IN104

Nastassia Bonetti et Agathe Pascal

23 mai 2023

Contents

1 Introduction

Nous disposons d'une carte avec l'ensemble des pistes d'une station de ski et un entier qui représente le plaisir généré lorsqu'on dévale cette piste. Ce chiffre peut être négatif si cette piste est ennuyeuse.

Les croisements des pistes sont numérotés et on part de celui qui a le numéro 0.

Notre objectif est de déterminer le meilleur itinéraire possible pour la journée. En revanche, rien ne nous interdit de passer plusieurs fois sur la même piste et donc de ressentir un plaisir infini. Nous pouvons aussi décider que la meilleure solution est de ne pas skier.

Mais alors quel itinéraire devons-nous emprunter?

2 Notre programme

Notre première tentative de résolution se trouve dans la branche version 1 de notre git. Nous avons essayé de résoudre le projet avec nos propres fonctions. C'était notre première tentative de résolution. Les autres méthodes ont été placées dans d'autres branches de notre git.

Pour résoudre ce problème, nous devons d'abord récupérer les données qui ont été mises dans un fichier.txt.
Pour ce faire, nous avons d'abord pensé à utiliser des structures et à retourner une liste de structures contenant le départ, l'arrivée et le plaisir associé. Cependant, avec cette méthode, les étapes qui suivent devenaient trop complexes, surtout avec l'utilisation de pointeurs. Ainsi, nous avons repris cette fonction et avons simplement créé une liste (affichée sous forme de matrice adjacente) qui contient les plaisirs associés au sommet i et au sommet j (ième ligne et jième colonne).

Une fois cela fait, nous commençons par trouver si notre graphe pondéré possède des cycles positifs. Si c'est le cas, le programme retourne "SKY IS THE LIMIT". Sinon, nous continuons le programme et affichons le plaisir maximum que nous pouvons atteindre.

Avec la version 1 la recherche de cycle et le parcours du graphe se font ainsi:

Dans un premier temps nous cherchons à savoir si le cycle possède un cycle à valeurs positives. Pour ce faire nous avons codé un programme qui permet de parcourir le graphe en profondeur de manière récursive.
Nous avons tenté que cette fonction retourne une liste de tous les cycles présents sous forme de structure (cycle et taille du cycle).
L'objectif était de calculer le plaisir de chacun de ces cycles grâce à une fonction annexe qui le permettait.

Cette idée a été aussi mise en place pour le parcours du graphe. Nous voulions écrire un programme qui après la détection des cycles puisse parcourir tout le graphe et déterminer tous les chemins existants entre le sommet 0 et les autres sommets.
On aurait alors une liste de chemin et il aurait fallu simplement appliquer encore une fois la fonction qui calcule le plaisir et déterminer le maximum.

Cependant, cette méthode se révèle particulièrement complexe c'est pourquoi nous avons abandonné cette idée pour la simplifier.

Nous avons donc repris les bases de cette version pour en faire une nouvelle version, la version 3.

Dans cette version on ne cherche qu'à savoir si le graphe possède un cycle positif en mettant directement un compteur de plaisir dans le parcours du graphe. De même, pour le parcours du graphe si celui-ci ne présente pas de cycle. On parcourt le graphe en profondeur de manière récursive et on calcul le plaisir maximal directement lors du parcours.

3 Bellman Ford

Trouver le chemin le plus long peut être réalisé en adaptant l'algorithme de Bellman Ford. En effet, cet algorithme recherche le chemin le plus court et nous pouvons le modifier pour qu'il trouve le plus long (maximum de plaisir) dans notre problème.
En l'adaptant, il résoudra le problème des plus longs chemins avec origine unique, dans le cas le plus général où les poids des arcs peuvent avoir des valeurs négatives. Étant donné un graphe orienté pondéré, de fonction de poids w, et une origine s, l'algorithme retournera "SKY IS THE LIMIT" s'il existe un cycle dans le graphe. S'il n'en existe pas, l'algorithme donnera le plaisir maximal qu'un skieur peut avoir en parcourant la station.

