Rapport - Projet théorie des graphes

ABDELMOUMENE Djahid
&
SEDDAR Hadjer

May 3, 2018

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 des pistes de serre chevalier - 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 mme sommets), mais on pourra voir dans les sections suivantes qu'on pourra le considérer ainsi.
Pour le langage, on a choisi javascript (avec html et css) parceque c'est plus facile a gérer l'affichage et les resultats en text (chemin avec les temps) avec les elements DOM.

2 Execution

Pour executer le program, il suffit d'ouvrir le fichier 'index.html' avec le navigateur firefox.

3 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.

3.1 le graphe

le graphe est stocké comme un liste d'adjacence, donc le graphe est un tableau des objets des sommets.

3.2 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 les pistes se croisent. La carte fournie a été simplifée 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 sommets n'ont pas des noms, mais il sont défini par leur indice dans le tableau de graph, et avec leur coordonnées (cartésiens) dans l'image des pistes (2362x1013).

Indices des sommets :

Chaque sommet a été repéré par un indice allant de 0 à 147 : **on a donc 148 sommets au total**.

Coordonnées des sommets :

Tout sommet à un coordonnèe cartésien dans l'image des pistes (2362x1013), et c'est représenté comme un tableau de taille 2 ou le premier element est le x et le deuxième est le y.

Les voisins de sommet Tout les sommet ont un objet *tableaudehachage* où chaque voisin est representé par son indice dans le tableau de graphe, et la valeur est l'objet de l'arc. Exemple d'un objet des voisins:

extrait de data.json

```
"voisins": {
  "116": {
    "coords": [
      [
        527,
        387
      ],
      [
        486,
        393
      ],
      [
        469,
        398
      ],
      [
        451,
        399
      ],
      [
```

```
            437,
            404
          ],
          [
            409,
            399
          ]
        ],
        "len": 120,
        "nom": "PREVOUX",
        "couleur": "V"
      },
      "123": {
        "coords": [
          [
            531,
            386
          ],
          [
            539,
            390
          ],
          [
            524,
            397
          ],
          [
            521,
            403
          ]
        ],
        "len": 32,
        "nom": "PREVOUX2",
        "couleur": "R"
      }
    }
```


3.3 Arcs du graphe

Considération des arcs :

Les arcs du graphe correspondent bien évidemment aux pistes de ski. On a alors deux types d'acrs : des *descentes* et des *remontées*.
Un 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 etre pris en compte, peu importe le nombre des autres arcs liant ces mmes sommets.

Infomations stockées dans les arcs :

Le nom: Chaque arc à un nom unique, vu que la majorité des arcs ont des noms, si un arc n'as pas un nom il prend le nom de l'arc le plus proche mais avec un indice ajouté. eg: arc ,arc1, arc2..

Les coordonnées Tout les arcs ont un tableau des points dans l'image (2362x1013), et ces coordonnées sont utilisées pour dessiner les arcs, où on connect chaque deux points consécutifs avec un ligne.

La longueur de l'arc Chaque arc a un longueur, qui est la somme des distance de chaque deux points consécutifs dans le tableau des coordonnées de l'arcs.

Couleur et poids des arcs les arcs ont un qui represente le couleur ou le type de l'arc, Les différents types d'arcs ont chacun leur propre couleur qui a été repérée par un chaine des caractéres. voici la liste des couleurs et types possibles avec le poids:

Les pistes (la couleur, poids un skieur debutant, poids pour un skieur expert):

- "V": vert, longueur*1.05/10, longueur*1/10
- "B": blue, longueur*1.4/10 , longueur*1.1/10
- "R": rouge, longueur*2.2/10 , longueur*1.2/10
- "N": noir, longueur*3/10 , longueur*1.3/10

Les remontées (le type, le poids):

- "TELECABINE": telecabine, longueur*0.5/10
- "OEUF": oeuf, longueur*0.75/10
- "TELESIEGE": telesiège, longueur*0.9/10
- "TELESKI": teleski, longueur*1/10

Voici la fonction qui permet la conversion du temps de parcours de l'arc et de sa couleur en poids:

dijkstra.js

```
function calculateArcCost(v){
        switch(v.couleur){
```

```
                case"TELESKI": return 1*v.len/10;
                case"TELESIEGE": return 0.9*v.len/10;
                case"OEUF": return 0.75*v.len/10;
                case"TELECABINE": return 0.5*v.len/10;
        }
        if(sel.value === 'Expert'){
                switch(v.couleur){
                        case"V": return 1*v.len/10;
                        case"B": return 1.1*v.len/10;
                        case"R": return 1.2*v.len/10;
                        case"N": return 1.3*v.len/10;
                }
        } else {
                switch(v.couleur){
                        case"V": return 1.05*v.len/10;
                        case"B": return 1.4*v.len/10;
                        case"R": return 2.2*v.len/10;
                        case"N": return 3*v.len/10;
                }
        }
}
```

3.4 le fichier d'entrée

Le fichier d'entrée est en format JSON, où il existe les objets (tableau de hachage) representé par des { }, et des tableaux representé par des [ ].

cette format nous permet de structurer l'objet de graphe dans le fichier de l'information.

4 L'algorithme

On va maintentant s'interésser 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** 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 arboresence des plus courts chemins).
Voici la fonction commentée étape par étape :

dijkstra.js

```
function dijkstra(graph, dep){
        var costs = [];
        var pere = [];
        var visited = [];
        for(var i=0; i<graph.length; i++){
                costs.push(Number.POSITIVE_INFINITY);
                pere.push(-1);
        }
        costs[dep] = 0;
        while(visited.length < graph.length) {
                // pour choisir un element non visit
                var u = 0;
                while(visited.includes(u))
                        u++;
                // on cherche le sommet avec le cout le plus bas
                for(var i=0; i<graph.length; i++){
                        if(costs[u] > costs[i] && !visited.includes(i))
                                u = i;
                }
                // on marque u comme vu
                visited.push(u);
                // on regarde tout les sommets voisins de u
                for(let v of Object.keys(graph[u].voisins)){
                        v = int(v);
                        // si v n'etait pas visit
                        if(!visited.includes(v)){
                                // calcul de cout d'arc
                                var arcCost = calculateArcCost(graph[u].voisins[
                                // le cout d'arc + le cout jusqua la sommet cour
                                var alt = costs[u] + arcCost;
                                // si le nouveau cout est plus petit que l'ancie
                                if(alt < costs[v]){
                                        // on prend le nouveau cout
                                        costs[v] = alt;
                                        // et on met u comme pere de v
                                        pere[v] = u;
                                }
                        }
                }
        }
        return pere;
}
```

Plus court chemin :

Comme on l'a dit précedemment, 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.

5 Conclusion

En conclusion, on peut dire que ce projet nous a permis d'avoir un très bon aperçu sur les champs d'application des algorithmes lis la thorie des graphes, surtout pour un problme qui nous est trs familier aujourd'hui.

Le code source de ce projet est disponible sur github :
`https://www.github.com/djahiddj13/PROJET_GRAPH`