Algoritmul Dijkstra

Miriam Costan

Februarie 2022

1 Descrierea Problemei.

- Se dă un graf ponderat (orientat sau neorientat) cu n - noduri și m - muchii.
- Ponderea pe orice muchie este non-negativă.
- Se dă un nod de **start** $start$.
- Se cere determinarea drumului de cost minim de la nodul de start $start$ până la toate celelalte noduri.

Vom prezenta în continuare o soluție la problema descrisă, formulată de Edsger W. Dijkstra in 1959.

2 Soluție $\mathcal{O}(n^2 + m)$ (pentru grafuri dense).

2.1 Inițializare.

1. Creăm un vector de soluții $dist$ unde $dist[start] = 0$ si $dist[nod] = \infty, \forall nod \neq start$.
2. Creăm un vector de vizitări. $vis[nod] = false, \forall nod \in Noduri$

2.2 Descrierea algoritmului.

Cât timp mai avem noduri nevizitate ($\exists nod \in Noduri, vis[nod] = false$):

1. Parcurgem nodurile și salvăm nod_{minim} a.i. $nod_{minim} = min(dist[nod]), \forall nod \in Noduri \& vis[nod] = false$.
2. Marcăm $vis[nod_{minim}] = true$.
3. Parcurgem vecinii nodului nod_{minim} și actualizăm $dist[vecin] = min(dist[vecin], dist[nod_{minim}] + pondere[nod_{minim}][vecin])$.

Soluția descrisă este omptimă pentru grafuri dense ($n^2 \approx m$), dar atunci când m este mult mai mic decât numărul maxim de muchii, problema poate fi optimizată la $\mathcal{O}(n * log(m) + m)$.

3 Soluție $\mathcal{O}(n * log(m) + m)$.

3.1 Inițializare.

1. Creăm un vector de soluții $dist$ unde $dist[start] = 0$ si $dist[nod] = \infty, \forall nod \neq start$.
2. Creăm un vector de vizitări. $vis[nod] = false, \forall nod \in Noduri$.
3. Creăm un heap (coadă cu priorități) de perechi $Q(nod, distanta)$, unde introducem inițial perechea $(start, 0)$.

3.2 Descrierea algoritmului.

Cât timp mai sunt elemente in Q:

1. Scoatem perechea cu $distanta$ minima.
2. Dacă nodul din perechea extrasă a fost deja vizitat ($vis[nod] = true$), continuăm.
3. Altfel, marcăm ($vis[nod] = true$) și ($dist[nod] = distanta$).
4. Parcurgem vecinii nodului nod și introducem în Q perechea $(vecin, distanta + pondere[nod][vecin])$.

Alternativ, pentru a optimiza și a nu introduce perechi redundante în heap, pasul 4 poate fi modificat după cum urmeaza:

Parcurgem vecinii nodului nod:

- Dacă $vis[vecin] = true$, continuăm.
- Dacă $dist[vecin] < distanta + pondere[nod][vecin]$, continuăm.
- $dist[vecin] = distanta + pondere[nod][vecin]$.
- Introducem in Q perechea $(vecin, dist[vecin])$.

4 Reconstrucția drumului.

Creăm un vector de tați, unde $tata[nod]$ este tatăl nodului nod în drumul de cost minim de la $start$ la nod. Initializăm $tata[nod] = -1, \forall nod \in Noduri$.

Acest vector poate fi folosit pentru reconstrucția drumurilor minime, mergând din tata în tata până se ajunde la nodul $start$.

4.1 Reconstrucția pentru soluția $\mathcal{O}(n^2 + m)$.

La pasul 3 din secținuea 2.2, de fiecare dată când $dist[vecin]$ este actualizat, actualizăm și $tata[vecin] = nod_{minim}$.

4.2 Reconstrucția pentru soluția $\mathcal{O}(n * log(m) + m)$.

- Modificam heap-ul Q, care în loc să rețină perechi, acum va reține tupluri $(nod, tata, distanta)$.
- Inițial vom introduce în heap tuplul $(start, -1, 0)$.
- La pasul 3 din secțiunea 3.2 actualizăm și $(tata[nod] = tata)$.
- La pasul 4 din secțiunea 3.2, când introducem elemente în Q, în loc de $(vecin, distanta + pondere[nod][vecin])$, vom introduce $(vecin, nod, distanta + pondere[nod][vecin])$