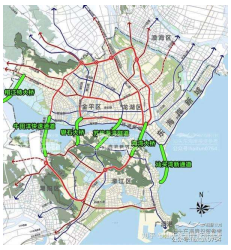
Network Flow
网络流

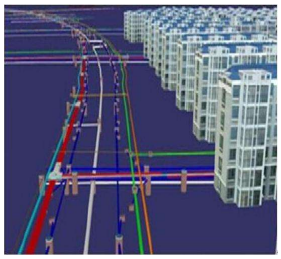
Max flow-Min Cut
最大流最小割定理

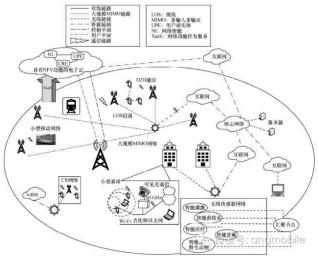
Outline

- Network flow 网络流
- Flows and Cuts 流和割
- Residual Graph 残差图
- Augmenting Paths 增广路径
- Ford Fulkerson Algorithm 福特-福克森算法
- Max flow-Min Cut Theorem 最大流最小割
- Application – Bipartite Matching

Network Flow

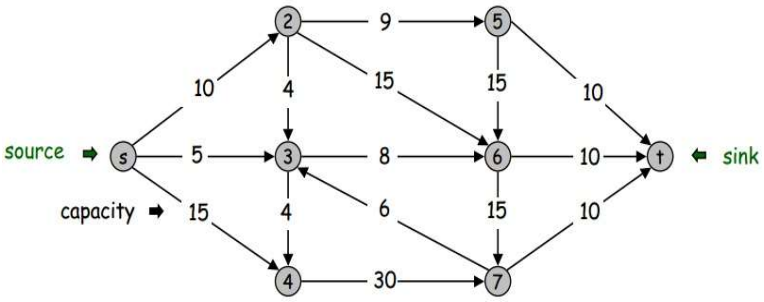






Network Flow Definitions

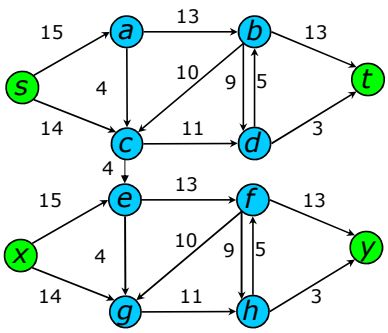
- Flowgraph: Directed graph with distinguished vertices s (source源点) and t (sink收点/汇点)
- Capacities（容量） on the edges: c(e) >= 0

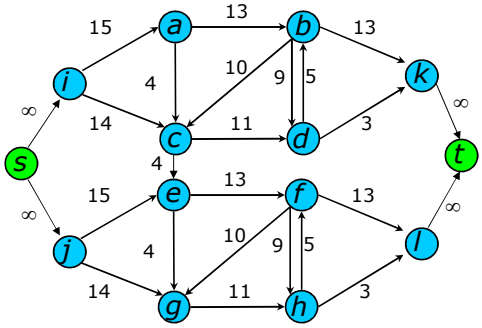


边上标注的是容量值

Multiple Sources or Sinks

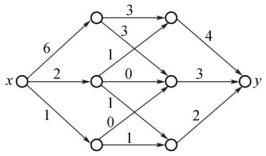
- What if you have a problem with more than one source and more than one sink?
- Modify the graph to create a single supersource and supersink





Network Flow

- Problem, assign flows f(e) to the edges such that:
  - 0 <= f(e) <= c(e) （容量约束）
  - Flow is conserved at vertices other than s and t
    - Flow conservation: flow going into a vertex equals the flow going out （守恒约束）



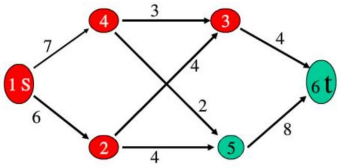
一个可行流，边上标注的是流量

  - 优化目标： The flow leaving the source is as large as possible
    - Denoted by |f|

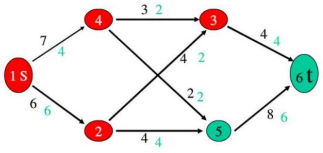
Network Flow

实例：

有一自来水管道输送系统，起点是S，目标是T，途中经过的管道都有一个最大的容量。



- 问题：问从S到T的最大水流量是多少？



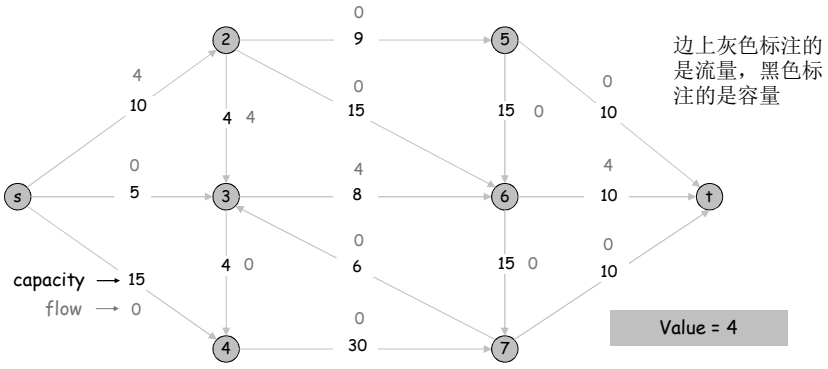
最大水流量是10

Flows

Def. An s-t flow （可行流）is a function that satisfies:

- For each e ∈ E: $0 \le f(e) \le c(e)$ [capacity]
- For each v ∈ V - {s, t}: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [conservation]
- 特例：零流

Def. The value of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$.



