ADA Final Exam Preparation

Kon Yi

November 21, 2025

Contents

Chapter 1

All-pairs distances problem

Lecture 1

13 Nov.

Problem 1.0.1.

- Input: an edge-weighted directed graph G with $V(G) = \{1, 2, \ldots, n\}$ that has no cycle of negative weight.
- Output: $d_G(i, j)$ for all vertices i and j of G.

Remark 1.0.1. Here we suppose G has no negative cycle to simplify the problem.

The Naive algorithm is to solve n single-source distances problems directly. Hence, the time complexity for using different algorithm is:

- General edge weights: Bellman-Ford's algorithm takes $O\left(mn^2\right)$ time, which can be $\Theta\left(n^4\right)$ when $m = \Theta\left(n^2\right)$.
- Acyclic: Lawler's algorithm takes $O(mn)$ time.
- Non-negative edge weights: Dijkstra's takes $O\left(mn + n^2 \log n\right)$ time.

However, naive method takes a lot of time to compute unnecessary things, so it takes a lot of times.

1.1 A DP algorithm

Definition 1.1.1. Let $w_k(i, j)$ be the length of a shortest (i, j)-path having at most k edges. Let it be ∞ if such a path does not exist.

We have

$w_1(i, j) = w(ij)$ if (i, j) if a edge, otherwise $w_1(i, j) = \infty$.
$w_{n-1}(i, j) = d_G(i, j)$ since a shortest path has at most $n - 1$ edges (note that there is not negative cycle).

Hence, we have

$$w_{2k}(i, j) = \min_{1 \le t \le n} w_k(i, t) + w_k(t, j) \text{ for all } k \ge 1.$$

Also, note that even if $k \ge \frac{n}{2}$ this recurrence relation is still correct, so we can just compute $w_1(i, j)$ then $w_2(i, j)$ then $w_4(i, j)$ and so on, and after $O(\log n)$ rounds we can get the answer.

Now we analyze the time complexity. For each (i, j)-pair, we need $O(\log n)$ rounds, where each round takes $O(n)$ times, so each (i, j)-pair takes $O(n \log n)$ times. Note that we have $O\left(n^2\right)$ (i, j)-pairs, so it takes totally $O\left(n^3 \log n\right)$ times. This is pretty slow, but in general a little bit faster than doing Bellman-Ford's algorithm for n times.

1.2 Floyd and Warshall's DP algorithm

Definition 1.2.1. Let $d_k(i,j)$ be the length of any shortest (i,j)-path whose internal indices are at most k. If there is no such a path, then let $d_k(i,j) = \infty$.

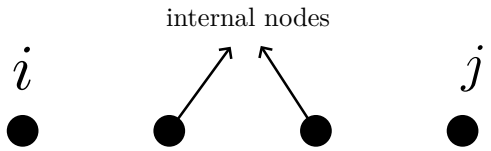

Figure 1.1: Internal Nodes

Thus, we have

$$d_0(i,j) = w(ij), \quad d_n(i,j) = d_G(i,j).$$

Hence, we can define a recurrence relation:

$$\begin{cases} d_0(i,j) = w(i,j) \\ d_{k+1}(i,j) = \min\{d_k(i,j), d_k(i,k+1) + d_k(k+1,j)\}. \end{cases}$$

Note that it corresponds to two cases: walk through $k+1$ or not. If not, then it corresponds to $d_k(i,j)$. If so, then it corresponds to $d_k(i,k+1) + d_k(k+1,j)$ since excluding $k+1$ and seperate this path into two parts, then internal nodes in each part can have indices of at most k.

Now we analyze the time complexity of Floyd and Warshall's DP algorithm: Fix (i,j), then it takes $O(n)$ time to compute from $d_0(i,j)$ to $d_n(i,j)$, and since we have $O\left(n^2\right)$ (i,j)-pairs, so it take totally $O\left(n^3\right)$ time.

1.3 Johnson's reweighting technique

As previously seen, if G is a non-negative weighted graph, then running Dijkstra's algorithm for n times needs $O(mn + n\log n)$ time. Now Johnson gives a method to reweight w into $\hat{w}$ s.t.

- $\hat{w}$ is non-negative
- any shortest (i,j)-path of G w.r.t. $\hat{w}$ is a shortest (i,j)-path of G w.r.t. w.

The idea of reweighting is to

- Assign a weight $h(i)$ to each vertex i of G.
- Let $\hat{w}(i,j) = w(i,j) + h(i) - h(j)$.
- Then, for any (i,j)-path P, we have

$$\hat{w}(P) = w(P) + h(i) - h(j).$$

- Hence, P is a shortest (i,j)-path of G w.r.t. $\hat{w}$ if and only if P is a shortest (i,j)-path of G w.r.t. w.

Remark 1.3.1. The challenge is to find a vertex weight h s.t. the resulting adjusted edge weight $\hat{w}$ is non-negative. If $\hat{w}$ is non-negative, then we can apply Dijkstra's algorithm to obtain all-pairs shortest path trees in $O\left(mn + n^2\log n\right)$ time.

