

# ADA Final Exam Preparation

Kon Yi

November 27, 2025

# Contents

<b>1</b>	<b>All-pairs distances problem</b>	<b>2</b>
1.1	A DP algorithm . . . . .	2
1.2	Floyd and Warshall's DP algorithm . . . . .	3
1.3	Johnson's reweighting technique . . . . .	3
<b>2</b>	<b>Maximum flow</b>	<b>5</b>
2.1	Ford-Fulkerson's algorithm . . . . .	6
2.2	Edmonds and Karp's theorem . . . . .	10

# Chapter 1

## All-pairs distances problem

### Lecture 1

#### Problem 1.0.1.

13 Nov.

- Input: an edge-weighted directed graph  $G$  with  $V(G) = \{1, 2, \dots, n\}$  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  $O(mn^2)$  time, which can be  $\Theta(n^4)$  when  $m = \Theta(n^2)$ .
- Acyclic: Lawler's algorithm takes  $O(mn)$  time.
- Non-negative edge weights: Dijkstra's takes  $O(mn + n^2 \log n)$  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  $\infty$  if such a path does not exist.

We have

$$w_1(i, j) = w(ij) \text{ if } (i, j) \text{ is an edge, otherwise } w_1(i, j) = \infty.$$

$w_{n-1}(i, j) = d_G(i, j)$  since a shortest path has at most  $n - 1$  edges (note that there is no negative cycle).

Hence, we have

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

Also, note that even if  $k \geq \frac{n}{2}$  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  $O(n^2)$   $(i, j)$ -pairs, so it takes totally  $O(n^3 \log n)$  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  $d_k(i, j) = \infty$ .

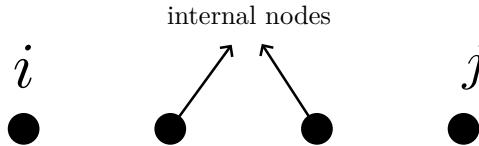


Figure 1.1: Internal Nodes

Thus, we have

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

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 separate 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  $O(n^2)$   $(i, j)$ -pairs, so it takes totally  $O(n^3)$  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  $O(mn + n^2 \log n)$  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  $\{1, 2, \dots, n\}$ , 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  $O(mn + n^2 \log n)$  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  $O(n) \cdot n = O(n^2)$  time since a tree has  $n - 1$  edges. Hence, it totally take  $O(mn + n^2 \log n)$  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  $c : E(G) \rightarrow \mathbb{R}^+$  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 forbiden.

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

$$\sum_{sv \in E(G)} f(sv) - \sum_{us \in E(G)} f(us).$$

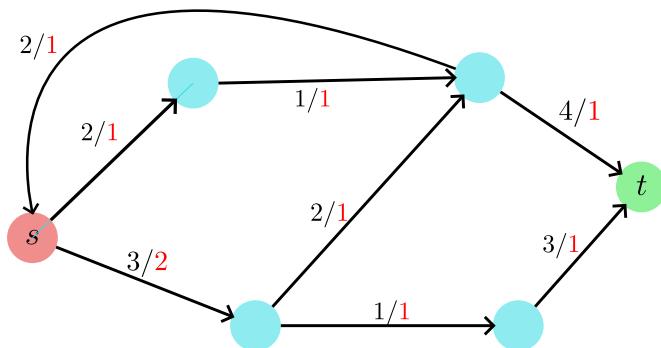


Figure 2.1: The maximum flow problem

## 2.1 Ford-Fulkerson's algorithm

**Intuition.** Reduce the maximum  $(s, t)$ -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  $f(uv) < c_f(uv)$ , 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  $G$ .

**Remark 2.1.2.** In (2), the  $g(uv)$  and  $g(vu)$  corresponds to the edges on  $R(f)$  and formed by  $uv$  in  $G$  and we give the flow value to this  $uv, vu$  pairs. For  $vu$  in  $G$ , it may also form a  $uv, vu$  pairs in  $R(f)$ , and these two cases should be handled separately.