边上灰色标注的是流量，黑色标注的是容量

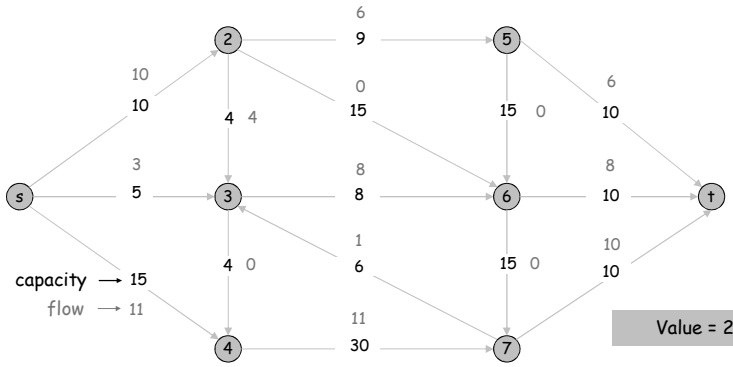
8

Flows

Def. An s-t flow is a function that satisfies:

- For each e ∈ E: $0 \le f(e) \le c(e)$ [capacity]
- For each v ∈ V – {s, t}: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [conservation]

Def. The value of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$.

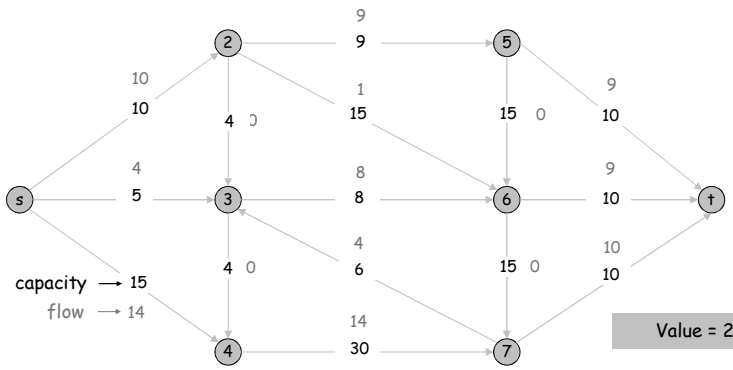


9

Maximum Flow Problem

Max flow problem. Find s-t flow of maximum value.

所有可行流中，流量最大的流的流量



10

Cuts in a graph

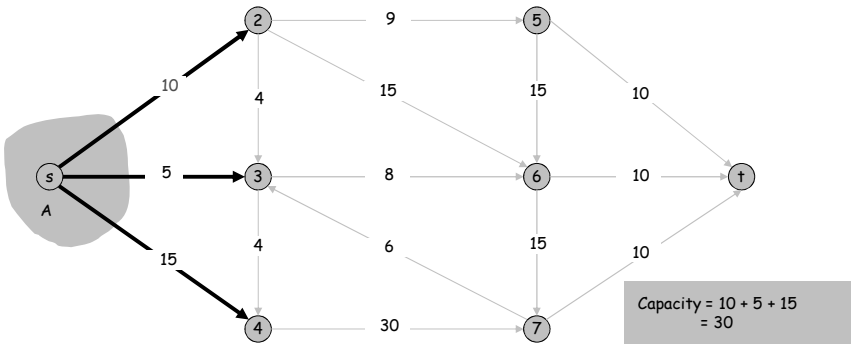
- Cut: Partition of V into disjoint sets S, T with s in S and t in T.
- Cap(S,T): sum of the capacities of edges from S to T
- Flow(S,T): net flow out of S
  - Sum of flows out of S minus sum of flows into S

- Flow(S,T) <= Cap(S,T)

Cuts

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

Def. The capacity of a cut (A, B) is: $cap(A, B) = \sum_{e \text{ out of } A} c(e)$

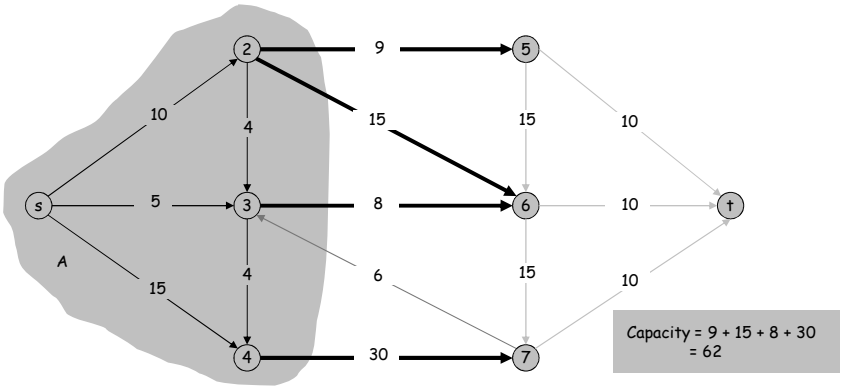


12

Cuts

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

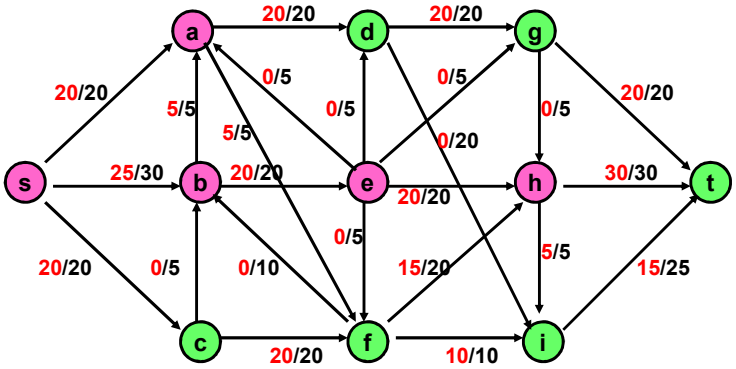
Def. The capacity of a cut (A, B) is: $cap(A, B) = \sum_{e \text{ out of } A} c(e)$



13

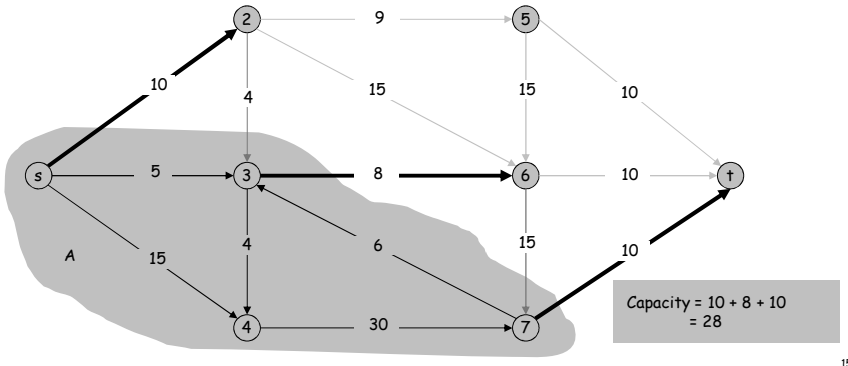
What is Cap(S,T) and Flow(S,T)

S={s, a, b, e, h}, T = {c, f, i, d, g, t}



Minimum Cut Problem

Min s-t cut problem. Find an s-t cut of minimum capacity.

