Jeronimo Louis IS2A3

Perrin-Zen Bastien

**Projet Structures de Données – Graphes**

**Rapport final**



**Lundi 16 mai 2022**

Table des matières

[I – Partie analyse et conception de notre application 3](#_Toc104878383)

[1 – Précisions du cahier des charges 3](#_Toc104878384)

[2 – Analyse et conception du problème 4](#_Toc104878385)

[3 – Choix et justification des différentes structures de données utilisées 5](#_Toc104878386)

[A – Tableau comparatif 5](#_Toc104878387)

[B – Schéma de la structure de données retenue pour le graphe auxiliaire 6](#_Toc104878388)

[4 – Algorithmes de résolution appliqués à nos structures de données 7](#_Toc104878389)

[A – Méthode pour créer le tour géant 7](#_Toc104878390)

[B – Procédure SPLIT 7](#_Toc104878391)

[C – Plus court chemin 8](#_Toc104878392)

[5 – Pseudo-code de la procédure globale d’enchainement de ces algorithmes 9](#_Toc104878393)

[II – Mode d’emploi de l’application 9](#_Toc104878394)

[III – Description des exemples traités 9](#_Toc104878395)

[IV – Conclusion 9](#_Toc104878396)

[V – Bilan personnel sur le projet 9](#_Toc104878397)

# I – Partie analyse et conception de notre application

## 1 – Précisions du cahier des charges

Pour ce projet, nous avons comme objectif de créer un programme qui répond au problème de planification de livraison expliqué dans le sujet fourni.

Nous allons préciser ci-dessous quelques éléments supplémentaires, non précisés dans le cahier des charges.

Premièrement, nous supposerons que le graphe global est non-orienté, i.e. qu’il est possible d’aller de A vers B mais aussi de B vers A. Ces deux chemins ont le même coût.

Ensuite, dans notre fichier de données, les sommets n’ont pas de nom. Nous leur donnerons un nom numérique (1,2,3 et non a,b,c) par soucis de simplicité dans l’application des algorithmes.

Enfin, nous admettons que l’application de notre programme n’est valable uniquement dans le cas où chaque véhicule de livraison a la même capacité.

## 2 – Analyse et conception du problème

Notre problème consiste à développer un programme qui détermine de manière optimale le nombre de véhicules de livraison nécessaires pour livrer des clients répertoriés dans un fichier de données prédéfini, afin de minimiser le coût total de l’opération. Nos véhicules ont tous la même capacité limitée.

**Modélisation du problème initial sous forme de graphes :**

Sommets : clients + dépôt.

Arêtes : Coûts affectés au chemin entre les 2 sommets.

Notre graphe sera par définition un graphe complet car chaque sommet est relié à tous les autres.

Afin de résoudre ce problème, nous allons découper notre raisonnement en 3 grandes étapes.

**1ère étape : le tour géant**

La première étape consiste à définir un tour géant, qui n’est autre qu’un tableau de taille égale au nombre de clients. Nous définissons le premier client, puis, par la méthode du plus proche voisin, nous explorons tous les sommets afin de déterminer le voisin (client) le plus proche de notre client actuel.

**2ème étape : Procédure SPLIT**

La deuxième étape consiste, à partir du tour géant produit précédemment, à construire le graphe auxiliaire, sachant que la capacité des véhicules ne peut excéder la quantité donnée dans le fichier d’entrée.

**Modélisation du graphe auxiliaire :**

Sommets : Etats associés au client.

Arcs : Livraisons.

**3ème étape : Plus court chemin**

Enfin, troisième étape, on définit dans le graphe auxiliaire, le plus court chemin qui permet de livrer tous les clients avec un coût minimal. Nous utiliserons l’algorithme de Dijkstra car tous nos coûts sont positifs donc Dijkstra est possible.