**proof of (1).** If  $d_{R(f)}(s, t) = \infty$ , and there exists  $f'$  s.t.  $|f'| > |f|$ , i.e.  $f$  is not a maximum  $(s, t)$ -flow. Then, we define

$$h(uv) := f'(uv) - f(uv) \text{ for all } uv \in E(G).$$

Note that for all  $v \neq s, t$  we have

$$\begin{aligned} \sum_x h(xv) &= \sum_x f'(xv) - f(xv) = \sum_x f'(xv) - \sum_x f(xv) = \sum_x f'(vx) - \sum_x f(vx) \\ &= \sum_x f'(vx) - f(vx) = \sum_x h(vx). \end{aligned}$$

Also, we have

$$\begin{aligned} |h| &= \sum_x h(sx) - h(xs) = \sum_x f'(sx) - f(sx) - (f'(xs) - f(xs)) \\ &= \sum_x f'(sx) - f'(xs) - \sum_x f(sx) - f(xs) = |f'| - |f| > 0. \end{aligned}$$

Now we convert  $h$  to a  $(s, t)$ -flow on  $R(f)$ . Note that

- If  $h(uv) \geq 0$ , then

$$0 \leq h(uv) = f'(uv) - f(uv) \leq c_f(uv) - f(uv),$$

so  $h(uv)$  fits within the forward residual capacity of  $uv$  if  $h(uv) \geq 0$ .

- If  $h(uv) < 0$ , then

$$0 < -h(uv) = f(uv) - f'(uv) \leq f(uv),$$

so  $-h(uv)$  fits within the reverse residual capacity on arc  $vu$  if  $h(uv) < 0$ .

Now we construct a new graph  $R(f)'$  by

- (1) If  $f(uv) < c_f(g)$ , then let  $R'(f)$  have an edge  $\textcolor{red}{uv}$  with capacity  $c_f(uv) - f(uv)$ . If  $f(uv) = c_f(g)$ , then have an edge  $\textcolor{red}{uv}$  with capacity 0.
- (2) If  $f(uv) > 0$ , then let  $R'(f)$  have an edge  $\textcolor{blue}{vu}$  with capacity  $f(uv)$ . If  $f(uv) = 0$ , then have an edge  $\textcolor{blue}{vu}$  with capacity 0.

Hence, for every vertices  $u, v \in V(R'(f))$ , there is a forward residual arc  $uv$ , which is formed by  $uv$  in (1), and a reverse residual arc  $vu$  formed by  $vu$  in (2). We can define  $g$  on  $R'(f)$  by

$$\begin{cases} g(uv) = g_f(uv) = \max\{h(uv), 0\} & \text{on the forward residual arcs } uv. \\ g(vu) = g_r(vu) = \max\{-h(uv), 0\} & \text{on the reverse residual arcs } vu. \end{cases}$$

Then, we claim that  $g$  is a flow of  $R'(f)$  with flow value  $> 0$ . Note that  $R'(f)$  is in fact an expansion of  $R(f)$ , but add some edges with capacity 0, and anything else is not changed.

- Capacity constraint: If the forward residual arc  $uv$  exists in  $R(f)$ , then we know

$$c_{R'(f)}(uv) = c_f(uv) - f(uv) > 0.$$

If  $g_f(uv) = 0$ , then  $g_f(uv) < c_{R'(f)}(uv)$ , and if  $g_f(uv) = h(uv)$ , then  $h(uv) > 0$  and thus

$$g_f(uv) = h(uv) \leq c_f(uv) - f(uv) = c_{R'(f)}(uv)$$

by the arguments of  $h$  above. Hence, if the forward residual arc  $uv$  exists in  $R(f)$ , then  $g_f(uv) \leq c_{R'(f)}(uv)$ . Now if the forward residual arc  $uv$  does not exist in  $R(f)$ , then  $c_{R'(f)}(uv) = 0$  and  $c_f(uv) = f(uv)$ , so we have

$$h(uv) = f'(uv) - f(uv) \leq c(uv) - f(uv) = f(uv) - f(uv) = 0,$$

so we must have

$$g_f(uv) = \max\{h(uv), 0\} = 0,$$

and thus in this case  $g_f(uv) \leq c_{R'(f)}(uv)$ .

Now we discuss the reverse residual arc. If the reverse residual arc  $vu$  exists in  $R(f)$ , then

$$c_{R'(f)}(vu) = f(uv) > 0.$$

Now if  $g_r(vu) = 0$ , then  $g_r(vu) \leq c_{R'(f)}(vu)$ . If  $g_r(vu) = -h(uv)$ , then  $h(uv) < 0$ , and thus

$$g_r(vu) = -h(uv) \leq f(uv) = c_{R'(f)}(vu)$$

by the above arguments about  $h$ . Now if the reverse residual arc  $vu$  does not exist in  $R(f)$ , then  $f(uv) = 0$  and  $c_{R'(f)}(vu) = 0$ . Hence,

$$h(uv) = f'(uv) - f(uv) = f'(uv) \geq 0,$$

so

$$g_r(vu) = \max\{-h(uv), 0\} = 0,$$

and thus  $g_r(vu) \leq c_{R'(f)}(vu)$ . Hence, we have shown that the capacity constraint is always correct.

