

# Rapport - Projet théorie des graphes

Younes Ben Yamna - Malek Zemni

2 mai 2016

## 1 Introduction

L'idée principale de ce projet est de trouver le plus court chemin entre deux sommets d'un graphe orienté.

Le graphe qui nous a été fourni - la carte de l'Alpe d'Huez - correspond à un graphe orienté muni d'une fonction de pondération positive, c'est à dire que tous les arcs ont un poids positif. Ce graphe n'est pas un graphe simple à première vue (existence de plusieurs arcs entre les même sommets), mais on pourra voir dans les sections suivantes qu'on pourra le considérer ainsi.

## 2 Graphe

La première étape de ce projet était d'analyser la carte de la station de ski fournie et d'en tirer le graphe correspondant. On a donc considéré les pistes de ski et les remontées mécaniques comme des arcs, et les points d'intersection, de départ et d'arrivée de ces pistes comme des sommets.

### 2.1 Sommets du graphe

#### Considération des sommets :

On considère comme sommets du graphe, les points de départ et d'arrivée des pistes et les points où se croisent plusieurs de ces pistes. La carte fournie a été simplifiée puisque les sommets correspondant à des points où se croisent plusieurs pistes sont en fait des zones (et non des points) où se croisent ces pistes.

Les noms des sommets sont quant à eux choisis par rapport au nom de la zone où se trouve le sommet. Si la zone ne porte pas de nom, le nom du sommet sera choisi par rapport au nom de la piste ou de la remontée la plus proche.

#### Indices des sommets :

Chaque sommet a été repéré par un indice allant de 0 à 44 : **on a donc 45 sommets au total**. Voici la liste complète des sommets, ainsi que leurs indices respectifs :

0 PIC BLANC  
1 GROTTE DE GLACE  
2 SOMMET 3060  
3 SARENNE BASSE  
4 CLOCHER DE MACLE  
5 LAC BLANC  
6 LIEVRE BLANC  
7 PLAT DES MARMOTTES  
8 MINE DE L'HERPIE  
9 SOMMET 2100  
10 SOMMET DES VACHETTES  
11 SIGNAL DE L'HOMME  
12 L'ALPETTE  
13 COL DU COUARD  
14 CASCADE  
15 CLOS GIRAUD  
16 MONFRAIS  
17 SIGNAL  
18 RIFNEL EXPRESS  
19 CHALVET  
20 AURIS EXPRESS  
21 FONTFROIDE  
22 LOUVETS  
23 POUTRAN  
24 CHAMP CLOTURE  
25 STADE  
26 SCHUSS  
27 ALPE D'HUEZ  
28 GRANDE SURE  
29 ECLOSE  
30 SURES  
31 COL  
32 AURIS EN OISANS  
33 LA VILLETTE  
34 VAUJANY  
35 L'EVERSIN D'OZ  
36 OZ EN OISANS  
37 PETIT PRINCE  
38 VILLAGE  
39 MARONNE  
40 VILLARD RECLAS  
41 HUEZ  
42 DOME DES PETITES ROUSSES  
43 ALPAURIS  
44 L'ALPETTE BASSE

sommets.txt

## 2.2 Arcs du graphe

### Considération des arcs :

Les arcs du graphe correspondent bien évidemment aux pistes de ski. On a alors deux types d'arcs : des *descentes* et des *remontées*.

Une simplification du graphe a été effectuée : s'il existe plusieurs arcs de même extrémité de départ et d'arrivée, on ne retient que l'arc de poids le plus faible.

La raison est que pour calculer un plus court chemin, l'arc de poids faible est le seul qui va être pris en compte, peu importe le nombre des autres arcs liant ces mêmes sommets.

### Couleur et poids des arcs :

Les différents types d'arcs ont chacun leur propre couleur qui a été repérée par un indice. Cette coloration influe directement dans le calcul du poids de l'arc pour le **cas d'un skieur débutant** :

- 0 - vert : descente de poids normal
- 1 - bleu : descente dont le poids est multiplié par 2
- 2 - rouge : descente dont le poids est multiplié par 3
- 3 - noir : descente dont le poids est multiplié par 4
- 4 - noir : remontée dont le poids dépend de son type (télésiège, téléski, funitel, etc.)

