

Graphes Pondérés

I - Mathématiquement

III - Recherche de chemin de poids minimal dans un graphe pondéré

1. Algorithme de Dijkstra

Parenthèse : structure de “sac”

On retrouve ces 4 fonctions :

- create
- add
- take
- is_empty

```
1  def whatever_first_search(G, s):  
2      n = len(G)  
3      visited = tous à faux  
4      bag = create  
5      add s bag  
6  
7      while not is_empty(bag):  
8          u = take bag  
9          if not visited[u]:  
10             visited[u] = True  
11             for each v neighbor:  
12                 add v bag
```

Description de l'algorithme

```
1  def dijsktra(G, s):  
2      n = len(G)  
3      pq = initialise une file de prio avec sommets à prio = +infini  
4      update pq s 0  
5  
6      Tant que pq non vide:  
7          u = extract_min pq  
8          mise à jour de tab_distance[u]  
9          Pour chaque voisin v:  
10             min_d = prio(u) + pond(u,v)  
11             if v in pq && min_d < prio(v):  
12                 update pq v min_d  
13  
14      renvoyer le tableau des distances
```

Correction de l'algorithme

Lemme : Soit s et t deux sommets d'un graphe orienté et pondéré G . Soit c un chemin $s = u_0, u_1, \dots, u_k = t$ de poids minimal de s à t . Alors quelque soit i , le chemin $s = u_0, \dots, u_i$ est de poids minimal de s à u_i .

Preuve : S'il y avait un meilleur chemin de s à u_i , alors on obtiendrait un meilleur chemin de s à t .

Invariant

- Si $\text{prio}(u)$ différent de $+\infty$ alors $\exists c : \rightarrow u$ de poids $\text{prio}(u)$
- Si $x \notin pq$ alors pour tout voisin w de x on a $\text{prio}(w) \leq \text{prio}(x) + \text{pond}(x, w)$
- Lorsque u sort de la file, $\text{prio}(u) = \delta(s, u)$

Préservation de l'invariant

On suppose u différent de s

On considère c un chemin optimal de s à u .

- Sur ce chemin, on note w le premier du chemin qui est dans la file (existe car u est dans la file)
- Sur ce chemin on note x le prédécesseur de w (existe car $s \notin pq$ donc s différent de w)

$\delta(s, w) \leq \text{prio}(w)$ par invariant

$x \notin pq$ puisque w est le premier du chemin à être dans pq .

Donc par le 2ème invariant : $\text{prio}(w) \leq \text{prio}(x) + \text{pond}(x, w)$ où $\text{prio}(x) = \delta(s, w)$.

Par le lemme, le préfixe du chemin c de s à w est optimal, donc de poids $\delta(s, w)$

De même pour x , donc $\delta(s, w) = \delta(s, x) + \text{pond}(w, x)$

On met tout ensemble :

$\delta(s, w) \leq \text{prio}(w) \leq \delta(s, w) + \text{pond}(w, x)$ et $\text{prio}(w) = \delta(s, w)$

Et comme $u = \text{extract_min}(pq)$.

On a $\text{prio}(u) \leq \text{prio}(w) = \delta(s, w) = \delta(s, u) - \text{pond}(c_2)$

Donc $\delta(s, u) \leq \delta(s, u) - \text{pond}(c_2)$

Donc $\text{pond}(c_2) = 0$

L'invariant est vérifié.

Complexité

Avant la boucle : $O(n)$ pour initialiser la file de priorité

Boucle while : exécutée exactement une fois par sommet

Extraction du min : $O(\log(n))$

Pour chaque voisin : $O(\log(n))$ à cause de la mise à jour de priorité

Finalement :

$$\begin{aligned}
O(n) + \sum_{u \in V} O\left(\log(n) + \sum_{v \in \text{voisins}} O(\log(n))\right) &= O(\log(n)) + \sum_{u \in V} O(\log(n))d_+(u) \\
&= O(n \log(n)) + O(m \log(n)) \\
&= O(\log(n)(n + m))
\end{aligned}$$

Conclusion

- L'algorithme donne pour un sommet s : les poids minimaux et plus courts chemins de s à tous les $t \in V$.

2. Algorithme de Floyd Warshall

Introduction

- On travaille avec la matrice d'adjacence
- On va déterminer tous les plus courts chemins de s à $t \forall (s, t)$.

Première idée - Adaptation du produit matriciel

Essayons d'adapter la méthode des puissances matricielles. On note A la matrice d'adjacence du graphe et suppose :

$$A_{ij} = +\infty \text{ si } (i, j) \notin E$$

$$A_{ij} = \text{pond}(i, j)$$

$$A_{jj} = 0$$

On aimerait que A_{ij}^k donne le poids minimal d'un chemin de longueur au plus k de i à j .

$$A_{ij}^k = \min_{l=0}^{n-1} (A_{il}^{k-1} + A_{lj})$$

Complexité

En supposant la multiplication matricielle modifiée en $O(n^3)$ le calcul de A^n est en $O(n^3 \log(n))$.

Description de l'algorithme

De manière similaire, on fractionne le problème "aller de i à j en un chemin de poids minimal" en des sous-problèmes "aller de i à j **en utilisant uniquement les sommets $[0, k-1]$** et de poids minimal".

On définit pm_{ij}^k le poids minimal d'un chemin de i à j dont les sommets intermédiaires sont dans $[0, k-1]$.

$$pm_{ij}^0 = A_{ij}$$

$$pm_{ij}^k = \min\left(pm_{ij}^{(k-1)}, pm_{i,k-1}^{k-1} + pm_{k-1,j}^{k-1}\right)$$

Complexité : $O(n^3)$

Pseudo-code : On applique la recette du cours de programmation dynamique

- Création du tableau
 - C'est un `int array array`
 - Convention : $T.(i).(j).(k) = pm_{ij}^k$
- Cas de base : facile

```
1   for i
2   for j
```

ocaml

```
3           T.(i).(j).(0) <- ...
```

- Remplissage : ne pas se tromper dans l'ordre des boucles

```
1   for k = 1 to ...
2   for i = 0 to ...
3   for j = 0 to ...
4           T.(i).(j).(k) <- ...
```

ocaml

Gain en espace : $T.(i).(j)$: table 2D.

Invariant : $T.(i).(j) = pm_{ij}^{k-1}$