



RAPPORT DE PROJET ALGO AVANCÉE

Projet N°17 : Cols des Alpes
LABARRE Axel - L3 Informatique
Université Paris 8

1) INTRODUCTION

Description du Projet

On voudra représenter les routes entre les cols de la chaîne de montagnes des Alpes dans un graphe implémenté avec les structures Lis et Vect, et déterminer le chemin permettant de passer d'un col à un autre en minimisant le dénivelé. Les dénivelés seront représentés par des poids positifs lorsqu'il y a un dénivelé positif, négatif sinon.

Ce projet du cours d'algorithmique avancée s'inscrit dans le domaine des graphes, c'est d'ailleurs pour ce type de structure de donnée que j'ai choisi ce projet et aussi par nostalgie d'avoir pu grimper le Col du Galibier en 2020 à vélo depuis Saint-Michel de Maurienne.

Pour ce projet, le but était de représenter par un graphe pondéré non orienté les différents cols des Alpes et leur altitude dans des structures Liste & Vecteur, et ainsi déterminer le chemin permettant de passer d'un col à un autre en minimisant le dénivelé.

Néanmoins, il faut faire la différence entre le dénivelé global et le dénivelé cumulé. Le dénivelé global est la différence d'altitude entre le point d'arrivée et le point de départ, tandis que le dénivelé cumulé est la somme de tous les dénivelés, séparés d'ailleurs en positif et négatifs.

Ce dernier est à mon sens plus pertinent à connaître.

2) Point de départ : Les données

Ce projet a donc démarré par une longue analyse des différents cols qui composent la chaîne de montagnes des Alpes afin d'en répertorier un certain nombre dans un tableau au format CSV `list_cols_alpes.csv`, constitué des cols et de leur altitude respectifs.

Pour approfondir ma recherche et concevoir un graphe avec davantage de connexités, j'ai écumé un bon nombre d'itinéraires cyclistes avec étapes, ce qui m'a permis de rajouter aussi ces fameuses villes ou lieux de départ dans un fichier CSV avec leur altitude également `list_ville_etapes.csv`.

L'étape suivante était de faire les liaisons directes entre les cols et les villes dans un 3e fichier CSV `list_access_cols.csv`

3) Lire et extraire les données

Une fois les données récoltées, il me fallait lire et extraire ces données afin de construire mon graphe et c'est donc à l'aide de la SDL que j'ai pu le construire. Pour ce faire j'ai ouvert et parcouru les fichiers en découplant à l'aide des enums et des méthodes de la `lib <string.h>` afin de découper et parser mes différentes données.

Pour construire mon graphe, j'ai utilisé une structure Graphe constituée du nombre de sommet et d'un vecteur de tête de liste de sommets adjacents chainés.

```
typedef struct graphe {
    int nbs;
    struct AdjList *table;
} Graphe;
```

Chaque liste de sommet adjacent représente les sommets accessibles directement depuis le sommet (Tête de liste) ayant pour identifiant l'index du vecteur `struct AdjList * table` de la structure Graphe.

```
/* Structure de liste d'adjacence */
typedef struct nodeAdjList {
    int id;                                // id et index dans la table du graphe
    char *nom;                             // Nom du sommet
    int altitude;                         // Altitude du sommet
    struct nodeAdjList *suivant;        // Sommet suivant accessible depuis celui-ci
} nodeAdjList;

typedef struct AdjList {
    int id;                                // id et index dans la table du graphe
    char *nom;                             // Nom du sommet
    int altitude;                         // Altitude du sommet
    struct nodeAdjList *tete;            // Tete de liste
} AdjList;
```

Chaque nœud de la liste possède alors le dénivelé ascendant ou descendant qu'il y a entre le sommet de départ (Tête de liste) et celui d'arrivée (nœud suivant)

4) Programmation modulaire

À ce stade, j'ai créé un Makefile afin d'organiser mon projet et m'octroyer la possibilité de tester séparément chaque portion de code afin de corriger quelques fonctions et avant d'envisager la suite. (Voir dossier Tests)

5) Algorithme de Floyd-Warshall

L'algorithme que j'ai décidé d'implémenter pour ce problème s'appelle l'algorithme de Floyd-Warshall, qui porte le nom de deux personnes, mais qui compte en réalité au moins quatre prétendants différents à ce titre. Cet algorithme commence avec la matrice d'adjacence d'un graphe, qui va être progressivement transformée en matrice des plus courts chemins.

