

CS711008Z Algorithm Design and Analysis

Lecture 10. Algorithm design technique: Network flow and its applications¹

Dongbo Bu

Institute of Computing Technology
Chinese Academy of Sciences, Beijing, China

¹The slides are made based on Chapter 7 of Introduction to algorithms, Combinatorial optimization algorithm and complexity by C. H. Papadimitriou and K. Steiglitz. Some slides are excerpted from the presentation by K. Wayne with permission.

- MAXIMUMFLOW problem: FORD-FULKERSON algorithm,
MAXFLOW-MINCUT theorem;
- A duality explanation of FORD-FULKERSON algorithm and
MAXFLOW-MINCUT theorem;

快速

- Scaling technique to improve FORD-FULKERSON algorithm;

- Solving the dual problem: Push-Relabel algorithm; 最常用

扩展

- Extensions of MAXIMUMFLOW problem: lower bound of
capacity, multiple sources & multiple sinks, indirect graph;

A brief history of MINIMUMCUT problem I

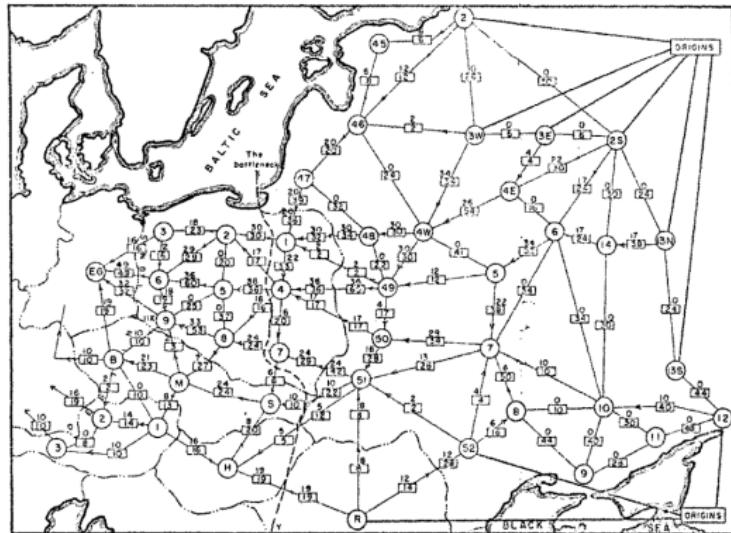


Figure: Soviet Railway network, 1955

- “From Harris and Ross [1955]: Schematic diagram of the railway network of the Western Soviet Union and Eastern European countries, with a maximum flow of value 163,000 tons from Russia to Eastern Europe, and a cut of capacity 163,000 tons indicated as ‘The bottleneck’ ...”
- A recently declassified U.S. Air Force report indicates that the original motivation of minimum-cut problem and Ford-Fulkerson algorithm is to disrupt rail transportation the Soviet Union [A. Shrijver, 2002].
最小割 ← 炸铁路.

A brief history of algorithms to MINIMUMCUT problem

Year	Developers	Time-complexity
1956	Ford and Fulkerson	$O(mC)$ and $O(m^2 \log C)$
1970	Dinitz	$O(n^2m)$
1972	Edmonds and Karp	$O(m^2n)$
1974	Karzanov	$O(n^3)$
1986	Sleator and Tarjan	$O(nm \log n)$
1988	Goldberg and Tarjan	$O(n^2m \log(\frac{n^2}{m}))$
2012	Orlin	<u>$O(nm)$</u>

MAXIMUMFLOW problem

INPUT:

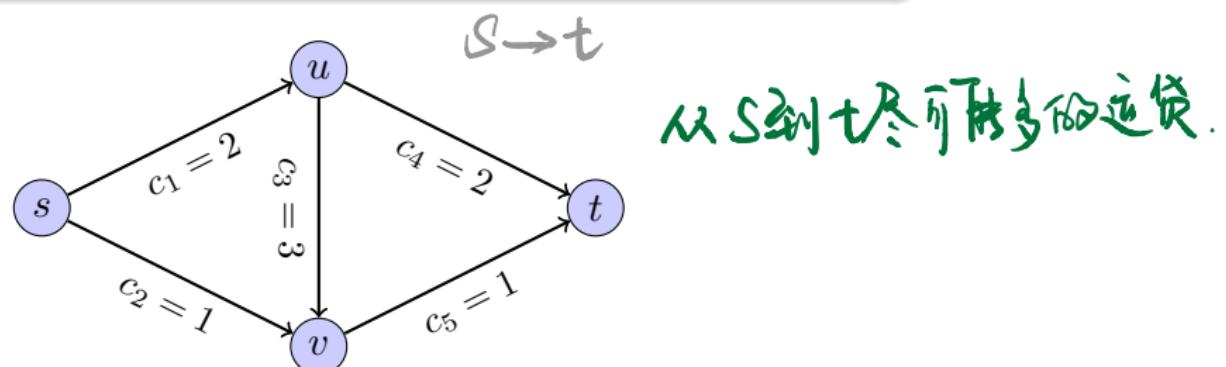
A directed graph $G = \langle V, E \rangle$. Each edge e has a capacity C_e .

Two special points: **source** s and **sink** t ;

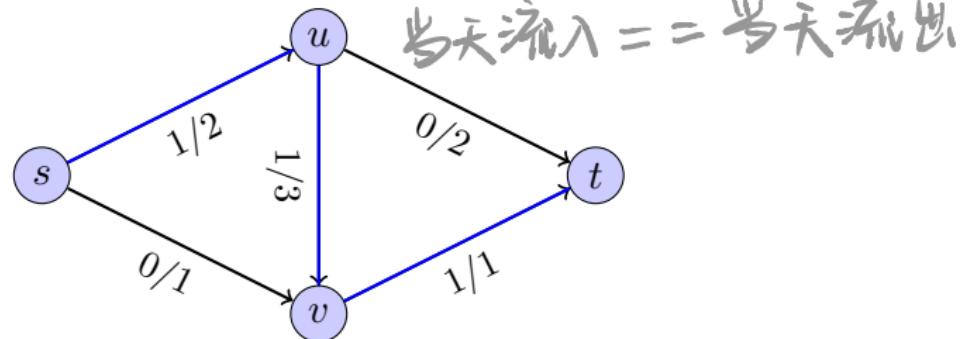
OUTPUT:

For each edge $e = (u, v)$, to assign a flow $f(u, v)$ such that

$\sum_{u,(s,u) \in E} f(s, u)$ is maximized.



Intuition: to push as many commodity as possible from **source** s to **sink** t .



“流”需满足
的两点要求：

Definition (Flow)

$f : E \rightarrow R^+$ is a $s - t$ flow if:

- ① (Capacity constraints): $0 \leq f(e) \leq C_e$ for all edge e ;
- ② (Conservation constraints): for any intermediate vertex $v \in V - \{s, t\}$, $f^{in}(v) = f^{out}(v)$, where $f^{in}(v) = \sum_{e \text{ into } v} f(e)$ and $f^{out}(v) = \sum_{e \text{ out of } v} f(e)$.
(Intuition: input = output for any intermediate vertex.)

The **value of flow f** is defined as $V(f) = f^{out}(s)$. 流量值.

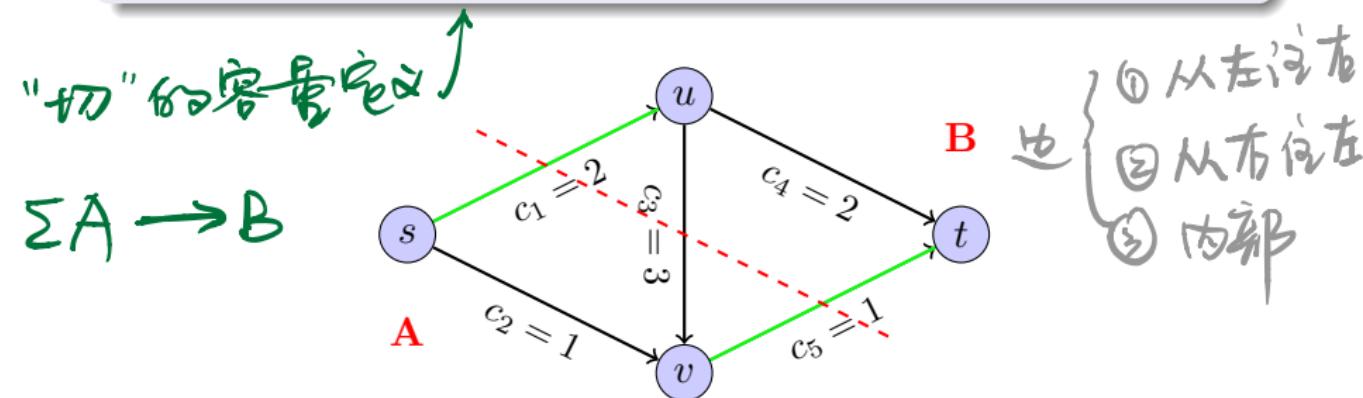
中间节点不持有
剩余.

Definition ($s - t$ cut)

An **$s - t$ cut** is a partition (A, B) of V such that $s \in A$ and $t \in B$.

The **capacity of a cut** (A, B) is defined as

$$C(A, B) = \sum_{e \text{ from } A \text{ to } B} C(e).$$

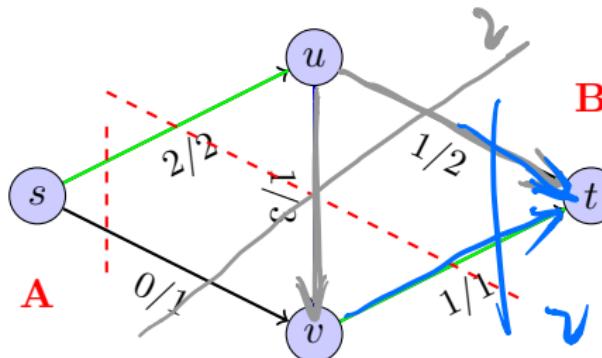


$$C(A, B) = 3$$

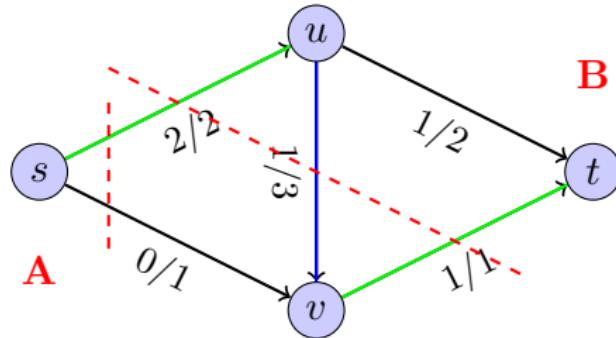
Lemma

(Flow value lemma) Give a flow f . For **any** $s - t$ cut (A, B) , the flow across the cut is a constant $V(f)$. Formally,
 $V(f) = f^{out}(A) - f^{in}(A)$.

流量和割没有关系



$$V(f) = 2 + 0 = 2$$
$$f^{out}(A) - f^{in}(A) = 2 + 1 - 1 = V(f)$$



Proof.

- We have: $\underbrace{f^{out}(v) - f^{in}(v)}$ for any node $v \neq s$ and $v \neq t$.
- Thus, we have:

$$\begin{aligned}
 V(f) &= f^{out}(s) - f^{in}(s) && // \text{Hint: } f^{in}(s) = 0; \\
 &= \sum_{v \in A} (f^{out}(v) - f^{in}(v)) \\
 &= (\sum_{e \text{ from } A \text{ to } B} f(e) + \sum_{e \text{ from } A \text{ to } A} f(e)) \\
 &\quad - (\sum_{e \text{ from } B \text{ to } A} f(e) + \sum_{e \text{ from } A \text{ to } A} f(e)) \\
 &= f^{out}(A) - f^{in}(A) \Rightarrow \text{和割没关系.}
 \end{aligned}$$

FORD-FULKERSON algorithm [1956]

Lester Randolph Ford Jr. and Delbert Ray Fulkerson



Figure: Lester Randolph Ford Jr. and Delbert Ray Fulkerson

- Dynamic programming doesn't seem to work.
- In fact, there is no algorithm known for MAXIMUM FLOW problem that can really be viewed as belonging to the dynamic programming paradigm.
- We know that the MAXIMUMFLOW problem is in **P** since it can be formulated as a linear program (See Lecture 8).
- However, the network structure has its own property to enable a more efficient algorithm, informally called **network simplex**.

网络单纯形.

① 分
② 不分。
③ 聰明的枚举

- Let's return to the general IMPROVEMENT strategy:

IMPROVEMENT(f)

```
1: x = x0; //starting from an initial solution;  
2: while TRUE do  
3:   x = IMPROVE(x); //move one step towards optimum;  
4:   if STOPPING(x,  $f$ ) then  
5:     break;  
6:   end if  
7: end while  
8: return x;
```

- Three key questions:

- ① How to construct an initial solution?