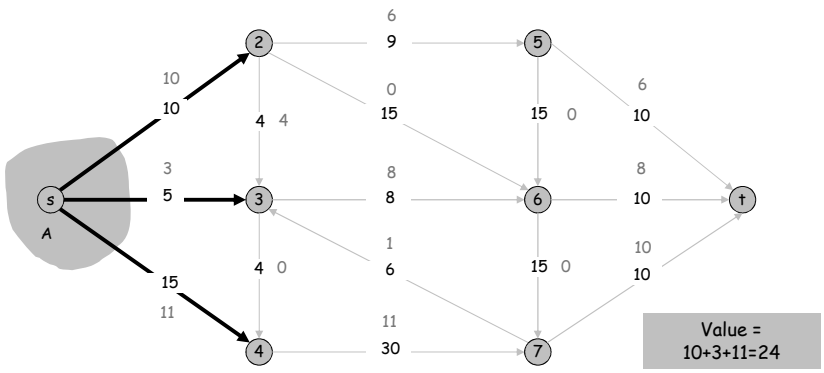


15

Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut.
Then, the net flow sent across the cut is equal to the amount leaving s.

$$\sum_{e \text{ out of } A} f(e) \; - \; \sum_{e \text{ in to A}} f(e) \; = \; v(f)$$

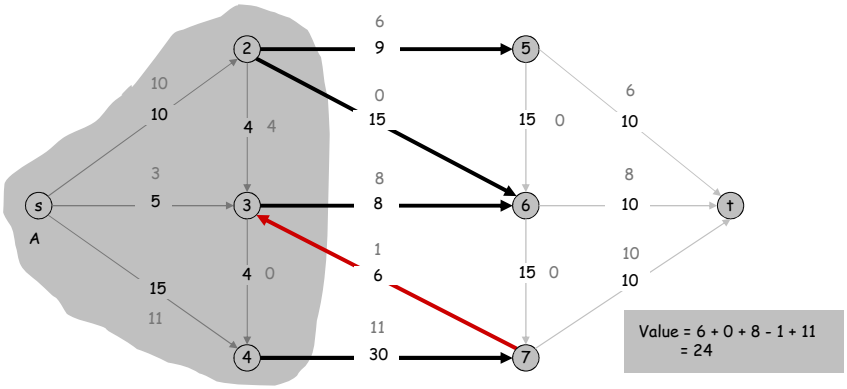


16

Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut.
Then, the net flow sent across the cut is equal to the amount leaving s.

$$\sum_{e \text{ out of } A} f(e) \; - \; \sum_{e \text{ in to A}} f(e) \; = \; v(f)$$

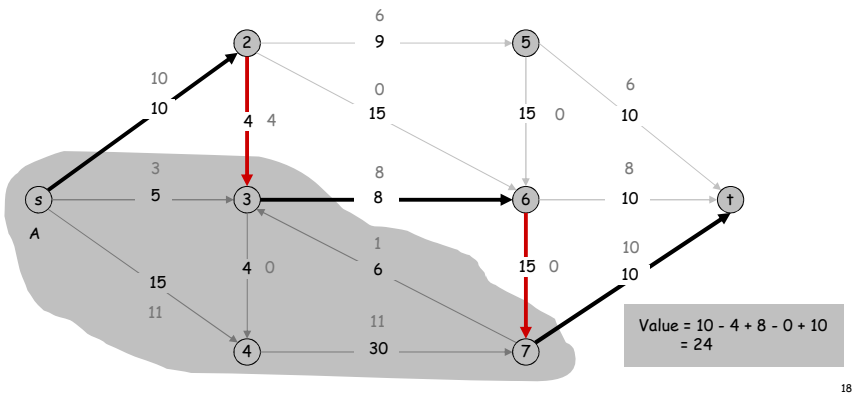


17

Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut.
Then, the net flow sent across the cut is equal to the amount leaving s.

$$\sum_{e \text{ out of } A} f(e) \; - \; \sum_{e \text{ in to A}} f(e) \; = \; v(f)$$



18

Flows and Cuts

Flow value lemma. Let f be any flow, and let (A, B) be any s-t cut. Then

$$\sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) = v(f).$$

Pf.

$$v(f) = \sum_{e \text{ out of } s} f(e)$$

by flow conservation, all terms except v = s are 0

$$= \sum_{v \in A} \left(\sum_{e \text{ out of } v} f(e) - \sum_{e \text{ in to } v} f(e) \right)$$

从顶点角度观察：除了s，其他顶点的出入流量都相互抵消

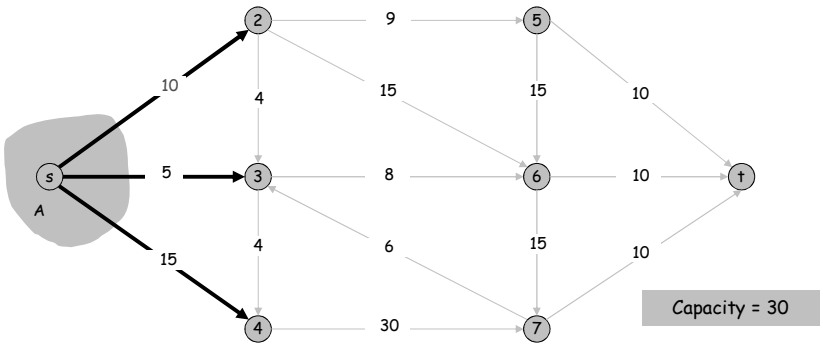
$$= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to A}} f(e).$$

从边角度观察：位于A内部的边出入流量都相互抵消，只有AB之间的边才起作用

Flows and Cuts

Weak duality. Let f be any flow, and let (A, B) be any s-t cut. Then the value of the flow is at most the capacity of the cut.

