Rapport de Projet en Algorithmique

Pablo Rauzy

Rendu le 13 février 2009

1 Introduction

Pour chaque module, je vais présenté la structure de donnée et les fonctions qui le compose.

Tout les noms commencent par `PA`, comme « Projet Algo ».

2 PACity

2.1 Structure de donnée

```
typedef struct pa_city_ {
  int id;
  int x, y;
  PALinkedCity *link;
} PACity;
```

La structure `PACity` représente une ville, elle contient son identifiant, ses coordonnées et un pointeur vers une liste chainée des villes auxquelles elle est directement reliée.

2.2 Fonctions

`PACity * PACity_new (int id, int x, int y);`
Alloue et initialise une ville, avec l'id `id` et les coordonnées (`x`, `y`).

`void PACity_addLinkedCity (PACity *city, int to);`
Ajoute la ville dont l'id est `to` aux voisines de la ville `city`.

`int PACity_linkExists (PACity *city, int to);`
Retourne `1` si un lien existe entre `city` et la ville d'id `to`, `0` sinon.

`void PACity_addRide (PACity *city, int to, int lineId, int depTime, int arrTime);`
Ajoute un trajet entre `city` et la ville d'id `to`, sur la ligne d'id `lineId`. Le train part à `depTime` et arrive à `arrTime`, tout deux données en minutes.

`void PACity_free (PACity *city);`
Libère la mémoire occupée par la ville `city`.

3 PALine

3.1 Structure de donnée

```
typedef struct pa_line_ {
  int id;
  int nbCities, *cities;
  int nbTravels;
  PADatetime ***stopDates;
} PALine;
```

La structure `PALine` représente une ligne, elle contient l'identifiant de la ligne, le nombre de ville dans lequel elle passe, un tableau avec les `id` de ces villes, le nombre de voyage sur cette ligne par jour et les heures de passages dans les villes indéxées dans un tableau à double entrées (numéro de voyage et numéro de ville).

3.2 Fonctions

`PALine * PALine_new (int id, int nbCities, int nbTravels);`
Alloue et initialise une ligne avec l'id `id`, qui passe par `nbCities` ville et fait le voyage `nbTravels` fois par jour.

`void PALine_setNbTravels (PALine *line, int nbTravels);`
Change le nombre de voyage de la ligne `line` pour `nbTravels`.

`void PALine_addCity (PALine *line, int cityNum, int cityId);`
Ajoute la ville dont l'id est `cityId` en `cityNum`ème position sur la ligne `line`.

`void PALine_addTravel (PALine *line, int travelNum, const char *stopDates);`
Ajoute un `travelNum`ème voyage à la ligne `line`, les horaires de passage dans les villes sont dans la chaine de caractère `stopDates`, formattés de la même manière que dans le format de données que l'ont doit lire.

`void PALine_free (PALine *line);`
Libère la mémoire occupée par la ligne `line`.

4 PADatetime

4.1 Structure de données

```
typedef struct pa_datetime_ {
  int days, hours, minuts;
  int ttlMinuts;
} PADatetime;
```

La structure `PADatetime` représente un temps (une durée ou une heures par exemple). Elle contient un nombre de jours, un nombre d'heures, un nombre de minutes, ainsi que le temps représenté en minutes.

4.2 Fonctions

`PADatetime * PADatetime_new (int days, int hours, int minuts);`
Alloue et initialitise un temps avec `days` jours, `hours` heures et `minuts` minutes. Des ajustements sont fait si nécessaire.

`void PADatetime_update (PADatetime **dt, int days, int hours, int minuts);`
Met à jour le temps `*dt` avec `days` jours, `hours` heures et `minuts` minutes. Des ajustements sont fait si nécessaire.

`void PADatetime_free (PADatetime *dt);`
Libère la mémoire occupée par le temps `dt`.

5 PALinkedCity

5.1 Structure de données

```
typedef struct pa_linked_city_ {
  int cityId;
  PARide *ride;
  struct pa_linked_city_ *next;
} PALinkedCity;
```

La structure `PALinkedCity` représente une « ville chainée », utilisé pour représenter la liste des villes voisines (directement reliées) à une ville. Elle contient l'`id` de la ville représentée, une liste chainée des trajets entre la ville et celle dont elle est voisine, et un lien vers la « ville chainée » suivante.