Algorithm 1.1: Johnson's Technique

1 Let H be obtained from G by adding a new vertex 0 and adding a weight-0 edge from vertex 0 to each vertex i of G.
2 H has no negative cycle if and only if G has no negative cycle.
3 Let $h(i)$ be the distance from vertex 0 to vertex i in H. That is, $h(i) = d_H(0,i)$.
4 The vertex weight function h can be obtained by Bellman-Ford in $O(mn)$ time.

Remark 1.3.2. H has no negative cycle if and only if G has no negative cycle since G is directed and vertex 0 has only out degree, so any cycle in H and G is induced by $\{1, 2, \ldots, n\}$, which does not include 0.

Remark 1.3.3. $d_H(0, i) \leq 0$ and $d_H(0, i) < 0$ if there is a path of negative weight from j to i for some $j > 0$ since we can go from 0 to j first, then go from j to i.

Theorem 1.3.1. $\hat{w}(i, j) \geq 0$ for all $i, j \in [n]$.

Proof. Since

$$\hat{w}(i, j) = w(i, j) + h(i) - h(j) = w(i, j) + d_H(0, i) - d_H(0, j),$$

and note that $d_H(0, i) + w(i, j)$ is the shortest distance of a path from 0 and go through i to j, which is $\geq$ the distance from 0 to j, which is $d_H(0, j)$. Hence, we have

$$d_H(0, i) + w(i, j) - d_H(0, j) \geq 0.$$

■

Now we analyze the time complexity of Johnson's algorithm for general edge weights: We first obtain h by doing one time Bellman-Ford's algorithm, which takes $O(mn)$ time. Then, we run Dijkstra's algorithm for all vertex i of G, which takes totally $O\left(mn + n^2 \log n\right)$ time. Note that to here we just store n shortest path tree, and we have to obtain the real distance by running through all n tree, which takes $O(n) \cdot n = O\left(n^2\right)$ time since a tree has $n - 1$ edges. Hence, it totally take $O\left(mn + n^2 \log n\right)$ time for Johnson's technique.

Chapter 2

Maximum flow

Problem 2.0.1 (The maximum flow problem)**.**

- Input: A directed graph G with edge capacity $c : E(G) \to \mathbb{R}^+$ and two vertices, the source s and the sink t.
- Output: An (s,t)-flow with maximum (flow) value.

Remark 2.0.1. For convenience, we allow G has multiple/parallel edges, though merging multiple edges and increase the capacity to get an equivalent graph is not forbbiden.

Remark 2.0.2. An (s,t)-flow is a function $f : E(G) \to \mathbb{R}^+ \cup \{0\}$ satisfying

- capacity constraint: $f(uv) \leq c(uv)$ for each edge uv of G.
- conservation law:
$$\sum_{uv \in E(G)} f(uv) = \sum_{vw \in E(G)} f(vw)$$
for each vertex v of G other than s and t.

The value of an (s,t)-flow f is
$$\sum_{sv \in E(G)} f(sv) - \sum_{us \in E(G)} f(us).$$

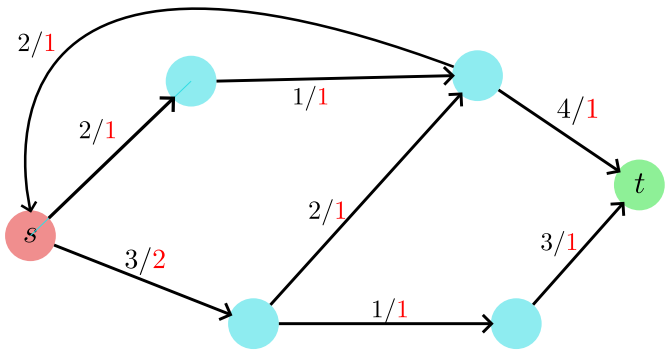

Figure 2.1: The maximum flow problem

2.1 Ford-Fulkerson's algorithm

Intuition. Reduce the mainum (s,t)-flow problem to the reachability problems for a sequence of graph R.

Definition 2.1.1 (Residual graph)**.** The residual graph $R(f)$ with respect to a flow f of G with $V(R(f)) = V(G)$ is defined as follows for each edge uv of G:

- If $f(uv) < c(fg)$, then let $R(f)$ have an edge uv with capacity $c(uv) - f(uv)$.
- If $f(uv) > 0$, then let $R(f)$ have an edge vu with capacity $f(uv)$.

Corollary 2.1.1. $R(f)$ is a graph with capacity of every edge positive, just like G.

Remark 2.1.1. Even if G has only one uv edge, $R(f)$ may have 2 uv edges if $f(uv) < c(uv)$ and $f(vu) > 0$.

Theorem 2.1.1. For any (s,t)-flow f of G, we have the following statements:

(1) If $d_{R(f)}(s,t) = \infty$, then f is a maximum (s,t)-flow of G.

(2) If $d_{R(f)}(s,t) < \infty$ and g is an (s,t)-flow of the residual graph $R(f)$, then $f + g$ remains an (s,t)-flow of G, where
$$(f+g)(uv) = f(uv) + g(uv) - g(vu)$$
for each edge uv of G.

Appendix