- For MAXIMUMFLOW problem, an initial solution can be easily obtained by setting $f(e) = 0$ for any e (called 0-flow).
 - It is easy to verify that both CONSERVATION and CAPACITY constraints hold for the 0-flow.

最简初始解

- ② How to improve a solution?

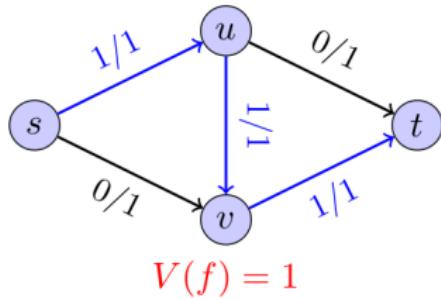
- ③ When shall we stop?

A failure start: augmenting flow along a path in the original graph

- Let p be a simple $s - t$ path in the network G .

- 1: Initialize $f(e) = 0$ for all e .
- 2: **while** there is an $s - t$ path in graph G **do**
- 3: **arbitrarily** choose an $s - t$ path p in graph G ;
- 4: $f = \text{AUGMENT}(p, f)$;
- 5: **end while**
- 6: **return** f ;

迭代一条路径增加该边的容量.



- We define $bottleneck(p, f)$ as the minimum residual capacity of edges in path p .

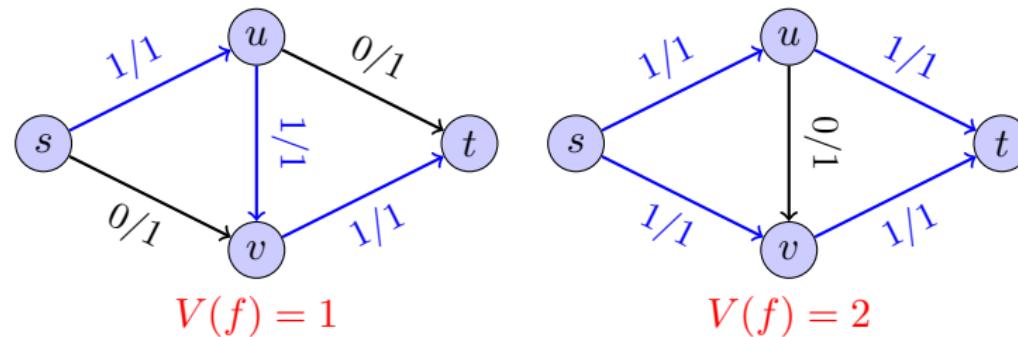
AUGMENT(p, f)

- 1: Let $b = bottleneck(p, f);$
- 2: **for** each edge $e = (u, v) \in p$ **do**
- 3: increase $f(u, v)$ by b ;
- 4: **end for**

选剩余容量。
边的最小容量↑

Why we fail?

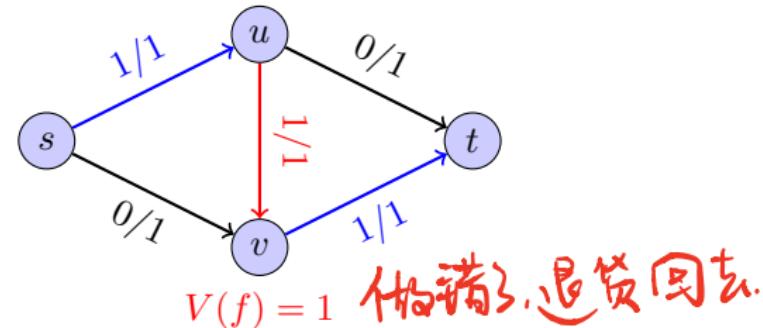
- We start from 0-flow. In order to increase the value of f , we find a $s - t$ path, say $p = s \rightarrow u \rightarrow v \rightarrow t$, to transmit one unit of commodity.
- However we cannot find a $s - t$ path in G to increase f further (left panel) although the maximum flow value is 2 (right panel).



Ford-Fulkerson algorithm: “undo” functionality

- Key observation:

- When constructing a flow f , one might commit errors on some edges, i.e. the edges should not be used to transmit commodity. For example, the edge $u \rightarrow v$ should not be used.

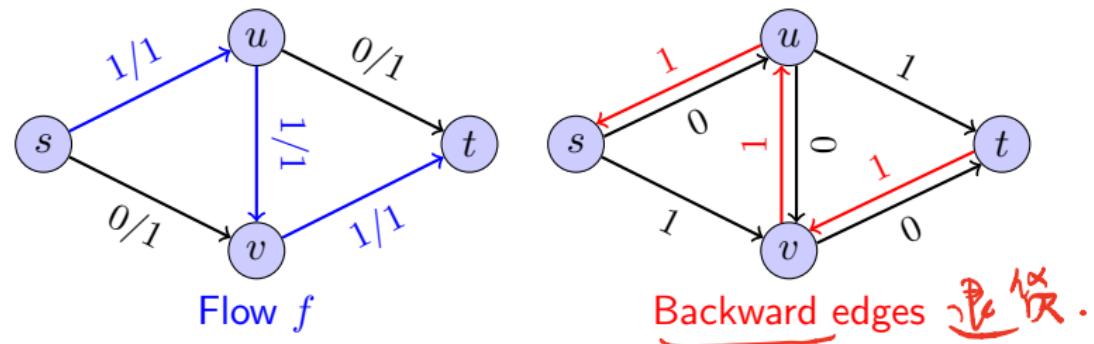


- To improve the flow f , we should work out ways to **correct these errors**, i.e. “undo” the transmission assigned on the edges.

Implementing the “undo” functionality

- But how to implement the "undo" functionality?
- **Adding backward edges!**
- Suppose we add a **backward** edge $v \rightarrow u$ into the original graph. Then we can correct the transmission via pushing back commodity from v to u .

? 怎么知道撤销？
→ 什么时候加逆边.



剩余图

Definition (Residual Graph)

Given a directed graph $G = \langle V, E \rangle$ with a flow f , we define

residual graph $G_f = \langle V, E' \rangle$. For any edge $e = (u, v) \in E$, two edges are added into E' as follows:

对每条边都

- ① (**Forward edge** (u, v) with leftover capacity):

If $f(e) < C(e)$, add edge $e = (u, v)$ with capacity

$$C(e) = C(e) - f(e).$$

前向边

剩余

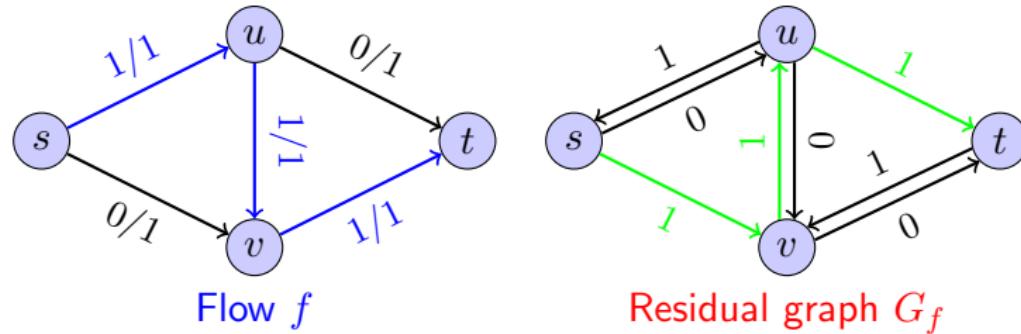
- ② (**Backward edge** (v, u) with undo capacity):

If $f(e) > 0$, add edge $e' = (v, u)$ with capacity $C(e') = f(e)$.

后向边

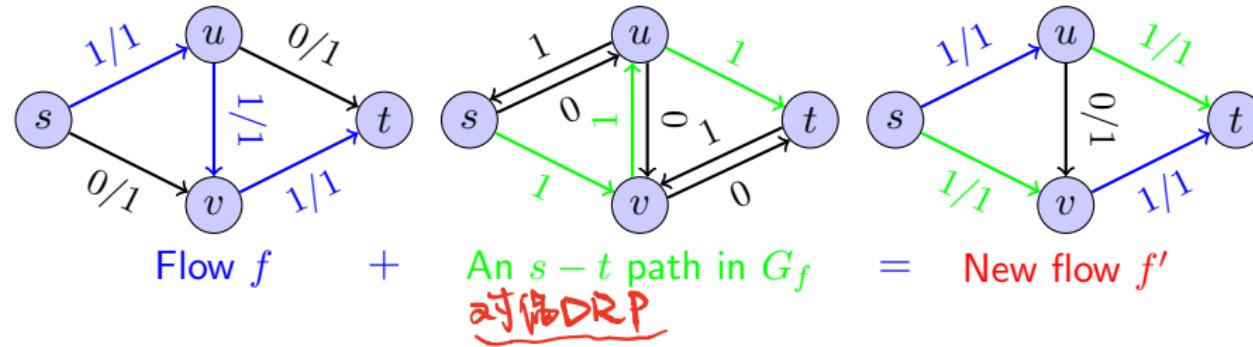
如果这条边过货，则添加后向边。

Finding an $s - t$ path in G_f rather than G



Note: the path contains a backward edge (v, u)

Augmenting flow along the path: from f to f'



Note:

- By using the backward edge $v \rightarrow u$, the initial transmission from u to v is pushed back.
- More specifically, the first commodity transferred through flow f changes its path (from $s \rightarrow u \rightarrow v \rightarrow t$ to $s \rightarrow u \rightarrow t$), while the second one uses the path $s \rightarrow v \rightarrow t$.

- Let p be a simple $s - t$ path in residual graph G_f , called augmentation path. We define $bottleneck(p, f)$ as the minimum capacity of edges in path p .

FORD-FULKERSON algorithm:

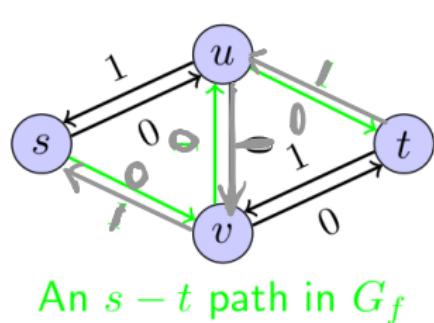
- 1: Initialize $f(e) = 0$ for all e .
- 2: **while** there is an $s - t$ path in residual graph G_f **do**
- 3: **arbitrarily** choose an $s - t$ path p in G_f ;
- 4: $f = \text{AUGMENT}(p, f)$;
- 5: **end while**
- 6: **return** f ;

Correctness and time-complexity analysis

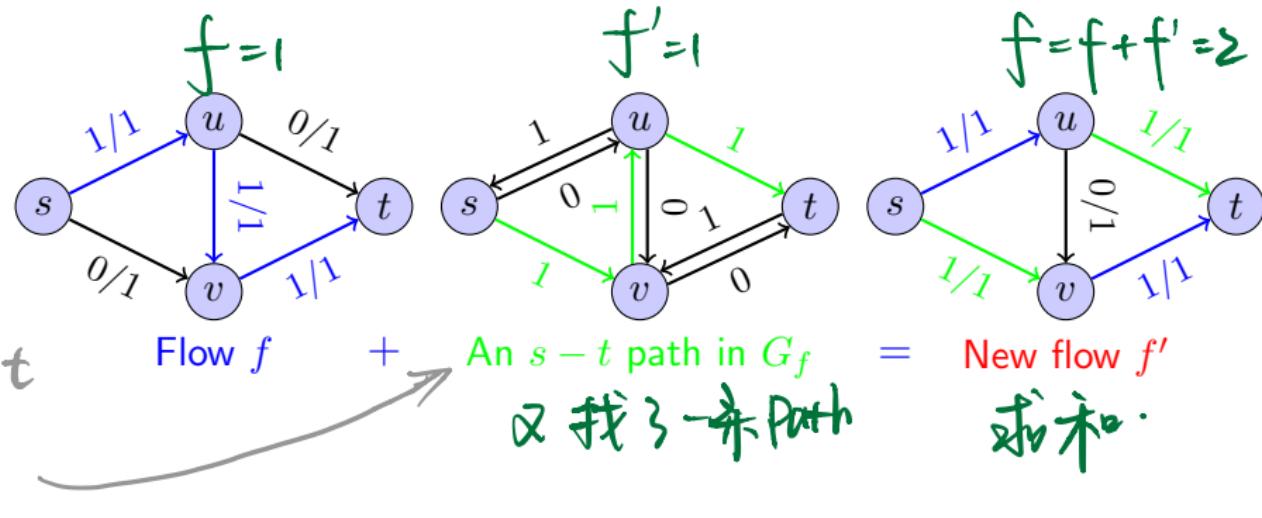
Property 1: augmentation operation generates a new flow

Theorem

The operation $f' = \text{AUGMENT}(p, f)$ generates a new flow f' in G .