5.2 Fonctions

`PALinkedCity * PALinkedCity_new (int to, PALinkedCity *next);`
Alloue et initialise une « ville chaînée » vers la ville d'id `to`, l'élément suivant de la liste chainée est pointé par `next`.

`PARide * PALinkedCity_nextRide (PALinkedCity *linkedCity, int time);`
Retourne le trajet du prochain départ pour la « ville chainée » `linkedCity` à partir du temps `time` en minutes.

`void PALinkedCity_free (PALinkedCity *linkedCity);`
Libère la mémoire occupée par la « ville chainée » `linkedCity`.

6 PARide

6.1 Structure de données

```
typedef struct pa_ride_ {
  int lineId;
  int depTime, arrTime;
  int depCity, arrCity;
  struct pa_ride_ *next;
```

```
} PARide;
```

La structure `PARide` représente un élément d'une liste chainée de trajet d'une ville à une autre. Elle contient l'`id` de la ligne, les heures de départ et d'arrivée en minutes, les `id` des villes de départ et d'arrivée et un lien vers le trajet suivant dans l'ordre chronologique de départ *(mieux par heures d'arrivée si on ne considère pas qu'un trajet à toujours la même durée)*.

6.2 Fonctions

`PARide * PARide_new (int lineId, int depTime, int arrTime, int depCity, int arrCity, PARide *next);`
Alloue et initialise un trajet sur la ligne `lineId`, qui part à `depTime` (en minutes) de la ville d'id `depCity` et qui arrive à `arrTime` (en minutes) dans la ville d'id `arrCity`, l'élément suivant de la liste chainée est pointé par `next`.

`PARide * PARide_copy (PARide *ride);`
Créer et retourne un pointeur vers une copie exacte du trajet `ride`. *utile ? je crois qu'on peut faire direct *ride1 = *ride2, à voir.*

`void PARide_free (PARide *ride);`
Libère la mémoire occupée par le trajet `ride`.

7 PAData

7.1 Structure de données

```
typedef struct pa_data_ {
  int nbCities;
  PACity **cities;
  int nbLines;
  PALine **lines;
} PAData;
```

La structure `PAData` stocke toutes les données. Elle contient le nombre de villes et un tableau de celles-ci, ainsi que le nombre de lignes et un tableau de celles-ci.

7.2 Fonctions

`PAData * PAData_new (FILE *in);`
Alloue et initialise des données avec les données lues dans le fichier `in`.

`void PAData_free (PAData *data);`
Libère la mémoire occupée par les données `data`.

8 PAio

Ce module n'a pas de structure de données, il sert à gérer tout ce qui est lecture / écriture dans le programme.

8.1 Fonctions

`PACity ** PAio_readCities (FILE *in, int *nbCities);`
Lit les villes données dans le fichier `in`, stocke le nombre de villes dans `nbCities` et retourne un tableau avec les villes lues.

`PALine ** PAio_readLines (FILE *in, int *nbLines);`
Lit les lignes données dans le fichier `in`, stocke le nombre de lignes dans `nbLines` et retourne un tableau avec les lignes lues.

`void PAio_printCities (FILE *out, PACity **cities, int nbCities);`
Écrit dans le fichier `out` les `nbCities` villes contenues dans le tableau `cities` de manière lisible par un humain.

`void PAio_printLines (FILE *out, PALine **lines, int nbLines);`
Écrit dans le fichier `out` les `nbLines` lignes contenues dans le tableau `lines` de manière lisible par un humain.

`void PAio_printDijkstraResult (FILE *out, PADijkstra *dijkstraResult);`
Écrit dans le fichier `out` le résulat de l'algorithme de Dijkstra `dijkstraResult` de manière lisible par un humain.

9 PAHeap

9.1 Structure de données

```
typedef struct pa_heap_ {
  int size, maxSize;
  int *cities;
  int *times;
} PAHeap;
```

La structure `PAHeap` sert à représenter un tas, utilisé pour l'algorithme de Dijkstra. Elle contient la taille maximum possible du tas, sa taille actuelle, un tableau qui est le tas des `id` des villes et un tableau avec des temps en minutes correspondant au villes et selon lesquels le tas est organisé.

9.2 Fonctions

`PAHeap * PAHeap_new (int maxSize);`
Alloue et initialise un tas dont la taille maximum est `maxSize`.

