Annee 2013{2014

Projet informatique (PI3) { L2

Itineraires de metro

Francois Laroussinie

francoisl@liafa.univ-paris-diderot.fr

Résumé : L'objectif de ce projet est de realiser un programme permettant de chercher des itineraires dans un reseau de transport comme le metro.

1 Les problemes a resoudre

Les donnees de depart seront les lignes de metro. A partir de ces donnees, le probleme a resoudre sera de chercher de "bons" itineraires pour rejoindre deux stations que fournira l'utilisateur du programme via une interface. Ici un bon itineraire sera le plus rapide (on estimera un temps de parcours en fonction du nombre de stations et du nombre de correspondances). Dans un deuxieme temps, on s'interessera au même probleme lorsque l'utilisateur peut limiter le nombre de correspondances. Le calcul de ces itineraires peut se faire avec di erents algorithmes : pour ce projet, nous en avons retenu deux et ce sont eux qu'il faudra programmer en priorite dans ce projet.

1.1 Description des lignes

La description des lignes de metro sera faite dans des chiers "texte" en utilisant le format suivant :

```
Ligne 14
Saint-Lazare
Madeleine
Pyramides
Châtelet
Gare de Lyon
Bercy
Cour Saint-Émilion
Bibliothèque François Mitterrand
Olympiades
--
Ligne 12
Porte de la Chapelle
Marx Dormoy
Marcadet - Poissonniers
Jules Joffrin
Lamarck - Caulaincourt
...
```

Remarque : certaines lignes de metro contiennent des embranchements et parfois les stations desservies di erent d'une direction a l'autre... Dans un premier temps, on laissera ces aspects de cote et on se contentera de decrire des lignes "simples". Dans un deuxieme temps, il est demande de les integrer au programme. On pourra noter les embranchements "[bloc$_1$||bloc$_2$]" ou chaque bloc$_i$ designe une suite de stations. Pour les "cycles", on pourra utiliser la notation "[bloc$_1$/bloc$_2$]" le bloc$_1$ designe la suite de stations dans le sens haut-bas, et le bloc$_2$ designe la suite pour l'autre sens.

Un chier pourra contenir plusieurs lignes di erentes. Et il devra aussi être possible de repartir les lignes dans plusieurs chiers.

On devra aussi prendre en compte les regroupements de stations permettant des correspondances (par exemple Châtelet-les-Halles avec Châtelet et les Halles ou St-Michel avec St-Michel Notre-Dame,...

Le programme a realiser devra être capable de lire des chiers et d'en extraire des informations pour construire une structure de donnees permettant les recherches d'itineraires decrits ci-dessous.

A n de faire des tests, un chier contenant les lignes du metro parisien sera donne.

1.2 Algorithmes de recherche d'itineraires

La recherche d'itineraires repose sur un calcul de *plus courts chemins* dans un graphe value (c'est-a-dire un graphe ou chaque transition est munie d'une valeur qui correspond a la longueur ou duree de la transition). Une solution classique de ce probleme est l'algorithme de Dijkstra. Pour calculer nos itineraires, on utilisera d'abord cette methode que l'on a nera ensuite pour tenir compte des correspondances. En n on va considerer une derniere methode tres di erente pour chercher des itineraires pour lesquels on xe le nombre maximum de correspondances.

1.2.1 Algorithme de plus court chemin : Dijkstra

Dans un premier temps, on demande de programmer l'algorithme de Dijkstra pour un graphe value G et deux sommets s et s' de G. Cet algorithme est tres classique et il existe une large documentation a son sujet, il est presente succinctement en annexe.

1.2.2 Algorithme de recherche d'itinéraires simples

A partir des lignes de metro, construire un graphe value dont les sommets sont les stations et ou les arêtes correspondent a des etapes des lignes. On supposera que la duree d'une etape est 1min30. Appeler l'algorithme de Dijkstra sur ce graphe pour en deduire des itineraires et des temps de parcours minimaux entre deux stations. On a chera aussi l'itineraire a suivre.

1.2.3 Algorithme de recherche incluant les temps de correspondances

A present, nous voulons tenir compte du temps de correspondance : on supposera que changer de ligne a une station prend 4 minutes. Deduire un nouveau graphe sur lequel on appliquera l'algorithme de Dijkstra.

1.2.4 Algorithme de recherche avec une borne sur le nombre de correspondances autorisées

Pour cette recherche on suppose que l'on dispose de deux stations s et s' et d'un entier k. Le probleme est alors de trouver l'itineraire le plus court (en temps) en utilisant au plus k correspondances.