There is no path $s-t$
 \Rightarrow end while loop



Proof.

- Checking **capacity constraints**: Consider two cases of edge $e = (u, v)$ in path p :

① (u, v) is a forward edge arising from $(u, v) \in E$:

$$0 \leq f(e) \leq f'(e) = f(e) + \text{bottleneck}(p, f) \leq f(e) + (C(e) - f(e)) \leq C(e)$$

② (u, v) is a backward edge arising from $(v, u) \in E$:

$$C(e) \geq f(e) \geq f'(e) = f(e) - \text{bottleneck}(p, f) \geq f(e) - f(e) = 0$$

- Checking **conservation constraints**:

On each node v , the change of the amount of flow entering v is the same as the change in the amount of flow exiting v .

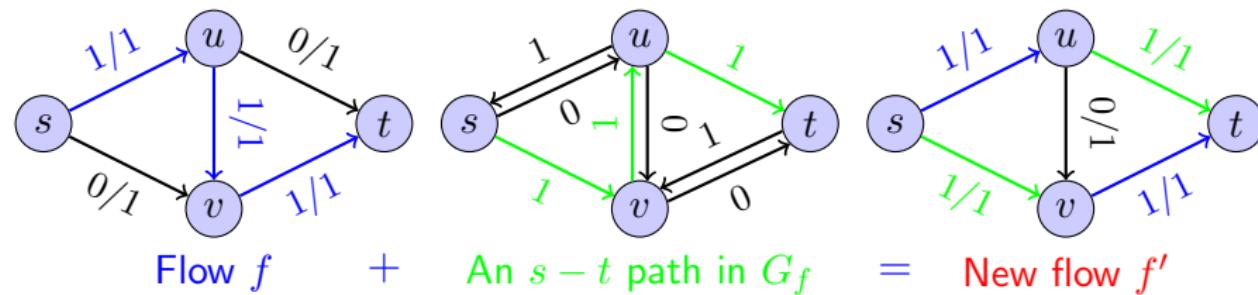


Property 2: Monotonically increasing

单调

Theorem

(Monotonically increasing) $V(f') > V(f)$



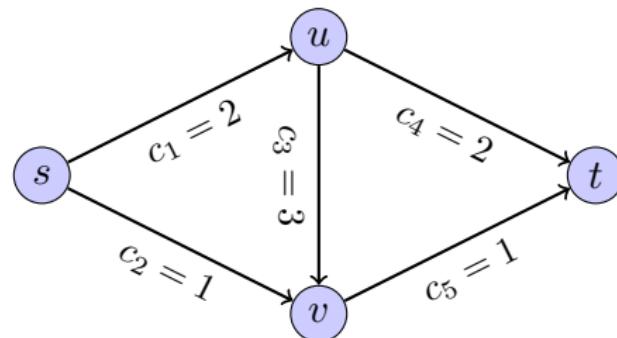
- Hint: $V(f') = V(f) + \text{bottleneck}(p, f) > V(f)$ since $\underbrace{\text{bottleneck}(p, f)}_{> 0} > 0$.

Property 3: a trivial upper bound of flow

Theorem

$V(f)$ has an upper bound $C = \sum_{e \text{ out of } s} C(e).$

(Intuition: the edges out of s are completely saturated with flow.)



Theorem

Assume all edges have integer capacities. At every intermediate stage of the Ford-Fulkerson algorithm, both flow value $V(f)$ and residual capacities are integers. Thus, $\text{bottleneck}(p, f) \geq 1$, and there is at most C iterations of the while loop.

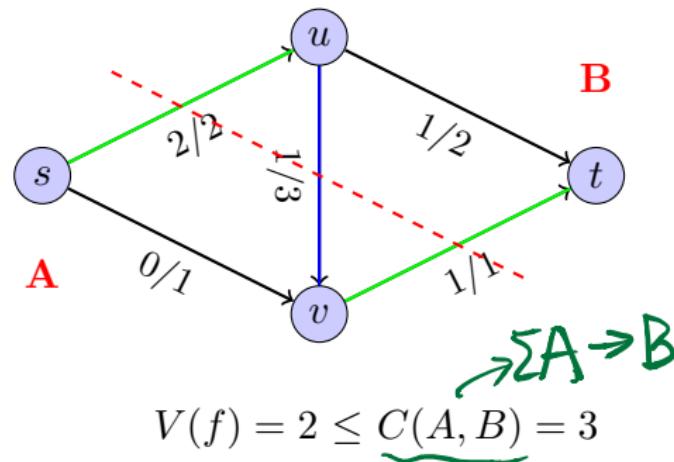
- Intuition: Under a reasonable assumption that all capacities are integers, we have $\text{bottleneck}(p, f) \geq 1$ at every stage; thus, $V(f') \geq V(f) + 1$.
- Time complexity: $O(mC)$.
 - $O(C)$ iterations
 - At each iteration, it takes $O(m + n)$ time to find an $s - t$ path in G_f using DFS or BFS technique.

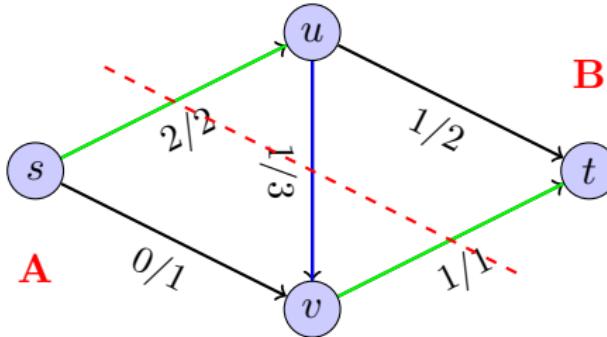


Property 5: A tighter upper bound

Theorem

(Tight upper bound) Given a flow f . For any $s - t$ cut (A, B) , we have $V(f) \leq C(A, B)$.





Proof.

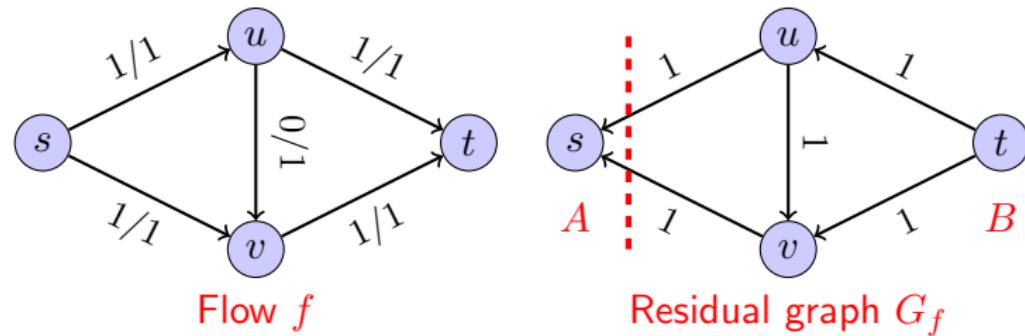
$$\begin{aligned}
 V(f) &= f^{out}(A) - f^{in}(A) && \text{(by flow value lemma)} \\
 &\leq f^{out}(A) && \text{(by } f^{in}(A) \geq 0\text{)} \\
 &= \sum_{e \in A \rightarrow B} f(e) \\
 &\leq \sum_{e \in A \rightarrow B} C(e) && \text{(by } f(e) \leq C(e)\text{)} \\
 &= C(A, B)
 \end{aligned}$$

□

Theorem

FORD-FULKERSON ends up with a maximum flow f and a minimum cut (A, B) .

最大流 = 最小切



Proof.

- FORD-FULKERSON algorithms ends when there is no $s - t$ path in the residual graph G_f .
- Let A be the set of nodes reachable from s in G_f , and $B = V - A$. (A, B) forms a $s - t$ cut. ($A \neq \emptyset, B \neq \emptyset$).
- Consider two types of edges $e = (u, v) \in E$ across cut (A, B) :
 - ① $u \in A, v \in B$: we have $f(e) = C(e)$ (Otherwise, A should be extended to include v since (u, v) is in G_f .)
 - ② $u \in B, v \in A$: we have $f(e) = 0$ (Otherwise, A should be extended to include u since (v, u) is in G_f .)
- Thus we have

$$\begin{aligned}V(f) &= f^{out}(A) - f^{in}(A) \\&= f^{out}(A) \quad (\text{by } f^{in}(A) = 0) \\&= \sum_{e \in A \rightarrow B} f(e) \\&= \sum_{e \in A \rightarrow B} C(e) \quad (\text{by } f(e) = C(e)) \\&= C(A, B)\end{aligned}$$

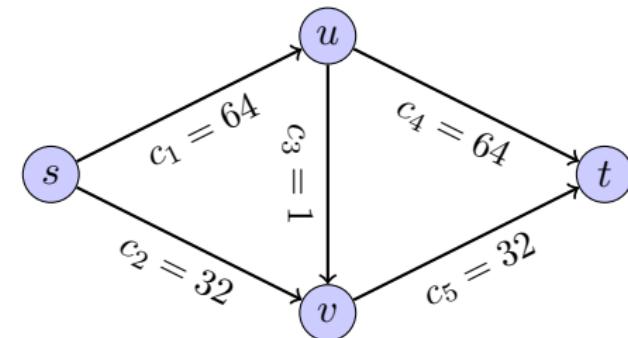
FORD-FULKERSON algorithm: bad example 1

The integer restriction is important

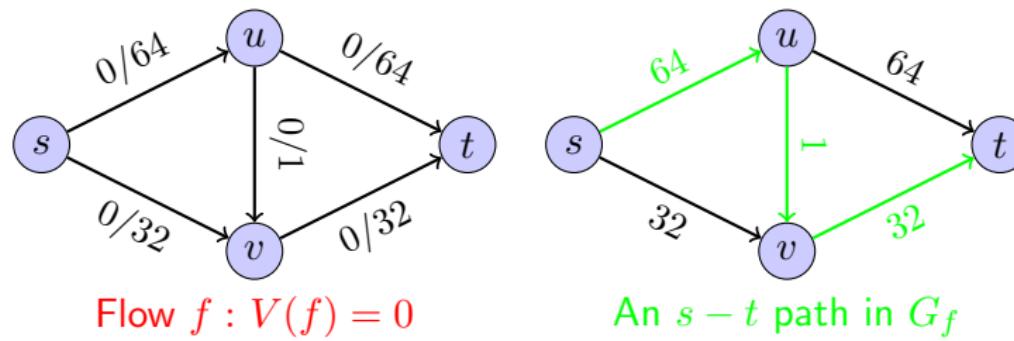
- In the analysis of FORD-FULKERSON algorithm, the integer restriction of capacities is important: the bottleneck edge leads to an increase of at least 1.
以成为整数
- The analysis doesn't hold if the capacities can be irrational.
不为理性的.
- In fact, the flow might be increased by a smaller and smaller number and the iteration will be endless.
- Worse yet, this endless iteration might not converge to the maximum flow.

(See an example by Uri Zwick)

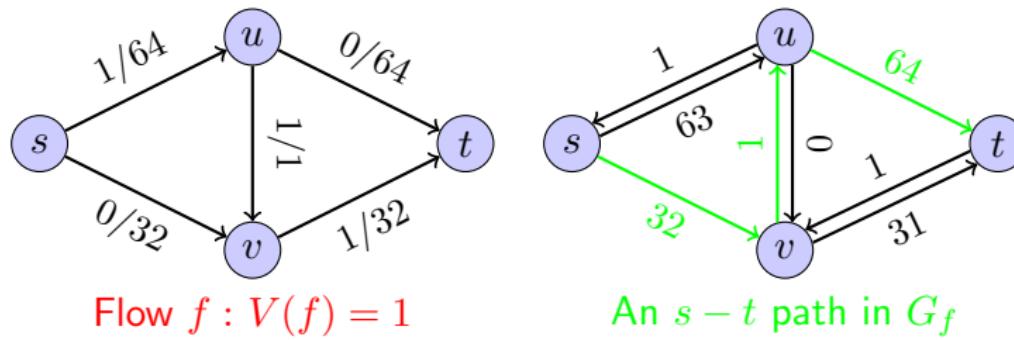
FORD-FULKERSON algorithm: bad example 2



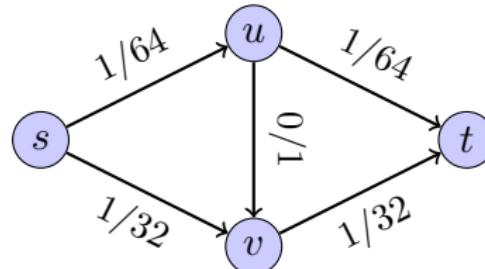
A bad example of FORD-FULKERSON algorithm: Step 1



A bad example of FORD-FULKERSON algorithm: Step 2



A bad example of FORD-FULKERSON algorithm: Step 3



Flow $f : V(f) = 2$

Note:

- ① After two iterations, the problem is similar to the original problem except for the capacities on $(s, u), (s, v), (u, t), (v, t)$ decrease by 1.
- ② Thus, FORD-FULKERSON algorithm will end after 64+32 iterations. (Why? *bottleneck = 1 at all stages.*)

- FORD-FULKERSON algorithm doesn't specify how to choose an augmentation path, leading to some weaknesses:
 - A path with small bottleneck capacity is chosen as augmentation path;
 - We put flow on too many edges than necessary,
- It should be pointed out that the original max-flow paper lists several heuristics for improvement.