Voici la fonction qui permet la conversion du temps de parcours de l'arc et de sa couleur en poids (l'entier experience vaut 0 pour un expert et 1 pour un débutant) :

```
int calculPoids(char* nomArc, int couleur, int temps, int
    experience)
{ //Convertit la couleur et le temps de l'arc en poids en fonction
  de l'experience du skieur
  //Couleurs : 0 - Vert, 1 - Bleu, 2 - Rouge, 3 - Noir
  double typeRemontee = 1; //nombre par lequel on diminue le poids
    de la remontée (varie en fonction de le type de remontée)

  if (couleur==0)
    return temps;
  if (couleur==1)
    return (experience*temps + temps);
  if (couleur==2)
    return (experience*2*temps + temps);
  if (couleur==3)
    return (experience*3*temps + temps);
  if (couleur==4)
  { //Les remontees sont des arcs de couleur 4
    //Plus ce type de remontée est rapide plus son poids va
    diminuer
    if (strstr(nomArc, "TELEPHERIQUE") != NULL)
      typeRemontee = 0.3;
    if (strstr(nomArc, "FUNITEL") != NULL)
      typeRemontee = 0.5;
    if (strstr(nomArc, "DMC") != NULL)
      typeRemontee = 0.6;
    if (strstr(nomArc, "TELECABINE") != NULL)
      typeRemontee = 0.7;
    if (strstr(nomArc, "TELEMIXSTE") != NULL)
      typeRemontee = 0.8;
    if (strstr(nomArc, "TELESIEGEBULLE") != NULL)
      typeRemontee = 0.85;
    if (strstr(nomArc, "TELESIEGE") != NULL)
      typeRemontee = 0.9;
    return (int)(temps*typeRemontee);
  }
}
```

```

|| return 1000;
|| }

```

graphe.c

### Indices des arcs :

Chaque arc dans le graphe est repéré par un indice.

Les descentes sont repérées par des indices allant de 0 à 56 : on a donc 57 descentes.

Les remontées sont repérées par des indices allant de 0 à 42 : on a donc 43 remontées.

Remarque : on ajoute 100 à chaque indice de remontée pour les différencier des descentes.

On a alors 57 descentes et 43 remontées, ce qui fait **100 arcs au total**.

Voici la liste des arcs, ainsi que leurs indices respectifs :

```

Liste des descentes :
0 descente SARENNE HAUTE / CHATEAU NOIR
1 descente GLACIER
2 descente BRECHE
3 descente TUNNEL
4 descente CRISTAILLERE
5 descente SARENNE BASSE
6 descente DOME
7 descente LAC BLANC
8 descente PROMONTOIRE
9 descente COMBE CHARBONNIERE
10 descente ROUSSES
11 descente CHAMOIS
12 descente ANCOLIES
13 descente CANYON
14 descente CAMPANULES
15 descente COL DE CLUY
16 descente VERNETTES
17 descente CHALVET-ALPAURIS
18 descente ETERLOUS
19 descente FONTFROIDE
20 descente PRE-ROND
21 descente COL 1
22 descente LES FARCIS
23 descente GENTIANES
24 descente COL 2
25 descente CORNICHE
26 descente FONTFROIDE-LOUVETS
27 descente BARTAVELLES
28 descente POUTRAN
29 descente JEUX
30 descente AGNEAUX
31 descente LES BERGES
32 descente LOUP BLANC
33 descente POUSSINS
34 descente ANEMONES
35 descente SIGNAL-STADE
36 descente SIGNAL