Cut capacity = 30 ⇒ Flow value ≤ 30

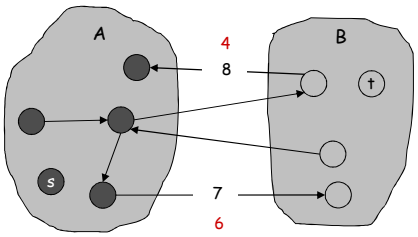


Flows and Cuts

Weak duality. Let f be any flow. Then, for any s-t cut (A, B) we have v(f) ≤ cap(A, B).

Pf.

$$\begin{aligned} v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\ &\le \sum_{e \text{ out of } A} f(e) \\ &\le \sum_{e \text{ out of } A} c(e) \\ &= \text{cap}(A, B) \quad \blacksquare \end{aligned}$$

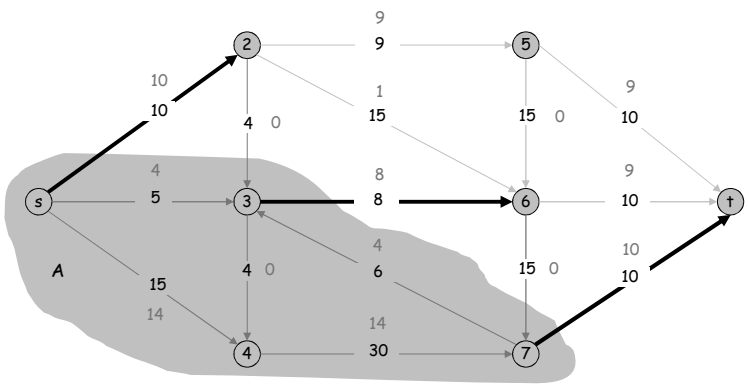


21

Certificate of Optimality

Corollary. Let f be any flow, and let (A, B) be any cut.
If v(f) = cap(A, B), then f is a max flow and (A, B) is a min cut.

Value of flow = 28
Cut capacity = 28 ⇒ Flow value ≤ 28

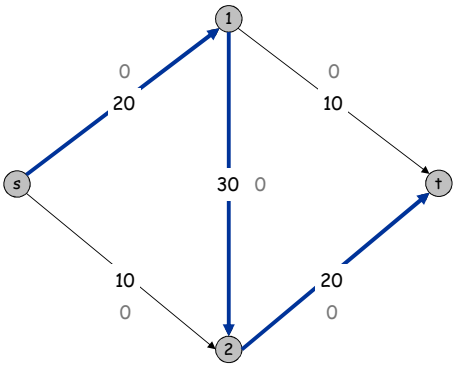


22

Towards a Max Flow Algorithm

Greedy algorithm.

- Start with f(e) = 0 for all edge e ∈ E.
- Find an s-t path P where each edge has f(e) < c(e).
- Augment flow along path P.
- Repeat until you get stuck.



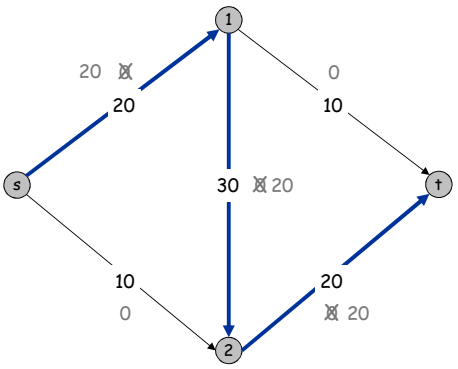
Flow value = 0

23

Towards a Max Flow Algorithm

Greedy algorithm.

- Start with f(e) = 0 for all edge e ∈ E.
- Find an s-t path P where each edge has f(e) < c(e).
- Augment flow along path P.
- Repeat until you get stuck.



Flow value = 20

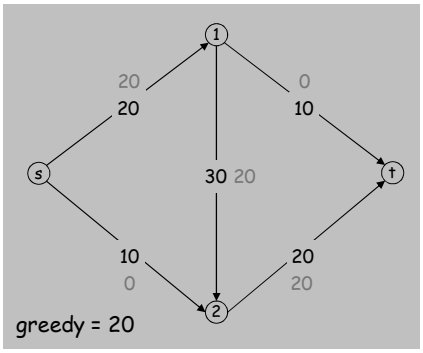
24

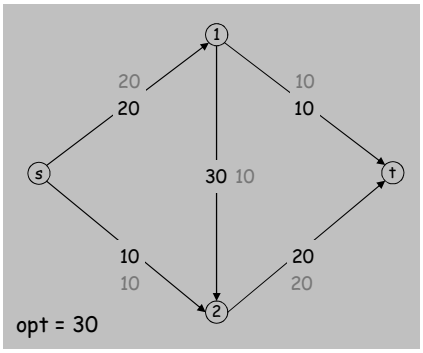
Towards a Max Flow Algorithm

Greedy algorithm.

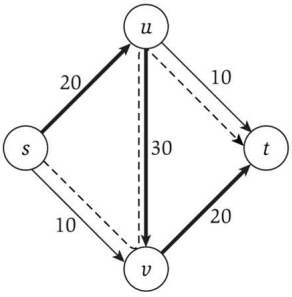
- Start with f(e) = 0 for all edge e ∈ E.
- Find an s-t path P where each edge has f(e) < c(e).
- Augment flow along path P.
- Repeat until you get stuck.