BTW Augment Path

- Various strategies to select augmentation path in G_f :
 - ① Fat pipes:
 - To select the augmentation path with the largest bottleneck capacity;
 - Scaling technique: an efficient way to find an augmentation path with **large** improvement;
 - ② Short pipes:
 - Edmonds-Karp: to find **the shortest $s - t$ path** in *BFS tree*.
 - Dinitz' algorithm: to extend *BFS tree* to **layered network**, find augmentation path in the layered network, and perform **amortized** analysis;
 - Dinic's algorithm: running **DFS** to find a collection of augmentation paths (called **blocking flow**) in the **layered network**.
- It should be pointed out that the complexity of EDMONDS-KARP and DINIC'S algorithms do not depend on edge capacities, and thus avoid the limitation of Ford-Fulkerson algorithm for the networks with irrational edge capacities.

Improvement 1: Scaling technique for speed-up

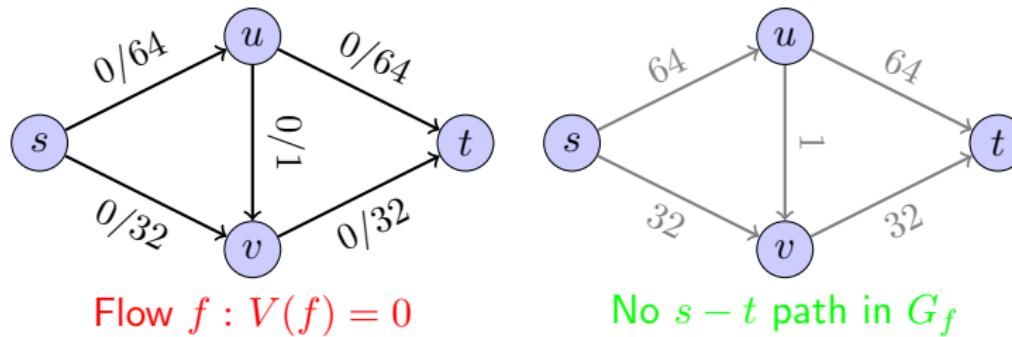
- Question: can we choose a large augmentation path? The larger $bottleneck(p, f)$, the less iterations.
- An $s - t$ path p in G_f with the largest bottleneck(p, f) can be found using binary search, or a slight change of Dijkstra's algorithm in $O(m + n \log n)$ time; however, it is still somewhat inefficient.
- Basic idea: we can relax the "**largest**" requirement to "**sufficiently large**".
- Specifically, we can set up a lower bound Δ for $bottleneck(P, f)$: simply removing the "small" edges, i.e. the edges with capacities less than Δ from $G(f)$. This residual graph is called $G_f(\Delta)$.
- Δ will be scaled as iteration proceeds.

- Scaling FORD-FULKERSON algorithm:

```
1: Initialize  $f(e) = 0$  for all  $e$ .  
2: Let  $\Delta = C$ ; but  $C = \sum_{e \in \text{out}(s)} C(e)$   
3: while  $\Delta \geq 1$  do  
4:   while there is an  $s - t$  path in  $G_f(\Delta)$  do  
5:     choose an  $s - t$  path  $p$ ;  
6:      $f = \text{AUGMENT}(p, f)$ ;  
7:   end while  
8:    $\underline{\Delta = \Delta/2}$ ;  
9: end while  
10: return  $f$ ;
```

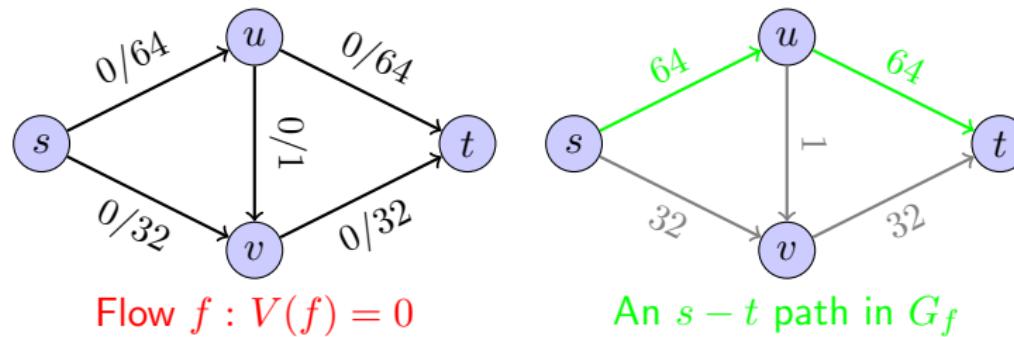
- Intuition: flow is augmented in a large step size whenever possible; otherwise, the step size is reduced. Step size is controlled via removing the “small” edges out of residual graph.
- Note: Δ turns to be 1 finally; thus no edge in residual graph will be neglected.

An example: Step 1



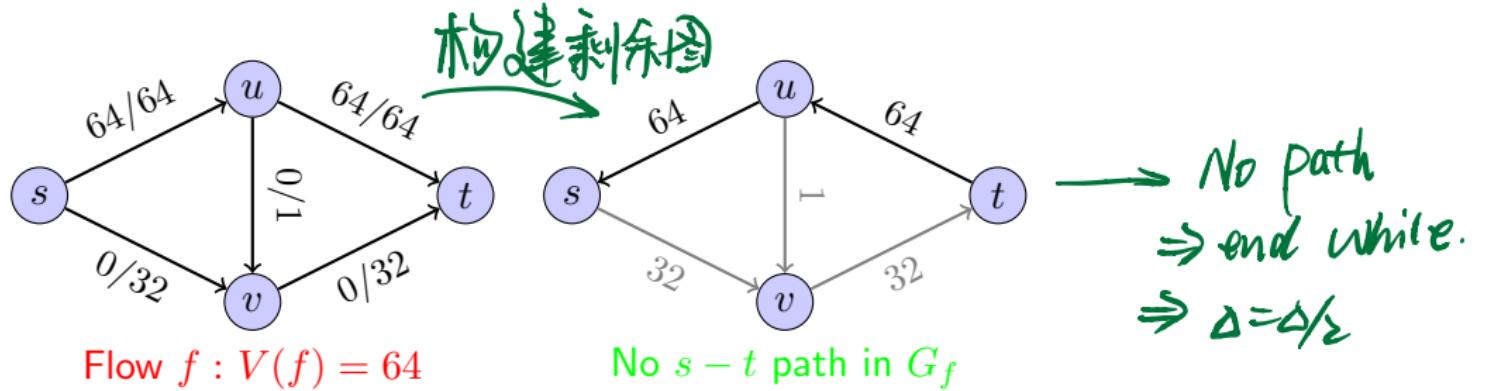
- Flow: 0 flow;
- Δ : $\Delta = \underline{96}$; 64+32
- $G_f(\Delta)$: the edges in light blue were removed since capacities are less than 96.
- $s - t$ path: cannot find. Thus Δ is scaled: $\Delta = \Delta/2 = 48$.

An example: Step 2



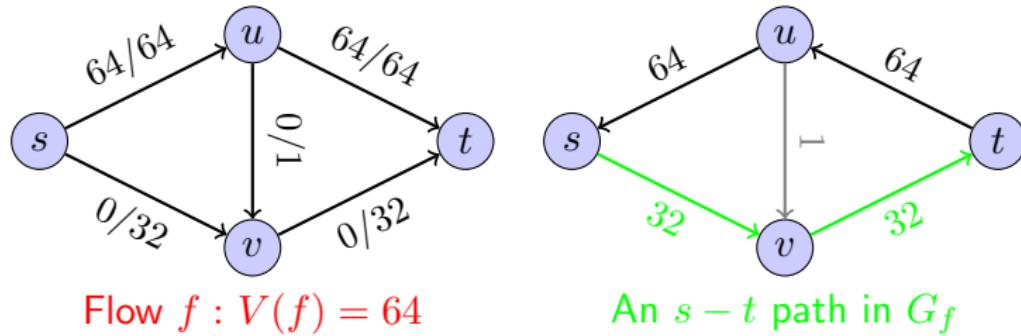
- Flow: 0 flow;
- Δ : $\Delta = 48$;
- $G_f(\Delta)$: the edges in light blue were removed since capacities are less than 48.
- $s - t$ path: a path $s - u - t$ appears. Perform augmentation operation.

An example: Step 3



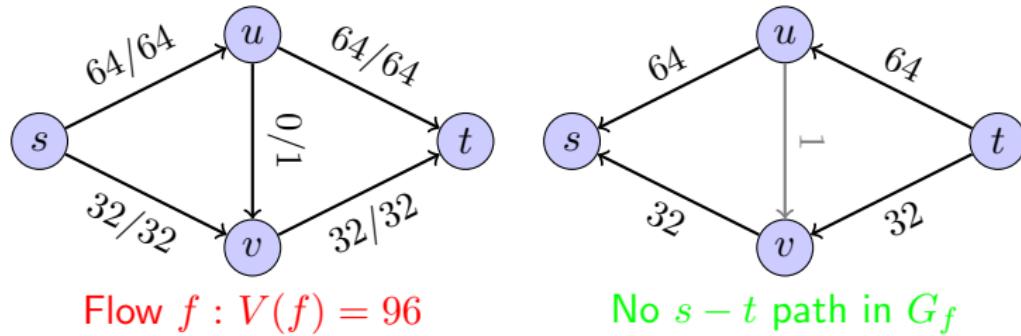
- Flow: 64;
- Δ : $\Delta = 48$;
- $G_f(\Delta)$: the edges in light blue were removed since capacities are less than 48.
- $s - t$ path: no path found. Perform scaling: $\Delta = \Delta/2 = 24$.

An example: Step 4



- Flow: 64;
- Δ : $\Delta = 24$;
- $G_f(\Delta)$: the edges in light blue were removed since capacities are less than 24.
- $s - t$ path: find a path: $s - v - t$. Perform augmentation.

An example: Step 5



- Flow: 96. Maximum flow obtained.
 - $\Delta: \Delta = 24;$
 - $G_f(\Delta)$: the edges in light blue were removed since capacities are less than 24.
 - $s - t$ path: cannot find a $s - t$ path.
- 可以擇到停止

Theorem

(Outer while loop number) The while iteration number is at most $1 + \log_2 C$.

SCALING FORD-FULKERSON algorithm:

- 1: Initialize $f(e) = 0$ for all e .
- 2: Let $\Delta = C$;
- 3: **while** $\Delta \geq 1$ **do**
- 4: **while** there is an $s - t$ path in $G_f(\Delta)$ **do**
- 5: choose an $s - t$ path p ;
- 6: $f = \text{AUGMENT}(p, f)$;
- 7: **end while**
- 8: $\Delta = \Delta/2$; *$\log_2 C$*
- 9: **end while**
- 10: **return** f ;

Theorem

(Inner while loop number) In a scaling phase, the number of augmentations is at most $2m$.

SCALING FORD-FULKERSON algorithm:

- 1: Initialize $f(e) = 0$ for all e .
 - 2: Let $\Delta = C$;
 - 3: **while** $\Delta \geq 1$ **do**
 - 4: **while** there is an $s - t$ path in $G_f(\Delta)$ **do**
 - 5: choose an $s - t$ path p ;
 - 6: $f = \text{AUGMENT}(p, f)$;
 - 7: **end while**
 - 8: $\Delta = \Delta/2$;
 - 9: **end while**
 - 10: **return** f ;
- $s-t$ 至多有 $2m$ 条.

Proof.

- ① Let f be the flow that a Δ -scaling phase ends up with, and f^* be the maximum flow. We have $V(f) \geq V(f^*) - m\Delta$. (Intuition: $V(f)$ is not too bad; the distance to maximum flow is small.)
- ② In the subsequent $\frac{\Delta}{2}$ -scaling phase, each augmentation will increase $V(f)$ at least $\frac{\Delta}{2}$.

Thus, there are at most $2m$ augmentations in the $\frac{\Delta}{2}$ -scaling phase. \square

- Time-complexity: $O(\underbrace{m^2} \log_2 C)$.
 - $O(\log_2 C)$ outer while loop;
 - $O(m)$ inner loops;
 - Each augmentation takes $O(m)$ time.

遍历一遍 path 上的边

Proof.