```

37 descente SCHUSS-ECLOSE  
 38 descente SCHUSS-GRANDE SURE  
 39 descente ECLOSE-GRANDE SURE  
 40 descente VILLAGE 1  
 41 descente HUEZ  
 42 descente VILLAGE 2  
 43 descente PETIT PRINCE  
 44 descente LA FORET  
 45 descente L'OLMET  
 46 descente CHAMPCLOTURY  
 47 descente ALPETTE  
 48 descente LUTINS  
 49 descente CARRELET  
 50 descente CHALETS  
 51 descente LA FARE  
 52 descente CASCADE  
 53 descente ETOURNEAUX  
 54 descente EDELWEISS  
 55 descente VAUJANIATE  
 56 descente VILLARD  
     Liste des remontées :  
 100 TELEPHERIQUE PIC BLANC  
 101 TELESIEGE GLACIER  
 102 TELESIEGE HERPIE  
 103 FUNITEL MARMOTTES III  
 104 TELESIEGE LAC BLANC  
 105 TELEPHERIQUE ALPETTE-ROUSSES  
 106 DMC 2EME TRONCON  
 107 TELESIEGE LIEVRE BLANC  
 108 TELECAABINE MARMOTTES II  
 109 TELECAABINE POUTRAN II  
 110 DMC 1ER TRONCON  
 111 TELESIEGE ROMAINS  
 112 TELESIEGEBULLE MARMOTTES I  
 113 TELEMIXTE RIFNEL EXPRESS  
 114 TELESIEGE SIGNAL  
 115 TELESKI STADE  
 116 TELESKI ECLOSE-SCHUSS  
 117 TELESIEGE GRANDE SURE  
 118 TELECAABINE TELEVILLAGE  
 119 TELESIEGE BERGERS  
 120 TELESKI PETIT PRINCE  
 121 TELESIEGE TSD LE VILLARAIS  
 122 TELESIEGE ALPAURIS-ALPE D'HUEZ  
 123 TELESIEGE CHALVET  
 124 TELESIEGE FONTFROIDE  
 125 TELESIEGE LOUVETS  
 126 TELESIEGE ALPAURIS  
 127 TELESIEGE AURIS EXPRESS  
 128 TELESKI COL  
 129 TELESIEGE SURES  
 130 TELESIEGE MARONNE  
 131 TELECAABINE L'ALPETTE  
 132 TELESKI L'ALPETTE  
 133 TELECAABINE POUTRAN I  
 134 TELESKI HAMP CLOTURE  
 135 TELEPHERIQUE VAUJANY-ALPETTE

```

136 TELESIEGE CLOS GIRAUD
137 TELESKI MONTFRAIS
138 TELESIEGE MONTFRAIS
139 TELESIEGE VALLONNET
140 TELECABINE VILLETTE-MONTFRAIS
141 TELECABINE VAUJANY-VILLETTE
142 TELECABINE VAUJANY-ENVERVIN

```

arcs.txt

## 2.3 Fichier du graphe

Un fois les sommets et les arcs considérés, on a pu constituer notre graphe et le rentrer dans un fichier texte en respectant ce format :

- le 1er entier est l'indice du sommet de depart
- le 2ème entier est l'indice du sommet d'arrivee
- le 3ème entier est l'indice de l'arc liant le sommet de départ au sommet d'arrivée
- le 4ème entier est le temps de parcours (mesuré directement sur la carte) de cet arc
- le 5ème entier est la couleur de cet arc

Voici un petit extrait de ce fichier :

```

|| 2 3 4 1 4
|| 3 19 5 3 15
|| 42 1 6 2 3

```

graphe.txt

## 3 Structure

La structure utilisée pour représenter le graphe est une matrice d'adjacence. Cette matrice est en fait un tableau à deux dimensions dont la première dimension (les lignes) correspond à l'indice du sommet de départ, et la deuxième dimension (les colonnes) correspond à l'indice du sommet d'arrivée.

Les éléments de ce tableau ont un type personnalisé **Arc** qui stocke toutes les informations relatives à un arc du graphe :

```

//Structure principale representant un arc du graphe
typedef struct
{
    char* nom; //noms de l'arc de poids le plus faible entre le
                sommet de depart et le sommet d'arrivee
    char* depart; //nom du sommet de depart
    char* arrivee; //nom du sommet d'arrivee
    int couleur; //couleur de l'arc, utilisée pour colorier l'arc
                lors de l'affichage graphique
    int poids; //poids de l'arc de poids le plus faible entre le
                sommet de depart et le sommet d'arrivee
} Arc;

```

definitions.h

Le graphe a été déclaré comme ceci (V étant le nombre de sommets) :

```

|| Arc G[V][V];      //matrice d'adjacence représentant les arcs du
||     graphe
||
||                                     definitions.h

```

Remarque : ce tableau a été déclaré comme variable globale par soucis de clareté du code : on a préféré le garder ainsi puisqu'à la fin, cce projet est un projet d'algorithmique et non de programmation.