↖ locally optimality ≠> global optimality

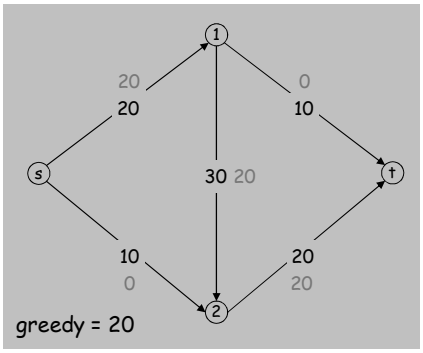


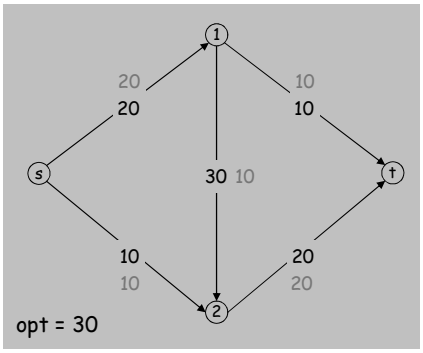


25



观察虚线：利用逆向边。



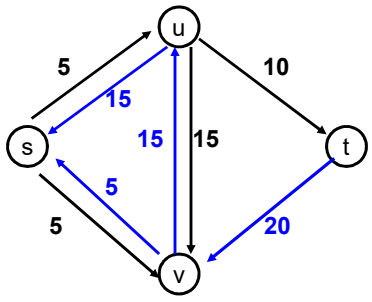


26

Residual Graph（残差图）

- Flow graph showing the remaining capacity
- Flow graph G, Residual Graph G_R
  - G: edge e from u to v with capacity c and flow f
  - G_R: edge e' from u to v with capacity c – f
  - G_R: edge e'' from v to u with capacity f

The residual graph

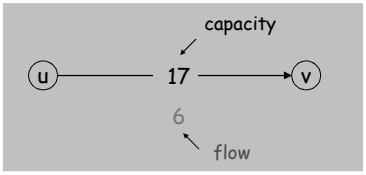


the residual graph

Residual Graph

Original edge: e = (u, v) ∈ E.

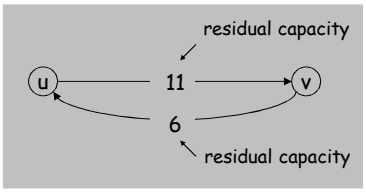
- Flow f(e), capacity c(e).



Residual edge.

- "Undo" flow sent.
- e = (u, v) and e^R = (v, u).
- Residual capacity:

$$c_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \in E \\ f(e) & \text{if } e^R \in E \end{cases}$$



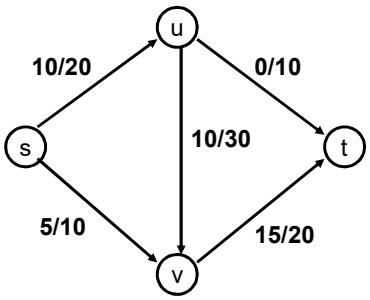
Residual graph: $G_f = (V, E_f)$.

- Residual edges with positive residual capacity.
- $E_f = \{e : f(e) < c(e)\} \cup \{e^R : f(e) > 0\}$.

29

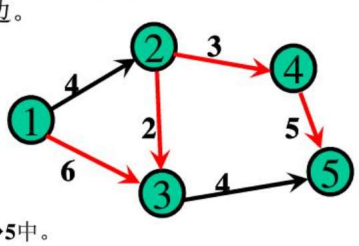
Augmenting Path Algorithm

- Augmenting path（增广路径）
  - Vertices $v_1, v_2, \ldots, v_k$
    - v_1 = s, v_k = t
    - Possible to add b units of flow between v_j and v_{j+1} for j = 1 … k-1



Augmenting Path Algorithm

- 从 s 到 t 的一条简单路径，若边 (u, v) 的方向与该路径的方向一致，称 (u, v) 为正向边，方向不一致时称为逆向边。



简单路：1→3→2→4→5中。

（1，3）（2，4）（4，5）是正向边。（3，2）是逆向边。

Augmenting Path Algorithm

增广路径：

若路径上所有的边满足：

①所有正向边有：f（u，v）< c（u，v）

②所有逆向边有：f（u，v）> 0

则称该路径为一条增广路径(可增加流量)

```
Augment(f, c, P) {
   b ← bottleneck(P)                    b > 0
   foreach e ∈ P {
      if (e ∈ E) f(e) ← f(e) + b        forward edge
      else       f(e^R)← f(e^R) - b     reverse edge
   }
   return f
}
```

找到这样一条路径，其流量未达到容量上限。增广后，总流量增加了b。

Ford-Fulkerson Algorithm (1956)

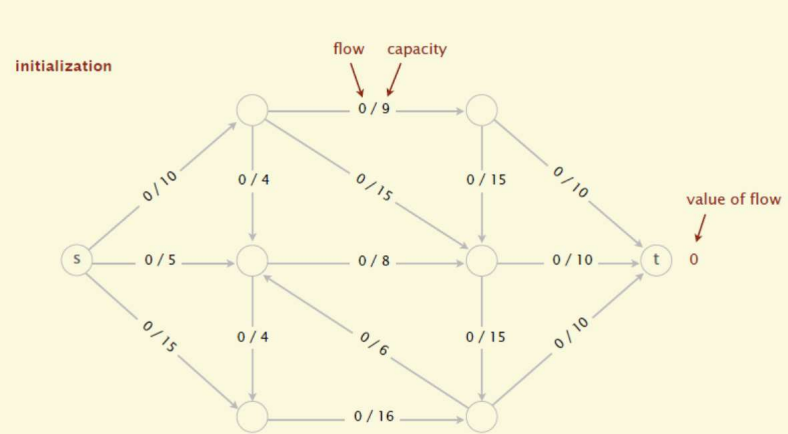
```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   G_f ← residual graph

   while (there exists augmenting path P) {
      f ← Augment(f, c, P)
      update G_f
   }
   return f
}
```