- Conservation law: For  $u \neq s, t$ , we know

$$\begin{aligned} & \sum_x g_f(ux) + g_r(ux) - \sum_x g_f(xu) + g_r(xu) \\ &= \sum_x \max\{h(ux), 0\} + \max\{-h(xu), 0\} - \max\{h(xu), 0\} - \max\{-h(ux), 0\} \end{aligned}$$

and we can observe that

$$\max\{h(ux), 0\} + \max\{-h(xu), 0\} - \max\{h(xu), 0\} - \max\{-h(ux), 0\} = h(ux) - h(xu)$$

no matter what the sign of  $h(ux)$  and  $h(xu)$  is. Hence,

$$\sum_x g_f(ux) + g_r(ux) - \sum_x g_f(xu) + g_r(xu) = \sum_x h(ux) - h(xu) = 0.$$

Also,

$$\begin{aligned} |g| &= \sum_x g_f(sx) + g_r(sx) - g_f(xs) - g_r(xs) \\ &= \sum_x \max\{h(sx), 0\} + \max\{-h(xs), 0\} - \max\{h(xs), 0\} - \max\{-h(sx), 0\} \\ &= \sum_x h(sx) - h(xs) = |h| > 0 \end{aligned}$$

by similar arguments.

Hence, we know  $g$  is a flow with flow value  $> 0$  on  $R'(f)$ . Also, notice that if the forward residual edge  $uv$  does not exist in  $R(f)$ , then  $g_r(uv) = 0$ , while the reverse residual edge  $vu$  does not exist in  $R(f)$  implies  $g_r(vu) = 0$ . Hence, if we define

$$g'(uv) := \begin{cases} g_f(uv), & \text{if } uv \text{ exists in } R(f) \text{ as a forward residual arc;} \\ g_r(uv), & \text{if } uv \text{ exists in } R(f) \text{ as a reverse residual arc.} \end{cases}$$

Then,  $g'$  is a flow in  $R(f)$  and  $|g'| = |g| > 0$ .

Now we claim that if a positive flow exists in  $R(f)$ , then  $s$  and  $t$  are reachable. Actually, for any multi-directed graph  $G'$ , if  $p$  is a  $(s, t)$ -flow of  $G'$  and  $|p| > 0$ , then  $s$  and  $t$  are reachable. We prove the general case. For all  $e_1, e_2, \dots, e_k$  are parallel edges between  $u$  and  $v$ , then we define

$$p'(uv) = p(e_1) + p(e_2) + \cdots + p(e_k).$$

Hence, we have

$$\sum_x p'(ux) = \sum_x p'(xu) \text{ for all } u \neq s, t \text{ and } |p| = \sum_x p'(sx) - \sum_x p'(xs).$$

Now if  $s$  and  $t$  are not reachable, then consider

$$S = \{v \in V(G') \mid \text{there is a directed } sv \text{ path made only of arcs } e \text{ with } p(e) > 0\}.$$

Hence,  $t \notin S$  and  $s \in S$ . First, note that no positive edge leave  $S$ , otherwise the endpoint of this point should be in  $S$ , which is a contradiction. Now since

$$\sum_{x \in S} \left( \sum_y p'(xy) - \sum_y p'(yx) \right) = |p| + \sum_{x \in S \setminus \{s\}} 0 = |p| > 0$$

and we know

$$\sum_{x \in S} \left( \sum_y p'(xy) - \sum_y p'(yx) \right) = \sum_{\substack{u \in S \\ v \notin S}} p'(uv) - \sum_{\substack{u \notin S \\ v \in S}} p'(uv)$$

since

$$\sum_{\substack{x \in S \\ y \in S}} p'(xy) - \sum_{\substack{x \in S \\ y \in S}} p'(yx) = 0 \text{ (these are two same things)}$$

However, we know there is no positive edge leaves  $S$ , so

$$\sum_{u \in S \setminus S} p'(uv) = 0,$$

and thus

$$0 < |p| = \sum_{x \in S} \left( \sum_y p'(xy) - \sum_y p'(yx) \right) = \sum_{\substack{u \in S \\ v \notin S}} p'(uv) - \sum_{\substack{u \notin S \\ v \in S}} p'(uv) = - \sum_{\substack{u \notin S \\ v \in S}} p'(uv) \leq 0,$$

which is a contradiction. Hence,  $s$  and  $t$  are reachable.