#### Lecture du fichier du graphe :

Une fois la structure du graphe bien définie, on a pu créer une fonction pour le remplissage de la variable G (le graphe) à partir du fichier texte :

```

|| void lectureGraphe(char* nomFichier, Arc G[V][V], int experience)
|| { //Lit le graphe à partir du fichier fourni
||   FILE* F = fopen(nomFichier,"r"); //doit etre deja present, sinon
||   NULL
||
||   if (F == NULL){
||     fprintf(stderr,"Erreur : fichier du graphe introuvable\n");
||     exit(EXIT_FAILURE);
||   }
||
||   int k;
||
||   initialise(G);
||
||   for (k = 0; k < E; k++) //boucle qui parcourt les lignes du
||     fichier : E lignes <=> E arcs
||   {
||     int i, j, indiceArc, couleur, temps;
||     //i : indiceSommetDepart, j : indiceSommetArrivee
||
||     fscanf(F,"%d %d %d %d %d",&i, &j, &indiceArc, &couleur, &temps)
||     ;
||
||     G[i][j].nom = nomArc(indiceArc);
||     G[i][j].depart = nomSommet(i);
||     G[i][j].arrivee = nomSommet(j);
||     G[i][j].couleur = couleur;
||     G[i][j].poids = calculPoids(G[i][j].nom, couleur, temps,
||     experience);
||   }
||   fclose(F);
|| }

```

graphe.c

## 4 Recherche du plus court chemin

On va maintenant s'intéresser à la recherche d'un plus court chemin entre un sommet de départ et un sommet d'arrivée.

La fonction principale qui permet de réaliser cette manipulation est la fonction **dijkstra** qui est une application directe de l'algorithme de meme nom.

Cet algorithme a été préféré à d'autres pour sa facilité d'implémentation, d'autant plus que le graphe fourni en entrée remplit les conditions (arcs de poids positifs).

La fonction dijkstra va se charger de remplir un tableau de pères **pere[V]** préalablement initialisé à -1 :  $pere[x] = y$  veut dire que le sommet d'indice x a pour père le sommet d'indice y. Elle va donc calculer les plus courts chemins à partir d'un sommet racine vers tous les sommets du graphe qui lui sont accessibles (construction d'une arborescence des plus courts chemins).

Voici la fonction commentée étape par étape :

```
void dijkstra(int pere[V], int sommetDepart)
{ //Modifie le tableau pere pour donner le tableau des plus courts
  chemins pour un sommet de départ donné
  //Structures et initialisations
  int sommetsTraites[V] = {0}; //quand le sommet i sera traité,
    sommetsTraites[i] vaudra 1
    //tous les sommets du graphe seront traités si ce
    tableau ne contient que des 1
  int plusCourteDistance[V]; //plusCourteDistance[i] vaut la
    plus courte distance entre le sommet de départ et le sommet i
  initTableauPere(pere); //pere[i] vaut le père du sommet i,
    -1 veut dire pas de père

  sommetsTraites[sommetDepart] = 1; //le sommet de départ est trait
    é

  int i;
  //Cette boucle initialise le tableau plusCourteDistance[]
  //Pour tout sommet du graphe
  for (i = 0; i < V; i++)
  {
    //Si c'est un successeur du sommet de départ, sa
    plusCourteDistance au sommet de départ vaut le poids de l'arc
    if (G[sommetDepart][i].poids != 1000)
    {
      plusCourteDistance[i] = G[sommetDepart][i].poids;
      pere[i] = sommetDepart;
    }
    //Sinon sa plusCourteDistance au sommet de départ vaut 1000 (
    infinie)
    else
      plusCourteDistance[i] = 1000;
  }

  plusCourteDistance[sommetDepart] = 0; //le sommet de départ n'a
    pas une distance infinie à lui-meme

  //Boucle principale
  //Tant que tous les sommets n'ont pas été traités
  while (!tousLesSommetsTraites(sommetsTraites))
  {
    //On cherche le prochain sommet à traiter : celui dont la
    plusCourteDistance au sommet de départ est minimale
  }
}
```



```

int min = 1000, aTraiter;
//Pour tout sommet
for (i = 0; i < V; i++)
{
    //Si le sommet n'est pas traité
    if (sommetsTraites[i] == 0)
    {
        //Si sa plusCourteDistance au sommet de départ est infé
        rieur au minimum
        if (plusCourteDistance[i] <= min)
        {
            aTraiter = i;
            min = plusCourteDistance[i];
        }
    }
}

//Traitement du sommet aTraiter
sommetsTraites[aTraiter] = 1;
for (i = 0; i < V; i++)
{
    //Pour chaque successeur du sommet aTraiter
    if (G[aTraiter][i].poids != 1000)
    {
        //Si la plus courteDistance de ce sommet au sommet de dé
        part est plus grande que celle passant par le sommet aTraiter
        if (plusCourteDistance[i] >= plusCourteDistance[aTraiter] +
        G[aTraiter][i].poids)
        { //On met cette distance à jour (la plus petite), et
        surtout on renseigne le père
            plusCourteDistance[i] = plusCourteDistance[aTraiter] + G[
            aTraiter][i].poids;
            pere[i] = aTraiter;
        }
    }
}
}
}