Cet algo a été choisi pour son efficacité à traiter les poids négatifs dans un graphe. Bien entendu il a fallu donc convertir notre liste d'adjacents précédente en matrice d'adjacence.

L'idée de l'algorithme de Floyd envisage pour chaque arc (i, j) si un passage par un sommet k peut raccourcir la distance entre i et j dans le cas où les chemins (i, k) et (k, j) existent, ces chemins n'étant pas forcément élémentaires, mais pouvant se constituer d'un chemin contenant des numéros de sommets inférieurs à k .

6) Difficulté

Comme évoqué précédemment, le résultat le plus intéressant étant le dénivelé cumulé positif, l'algorithme de Floyd-Warshall standard n'étant pas suffisant, il va falloir lui apporter quelques modifications avec l'ajout d'une fonction d'accumulateur de dénivelé `elevation_gain` qui va permettre de stocker le dénivelé cumulé positif entre chaque sommet.

À chaque itération de l'algorithme , je vais comparer le dénivelé cumulé de l'itinéraire actuel avec celui du sommet intermédiaire k et garder le moins élevé.

7) Conclusion

Mon projet n'étant pas terminé notamment à cause des contraintes de temps (Alternance).

À cet instant, l'algorithme de Floyd-Warshall implémenté est standard, néanmoins un résumé de l'itinéraire emprunté est visible grâce à la matrice des prédecesseurs.

Mon implémentation de l'algorithme de Floyd-Warshall qui minimise le dénivelé cumulé positif n'est pas encore concluante dans les tests de terrain. Il s'avère que je me suis rendu compte de l'insuffisance de Floyd-Warshall standard trop tardivement, néanmoins j'espère pouvoir réussir ma modification d'ici là au jour de la présentation, ma piste de réflexion concernant la modification avec l'accumulateur de dénivelé comparé à chaque itération me semble bonne.

Enfin cet algorithme pourrait être utile pour planifier des itinéraires de randonnée ou de vélo en minimisant l'effort physique requis.

Références

- Définition de Dénivelé global et cumulé : [https://www.randonner-malin.com/le-denivele-en-randonnee-simplement-explique/#:~:text=Le%20d%C3%A9nivel%C3%A9%20est%20la%20diff%C3%A9rence,m\(1500%20-%201000\).](https://www.randonner-malin.com/le-denivele-en-randonnee-simplement-explique/#:~:text=Le%20d%C3%A9nivel%C3%A9%20est%20la%20diff%C3%A9rence,m(1500%20-%201000).)
- **Algorithme sur les graphes : Tous les plus courts chemins - Floyd-Warshall** <https://www.youtube.com/watch?v=0skKIMB2bkk>
- <https://jj.up8.site/>
- <https://bdupont.up8.site/index.html>
- <https://www.geeksforgeeks.org/graph-and-its-representations/>
- Livre : Algorithmes et structures de données génériques Cours et exercices corrigés en langage C (Divay M.)
- Un bel itinéraire de Thonon-Les-Bains à Nice [Google Maps](#)
- Wikipedia

Utilisation

La présence d'un Makefile vous permettra de compiler facilement ce projet.

src = Sommet source

dst = Sommet de destination

Quelques fonctions d'affichages sont présentes, n'y faites pas trop attention.

Compiler le projet :

```
> make
```

Exécuter le projet :

```
> ./graph_alpes
```

Nettoyer le répertoire du projet :

```
› make clean
```