Now from this claim, we know  $d_{R(f)}(s, t) < \infty$ , which is a contradiction, so  $f$  is a maximum flow.  $\blacksquare$

### proof of (2).

- Capacity constraint: We want to show  $(f + g)(uv) \leq c_f(uv)$ . Note that

$$\begin{aligned} (f + g)(uv) &= f(uv) + g(uv) - g(vu) \leq f(uv) + (c_f(uv) - f(uv)) - g(vu) \\ &= c_f(uv) - g(vu) \leq c_f(uv). \end{aligned}$$

Hence, this is true.

- Conservation law: For  $v \neq s, t$ , we have

$$\begin{aligned} \sum_x (f + g)(vx) - \sum_x (f + g)(xv) &= \sum_x f(vx) + g_f(vx) - g_r(xv) - (f(xv) + g_f(xv) - g_r(vx)) \\ &= \sum_x g_f(vx) + g_r(vx) - g_r(xv) - g_f(xv) = 0 \end{aligned}$$

since

$$\sum_x f(vx) - f(xv) = 0$$

and  $g$  is a flow on  $R(f)$ .  $\blacksquare$

---

### Algorithm 2.1: Ford-Fulkerson's algorithm

**Setup:** Let  $f(uv) = 0$  for each edge  $uv$  of  $G$ . Repeat the following steps until  $R(f)$  does not contain an  $st$ -path.

- 1 Obtain an  $st$ -path  $P$  of  $R(f)$ . Let  $q$  be the minimum capacity of the edges of  $P$  in  $R(f)$ .
  - 2 Obtain an  $st$ -flow  $g$  of  $R(f)$  by letting  $g(uv) = q$  for each edge  $uv$  of  $P$  and  $g(uv) = 0$  for all other edges  $uv$  of  $G$ .
  - 3 Let  $f = f + g$ .
- 

Initially,  $f$  is a legal  $st$ -flow. In each round,  $g$  is a legal  $st$ -flow of  $R(f)$  (Easy to check). Then, by previous lemma, we know  $f + g$  is also a legal flow of  $G$ , so the algorithm seems correct. However, will the algorithm terminate?

We claim that when all edge capacities are integers, then Ford-Fulkerson's algorithm terminates.

Note that  $q \geq 1$  since  $R(f)$  has all edges' capacities  $> 0$  and all edges' capacities are integers. Also,

$$\begin{aligned} |f + g| &= \sum_x (f + g)(sx) - (f + g)(xs) \\ &= \sum_x f(sx) + g_f(sx) - g_r(xs) - (f(xs) + g_f(xs) - g_r(sx)) \\ &= \left( \sum_x f(sx) - f(xs) \right) + \sum_x g_f(sx) + g_r(sx) - g_f(xs) - g_r(xs) \\ &= |f| + |g| = |f| + q \geq |f| + 1, \end{aligned}$$

so  $|f + g|$  is strictly increasing in each round. Hence, if  $G$  is a finite graph, then Ford-Fulkerson's algorithm must terminate.

Now we analyze the time complexity. In each round, searching an  $st$ -path takes  $O(|E|)$  times, where  $E$  is the set of edges of  $G$ . Then, suppose the sum of capacity of all edges are  $C$ , then we need  $O(C)$  round since after each round  $|f|$  increases at least 1 and  $|f| \leq C$  for all flow  $f$  of  $G$ . Hence, it takes  $O(|E|C)$  times.

**Question.** Is this algorithm polynomial time or exponential time?

**Definition 2.1.2.** An algorithm is polynomial time only if its running time is bounded by a polynomial in the size of the input encoding, i.e.

$$T(\text{algorithm}) = (\text{size of the input encoding})^{O(1)}.$$

Also, we can similarly define exponential time, and linear time, e.t.c.

If  $C = O(1)$ , then the size of input encoding is  $O(|E| \cdot 1) = O(1)$ , and thus the time complexity  $O(|E|C) = O(|E|)$  is linear time.

If  $C$  has no restriction, then the space complexity for storing all edges' capacities is  $O(|E| \log C)$  (we need  $O(\log C)$  bits to store the capacity of an edge). Hence,

$$O(|E|C) \neq O(|E| \log C)^{O(1)},$$

so Ford-Fulkerson's algorithm is not polynomial time.

**Question.** What about when the capacity is not integers?

**Answer.** Then Ford-Fulkerson's algorithm may not stop, and although the flow value converges, but the limit value is not the maximum flow value. (\*)

## 2.2 Edmonds and Karp's theorem

Now we introduce the first polynomial time algorithm for maximal flow problem.

