Réseaux de transports

L'implémentation se fera en langage C dans ce sujet.

1 Graphes temporels

On fournit une implémentation de listes chaînées dans les fichiers *list.h* et *list.c* que vous êtes libres d'utiliser et de modifier. **Pensez à les copier avant de faire toute modification.** Il faut d'une part penser à inclure *list.h* dans tout fichier voulant l'utiliser : c'est déjà fait dans *main.c*. D'autre part, on rappelle comment compiler avec plusieurs fichiers :

```
gcc -Wall main.c list.c -o nomExecutable
```

Question 1. *Introduire un nouveau type* `temps` *stockant un instant de la journée sous la forme d'un couple (heure,minute).*

Question 2. *Implémenter la fonction* `Difference` *qui prend deux* `temps` *en paramètre et renvoie la différence en minute.*

Les graphes orientés temporels sont des graphes orientés tels que les arcs sont des quadruplets (u, v, t_d, t_a) avec :

— u sommet de départ,
— v sommet d'arrivée,
— t_d temps de départ,
— t_a temps d'arrivée, avec $t_a > t_d$.

Il peut y avoir plusieurs arcs entre un même couple de sommet.

La figure **??** est un exemple de graphe temporel. Les arcs ont une étiquette au début (le temps de départ) et une à la fin (le temps d'arrivée). Pour une représentation plus claire, on considère dans cet exemple que tous les temps sont entre 10h et 11h. Les valeurs affichées correspondent aux minutes.

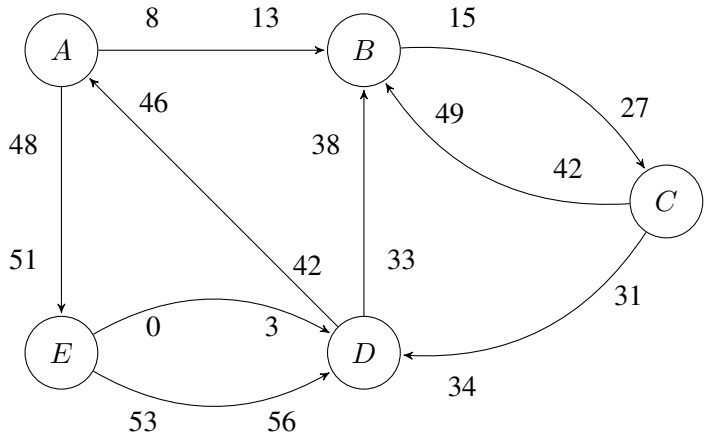

FIGURE 1 – Exemple de graphe temporel

Question 3. *Proposer une implémentation d'un graphe temporel en C.*

Un chemin dans un graphe temporel est une séquence d'arcs $(a_1, \ldots, a_n)$ telle que pour tout $i \in [\![1, n-1]\!]$, si $a_i = (u, v, t_d, t_a)$ et $a_{i+1} = (u', v', t'_d, t'_a)$, alors $v = u'$ et $t_a \leq t'_d$. Sa durée n'est pas la somme des durée des arcs le composant car il faut aussi compter le temps d'attente aux sommets. C'est la différence entre le temps d'arrivée du dernier arc et le temps de départ du premier.

Question 4. *Ecrire une fonction qui prend une liste de sommets en paramètre et un temps t, et qui renvoie la durée minimale d'un chemin parcourant les sommets donnés (dans l'ordre donné), en partant au temps t.*

Un graphe temporel est fortement connexe si, pour tout couple de sommets (u, v), il existe un chemin temporel de u à v.

Question 5. *Le graphe donné en figure* **??** *n'est pas fortement connexe. Pourquoi ? Et si on ajoute l'arc* $(D, A, 4, 6)$ *?*

Question 6. *Implémenter le graphe de la figure* **??** *en ajoutant l'arc* $(D, A, 4, 6)$.

A partir de maintenant, **on considère que les temps des arcs sont 24h-périodiques.** C'est à dire que si on note T le nombre de minutes dans $24h$ et si il existe un arc (u, v, t_d, t_a), alors l'arc (u, v, t_d+T, t_a+T) existe. Dans l'exemple de la figure **??**, après avoir pris l'arc de C vers B, on peut prendre celle de B vers C mais le lendemain.