- Let A be the set of nodes reachable from s in the residual graph $G_f(\Delta)$, and $B = V - A$. Thus (A, B) forms a cut ($A \neq \emptyset, B \neq \emptyset$).
- Consider two types of edges $e = (u, v) \in E$.
 - $u \in A, v \in B$: we have $f(e) \geq C(e) - \Delta$ (Otherwise, A should be extended to include v since (u, v) in $G_f(\Delta)$.)
 - $v \in A, u \in B$: we have $f(e) \leq \Delta$ (Otherwise, A should be extended to include v since (u, v) in $G_f(\Delta)$.)
- Thus we have:

$$\begin{aligned}V(f) &= \sum_{e \in A \rightarrow B} f(e) - \sum_{e \in B \rightarrow A} f(e) \\&\geq \sum_{e \in A \rightarrow B} (C(e) - \Delta) - \sum_{e \in B \rightarrow A} \Delta \\&\geq \sum_{e \in A \rightarrow B} C(e) - m\Delta \\&= C(A, B) - m\Delta \\&\geq V(f^*) - m\Delta\end{aligned}$$

Implementation 2: Edmonds-Karp algorithm using $\underline{O(m^2n)}$ time

Edmonds-Karp algorithm [1972]



Figure: Jack Edmonds, and Richard Karp

Note: The algorithm was first published by Yefim Dinic in 1970 and independently published by Jack Edmonds and Richard Karp in 1972.

EDMONDS-KARP algorithm:

- 1: Initialize $f(e) = 0$ for all e .
- 2: **while** there is a $s - t$ path in G_f **do**
- 3: choose **the shortest $s - t$** path p in G_f using BFS ;
- 4: $f = \text{AUGMENT}(p, f)$;
- 5: **end while**
- 6: **return** f ;

(a demo)

Theorem

Edmonds-Karp algorithm runs in $O(m^2n)$ time.

Proof.

- During the execution of Edmonds-Karp algorithm, an edge $e = (u, v)$ serves as **bottleneck** edge at most $\frac{n}{2}$ times
- Thus, the while loop will be executed at most $\frac{n}{2}m$ times since there are m edges in total
- It takes $O(m)$ time to find the shortest path using BFS, and augment flow using the path.



Lemma

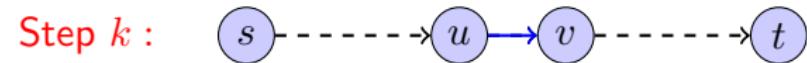
An edge $e = (u, v)$ serves as a **bottleneck** edge at most $\frac{n}{2}$ times.

Proof.

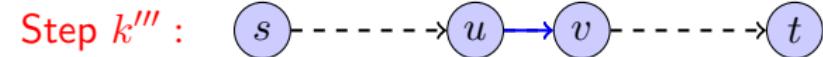
- For a residual graph G_f , we first categorize all nodes into levels L_0, L_1, \dots , where $L_0 = \{s\}$, and L_i contains all nodes v such that the shortest path from s to v has i edges. We use $L(u)$ to denote the level number of node u .
- Consider the two consecutive occurrences of edge $e = (u, v)$ in G_f as bottleneck, say at step k and step k'' .
- We have $L(v) = L(u) + 1$ at step k , and after flow augmentation, the bottleneck edge $e = (u, v)$ will be reversed in G_f .
- At step k'' , $e = (u, v)$ becomes a **bottleneck** edge again.
- This means that $e' = (v, u)$ should be reversed first before step k'' , say at step k''' . We have $L''(u) = L''(v) + 1$. Thus $L''(u) = L''(v) + 1 \geq L'(v) + 1 \geq L(u) + 2$.
- For any node, its maximal level is at most n and its level never decreases (why?). Thus the lemma holds.

Analysis of the Edmonds-Karp algorithm

$$L(u) \quad L(v) \quad L(v) = L(u) + 1$$

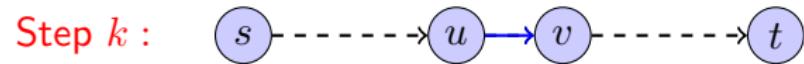


$$L'''(u) \quad L'''(v) \quad L'''(u) \geq L(u) + 2$$

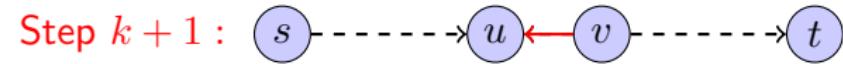


Analysis of the Edmonds-Karp algorithm

$$L(u) \quad L(v) \quad L(v) = L(u) + 1$$

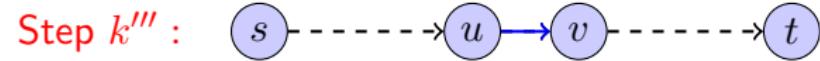


$$L'(u) \quad L'(v) \quad L'(v) \geq L(v)$$



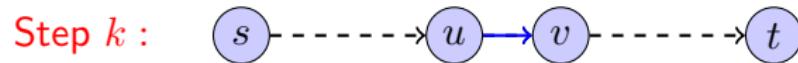
:

$$L'''(u) \quad L'''(v) \quad L'''(u) \geq L(u) + 2$$

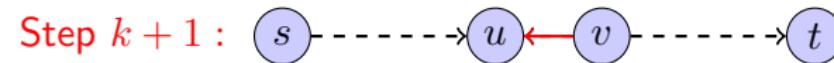


Analysis of the Edmonds-Karp algorithm

$$L(u) \quad L(v) \quad L(v) = L(u) + 1$$



$$L'(u) \quad L'(v) \quad L'(v) \geq L(v)$$



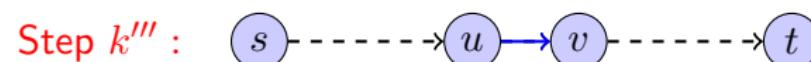
⋮

$$L''(u) \quad L''(v) \quad L''(u) = L''(v) + 1$$



⋮

$$L'''(u) \quad L'''(v) \quad L'''(u) \geq L(u) + 2$$



Lemma

For any node v , its shortest-path distance $L_f(v)$ in residual graph G_f never decreases during the execution of Edmonds-Karp algorithm.

Proof.

- Consider a flow f , and the Edmonds-Karp algorithm selects the shortest-path p from s to t in the residual graph G_f for augmentation, forming a new flow f' .
- Assume for contradiction that there is a node v such that $L_{f'}(v) < L_f(v)$
- Without loss of generality, let v be such a node with the minimum $L_{f'}(v)$ in $G_{f'}$. Let $p' = s \rightsquigarrow u \rightarrow v$ be the shortest-path to v in G'_f . Thus we have:
 - $L_{f'}(v) = L_{f'}(u) + 1$
 - $L_{f'}(u) \geq L_f(u)$ due to the selection of v .
- Let's investigate whether $u \rightarrow v$ appears in G_f . There are two cases:
 - $u \rightarrow v \in G_f$
 - $u \rightarrow v \notin G_f$ but $\in G_{f'}$and in both cases, a contradiction occurs. Thus the lemma follows.

- $u \rightarrow v \in G_f$:
 - We have $L_f(u) + 1 \geq L_f(v)$.
 - Thus $L_{f'}(v) = L_{f'}(u) + 1 \geq L_f(u) + 1 \geq L_f(v)$. A contradiction with the assumption.
- $u \rightarrow v \notin G_f$ but $\in G_{f'}$:
 - This can happen only if $v \rightarrow u$ exists in p , one of the shortest paths from s to t in G_f .
 - Thus, we have $L_f(u) = L_f(v) + 1$.
 - Thus another contradiction occurs:
$$L_{f'}(v) = L_{f'}(u) + 1 \geq L_f(u) + 1 = L_f(v) + 2.$$

Implementation 3: Dinitz' algorithm and its variant Dinic's algorithm

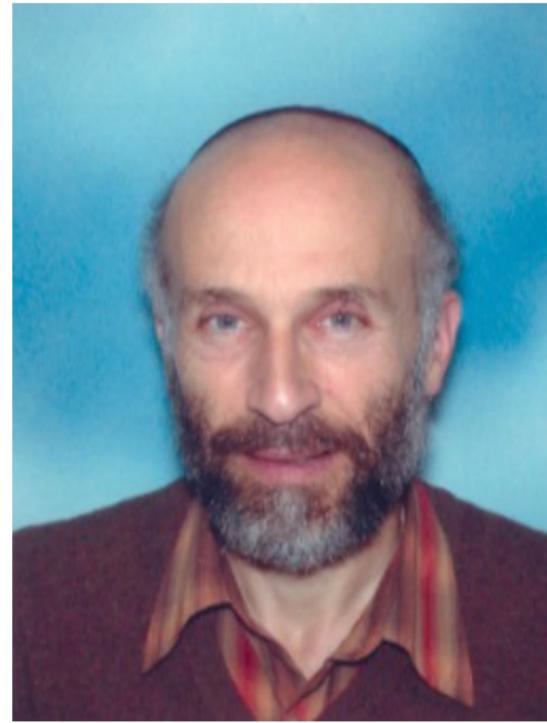


Figure: Yefim Dinitz

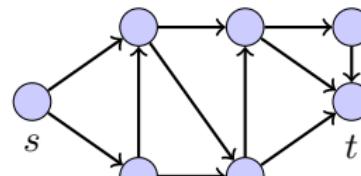
The original Dinitz' algorithm

Motivations:

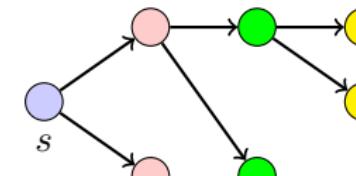
- The initial intention was just to accelerate FORD-FULKERSON algorithm by means of a smart data structure;
- Notice that finding an augmentation path takes $O(m)$ time and becomes a bottleneck of FORD-FULKERSON algorithm;
- It is valuable to save all information achieved at a BFS search for subsequent iterations;
- Specifically, the **BFS tree** is enriched to **layered network**:
 - BFS tree: includes only the first edge found to a node v ;
 - Layered network: keeping all the edges residing on all the shortest $s - v$ path. Note that layered network has an advantage to record all shortest $s - t$ paths.

水流下阶有最短路。

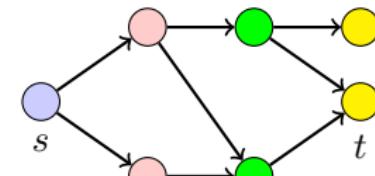
D, 进下一级。



Residual graph G_f



BFS tree



Layered network G_L

- Shimon Even and Alon Itai understood the paper by Y. Dinitz and that by A. Karzanov except for the **layered network maintenance** (removing the “dead-end” nodes). The gaps were spanned by using:
 - ① **blocking flow** (first proposed by A. Karzanov) to prove that the levels of layered network increases from phase to phase; **level ↑**
 - ② **DFS** to search an augmentation path guided by the layered network.
- Note: when running on bi-partite graph, the Dinic's algorithm turns into the Hopcroft-Karp algorithm.

```

1: Initialize  $f(e) = 0$  for all  $e$ .
2: while TRUE do
3:   Construct layered network  $G_L$  from residual graph  $G_f$ 
      using BFS;
4:   if  $dist(s, t) = \infty$  then 没有这个出循环方法.
5:     break;
6:   end if
7:   find a blocking flow  $f'$  in  $G_L$  using DFS technique guided
      by the layered network;
8:   augment flow  $f$  by  $f'$ ;
9: end while
10: return  $f$ ;

```

- The execution of the algorithm can be divided into phases, each phase consisting of construction of layered network, and finding blocking flow in it.

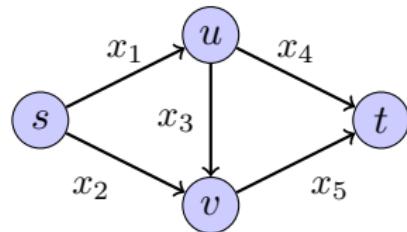
- Here, a **blocking flow** refers to a flow such that in the corresponding residual graph, there is no $s - t$ path.
- Intuition: after acquiring a layered network using $O(m)$ time, a blocking flow (containing a collection of $s - t$ paths) is found for further augmentation. In contrast, EDMONDS-KARP algorithm augment only one $s - t$ path.
→ 每次找一组 $s - t$ path.