`void PAHeap_add (PAHeap *heap, int cityId, int time);`
Ajoute dans le tas `heap` la ville d'id `cityId` avec le temps `time` en minutes.

`void PAHeap_swap (PAHeap *heap, int a, int b);`
Échange dans le tas `heap` les éléments d'index `a` et `b`.

`int PAHeap_pop (PAHeap *heap);`
Retire du tas `heap` la valeur du dessus (la plus petite) et la retourne. C'est à dire renvoie la ville la plus ≪ proche ≫.

`int PAHeap_isEmpty (PAHeap *heap);`
Retourne `1` si le tas `heap` est vide, `0` sinon.

`void PAHeap_update (PAHeap *heap, int cityId, int time);`
Met à jour le tas `heap` et changeant le temps de la ville d'id `cityId` pour `time` donné en minutes.

`void PAHeap_free (PAHeap *heap);`
Libère la mémoire occupée par le tas `heap`.

10 PADijkstra

10.1 Structure de données

```
typedef struct pa_dijkstra_ {
  PARide **rides;
  int nbCities;
  int depCity, arrCity, depTime;
} PADijkstra;
```

La struct `PADijkstra` stocke le résultat de l'algorithme de Dijkstra. Elle contient la ville de départ, la ville d'arrivée, l'heure de départ, le nombre de ville par lequel on passe et un tableau des trajets que l'on va emprunter.

10.2 Fonctions

`PADijkstra * PADijkstra_run (PAData *data, int startCity, int arrCity, int startTime);`
Lance l'algorithme de Dijkstra pour récupérer le moyen le plus rapide de se rendre à la ville d'id `arrCity` sachant qu'on part à `startTime` (en minutes) de la ville d'id `startCity` et que les données du problème sont dans `data`.

`PADijkstra * PADijkstra_getResult (PADijkstra *dijkstraResult);`
L'algorithme de Dijkstra retourne des résultats pas très pratique à utiliser : on doit naviguer dedans « à l'envers » en partant de la ville d'arrivée et en récupérant la ville précédente à chaque fois. Cette transforme le résultat brut qui sont dans `dijkstraResult` en résultas utilisable simplement pour l'affichage et le renvoi.

`void PADijkstra_freeResult (PADijkstra *dijkstraResult);`
Libère la mémoire utilisée par le résultat de l'algorithme de Dijkstra `dijkstraResult`.

11 projalgo

Ce module est celui qui contient le main.

11.1 Options en ligne de commande

`--from` ou `-f` :
l'id de la ville de départ.

`--to` ou `-t` :
l'id de la ville d'arrivée.

`--at` ou `-a` :
l'heure de départ au format `%02dh%02d`.

`--minuts` ou `-m` :
l'heure de départ en minutes.

`--data` ou `-d` :
le fichier dans lequel lire les données, par défaut `stdin`.

`--output` ou `-o` :
le fichier dans lequel écrire les résultats, par défaut `stdout`.

Rapport de Projet en Algorithmique

Pablo Rauzy

Rendu le 10 avril 2009

1 Introduction

Ce rapport ne contient que les nouveautés et les principaux changements depuis la fin de la première étape.

2 Le générateur

Le générateur prend en argument le nombre de villes (option `-c` sur la ligne de commande), le nombre de lignes (option `-l`), la densité du graphe qui va servire à la génération (option `-d`), les nombres minimum et maximum de passage journalier que peut faire un ligne (options `-m` et `-M`).

Il génère un graph du nombre de sommet spécifié par le nombre de villes, et de la densité choisie.

Ensuite, tant qu'on a pas le nombre de lignes demandées, il choisi deux villes au hasard et envoie un Dijkstra pour avoir un chemin entre ces deux villes. Si le chemin n'existe pas, on recommence tout depuis le début. Sinon, la ligne est créée si elle passe par au moins 3 villes (une étape entre le départ et l'arrivée au minimum) et part au plus deux tiers du nombres total de villes.

Enfin, on teste avec Dijkstra que chaque ville peut atteindre chacune des autres, si ce n'est pas le cas, on recommence tout.

