STRUCTURES DE DONNÉES ET ALGORITHMES · CHAPITRE 20

Choisir le plus proche sommet avec Dijkstra

Calculez des distances à poids non négatifs par relaxation. Distinguez le modèle quadratique par sélection linéaire et la version à file de priorité.

Par Ali ZAINOUL · Cours gratuit

Ce que vous allez apprendre

- Relâcher des arêtes et justifier le choix de Dijkstra.
- Dijkstra fixe le sommet de distance provisoire minimale, avec des poids non négatifs.
- Une relaxation ne change la distance d'un voisin que si un chemin moins coûteux a été trouvé.

Prérequis. Avoir étudié les chapitres DSA précédents ; connaître files, piles et structures indexées.

Partir d'un besoin

Un trajet de deux routes peut être plus rapide qu'un trajet d'une seule route. Quand les arêtes ont des durées différentes, BFS ne suffit plus à minimiser la durée totale.

Définir la structure et les conventions

L'algorithme de **Dijkstra** cherche les plus petites sommes de poids depuis une source, avec des poids finis **non négatifs**. Une distance provisoire est une borne supérieure obtenue par un chemin déjà connu ; l'infini, écrit inf, signifie qu'aucun chemin n'a encore été trouvé.

À chaque étape, on choisit parmi les sommets non fixés celui dont la distance provisoire est minimale, puis on fixe cette distance. **Relâcher** une arête u vers v consiste à remplacer distance[v] si distance[u] + poids(u, v) est plus petit. Avec des poids non négatifs, un détour par

un sommet non fixé ne peut améliorer le minimum choisi.

L'exemple est orienté : A vers B coûte 4, A vers C coûte 1, C vers B coûte 2. Python stocke des couples (voisin, poids) ; la boucle les décompose. MATLAB stocke dans chaque cellule une matrice à deux colonnes, identifiant puis poids. Pour garder la sélection visible, les deux versions parcourent tous les sommets non fixés plutôt que d'utiliser un tas.

Suivre les états

Distances provisoires depuis A

Sommet fixé	A	B	C
A	0	4	1
C	0	3	1
B	0	3	1

Comparer les réalisations

Les distances de A, B et C valent [0, 3, 1]. Le trajet vers B passe par C et améliore le coût direct 4. Les points-virgules dans une matrice MATLAB séparent ses lignes ; zeros(0, 2) crée une matrice vide avec deux colonnes. size(outgoing, 1) compte les arêtes sortantes ; isinf teste l'infini. Le script MATLAB conserve sa fonction locale à la fin.

Pseudo-code

```
// example.pseudo
// dijkstra
// Input: valid graph, finite non-negative weights, valid source.
// Set source distance to 0 and all others to infinity.
// Repeatedly fix the unfixed vertex with smallest finite distance.
// Relax its outgoing edges; stop if no finite candidate remains.
```

Python

```python
# dijkstra_demo.py
def dijkstra(edges, source):
    n = len(edges)
    distance = [float("inf")] * n
    fixed = [False] * n
    distance[source] = 0
    for step in range(n):
        node = -1
        for candidate in range(n):
            if not fixed[candidate] and (node == -1 or distance[candidate] <
distance[node]):
                node = candidate
        if node == -1 or distance[node] == float("inf"):
            break
        fixed[node] = True
        for neighbor, weight in edges[node]:
            proposal = distance[node] + weight
            if proposal < distance[neighbor]:
                distance[neighbor] = proposal
    return distance
edges = [[(1, 4), (2, 1)], [], [(1, 2)]]
distance = dijkstra(edges, 0)
print(distance)
```

MATLAB

```
% dijkstra_demo.m
edges = {[2, 4; 3, 1], zeros(0, 2), [2, 2]};
distance = dijkstra(edges, 1);
disp(distance);
function distance = dijkstra(edges, source)
    n = numel(edges);
    distance = inf(1, n);
    fixed = false(1, n);
    distance(source) = 0;
    for step = 1:n
        node = 0;
        for candidate = 1:n
            if ~fixed(candidate) && (node == 0 || distance(candidate) <
distance(node))
                node = candidate;
            end
        end
        if node == 0 || isinf(distance(node))
            break;
        end
        fixed(node) = true;
        outgoing = edges{node};
        for row = 1:size(outgoing, 1)
            neighbor = outgoing(row, 1);
            weight = outgoing(row, 2);
            proposal = distance(node) + weight;
            if proposal < distance(neighbor)
                distance(neighbor) = proposal;
            end
        end
    end
end
```

Justifier et compter

La sélection linéaire répétée n fois coûte $O(n^2)$; les relaxations totalisent $O(m)$. Ces versions ont donc $O(n^2 + m)$ temps et $O(n)$ mémoire auxiliaire. Une file de priorité avec tas et mises à jour adaptées permet $O((n + m) \log n)$ pour un graphe simple ; cette garantie ne décrit pas le code montré ici. Des poids négatifs invalident le raisonnement : ils demandent un autre algorithme, par exemple Bellman-Ford avec traitement des cycles négatifs.

Prévoir puis expérimenter

Ajoutez un sommet sans arête : sa distance reste infinie. Remplacez le coût C vers B par 5 : le trajet direct de coût 4 redevient optimal. Des poids nuls sont autorisés ; des poids négatifs ne le sont pas.

Exercice corrigé

▼ Comment conserver aussi le chemin choisi ?

Lors de chaque amélioration, mémorisez dans parent[v] le sommet u qui produit la nouvelle distance. Remontez les parents depuis la destination, puis inversez la suite obtenue. Un sommet à distance infinie n’a pas de chemin à reconstruire.

À retenir

- Dijkstra fixe le sommet de distance provisoire minimale, avec des poids non négatifs.
- Une relaxation ne change la distance d’un voisin que si un chemin moins coûteux a été trouvé.
- La sélection linéaire du code est quadratique ; une version à tas relève d’un autre coût.

Version web : https://www.clearcode.fr/courses/programming/dsa/graphs/dijkstra