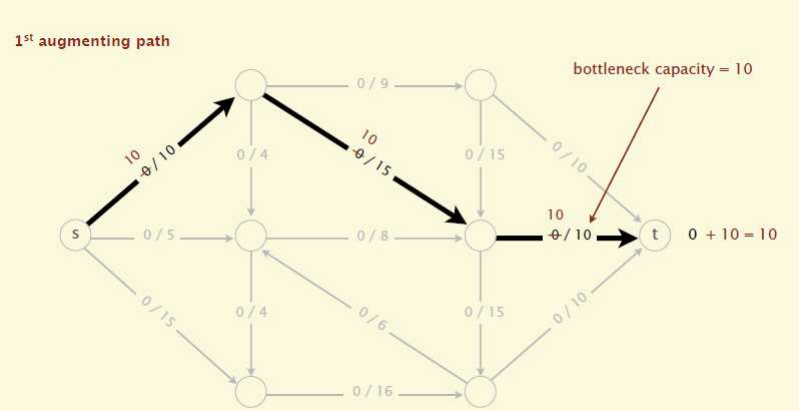
先假定容量为非负**整数**：

If the sum of the capacities of edges leaving S is at most C, then the algorithm takes at most C iterations.（每次都增加，但总量有限，一定会结束）

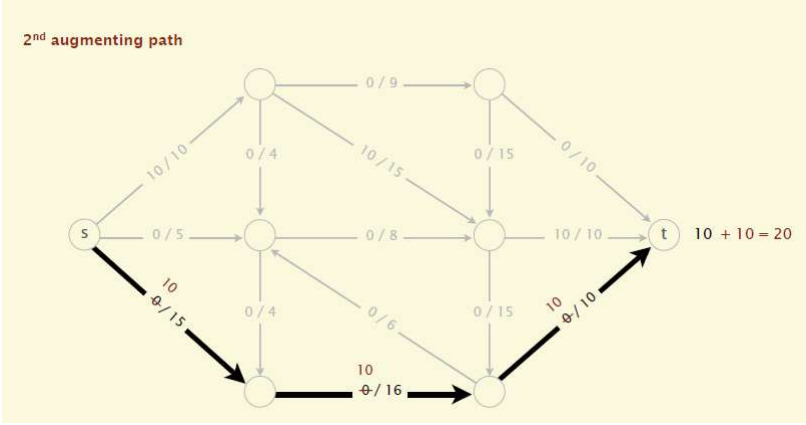
Ford-Fulkerson Algorithm (1956)



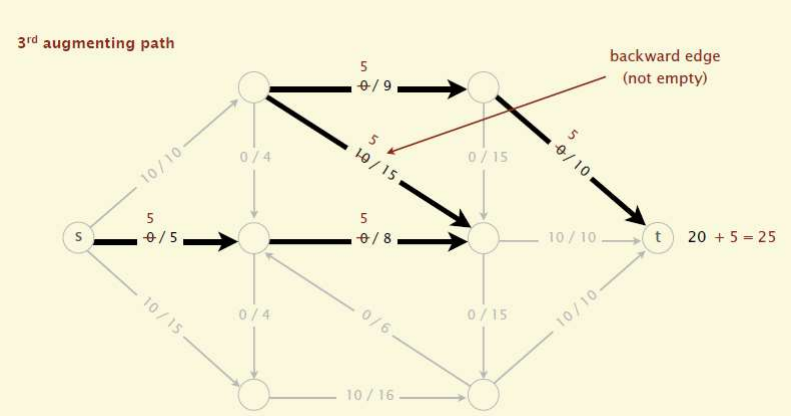
Ford-Fulkerson Algorithm (1956)



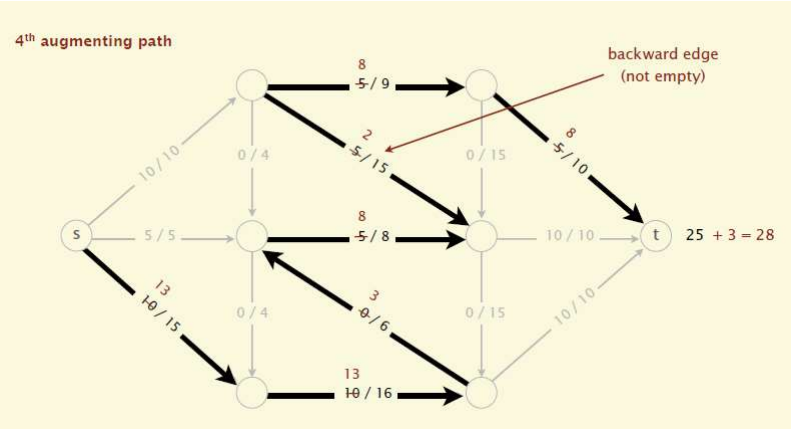
Ford-Fulkerson Algorithm (1956)



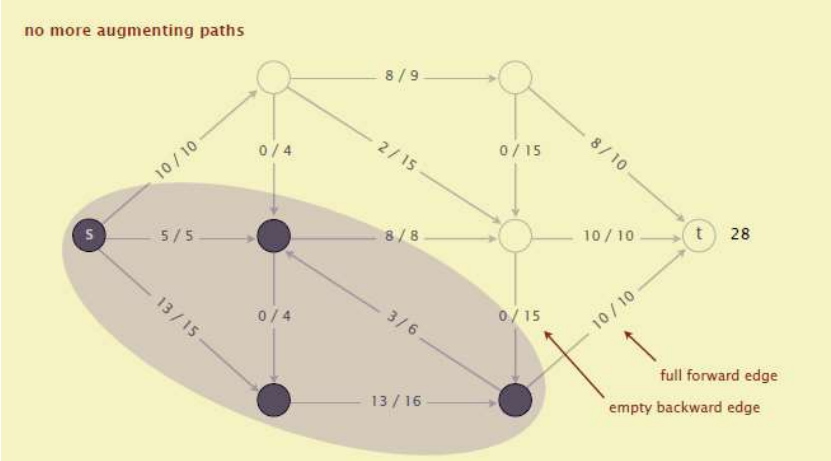
Ford-Fulkerson Algorithm (1956)



Ford-Fulkerson Algorithm (1956)



Ford-Fulkerson Algorithm (1956)



Max-Flow Min-Cut Theorem

Augmenting path theorem. Flow f is a max flow iff there are no augmenting paths.