(a demo here)

- Total time: $O(mn^2)$
 - #WHILE = $O(n)$. (why? Each phase increases the level of t by at least 1. Similar argument to the analysis of Edmonds-Karp algorithm.)
 - At each step, it takes $O(m)$ time to construct layered network using extended BFS;
 - and it takes $O(mn)$ time to find blocking flow. The reason is:
 - ① it takes $O(n)$ time to find a $s - t$ path using DFS in a layered network;
 - ② at least one bottleneck edge in the path will be saturated;
 - ③ thus it needs at most m iterations to find a blocking flow;

Understanding network-flow from the dual point of view

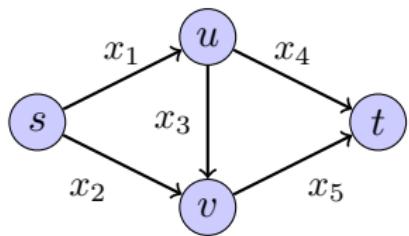
Duality explanation of MaxFlow-MinCut: Dual problem



DUAL: set variables for **edges** (Intuition: x_i denotes *flow* via edge i)

$$\begin{array}{lllll}
 \max & & & f & \\
 \text{s.t.} & x_1 & +x_2 & -f & = 0 \text{ vertex } s \\
 & & & -x_4 & -x_5 & +f & = 0 \text{ vertex } t \\
 & -x_1 & & +x_3 & +x_4 & & = 0 \text{ vertex } u \\
 & & -x_2 & -x_3 & & +x_5 & = 0 \text{ vertex } v \\
 & x_1 & & & & & \leq C_1 \\
 & x_2 & & & & & \leq C_2 \\
 & & x_3 & & & & \leq C_3 \\
 & & & x_4 & & & \leq C_4 \\
 & & & & x_5 & & \leq C_5 \\
 & x_1, & x_2, & x_3, & x_4, & x_5 & \geq 0
 \end{array}$$

An equivalent version



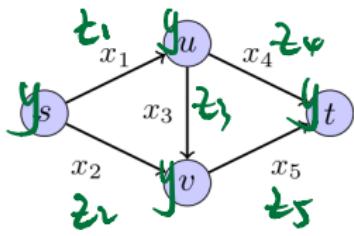
$$\left. \begin{array}{l} \textcircled{1} + \textcircled{2} + \textcircled{3}: \\ x_2 + x_3 - x_5 \leq 0 \\ \textcircled{4}: \\ -x_2 - x_3 + x_5 \leq 0 \end{array} \right\} \Rightarrow -x_2 - x_3 + x_5 = 0$$

? ↘ 拆化

					f			
\max								
$s.t.$	x_1	$+x_2$			$-f$	≤ 0	vertex s	
			$-x_4$	$-x_5$	$+f$	≤ 0	vertex t	
	$-x_1$		$+x_3$	$+x_4$		≤ 0	vertex u	
		$-x_2$	$-x_3$	$+x_5$		≤ 0	vertex v	
	x_1					$\leq C_1$		
	x_2					$\leq C_2$		
		x_3				$\leq C_3$		
			x_4			$\leq C_4$		
				x_5		$\leq C_5$		
	$x_1, \quad x_2, \quad x_3, \quad x_4, \quad x_5$					≥ 0		

Note: the constraints (1), (2), (3), and (4) force
 $-x_2 - x_3 + x_5 = 0$. So do other constraints.

Duality explanation of MaxFlow-MinCut: Primal problem



PRIMAL: set variables for nodes.

$$\begin{array}{llllllll}
 \min & & C_1z_1 & +C_2z_2 & +C_3z_3 & +C_4z_4 & +C_5z_5 \\
 \text{s.t.} & y_s & -y_u & +z_1 & & & \geq 0 \\
 & y_s & & -y_v & +z_2 & & \geq 0 \\
 & & y_u & -y_v & & +z_3 & \geq 0 \\
 & -y_t & +y_u & & & & +z_4 & \geq 0 \\
 & -y_t & & +y_v & & & +z_5 & \geq 0 \\
 & \underline{-y_s} & \underline{+y_t} & & & & & \geq 1 \\
 & \underline{y_s}, & \underline{y_t}, & y_u, & y_v, & z_1, & z_2, & z_3, & z_4, & z_5 & \geq 0
 \end{array}$$

Note:

- ① Since the constraints involves the difference among y_s, y_u, y_v and y_t , one of them can be fixed without effects. Here, we fix $y_s = 0$. Thus we have $y_t \geq 1$ (by the constraint $-y_s + y_t \geq 1$).
- ② Constraint (4) requires $z_4 \geq y_t - y_u$, and the objective is to minimize a function containing C_4z_4 , forcing $y_t = 1$.
- ③ Constraint (1) requires $z_1 \geq y_u$, and the objective is to minimize a function containing C_1z_1 , forcing $z_1 = y_u$. So

PRIMAL: set variables for **nodes**.

$$y_s \geq 0, y_t \geq 1$$

$$\begin{array}{lllllll}
 \min & C_1 z_1 & + C_2 z_2 & + C_3 z_3 & + C_4 z_4 & + C_5 z_5 \\
 s.t. & -y_u & +z_1 & & & & = 0 \\
 & -y_v & & +z_2 & & & = 0 \\
 & y_u & -y_v & & +z_3 & & \geq 0 \\
 & y_u & & & & +z_4 & \geq 1 \\
 & & y_v & & & +z_5 & \geq 1 \\
 & y_s & & & & & = 0 \\
 & y_t & & & & & = 1 \\
 & y_u, & y_v, & z_1, & z_2, & z_3, & z_4, & z_5 & \geq 0
 \end{array}$$

Note: the coefficient matrix of constraints (3), (4) and (5) is totally uni-modular, implying the optimal solution is an integer solution.

PRIMAL: set variables for **nodes**.

$$\begin{array}{lllllll}
 \min & C_1 z_1 & + C_2 z_2 & + C_3 z_3 & + C_4 z_4 & + C_5 z_5 & \\
 s.t. & -y_u & +z_1 & & & & = 0 \\
 & & -y_v & +z_2 & & & = 0 \\
 & y_u & -y_v & & +z_3 & & \geq 0 \\
 & y_u & & & & +z_4 & \geq 1 \\
 & & y_v & & & +z_5 & \geq 1 \\
 & y_s & & & & & = 0 \\
 & y_t & & & & & = 1 \\
 & y_u, & y_v, & z_1, & z_2, & z_3, & z_4, & z_5 & = 0/1
 \end{array}$$

MaxFlow-MinCut: strong duality

$$\begin{array}{lllllll}
 \min & C_1 z_1 & +C_2 z_2 & +C_3 z_3 & +C_4 z_4 & +C_5 z_5 & \\
 s.t. & -y_u & +z_1 & & & & = 0 \\
 & & -y_v & +z_2 & & & = 0 \\
 & y_u & -y_v & & +z_3 & & \geq 0 \\
 & & y_u & & & +z_4 & \geq 1 \\
 & & & y_v & & & +z_5 \geq 1 \\
 & y_s & & & & & = 0 \\
 & y_t & & & & & = 1 \\
 & y_u, & y_v, & z_1, & z_2, & z_3, & z_4, & z_5 = 0/1
 \end{array}$$

Observations:

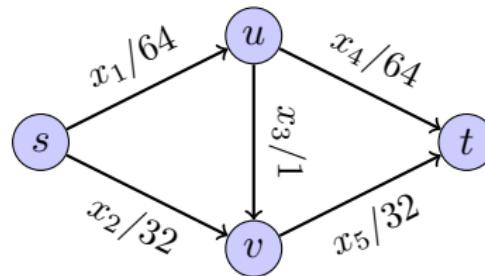
- Intuition of primal variables: if node i is in A , $y_i = 0$; and $y_i = 1$ otherwise.
- The primal problem is essentially to find a cut. (Note: $z_1 = 1$ iff $y_s = 0$ and $y_u = 1$, i.e., edge (s, u) is a cut edge.)
- By weak duality, we have $f \leq c$. This is exactly the MaximumFlow-MinimumCut theorem.

FORD-FULKERSON algorithm is essentially a primal-dual algorithm

- Recall that the generic primal-dual algorithm can be described as follows.

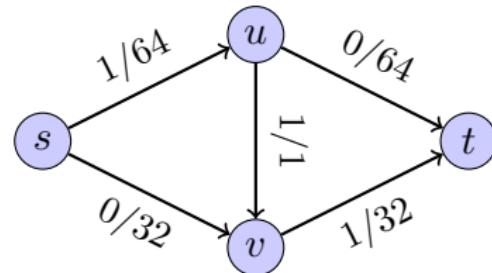
```
1: Initialize  $x$  as a dual feasible solution;  
2: while TRUE do  
3:   Construct DRP corresponding to  $x$ ;  
4:   Let  $\omega_{opt}$  be the optimal solution to DRP;  
5:   if  $\omega_{opt} = 0$  then  
6:     return  $x$ ;  
7:   else  
8:     Improve  $x$  according to the optimal solution to DRP;  
9:   end if  
10:  end while
```

- We will show that solving the DRP is equivalent to finding an augmentation path in residual graph.



- DUAL D: set variables for edges:

$$\begin{array}{lllll}
 \max & & & f \\
 \text{s.t.} & x_1 & +x_2 & -f & \leq 0 \text{ vertex } s \\
 & & & -x_4 & -x_5 & +f & \leq 0 \text{ vertex } t \\
 & -x_1 & & +x_3 & +x_4 & & \leq 0 \text{ vertex } u \\
 & & -x_2 & -x_3 & & +x_5 & \leq 0 \text{ vertex } v \\
 & x_1 & & & & & \leq 64 \\
 & x_2 & & & & & \leq 32 \\
 & x_3 & & & & & \leq 1 \\
 & & & x_4 & & & \leq 64 \\
 & & & & x_5 & & \leq 32 \\
 & x_1, & x_2, & x_3, & x_4, & x_5 & \geq 0
 \end{array}$$



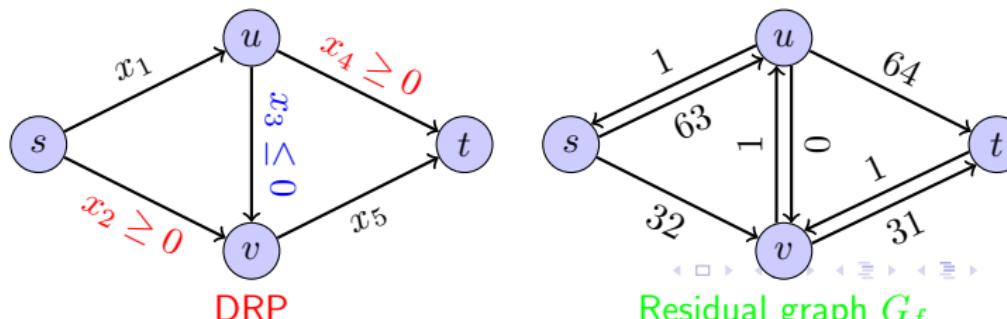
- Let's consider a dual feasible solution $\mathbf{x} = (1, 0, 1, 0, 1)$. Recall how to write DRP from D :
 - Replacing the right-hand side C_i with 0;
 - Adding constraints: $x_i \leq 1$, $f \leq 1$;
 - Keep only the tight constraints J . Here we category J into two sets, i.e. $J = J^S \cup J^E$, where the saturated arcs $J^S = \{i | x_i = C_i\}$, and the empty arcs $J^E = \{i | x_i = 0\}$. Here, $J_S = \{3\}$, and $J_E = \{2, 4\}$.

DRP corresponds to finding an augmentation path

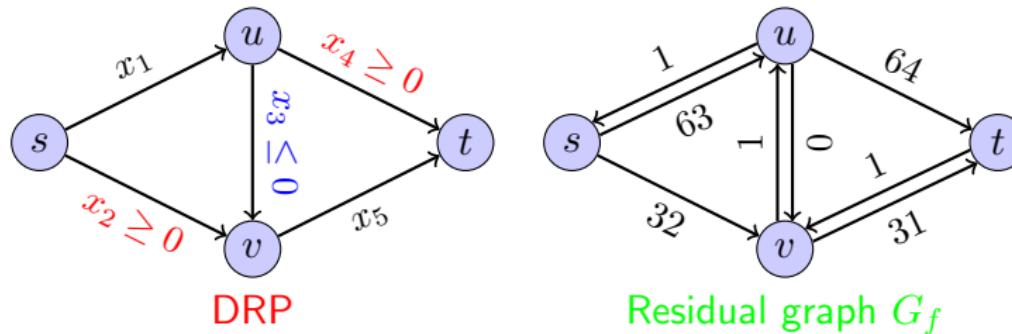
- DRP:

$$\begin{array}{llllll}
 \max & & & f & \\
 \text{s.t.} & x_1 & +x_2 & -f & = 0 & \text{vertex } s \\
 & & & -x_4 & -x_5 & +f = 0 & \text{vertex } t \\
 & -x_1 & & +x_3 & +x_4 & & = 0 & \text{vertex } u \\
 & & -x_2 & -x_3 & & +x_5 & = 0 & \text{vertex } v \\
 & & & x_i & & & \leq 0 & i \in J^S \\
 & & & x_j & & & \geq 0 & j \in J^E \\
 & x_1, & x_2, & x_3, & x_4, & x_5, & f & \leq 1
 \end{array}$$

- $\omega_{OPT} = 0$ implies that optimal solution is found.
- $\omega_{OPT} = 1$ implies a $s - t$ path (with unit flow) in residual graph G_f .