3 Clustering

3.1 Structure de donnée

```
typedef struct pa_cluster_ {
  int ttlCities;
  int nbGroup;
  int *nbCities;
  int **groups;
  double *maxDist;
} PACluster;
```

La structure `PACluster` représente un cluster. Elle contient le nombre de groupes qui compose le cluster, un tableau qui indique combien il y a de ville dans chaque groupe, un tableau de tableau d'id de villes, représentant les groupes ainsi qu'un tableau contient les distances entre les deux villes les plus éloignées de chaque cluster. Elle contient aussi le nombre total de ville pour des raisons pratiques.

3.2 Fonctions

`PACluster * PACluster_new (PAData *d, int nbGroup);`
Alloue et créer un cluster de `nbGroup` groupes à partir des données dans `d`.

`void PACluster_addCity (PACluster *cluster, int group, int cityId);`
Ajoute la ville `cityId` au groupe numéro `group` du cluster `cluster`.

`double PACluster_averageTime (PAData *d, int depCity, int arrCity);`
Calcul en minutes le temps moyen nécéssaires à faire le trajet de `depCity` à `arrCity`.

`double ** PACluster_getDistances (PAData *d);`
Renvoie une matrice de distances (toujours en minutes) entre les villes.

`void PACluster_optimize (PACluster *cluster, double **dist);`
Optimise le cluster obtenue par une solution initiale en simulant des changements de groupes plus ou moins au hasard. Si un changement est concluant il est effectués.

`double PACluster_getMaxDistanceForGroup (PACluster *cluster, double **dist, int group);`
Renvoie la distance maximum entre deux villes du groupe `group`.

`double PACluster_maxDistIfMove (PACluster *cluster, double **dist, int cityId, int group)`
Renvoie la distance maximum entre deux villes du groupe `group` en faisant comme si la ville `cityId` était dedans.

`double PACluster_maxDistIfRemove (PACluster *cluster, double **dist, int cityNum, int group);`
Renvoie la distance maximum entre deux villes du groupe `group` en faisant comme si la ville `cityId` n'était pas dedans.

`void PACluster_free (PACluster *cluster);`
Libère la mémoire occupée par le cluster `cluster`.

4 PAMem

Ce module contient juste deux fonctions : `PA_malloc` et `PA_free`. Ces deux fonctions sont juste des encapsuleurs de `malloc` et `free` qui permettent de tracer les allocations et le libérations de mémoires. Il y a eu beacoup de progrès de ce côté là depuis le rendu du premier rapport.

5 Quelques changements

`void PAio_printCluster (FILE *out, PACluster *cluster);`
Le module `PAio` sait maintenant afficher les résultats d'un calcul de cluster.

`int * PADijkstra_getDistances (PAData *data, int depCity);`
Le module `PADijkstra` contient maintenant une fonctions qui renvoie un tableau de distances d'une ville à toutes les autres.

Il n'y a plus de mémoire perdue pendant la lecture des données et l'algorithme de Dijkstra.

Le programme en ligne de commande permet de calculer la distance moyenne entre deux villes avec l'option `--travel-time`, un cluster de n groupes avec `--cluster n` ou `-c n`. On peut l'empêcher de faire le calcul de trajet (option par défaut) en lui spécifiant `--no-travel`.

6 L'interface graphique

L'interface graphique a été faite en moins de deux jours, je n'ai pas eu le temps de faire tout ce que je souhaitais, mais il est possible de l'utiliser pour faire tout ce que fait le programme en ligne de commande : calcul de trajet optimal d'une ville à une autre à une heure précise, calcul de temps de trajet moyen d'une ville à une autre, calcul de cluster.

Il n'y a pas la possibilité comme je l'aurait souhaité de créer / éditer des réseaux. Par contre la gestion des documents (nouveau, ouverture, enregistrement, enregistrement sous) fonctionne déjà.

J'ai fait en sorte que mon générateur place les villes sur une grille de 500 par 500, et c'est aussi la taille par défaut de la zone d'affichage. Cependant il est possible de zoomer (x1, x2, x3, x4 et x5), c'est utile par exemple pour les fichiers de teste qui nous ont été fournis au début est qui sont sur une grille de 100 par 100. Il est aussi possible de se déplacer dans la zone si jamais on ne voit pas tout à l'écran en faisant un cliqué-glissé avec la souris.

Elle est codée en GTK+.

7 The end ?

C'est déjà fini, ça fait que trois page à la place de 5, mais mon premier rapport en fait 7 à la place de 2...