Max-flow min-cut theorem. [Elias-Feinstein-Shannon 1956, Ford-Fulkerson 1956] The value of the max flow is equal to the value of the min cut. (There exists a flow which has the same value of the minimum cut)

Pf. We prove both simultaneously by showing TFAE (the following are equivalent):

(i) There exists a cut (A, B) such that v(f) = cap(A, B).
存在一个割的容量等于flow f的值

(ii) Flow f is a max flow.
f是最大流

(iii) There is no augmenting path relative to f.
对于f没有增广路径

40

Max-Flow Min-Cut Theorem

(i) ⇒ (ii) This was the corollary to weak duality lemma.
假设我们有一个割（A,B）的容量等于f的值，那么所有流的值<=(A,B)的容量，从而（2）成立

(ii) ⇒ (iii) We show contrapositive.

- Let f be a flow. If there exists an augmenting path, then we can improve f by sending flow along path.
  我们来证明它的逆否命题。对于f如果还有还有增广路径，那f不是最大流。这很显然。如果按照FF算法的话，我们还可以增加flow f的值，因此f就不会是最大流，因此逆否命题成立，也就代表（2）->(3)成立。

41

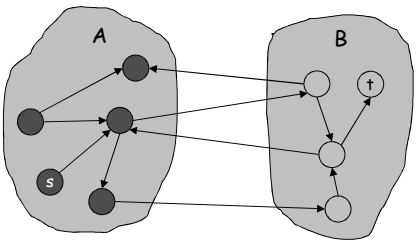
Proof of Max-Flow Min-Cut Theorem

(iii) ⇒ (i)

- Let f be a flow with no augmenting paths.
- Let A be set of vertices reachable from s in residual graph.
  - 通过这些边可达：要么是不是满的前向边，要么是非空的反向边。
- By definition of A, $s \in A$.
- By definition of f, $t \notin A$.（因为没有增广路径）

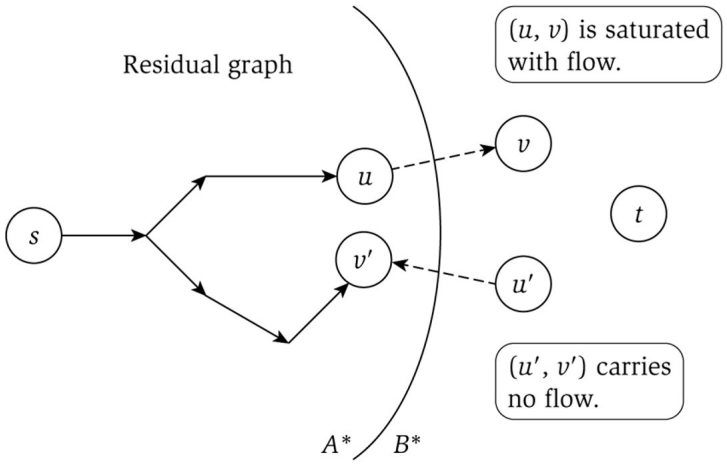
$$\begin{aligned} v(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to A}} f(e) \\ &= \sum_{e \text{ out of } A} c(e) \\ &= cap(A, B) \end{aligned}$$

Explained next page



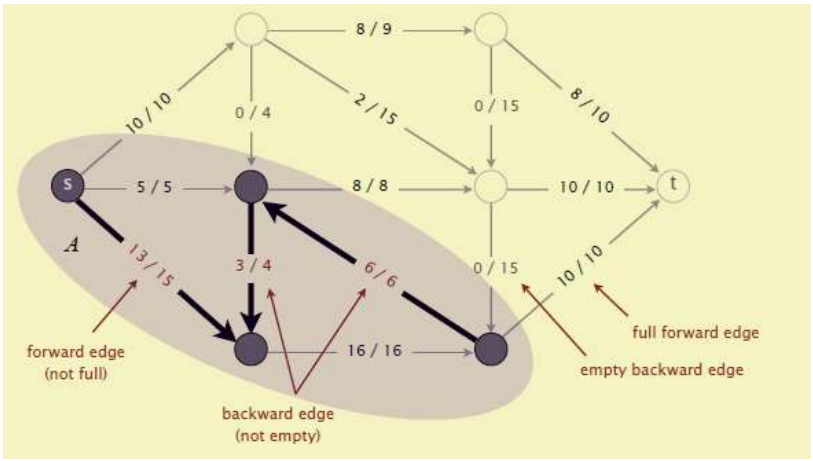
original network

Proof of Max-Flow Min-Cut Theorem



43

Proof of Max-Flow Min-Cut Theorem



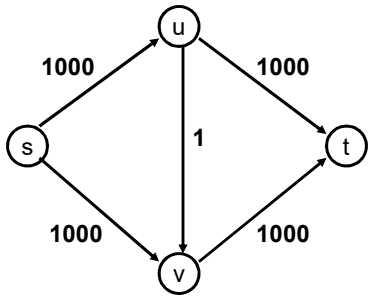
44

Max Flow - Min Cut Theorem

- Ford-Fulkerson algorithm finds a flow where the residual graph is disconnected, hence FF finds a maximum flow.
- If we want to find a minimum cut, we begin by looking for a maximum flow.

Performance

- The worst case performance of the Ford-Fulkerson algorithm is horrible

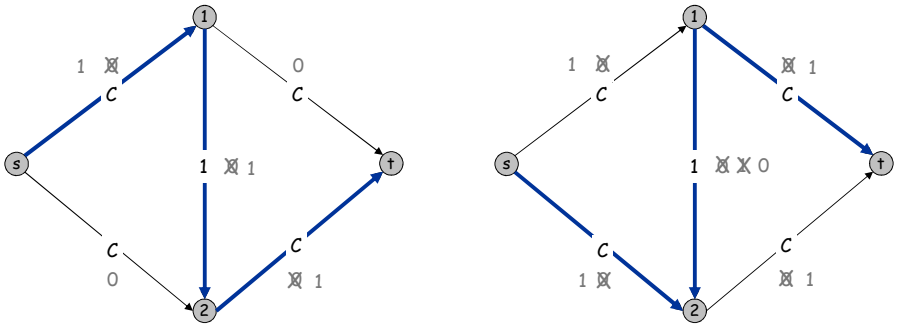


Ford-Fulkerson: Exponential Number of Augmentations

Q. Is generic Ford-Fulkerson algorithm polynomial in input size?

m, n, and log C

A. No. If max capacity is C, then algorithm can take C iterations.



47

Running Time

Assumption. All capacities are integers between 1 and C.