**Theorem 2.2.1 (Edmonds and Karp, JACM 1972).** If one makes sure that the augmenting  $st$ -path  $P$  in  $R(f)$  is an  $st$ -path in  $R(f)$  having a **minimum number of edges**, then the time complexity of Ford-Fulkerson's algorithm is  $O(m^2n)$ .

**Remark 2.2.1.** If we ensure we use the  $st$ -path of least number of edges in each round, then we can make sure the algorithm terminates within  $mn$  rounds.

**Remark 2.2.2.** We do not assume  $G$  has integer capacities under this circumstance.

From now on, we assume we pick the shortest  $st$ -path in each round. We need two observations to prove the theorem: **現邊** and **遞增觀察**.

**Lemma 2.2.1 (現邊觀察).** If in some round the residual graph  $R(f + g)$  has some edge  $uv$  where  $uv$  does not exist in  $R(f)$ , then

$$d_{R(f)}^*(s, u) = d_{R(f)}^*(s, v) + 1,$$

where  $d_{R(f)}^*(s, w)$  for a vertex  $w$  is the distance of  $s$  to  $w$  in the unweighted version of  $R(f)$ .

**Proof.** If  $R(f)$  has no  $uv$  this edge, but  $R(f + g)$  has  $uv$  this edge, then the only possibility is the unweighted shortest  $st$ -path  $P$  of  $R(f)$  go through  $vu$  this edge. The reason is as follows:

Since  $uv$  is not in  $R(f)$ , so  $P$  does not go through  $uv$ . If  $P$  does not go through  $vu$ , then  $g(uv) = g(vu) = 0$  no matter it is an forward residual arc or an reverse residual arc, and thus

$$\begin{aligned} (f + g)(uv) &= f(uv) + g_f(uv) - g_r(vu) = f(uv) \\ (f + g)(vu) &= f(vu) + g_f(vu) - g_r(uv) = f(vu). \end{aligned}$$

Since  $R(f)$  does not have  $uv$ , and  $f + g$  and  $f$  share same flow value between  $u$  and  $v$ , so it is impossible that  $R(f + g)$  contains  $uv$ , which is a contradiction. Now that  $vu$  is in  $P$ , which is an unweighted shortest  $st$ -path of  $R(f)$ , so the path is like

$$s \rightarrow \dots \rightarrow v \rightarrow u \rightarrow \dots \rightarrow t,$$

and thus

$$d_{R(f)}^*(s, u) = d_{R(f)}^*(s, v) + 1.$$

■

**Lemma 2.2.2 (遞增觀察).** Let the augmenting path  $P$  be an  $st$ -path whose number of edges is minimized in the residual graph  $R(f)$ . Let  $g$  be the saturating flow for  $R(f)$  corresponding to  $P$ . For each vertex  $v$  of  $G$ , we have

$$d_{R(f)}^*(s, v) \leq d_{R(f+g)}^*(s, v).$$

**Proof.** Assume for contradiction that there is a vertex  $v$  of  $G$  with

$$d_{R(f)}^*(s, v) > d_{R(f+g)}^*(s, v). \quad (2.1)$$

Thus,  $d_{R(f+g)}^*(s, v) \neq \infty$ . Let  $v$  be such a vertex closest to  $s$  in the unweighted version of  $R(f + g)$ . We know  $v \neq s$  since

$$d_{R(f)}^*(s, s) = 0 = d_{R(f+g)}^*(s, s).$$

Let  $Q$  be an unweighted shortest  $sv$ -path of  $R(f + g)$ . Let  $uv$  be the last edge of  $Q$ . (Note that  $u$  could be  $s$ .) We have

$$d_{R(f)}^*(s, u) \leq d_{R(f+g)}^*(s, u) \quad (2.2)$$

since we suppose  $v$  is the vertex closest to  $s$  in  $R(f + g)$  which violates the assumption in the lemma and  $d_{R(f+g)}^*(u) + 1 = d_{R(f+g)}^*(v)$ .

- Case 1:  $uv \subseteq R(f)$ , then we know

$$d_{R(f)}^*(s, v) \leq d_{R(f)}^*(s, u) + 1 \leq d_{R(f+g)}^*(s, u) + 1 = d_{R(f+g)}^*(s, v),$$

which contradicts to Equation 2.1.

- Case 2:  $uv \not\subseteq R(f)$ , then since  $uv \subseteq R(f + g)$ , so by 現邊觀察 we have

$$d_{R(f)}^*(s, v) = d_{R(f)}^*(s, u) - 1 \leq d_{R(f+g)}^*(s, u) - 1 = d_{R(f+g)}^*(s, v) - 2,$$

which contradicts to Equation 2.2.

Hence, it is impossible that such  $v$  exists.

■

# Appendix