Question 7. *Adapter la question* **??** *afin de prendre en compte cette périodicité.*

Question 8. *Expliquer sur son compte rendu comment déterminer si un graphe temporel $24h$-périodique est fortement connexe.*

2 Parsing de fichier

Ces graphes permettent en autres de représenter les réseaux de transports en commun. Considérons une ligne de bus par exemple. On note ses arrêts $u_1, \ldots, u_n$ dans l'ordre de parcours. Si un bus part de l'arrêt u_i au temps t_d et arrive à l'arrêt u_{i+i} au temps t_a, alors on a l'arc (u_i, u_{i+1}, t_d, t_a). Il y a donc potentiellement plusieurs arcs entre un même couple de sommet, chacune correspondant à un bus différent.

Afin de représenter les horaires de transports en communs, un format classique est le format *gtfs*. Dans ce format, les temps sont sous la forme *heure :minute :seconde*, ou plus précisément : "HH :MM :SS". Par exemple, pour 16h34 et 18 secondes, le fichier contient la chaîne de caractères "16 :34 :18".

Question 9. *Ecrire une fonction qui convertit une chaîne de caractère sous ce format en une variable de type* temps*. Le nombre de secondes sera ignoré.*

Le format *gtfs* est en fait un ensemble de fichiers. Le fichier *stops* contient les informations sur les sommets du graphe et le fichier *stop_times* contient celles sur les arcs. La première ligne de chaque fichier est l'entête. La colonne :

- *stop_id* est l'identifiant d'un sommet. Si le graphe a n sommets, les identifiants vont de 0 à $n-1$.
- *arrival_time* (resp. *departure_time* est l'heure d'arrivée (resp. de départ) à un sommet. Ce ne sont donc pas directement les arcs qui sont stockés.
- *trip_id* correspond à l'identifiant d'un trajet particulier. Par exemple, au début du fichier, on voit que le trajet d'identifiant *674330004 :0* part du départ à 9h00 et arrive à 9h09 au terminus en passant par 7 arrêts intermédiaires.

Afin de récupérer les informations de ces fichiers, on rappelle quelques fonctions :
- `FILE *fopen(const char *pathname, const char *mode)` : permet d'ouvrir un fichier. Par exemple, `fopen("monFichier","r")` permet d'ouvrir le fichier de nom *monFichier* en mode lecture et renvoie un FILE* (descripteur de fichier).
- `int fclose(FILE *stream)` : permet de fermer le fichier dont le descripteur est pointé par la variable *stream*.
- `int fscanf(FILE *stream, const char *format, ...)` : permet de récupérer une partie du contenu du fichier. Par exemple, `fscanf(f,"%s %d %lf\n",&var1,&var2,&var3)` permet de récupérer une ligne contenant un mot (séquence de caractères sans espace et directement suivi d'un espace) puis un entier, et enfin un double, en plaçant chacun dans var1, var2 et var3.
- `int atoi(const char *nptr)` : permet de convertir une chaîne de caractère en entier,
- `int atof(const char *nptr)` : permet de convertir une chaîne de caractère en flottant.

Question 10. *Ecrire une fonction qui génère le graphe correspondant aux fichiers* stops *et* stop_times *donnés dans le dossier* gtfs_saclay.

3 Plus courts chemins

Dans les graphes temporels, il existe plusieurs types de plus courts chemins :
- "arrivée le plus tôt" : pour un temps de départ fixé, on minimise la durée du chemin,
- "départ le plus tard" : pour un temps d'arrivée fixé, on minimise la durée du chemin,
- "le plus rapide" : dans un intervalle de temps donné, on minimise la durée du chemin,
- "le plus court" dans un intervalle de temps donné, on minimise le temps de trajet sans compter les temps de correspondances. Ce dernier ne sera pas étudié. Une solution classique consiste à appliquer l'algorithme de Dijkstra.

Question 11. *Proposer des exemples de la vie courante correspondant à chaque type de plus court chemin.*

3.1 Le plus tôt/tard

En pratique, un calculateur d'itinéraire va faire un pré-traitement sur le graphe afin de pouvoir ensuite répondre plus rapidement à de multiples requêtes d'utilisateurs. Par exemple, on pourrait en amont calculer une énorme matrice contenant les distances entre tous les couples de sommets. Ensuite, dès qu'un utilisateur veut aller d'un sommet u à un sommet v, il suffit de lire le coefficient d'indice u, v dans la matrice pour connaître la distance. Cette dernière méthode est déraisonnable en pratique : on ne s'autorisera qu'un pré-traitement en $O(n^2)$ en temps et en mémoire, avec n étant le nombre de sommets.

On donne un algorithme permettant de calculer un plus court chemin du premier type.