Invariant. Every flow value f(e) and every residual capacity $c_f(e)$ remains an integer throughout the algorithm.

Theorem. The algorithm terminates in at most $v(f^*) \leq nC$ iterations.
Pf. Each augmentation increase value by at least 1. ▪

Corollary. If C = 1, Ford-Fulkerson runs in O(mn) time.

Integrality theorem. If all capacities are integers, then there exists a max flow f for which every flow value f(e) is an integer.
Pf. Since algorithm terminates, theorem follows from invariant. ▪

48

Choosing Good Augmenting Paths

Use care when selecting augmenting paths.

- Some choices lead to exponential algorithms.
- Clever choices lead to polynomial algorithms.
- If capacities are irrational, algorithm not guaranteed to terminate!

Goal: choose augmenting paths so that:

- Can find augmenting paths efficiently.
- Few iterations.

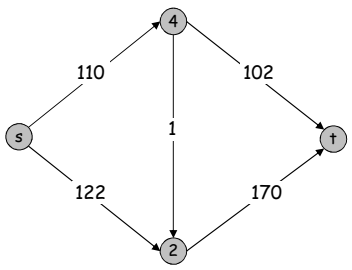
Choose augmenting paths with: [Edmonds-Karp 1972, Dinitz 1970]

- Max bottleneck capacity.
- Sufficiently large bottleneck capacity.
- Fewest number of edges.

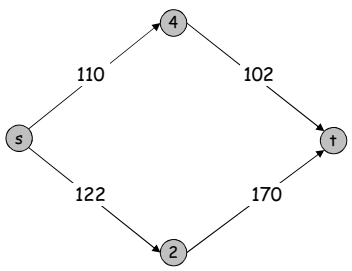
Capacity Scaling

Intuition. Choosing path with highest bottleneck capacity increases flow by max possible amount.

- Don't worry about finding exact highest bottleneck path.
- Maintain scaling parameter Δ.
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least Δ.



G_f



$G_f(100)$

Capacity Scaling

```
Scaling-Max-Flow(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Δ ← smallest power of 2 greater than or equal to C
   G_f ← residual graph

   while (Δ ≥ 1) {
      G_f(Δ) ← Δ-residual graph
      while (there exists augmenting path P in G_f(Δ)) {
         f ← augment(f, c, P)
         update G_f(Δ)
      }
      Δ ← Δ / 2
   }
   return f
}
```

51

Capacity Scaling: Correctness

Assumption. All edge capacities are integers between 1 and C.

Integrality invariant. All flow and residual capacity values are integral.

Correctness. If the algorithm terminates, then f is a max flow.

Pf.

- By integrality invariant, when $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$.
- Upon termination of $\Delta = 1$ phase, there are no augmenting paths. ▪

Theorem. The scaling max-flow algorithm finds a max flow in $O(m \log C)$ augmentations. It can be implemented to run in $O(m^2 \log C)$ time. ▪

52

Performance of finding augmenting paths

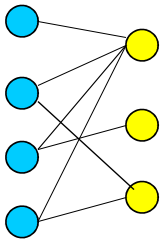
- Find the maximum capacity augmenting path
  - $O(m^2 \log(C))$ time algorithm for network flow
- Find the shortest augmenting path
  - $O(m^2 n)$ time algorithm for network flow
- Find a blocking flow in the residual graph
  - O(mnlog n) time algorithm for network flow

Application – Bipartite Matching

- Example – given a community with *n* men and *m* women
- Assume we have a way to determine which couples (man/woman) are compatible for marriage
  - E.g. (Joe, Susan) or (Fred, Susan) but not (Frank, Susan)
- Problem: Maximize the number of marriages
  - No polygamy allowed
  - Can solve this problem by creating a flow network out of a bipartite graph

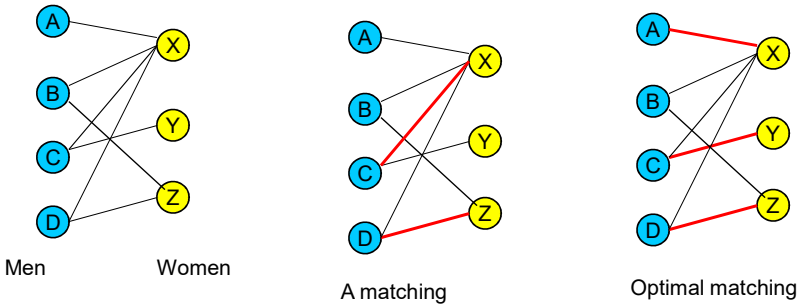
Bipartite Graph

- A bipartite graph is an undirected graph G=(V,E) in which V can be partitioned into two sets V_1 and V_2 such that (u,v) $\in$ E implies either u $\in$ V_1 and v $\in$ V_2 or vice versa.
- That is, all edges go between the two sets V_1 and V_2 and not within V_1 and V_2.



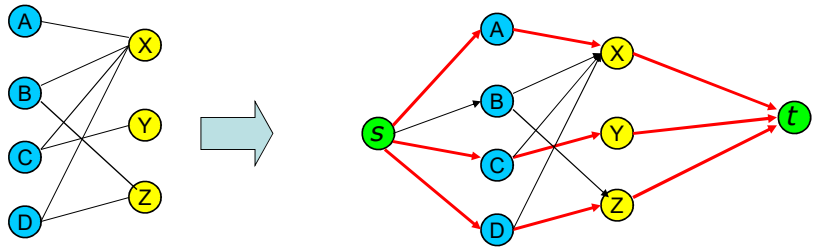
Model for Matching Problem

- Men on leftmost set, women on rightmost set, edges if they are compatible



Solution Using Max Flow

- Add a supersouce, supersink, make each undirected edge directed with a flow of 1



Since the input is 1, flow conservation prevents multiple matchings

Homework

1、In the network below, find a maximum flow from x to y, calculate its flow value, and prove that it is the maximum flow.