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`

```
def whatever_first_search(G, s):
    n = len(G)
    visited = tous à faux
    bag = create
    add s bag

    while not is_empty(bag):
        u = take bag
        if not visited[u]:
            visited[u] = True
            for each v neighbor:
                add v bag
```

Description de l’algorithme

```
def dijsktra(G, s):
    n = len(G)
    pq = initialise une file de prio avec sommets à prio = +infini
    update pq s 0

    Tant que pq non vide:
        u = extract_min pq
        mise à jour de tab_distance[u]
        Pour chaque voisin v:
            min_d = prio(u) + pond(u,v)
            if v in pq && min_d < prio(v):
                update pq v min_d

    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 $s = u_0, u_1, ..., u_k = t$ de poids minimal de s à t. Alors quelque soit i, le chemin $s = u_0, ..., u_i$ 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 prio(u) différent de $+\infin$ alors $\exist c : s\rightarrow u$ de poids prio(u)
- Si $x \notin pq$ alors pour tout voisin w de x on a prio(u) $\leq$ prio(x) + pond(x, w)
- Lorsque u sort de la file, 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)

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

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

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

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 $\delta(s, w) = \delta(s, x) + pond(w, x)$

On met tout ensemble :

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

Et comme $u = extract\_min \space pq$.

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

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

Donc $pond(c_2) = 0$

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 :

$O(n) + \sum_{u \in V}(O\Big(log(n) + \sum_{vvoisin} O(log(n))\Big)$

$= O(log(n)) + \sum_{u \in V} O(log(n))d_+(u)$

$= O(nlog(n)) + O(mlog(n))$

$= O(log(n)(n + m))$

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 :

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

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

$A_{jj} = 0$

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

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

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

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

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

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

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

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