Dans la suite on suppose qu'il y a n stations di erentes que l'on designera par les entiers $1, \dots, n$.

Pour resoudre ce probleme, nous allons utiliser l'algorithme suivant :

{ Calculer une matrice Direct de dimension $n \times n$ contenant les distances minimales entre chaque station lorsqu'on utilise qu'une seule ligne (sans aucune correspondance) : Direct$[\alpha, \beta]$ (avec α et β dans $\{1, \dots, n\}$) sera la duree minimale d'un itineraire entre la station α et la station β en suivant une ligne directe. Lorsqu'il n'y a pas de trajet possible entre α et β, on utilisera une valeur arbitraire representant l'in ni.
Pour calculer Direct, on pourra calculer d'abord une matrice Direct$_\ell$ pour chaque ligne ℓ du reseau.
A n de pouvoir retrouver les itineraires correspondant aux durees de la matrice Direct, il faudra indiquer dans une matrice Ligne$_D$ les numeros de ligne correspondant : Ligne$_D[\alpha, \beta]$ sera le numero de la ligne (ou d'une ligne si il y a plusieurs possibilites) permettant d'aller directement de α a β en temps Direct$[\alpha, \beta]$.

{ Ensuite on calculera la matrice D$_i$ de dimension $n \times n$ contenant les distances minimales entre chaque station lorsqu'on utilise au plus i correspondances de la maniere suivante : D$_0$ = Direct et

$$\mathrm{D}_{i+1}[\alpha, \beta] = \min \ \mathrm{D}_i[\alpha, \beta] \ , \ \min_{\gamma \neq \alpha, \beta} \{\mathrm{D}_i[\alpha, \gamma] + \mathrm{D}_0[\gamma, \beta] + \quad \}$$

L'idee de cette formule est que la duree minimale entre α et β en autorisant $i + 1$ correspondances est soit celle obtenue avec i correspondances, soit elle correspond a la duree minimale d'un itineraire de α a une station γ en au plus i correspondances PLUS la duree d'un trajet direct entre γ et β PLUS le temps de correspondance (ici 4 minutes)...
Pour permettre de retrouver le detail des itineraires, on va utiliser des matrices Via$_i$: Via$_i[\alpha, \beta]$ sera le sommet par lequel il faut passer pour aller de α a β en temps D$_i[\alpha, \beta]$ avec un trajet direct entre γ et β. On initialisera Via$_0[\alpha, \beta]$ avec α puisque ce trajet se fait sans changement...

1.2.5 Statistiques sur le réseau

A partir des algorithmes de la question precedente, on ajoutera le calcul de plusieurs mesures sur le reseau :

{ Nombre minimal de correspondances permettant de se rendre partout dans le reseau depuis n'importe quelle station.

{ Nombre minimal de correspondances permettant de se rendre partout dans le reseau depuis n'importe quelle station en *un temps minimal*.

{ Les stations les plus eloignees dans le reseau.

1.3 Interface

Le programme devra contenir une interface pour permettre a un utilisateur de :

{ Lire un chier de lignes de metro.
{ Chercher un itineraire simple.
{ Chercher un itineraire prenant en compte le temps de correspondance.
{ Chercher un itineraire permettant de borner le nombre de correspondances.
{ Donner des statistiques sur le reseau.

1.4 Extensions

Une fois que le programme sera operationnel, on pourra s'interesser aux extensions suivantes :

{ Utiliser les les de priorites de Java pour ameliorer l'implementation de l'algorithme de Dijkstra.
{ Permettre la gestion de perturbations : une partie d'une ligne pourra être declaree comme inactive et le calcul des itineraires devra en tenir compte...
{ Proposer d'autres algorithmes...

A Algorithme de Dijkstra

Cet algorithme permet de trouver les plus courts chemins depuis un sommet s dans un graphe value $G = (S, A, w)$ avec S un ensemble de sommets, A un ensemble de transitions et w une fonction qui associe a chaque transition une valeur positive ou nulle (on note $w(s, s')$ la duree ou la longueur de la transition (s, s').