Algorithm 1: Plus court chemin de type "arrivée le plus tôt"

Input: $G = (S, A)$ un graphe temporel, $s \in S$ la source, $d \in S$ la destination et t_d un temps de départ

Output: t_a le temps d'arrivée minimal en d en partant de s au temps t_d.

```
1 T ← tableau de temps de taille Card(S)
2 T[s] ← t_d
3 for u ∈ S\{s} do
4     T[u] ← +∞
5 On parcourt les arcs dans l'ordre de départ croissant
6 foreach (u, v, t_δ, t_α) ∈ A, tel que t_δ ≥ t_d do
7     if t_α < T[v] et t_δ ≥ T[u] then
8         T[v] ← t_α
9 return T[d]
```

Question 12. *Implémenter cet algorithme. Voyez vous un lien avec l'algorithme de Dijkstra ?*

Question 13. *Il est inutile de considérer les arcs dont le temps de départ est postérieur à $T[d]$. Pourquoi ? Modifier votre code pour prendre efficacement cette remarque en compte.*

Question 14. *Adapter la fonction précédente pour obtenir calculer un plus court chemin de type "départ le plus tard".*

3.2 Le plus rapide

Question 15. *Ajouter un type* intervalle *permettant de stocker un couple de temps (départ,arrivée).*

Question 16. *Ecrire une fonction* `meilleur` *qui prend en paramètre deux couples c_1, c_2 de type* intervalle *et qui renvoie si c_1 est inclus dans c_2.*

On va stocker des ensembles d'intervalles avec des listes chaînées. On donne maintenant un algo-

rithme permettant de calculer un plus court chemin de type "le plus rapide".

Algorithm 2: Plus court chemin de type "le plus rapide"

Input: $G = (S, A)$ un graphe temporel, $s \in S$ la source, $d \in S$ la destination, $c = (t_d, t_a)$ un intervalle de temps dans lequel le trajet doit se faire
Output: t_a le temps d'arrivée minimal en d en partant de s au temps t_d.

$T \leftarrow$ tableau de temps de taille $Card(S)$
$T[s] \leftarrow t_d$
$L[s] \leftarrow$ Liste vide de couples d'intervalles
for $u \in S \backslash \{s\}$ **do**
    $T[u] \leftarrow +\infty$
    $L[u] \leftarrow$ Liste vide de couples d'intervalles
On parcourt les arcs dans l'ordre de départ croissant
foreach $(u, v, t_\delta, t_\alpha) \in A$, *tel que* $t_\delta \geq t_d$ *et* $t_\alpha \leq t_a$ **do**
    if $u = s$ *et* $(t_\delta, t_\delta) \notin L[s]$ **then**
        Inserer (t, t) dans $L[s]$
    Soit $(t'_\delta, t'_\alpha) \in L[u]$ tel que $t'_\alpha = \max\{y | (x, y) \in L[u], y \leq t_\alpha\}$
    Inserer (t'_δ, t_α) dans $L[v]$
    Retirer les intervalles $c \in L[v]$ tels qu'il existe $c' \in L[v]$ tel que c' soit strictement inclus dans c.
    if $t_\alpha - t'_\delta < T[v]$ **then**
        $T[v] \leftarrow t_\alpha - t'_\delta$
return $T[d]$

Question 17. *Implémenter cet algorithme.*

4 Temps de marche

Les réseaux de transports multi-modaux sont des réseaux incluant :
— des moyens de transports dits contraint : en arrivant à un sommet, il faut attendre le prochain transport pour repartir,
— des moyens de transports dits non contraint : on repart quand on veut (exemple : à pied, en voiture).

Question 18. *Afin d'être plus réaliste, on s'autorise à marcher entre deux stations à une distance inférieure à* $100m$*. Modifier le programme de la question* **??** *afin de permettre de passer de l'une à l'autre. On pourra utiliser la fonction* `distance` *du fichier* distance.c*. Elle prend en paramètre quatre doubles correspondant à un couple de coordonnées et renvoie la distance en kilomètres. Ajouter l'option* -lm *à la fin de votre commande permettant de compiler.*

On s'autorise maintenant à marcher aussi longtemps que nécessaire, c'est à dire qu'il est possible de marcher de n'importe quel arrêt à n'importe quel autre en ligne droite. On considère que la vitesse de marche est de $4km.s^{-1}$.

Question 19. *Les plus courts chemins de type "arrivée le plus tôt" sont-ils bien plus courts sur le graphe généré à la question* **??** *?*