DRP and augmentation path in residual graph



- Note that DRP corresponds to finding an augmentation path in the residual graph G_f .
 - $x_i \leq 0, i \in J^S$ denotes a backward edge,
 - $x_j \geq 0, j \in J^E$ denotes a forward edge,
 - and for other edges, there is no restriction for x_i .
- Thus FORD-FULKERSON algorithm is essentially a primal_dual algorithm.

Push-relabel algorithm [A. V. Goldberg, R. E. Tarjan, 1986]

The push-relabel algorithm is one of the most efficient algorithms to compute a maximum flow. The general algorithm has $O(n^2m)$ time complexity, while the implementation with FIFO vertex selection rule has $O(n^3)$ running time, the highest active vertex selection rule provides $O(n^2\sqrt{m})$ complexity, and the implementation with Sleator's and Tarjan's dynamic tree data structure runs in $O(nm\log(n^2/m))$ time. In most cases it is more efficient than the Edmonds-Karp algorithm, which runs in $O(nm^2)$ time.

- Basic idea: the optimal solution f should meet two constraints simultaneously, namely, f is a flow, and there is no $s - t$ path in the residual graph G_f . FORD-FULKERSON algorithms maintains the first constraint while PUSH-RELABEL maintains the second constraint.
 - ① Ford-Fulkerson: set variables for edges. Update flow on edges until G_f has no $s - t$ path;
 - ② Push-relabel: set variables for nodes. Update a pre-flow f , maintaining the property that G_f has no $s - t$ path, until f is a flow.

预流

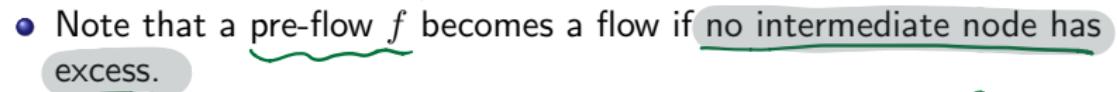
Definition (Pre-flow)

f is a pre-flow if

- (Capacity condition): $f(e) \leq C(e)$; 易满足.

 $E_f(v) = \sum_{e \text{ into } v} f(e) - \sum_{e \text{ out of } v} f(e) \geq 0$
宽松. ($E_f = E_{\text{into } v} f(e) - \sum_{e \text{ out of } v} f(e)$)

进 > 出



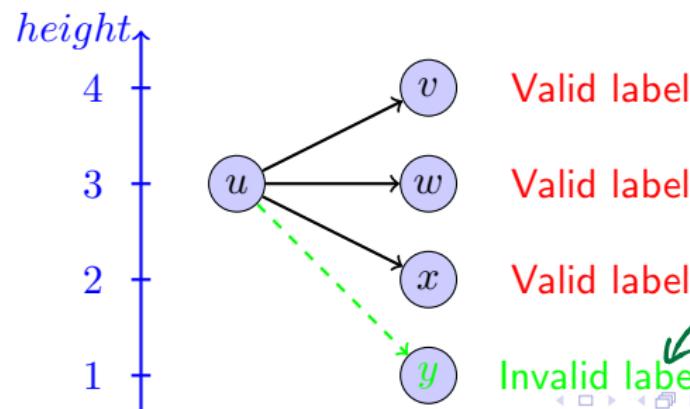
流:
 $\begin{cases} ① f(e) \leq C(e) \\ ② \text{in} = \text{out} \end{cases}$

Definition (Valid label)

Consider a pre-flow f . A **valid labeling** of nodes is:

- $\underline{h(s) = n}$, and $\underline{h(t) = 0}$;
- For each edge (u, v) in the residual graph G_f , we have $\underline{h(v) \geq h(u) - 1}$;

(Intuition: $h(v)$ is height of the node v , and for an edge in G_f , its end cannot be too lower than its head.)

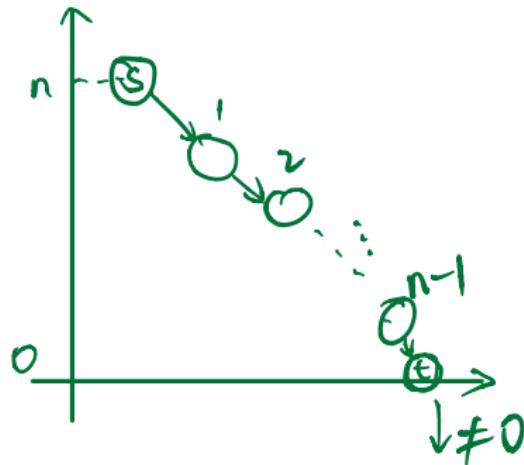


用于判断 $s-t$ 有没有合法的路.

too lower than its head.

Theorem

There is no $s - t$ path in a residual graph G_f if there exist valid labels.



Proof.

- Suppose there is a $s - t$ path in G_f . $\neg \exists \text{ path}$.
- Notice that $s - t$ path contains at most $n - 1$ edges.
- Since $h(s) = n$ and $h(u) \leq h(v) + 1$, the height of t should be greater than 0. A contradiction with $h(t) = 0$.



目标：像一个流

```

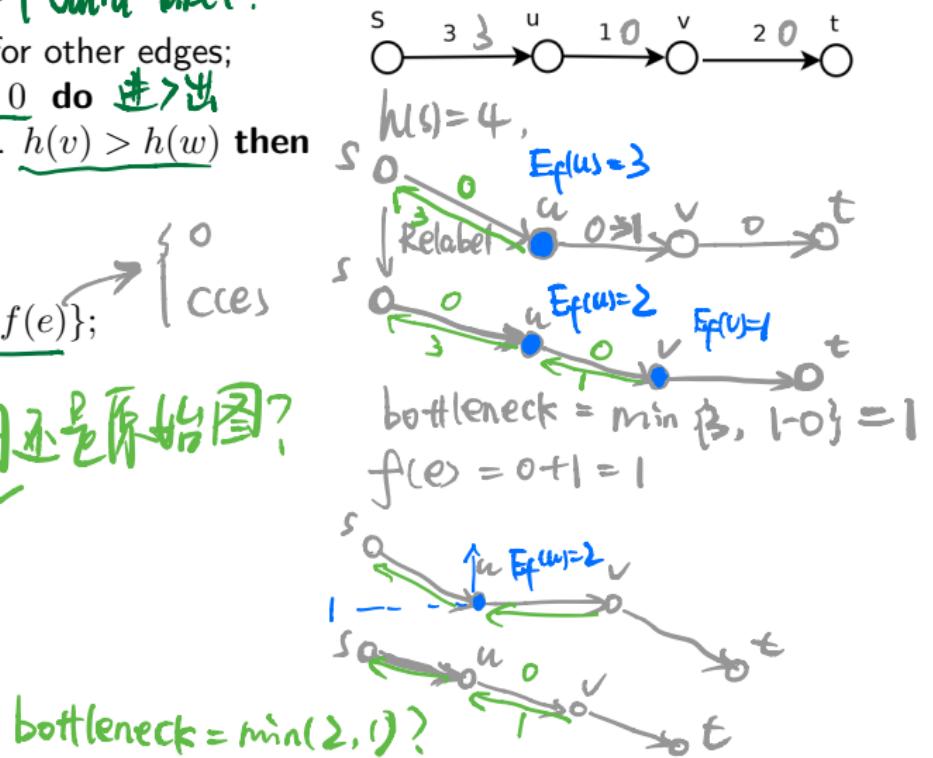
1: Set  $f$  as a pre-flow;
2: Set valid labels for nodes;
3: while TRUE do
4:   if no node has excess then } 是流
5:     return  $f$ ;
6:   end if
7:   Select a node  $v$  with excess;
8:   if  $v$  has a neighbor  $w$  such that  $h(v) > h(w)$  then }  $v$  比邻居高
9:     Push some excess from  $v$  to  $w$ ;
10:    else
11:      Perform relabeling to increase  $h(v)$ ;
12:    end if
13:  end while

```

把剩余货放下
推一推.

Push-relabel algorithm

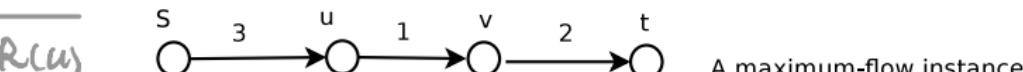
```
1:  $h(s) = n$ ;  $h(v) = 0$ ; for any  $v \neq s$ ; 不是有效 label.  
2:  $f(e) = C(e)$  for all  $e = (s, u)$ ;  $f(e) = 0$ ; for other edges;  
3: while there exists a node  $v$  with  $E_f(v) > 0$  do 进入  
4:   if there exists an edge  $(v, w) \in G_f$  s.t.  $h(v) > h(w)$  then  
5:     //Push excess from  $v$  to  $w$ ;  
6:     if  $(v, w)$  is a forward edge then  
7:       push //注释  
8:        $e = (v, w)$ ;  
9:       bottleneck =  $\min\{E_f(v), C(e) - f(e)\}$ ;  
10:       $f(e)_+ = \text{bottleneck}$ ;  
11:     else  
12:        $e = (w, v)$ ;  
13:       bottleneck =  $\min\{E_f(v), f(e)\}$ ;  
14:        $f(e)_- = \text{bottleneck}$ ;  
15:     end if //退回。  
16:      $h(v) = h(v) + 1$ ; //Relabel node  $v$ ;  
17:   end if  
18: end while
```



(height, residue)

A demo of push-relabel algo: initialization

STATUS		ACTION	
S	u	v	T
(4,0)	(0,3)	(0,0)	(0,0)
(4,0)	(1,3)	(0,0)	(0,0)
(4,0)	(1,2)	(0,1)	(0,0)
(4,0)	(1,2)	(1,1)	(0,0)
(4,0)	(1,2)	(1,0)	(0,1)
(4,0)	(5,2)	(1,0)	(0,1)
(4,2)	(5,0)	(1,0)	(0,1)



A maximum-flow instance

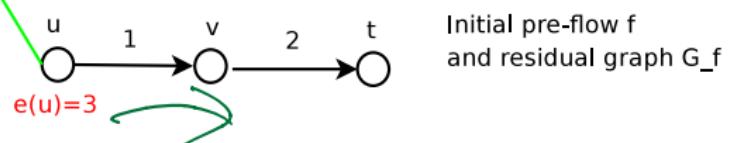


还是要有剩余图.



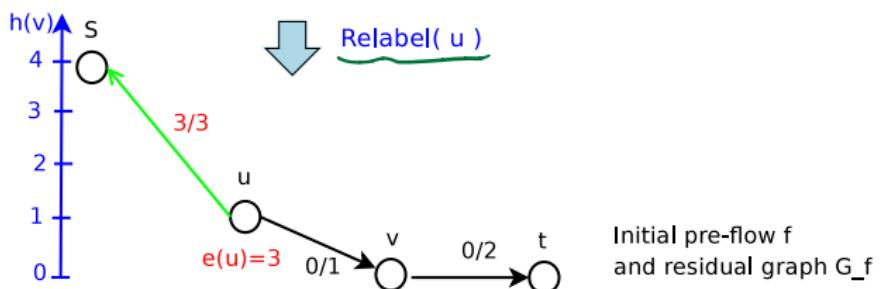
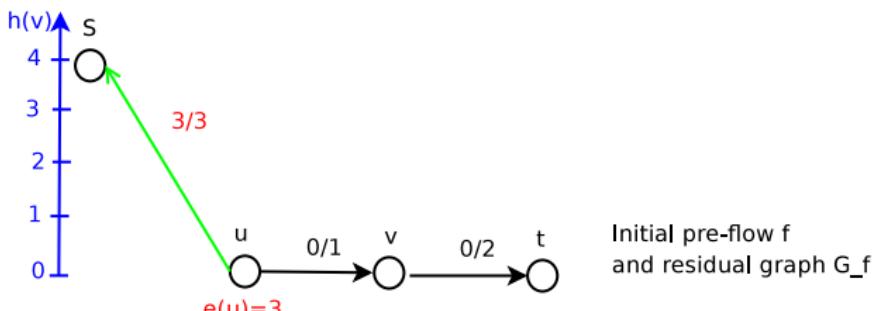
剩余图, 算法中P, 没有label. 没有剩余图.

label就表示剩余图.