```

dijkstra.c

### Plus court chemin :

Comme on l'a dit précédemment, la fonction dijkstra va calculer tous les plus courts chemins à partir d'un sommet racine. Cependant, notre objectif ici est de rechercher le plus court chemin entre un sommet de départ et un sommet d'arrivée précis.

L'idée est d'exploiter le tableau pere[] en le parcourant à partir du sommet d'arrivée et en remontant dans la hiérarchie des pères jusqu'à retrouver le sommet de départ. Cette séquence sera ensuite stockée dans une liste chaînée (plus performante ici qu'un tableau car on ne connaît pas la taille du chemin) pour être finalement affichée.

Voici la fonction qui s'occupe de cette manipulation (elle comporte quelques éléments de l'affichage graphique, n'y portez pas attention) :

```

void plusCourtChemin(Arc G[V][V], int sommetDepart, int
    sommetArrivee)
{ //Calcule le plus court chemin entre deux sommets et affiche ce
    chemin
    POINT HG = {880,170};

    if (sommetDepart == sommetArrivee)
    {
        printf("Vous etes deja a destination '%s'\n\n",nomSommet(
            sommetDepart));
        afficheTexte("A DESTINATION",HG, blanc);
        return;
    }

    int pere[V]; //Tableau des pères modifié par dijkstra et va etre
        exploité pour afficher le plus court chemin
    Liste chemin = NULL; //Liste qui va stocker le plus court chemin
    int tempsTotal = 0;

    dijkstra(pere,sommetDepart); //Modifie le tableau pere

    //On exploite le tableau pere pour remonter du sommetArrivee au
        sommetDepart
    int depart, arrivee = sommetArrivee;

    chemin = ajoutDebut(chemin,arrivee); //On ajoute le sommet d'
        arrivée actuel à la fin de la liste

    do {
        depart = pere[arrivee]; //Le sommet de départ actuel est le
            pere du sommet d'arrivee actuel
        if (depart == -1)
        {
            printf("Il n'existe pas de chemin entre '%s' et '%s'\n\n",
                nomSommet(sommetDepart),nomSommet(sommetArrivee));
            afficheTexte("PAS DE CHEMIN",HG, blanc);
            return;
        }
        drawArc(depart, arrivee);
        tempsTotal += G[depart][arrivee].poids;
        chemin = ajoutDebut(chemin,depart);

        arrivee = depart; //Le nouveau sommet d'arrivee est l'
            ancien sommet de départ
    } while (depart != sommetDepart);

    //Affichage du chemin et du temps (en console et partiellement en
        graphique)
    draw_fill_circle(points[sommetDepart],5,rouge);
    printListeChemin(chemin,G);
    printf("Vous etes arrivé en %d minutes\n",tempsTotal);
    char ch[3];
    sprintf(ch,"%d MINUTES",tempsTotal);
    afficheTexte(ch,HG, blanc);
    affiche_all();
}

```

fonctions.c

## 5 Conclusion

En guise de conclusion, on peut dire que ce projet nous a permis d'avoir un très bon aperçu sur les champs d'application des algorithmes liés à la théorie des graphes et nous a donné l'occasion de plonger au coeur de l'action, surtout pour un problème qui nous est très familier aujourd'hui.

Le code source de ce projet est "gratuitement" disponible sur github : [\*\*https ://github.com/Mzem/Ski\*\*](https://github.com/Mzem/Ski)