## 3 – Choix et justification des différentes structures de données utilisées

### A – Tableau comparatif

**Structure de données du tour géant :**

Pour représenter le tour géant T, nous utiliserons un tableau d’entiers.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| T = |  |  |  |  |  |

**Tableau comparatif des structures de données du graphe auxiliaire :**

|  |  |  |  |
| --- | --- | --- | --- |
| **Propositions** | **1ère SD** | **2ème SD** | **3ème SD** |
| **Création de types** | Pas de création de types | Type arête composé d’une distance et d’un indice.  Type sommet composé d’une liste d’arêtes. | Type arête composé d’une distance et d’un indice.  Type sommet composé d’une liste d’arêtes. |
| **Description** | Notre graphe auxiliaire sera représenté comme une matrice d’entiers (poids) correspondant à la matrice dist. | Notre graphe auxiliaire sera représenté comme un tableau de sommets (tableau de listes d’arêtes). | Notre graphe auxiliaire sera représenté par 2 tableaux, un tableau de sommets Succ des successeurs et un tableau de sommets Head des indices de début des successeurs dans Succ. |
| **Complexité** | o(n²) | o(m+n) avec m<n | o(m+n) avec m<n |

**Schéma des types définis dans nos SD :**

|  |  |  |
| --- | --- | --- |
| **Arête** |  | **Sommet** |
| **Int distance**  **Int indice** |  | **Arête[] arêtes** |

**Reprendre les noms : modifs**

**Structure de données du plus court chemin :**

La structure de données que l’on utilisera ici est la structure de données sélectionnée à la deuxième étape, i.e. la deuxième proposition :

Définition du type arête composé d’une distance et d’un indice.

Définition du type sommet composé d’une liste d’arêtes.

Notre graphe auxiliaire sera représenté comme un tableau de sommets (tableau de listes d’arêtes).

Nous avons choisi cette solution car nous avons une complexité en espace mémoire de o(m+n) avec m strictement inférieur à n car notre graphe n’est pas complet. Les complexités des deux autres structures de données ne sont pas meilleures (o(n²) pour la 1ère, o(m+n) pour la 3ème avec m aussi inférieur à n car notre graphe n’est pas complet). La deuxième SD a donc la complexité la plus faible. Ce n’est pas le seul critère de choix : choisir une SD composée d’une matrice n’est pas du tout optimal en cas d’un grand nombre de clients, et la SD avec les tableaux Succ et Head n’est pas très pratique car on ne sait pas à l’avance quelle taille doit-on allouer au tableau Succ. De plus, notre réflexion de sélectionner la deuxième SD s’est aussi motivé par la facilité de parcourir une liste de sommets par rapport aux méthodes des autres SD.

### B – Schéma de la structure de données retenue pour le graphe auxiliaire

Une image contenant table

Description générée automatiquement

## 4 – Algorithmes de résolution appliqués à nos structures de données

### A – Méthode pour créer le tour géant

**Méthode du plus proche voisin :**

A partir du sommet initial défini en paramètre, l’algorithme recherche le sommet non-marqué le plus proche et le marque dans le tableau Mark, puis recherche le sommet le plus proche de ce sommet, qu’il marque à son tour etc.

Enfin, l’ordre de livraison dépend du premier client livré car avec la méthode du plus proche voisin qui sera utilisée pour créer le tour géant, la définition du client suivant dépend de son successeur, donc le premier client livré impacte l’ordre de tous les clients suivants.

La procédure ressemblerait à ceci :

Action plusProcheVoisin(T,dist,sommetInit) :

D : dist Matrice[N][N] d’entiers, sommetInit Entier

D/R : T Vecteur[N] de sommets

Locales : i et j entiers

j=0

i= valeurMinLigne(dist[sommetInit])

Tant que j<N

Remplacer(T[j],T[i])

i = valeurMinLigne(dist[i])

j++

Fin Pour

Fin Action

Dans ce pseudo code nous supposons qu’il existe une méthode valeurMinLigne prenant en paramètre un tableau et permettant de renvoyer l’indice de la valeur minimale non-marquée du vecteur.