Pour changer Bellman Ford afin qu'il trouve le plus long chemin et non le plus court, nous avons modifié l'initialisation des valeurs de plaisir à moins l'infini au lieu de plus l'infini. De plus, nous avons modifié les conditions pour remplir les plaisirs. En effet, nous avons rempli ces conditions de la manière suivante :
Si la distance actuelle du sommet de départ au sommet d'arrivée est inférieure à la somme de la distance actuelle du sommet de départ au sommet de départ + le poids de l'arête entre ces deux sommets, alors nous mettons à jour la distance du sommet d'arrivée avec cette nouvelle valeur.
De même, nous avons changé les conditions pour trouver des cycles :
Après avoir effectué les N-1 itérations, nous effectuons une dernière itération de relaxation pour détecter les cycles négatifs. Si une distance peut encore être

mise à jour lors de cette dernière itération, cela signifie qu'il existe un cycle négatif dans le graphe. Dans ce cas, il n'y a pas de plus long chemin dans le graphe, car un cycle négatif signifie qu'il est possible de parcourir ce cycle indéfiniment, en augmentant la distance à chaque fois.

Pour pouvoir l'effectuer, nous devons tout d'abord récupérer les données pour construire le graphe pondéré. Nous utilisons la même fonction que dans la version 1 et obtenons une matrice de plaisirs associés à chaque sommet.

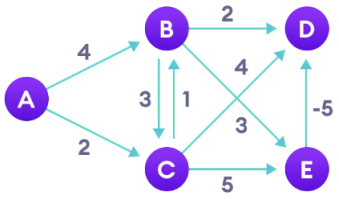


Figure 1: Exemple de graphe pondéré

Avec ce graphe, nous construisons l'algorithme qui fonctionne comme suit :

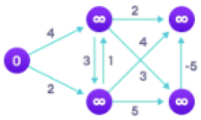
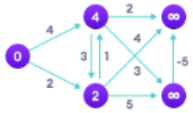
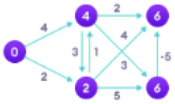
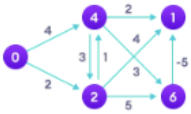
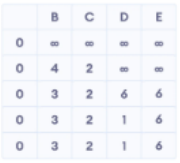
1. Initialisation : Nous attribuons une valeur de distance infinie à tous les sommets, sauf au sommet de départ où la distance est mise à zéro. Nous initialisons également un tableau de prédécesseurs pour chaque sommet.

2. Relaxation des arêtes : Nous effectuons V-1 itérations, où V est le nombre de sommets du graphe. À chaque itération, nous parcourons toutes les arêtes du graphe et mettons à jour les distances des sommets adjacents si une distance plus longue est trouvée. La distance entre le sommet de départ et chaque sommet est progressivement mise à jour.

3. Détection des cycles : Après V-1 itérations, nous effectuons une itération supplémentaire pour vérifier la présence de cycles. Si une distance plus longue est trouvée à cette étape, cela signifie qu'il y a un cycle de poids négatif dans le graphe. Dans ce cas, l'algorithme ne peut pas trouver de solution optimale.

4. Récupération du plus long chemin : Si aucun cycle n'est détecté, nous pouvons récupérer le plus long chemin entre le sommet de départ et chaque sommet en suivant les prédécesseurs enregistrés lors de l'étape de relaxation des arêtes.

Une fois que nous avons la liste de tous les plaisirs possibles, nous recherchons le maximum dans la liste et nous pouvons afficher le plaisir maximal s'il n'y a pas de cycles, sinon "SKY IS THE LIMIT".

étape 1: on choisit l'origine 0 et on initialise les plaisirs à l'infini.	
étape 2: on visit chaque sommet et on met le plaisir associé au chemin effectué	
étape 3: on fait cette étape V fois	
étape 4: on change les plaisirs sur chaque sommet	
étape 5: on obtient le tableau des plaisirs et on regarde s 'il y a un cycle	

	B	C	D	E
0	∞	∞	∞	∞
0	4	2	∞	∞
0	3	2	6	6
0	3	2	1	6
0	3	2	1	6

Figure 2: Étapes de l'algorithme de Bellman Ford

4 Conclusion

En comparant ces deux algorithmes, nous remarquons que celui de Bellman Ford est plus simple à coder. Nous avons dans une seule fonction la recherche de cycle et le plaisir maximal.

De plus, nous nous sommes rendu compte que partir sur des structures n'est pas nécessairement la bonne solution. Cela peut rapidement devenir complexe et pas du tout bénéfique pour la résolution du problème. Il faut aller à l'essentiel et ne pas se compliquer l'esprit.