L'algorithme de Dijkstra consiste a decouvrir, en partant de s, tous ses voisins en procedant par distance croissante : on cherche d'abord le plus proche, puis le deuxieme plus proche, etc. Pour le premier sommet s_1 a trouver (le plus proche de s), nous savons qu'il est accessible par une seule transition (car w associe des valeurs positives ou nulle aux transitions). Le second sommet s_2 est accessible par une seule transition a partir de s, ou par deux transitions en passant par s_1. Le troisieme plus proche sommet de s sera accessible par une, deux ou trois transitions (en passant par s_1 et/ou s_2)... A chaque fois, qu'on decouvre un nouveau "plus proche sommet", on voir comment celui-ci permet de rapprocher s d'autres sommets du graphe.

On va utiliser un tableau d$[-]$ qui donne pour chaque sommet sa distance depuis s en utilisant les sommets deja decouverts (il est facile de voir que d$[s']$ correspond a une surapproximation de la distance minimale de s a s' et cette distance n'est plus une approximation lorsque s' est decouvert). A chaque fois qu'on decouvre un sommet s', on doit mettre a jour le tableau d$[-]$ pour tenir compte des chemins qui passent par s' : il est possible que la distance entre s et s'' soit inferieure en passant par s' et dans ce cas, on aura d$[s'']$ = d$[s']$ + $w(s', s'')$.

On utilise aussi un tableau Pred pour memoriser le chemin decouvert par l'algorithme : Pred$[x]$ sera le sommet y par lequel on a decouvert le sommet x, c'est-a-dire qu'il existe un plus court chemin entre s et x dont la derniere transition est (y, x).

L'algorithme 1 decrit l'algorithme de Dijkstra. Il faut savoir que cet algorithme utilise generalement une le de priorite pour gerer la recherche des sommets, la version presentee ici est donc simpli ee (et moins e cace) que le veritable algorithme de Dijkstra.

Un exemple de l'application de cet algorithme est donne a la gure 1. A gauche se trouve le graphe initiale, et a droite celui avec les distances minimales indiquees en gras (le numero entre parentheses indique l'ordre dans lequel les sommets ont ete decouverts) et les transitions (Pred$[x], x$) sont indiquees en gras.

Procédure PCC-Dijkstra(G, s)

//$G = (S, A, w)$: un graphe orienté, valué avec $w : A \to \mathbb{R}_+$.

//$s \in S$: un sommet origine.

begin

  pour chaque $u \in S$ **faire**

    Pred[u] := nil

    d[u] := $\begin{cases} 0 & \text{si } u = s \\ \infty & \text{sinon} \end{cases}$

  E := Ensemble(S)

  tant que $E \neq \emptyset$ **faire**

    Soit u := le plus sommet dans E ayant la valeur d[$-$] minimale

    Extraire u de E

    pour chaque $(u, v) \in A$ **faire**

      si $d[v] > d[u] + w(u, v)$ **alors**

        //on met à jour d[$-$]

        d[v] := d[u] + $w(u, v)$

        Pred[v] := u

  return *d*, *Pred*

end

Algorithme 1 : algorithme de Dijkstra (simpli e)

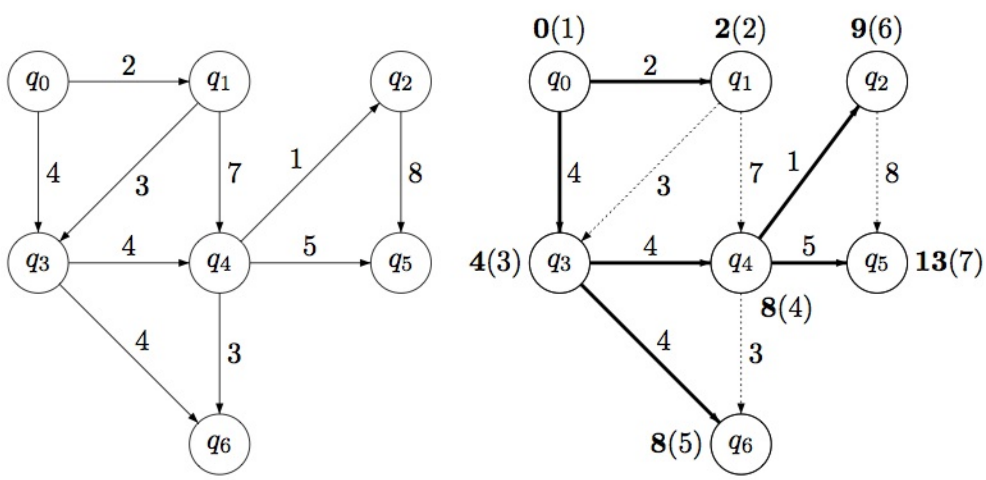

Figure 1 { Exemple d'application de la'lgorithme de Dijkstra