### B – Procédure SPLIT

Notre sous-graphe auxiliaire étant représenté comme un tableau de liste de sommets, contenant eux même des arêtes, l’algorithme de la méthode split ne va pas avoir de grande modification. En effet nous avons décidé de représenter dist comme une matrice et T sous forme d’un tableau. Le vecteur q est lui aussi représenté sous la forme d’un vecteur de n cases. La seule méthode à approfondir est donc la création de l’arc dans H. Celle-ci sera donc :

Action add(H,i,j,cost):

Données: i, j et cost Entier

Données-Resultat:H Vecteur de Sommets [N]

addArete(j,cost, H[i-1].sommets) # Détailler

Fin Action

### C – Plus court chemin

**Méthode du plus court chemin avec l’algorithme de Dijkstra :**

Le principe de l’algorithme de Dijkstra est de calculer les plus courtes longueurs de proche en proche par ajustements successifs. On initialise 3 tableaux Pi, Père et Mark, puis tant qu’il existe des sommets non marqués, on choisit parmi les sommets non marqués le sommet x avec le coût minimum, on marque x, puis pour tous les successeurs y de x, si pi[y] est supérieur à pi[x] + w(xy), alors on affecte à pi[y] la valeur pi[x] + w(xy) et à Père[y] la valeur x.

Dans notre problème, le tableau Head est remplacé par le tableau des sommets et le tableau Succ est la liste des arêtes de chaque sommet. (Le pseudo code de cet algorithme est détaillé dans le cours de Graphes et Combinatoire)

Fonction coutPlusCourtChemin(H) :

Données : H Vecteur [N] de Sommet

Locales : Mark Vecteur[N] de Booléen, pi Vecteur[N] d’Entier, Pere Vecteur[N] d’entiers, fini booleen, x Entier, ret et i Entier

On initialise Mark à false, pi à INFINI et Père à 0

fini=false

Pi[0] = 0

Père[0]=0

Tant que !fini faire

fini=true

x=choisirX(Mark,pi)

Si x !=NIL alors

fini =false

Mark[x]=true

Pour k de H[x].num à H[x+1].num-1 faire

y = getSommet(H[x].listeAretes, k) //Renvoie un entier

Si pi[y] > pi[x]+getPoids(H[x].listeAretes,k) alors

pi[y] = pi[x]+getPoids(H[x].listeAretes,k)

Pere[y]=x

FinSi

FinPour

FinSi

FinTantQue

ret=0

pour i de 0 à N-1

ret+=pi[i]

finPour

return ret

FinFonction

## 5 – Pseudo-code de la procédure globale d’enchainement de ces algorithmes

Action enchainement() :

Locales : T et q Vecteur[N] d’entiers, H vecteur[N] de sommets, dist Matrice[N][N] d’entiers, Q et n entiers

LireFichier(T,q,Q,n,dist)

plusProcheVoisin(T,dist,sommetInit)

H=Split(T,Q,n,dist,q)

Afficher(coutPlusCourtChemin(H))

FinAction

# II – Mode d’emploi de l’application

Nous allons présenter ci-dessous le mode d’emplois de notre application, en décrivant les fichiers d’entrées et de sorties et en présentant les différentes commandes de compilation.

# III – Description des exemples traités

Dans cette partie, nous allons utiliser notre application sur un exemple précis.

# IV – Conclusion

Pour conclure, nous avons réussi à arriver au terme de ce projet mêlant structures de données et graphes et ayant pour but de développer une application qui détermine de manière optimale le nombre de véhicules de livraison nécessaires pour livrer des clients répertoriés dans un fichier de données prédéfini, afin de minimiser le coût total de l’opération.

Nous avons fait … Cependant, nous n’avons pas fait … .

# V – Bilan personnel sur le projet

Nous tenons à dire que ce projet nous a beaucoup intéressé, notamment grâce au fait qu’il mélangeait plusieurs disciplines vues en cours cette année dans le but de les appliquer de manière pratique sur un cas concret. Ce projet nous a permis d’éclaircir certains points qui nous étaient encore un peu vagues. Nous apprécions ce genre de projet formateur pour nous et qui reprend de manière simplifiée ce que nous pourrions être amenés à faire plus tard en entreprise. Nous avons quelques remarques à apporter vis-à-vis des futurs projets de ce type qui seront proposés aux futures promotions IS2A3 :