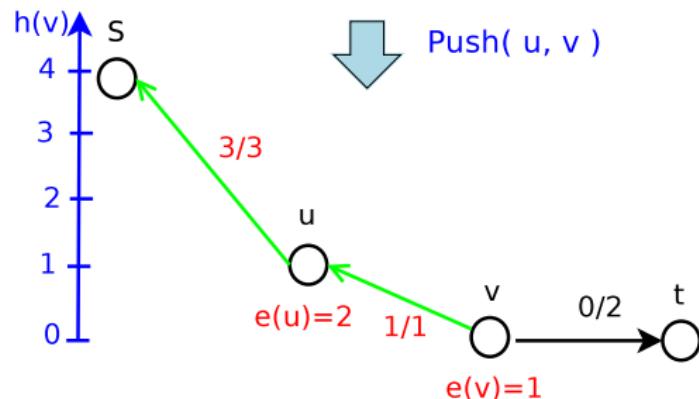
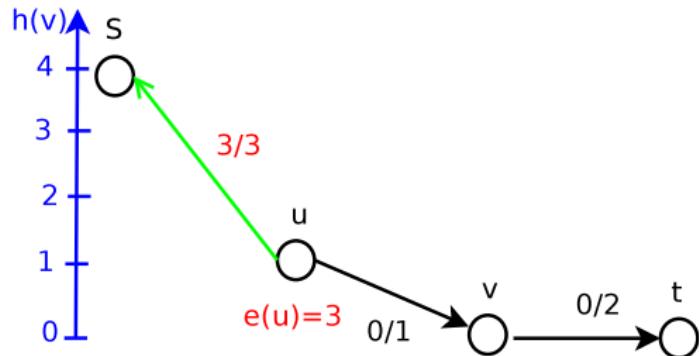


Initial pre-flow f
and residual graph G_f

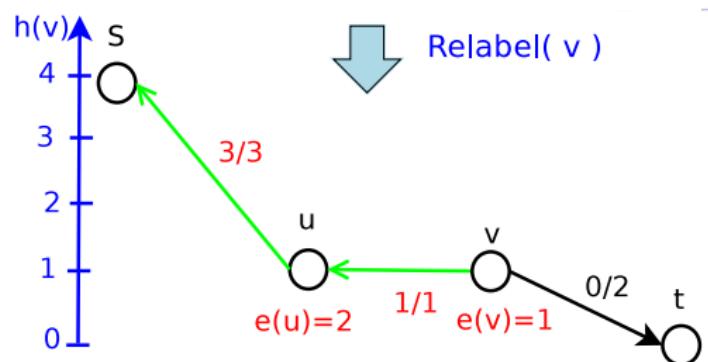
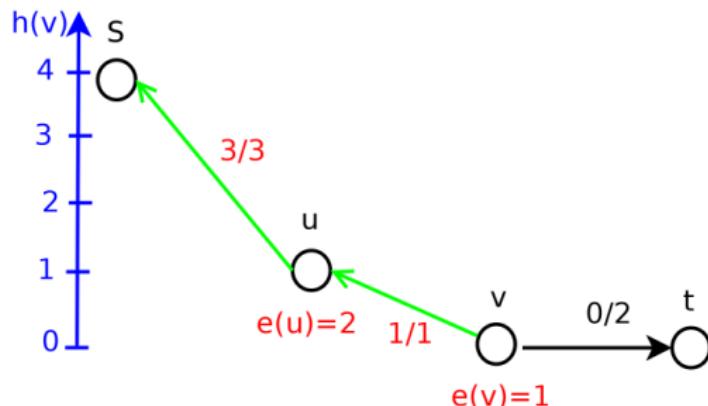
A demo of push-relabel algo: Step 1



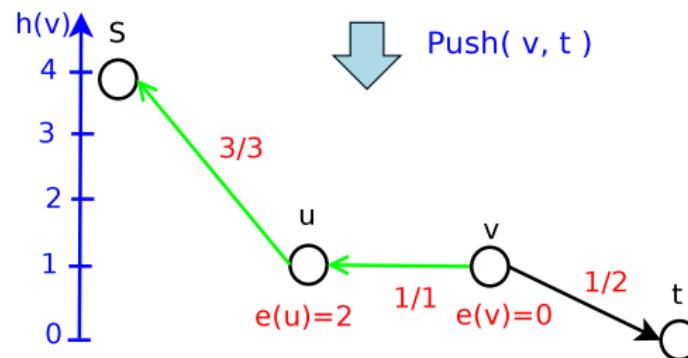
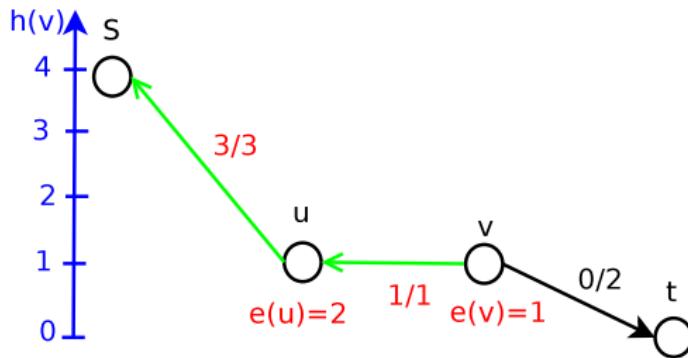
A demo of push-relabel algo: Step 2



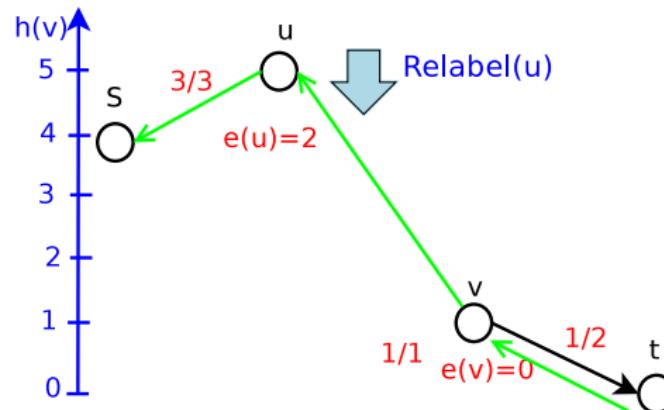
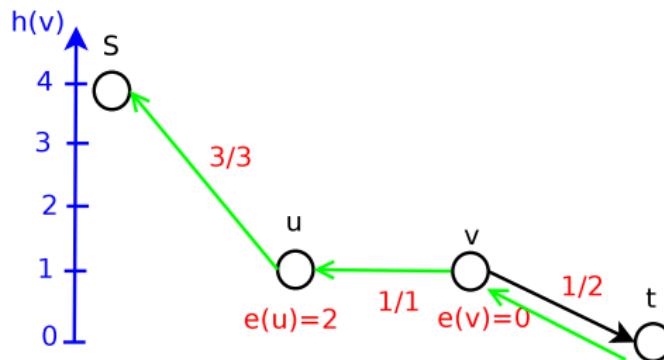
A demo of push-relabel algo: Step 3



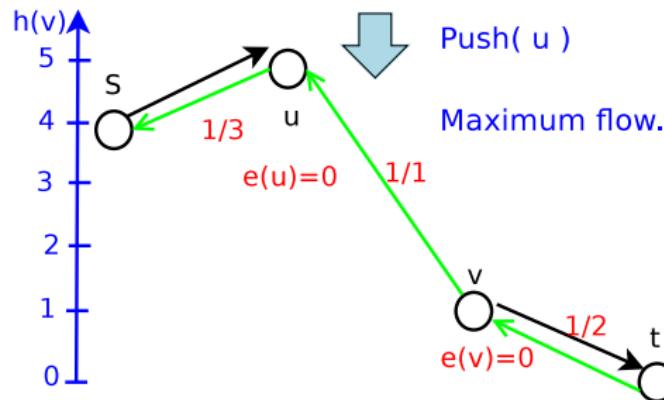
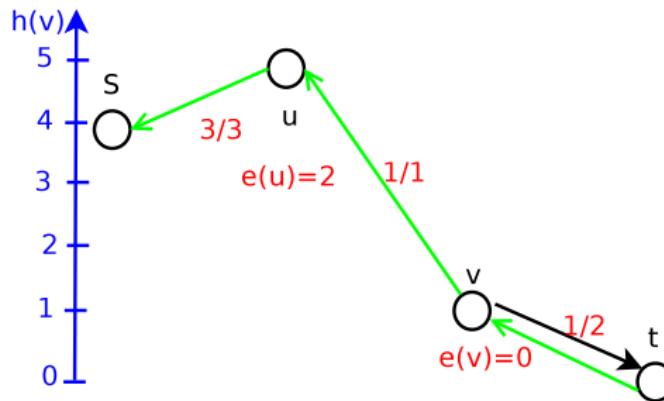
A demo of push-relabel algo: Step 4



A demo of push-relabel algo: Step 5



A demo of push-relabel algo: Step 6



Theorem

Push-relabel algo keeps label valid, and thus outputs a maximum flow when ends.

① $\text{if } \text{push}(u, v) \text{ 时}$
 $h(v) < h(u)$?

保证新边的
 $v \rightarrow u$

合法.

始終保證标记
 合法.

Proof.

(Induction on the number of push and relabel operations.)

- Push operation: the new f is still a pre-flow since the capacity condition still holds.

$\text{Push}(f, v, w)$ may add edge (w, v) into G_f . We have $h(w) < h(v)$. (pre-condition). Thus, the label is valid for the new G_f .

- Relabel operation: The pre-condition implies $h(v) \leq h(w)$ for any $(v, w) \in G_f$. $\text{relabel}(f, h, v)$ changes $\underline{h(v) = h(v) + 1}$. Thus, the new $h(v) \leq h(w) + 1$.

保证下场不会太多.



保证所有的下场都不比现
 在低.

while 亂序未結束

計算：

① 每一 Δ 狀態 STATUS φ

② $\phi(s) \leq h$

③ $\phi = 1$

Time-complexity: #Relabel |

Theorem

For any node v , $\#Relabel \leq 2n - 1$. Thus, the total label operation number is less than $2n^2$.

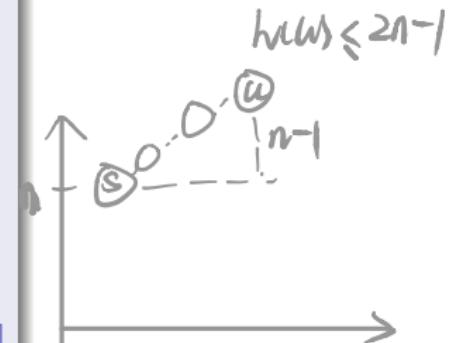
Proof.

- ① (Connectivity): For a node w with $E_f(w) > 0$, there should be a path from w to s in G_f .

(Intuition: node w obtain a positive $E_f(w)$ through a node v by $Push(f, v, w)$. This operation also causes edge (w, v) to be added into G_f . Thus, there should be a path from w to s .

)

- ② (Upper bound of $h(v)$): $h(v) < 2n - 1$ since there is a path from v to s . The length of the path is less than $n - 1$, $h(s) = n$, and $h(v) \leq h(w) + 1$ for any edge (v, w) in G_f .



Two types of *Push* operations:

- ① Saturated push (s-push): if $Push(f, v, w)$ causes (v, w) removed from G_f .
- ② Unsaturated push (uns-push): other pushes.

$$\#Push = \#s\text{-push} + \#uns\text{-push}.$$

饱和 不饱和. $\Rightarrow Push_{f_0} E(v) = 0$.

Theorem

$$\#s\text{-push} \leq 2nm.$$

Proof.

Consider an edge $e = (v, w)$. We will show that during the execution of algo, (v, w) appears in G_f at most $2n$ times.

- (Removing): a saturated $Push(f, v, w)$ removes (v, w) from G_f . We have $h(v) = h(w) + 1$.
- (Adding): Before applying $Push(f, v, w)$ again, (v, w) should be added to G_f first. The only way to add (v, w) to G_f is $Push(f, w, v)$. The pre-condition of $Push(f, w, v)$ requires that $h(w) \geq h(v) + 1$, i.e., $h(w)$ should be increased at least 2 since the previous $Push(f, v, w)$ operation. And we have $h(w) \leq 2n - 1$.



Theorem

$$\#\text{uns-push} \leq 2n^2m.$$

Proof.

Define a measure $\Phi(f, h) = \sum_{v:E_f(v)>0} h(v)$.

- (Increase and upper bound) $\Phi(f, h) < 4n^2m$:

- ① Relabel: a relabel operation increase $\Phi(f, h)$ by 1. The total $O(2n^2)$ relabel operations increase $\Phi(f, h)$ at most $O(2n^2)$.
- ② Saturated push: A saturated $Push(f, v, w)$ operation increases $\Phi(f, h)$ by $h(w)$ since w has excess now. $h(w) \leq 2n - 1$ implies an upper bound for each operation. The total $2nm$ ~~最多~~ ~~每次操作增加 $2n - 1$~~ pushes increase $\Phi(f, h)$ by at most $4n^2m$.

- (Decrease) An unsaturated $Push(f, v, w)$ will reduce $\Phi(f, h)$ at least 1.

(Intuition: after unsaturated $Push(f, v, w)$, we have $E_f(v) = 0$, which reduce $h(v)$ from $\Phi(f, h)$; on the other side, w obtains excess from v , which will increase $\Phi(f, h)$ by $h(w)$. From $h(v) \leq h(w) + 1$, we have that $\Phi(f, h)$ reduces at least 1.)