'un chemin avance. De plus, la mémoire utilisée par l'arbre final est beaucoup plus grand en raison de la duplication d'éléments.

Cela dit, on sauve du temps d'exécution lors de l'avancement en utilisant des listes simplement liées, car aucune copie ne doit être faite. De plus, notre arbre utilise moins de mémoire. Par contre, lorsqu'il effectuer des opérations avec le trajet d'un chemin, il faut parcourir le chemin à chaque fois. Bref, il est difficile de déterminer laquelle des deux approches est la plus efficace.

3.2 Valider l'algorithme écrit

Par fois, l'algorithme retournait un chemin valide, mais pas nécessairement le plus sécuritaire selon les critères du problème. Il fallait donc manuellement vérifier les résultats plusieurs fois pour augmenter notre certitude envers l'algorithme. Une amélioration à faire à ce problème serait de fournir une série de tests contenant les entrées et la sortie attendue. Cela rendrait le processus de validation et débogage de l'algorithme plus rapide.

4 Conclusion

Pour conclure, ce laboratoire nous a permis de mettre en pratique la théorie vue en classe. En effet, on a pu implémenter des graphes et utiliser l'algorithme de Dijkstra afin d'optimiser nos méthodes de recherche de chemins. Le temps passé dans ce laboratoire a été adéquat. Nous nous attendons a ce que le prochain TP soit comme celui-ci, c'est a dire, nous permettant d'utiliser la théorie dans un cas de pratique concret.