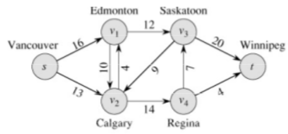
算法设计与分析
Algorithms Design & Analysis

第十四讲：最大网络流

华中科技大学软件学院
邱德红　主讲

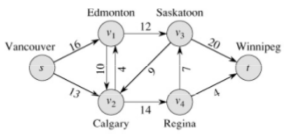
The Problems

- Road network to transport material (e.g.,) (公路物流)
  - Edges – roads, vertices – cities (边-公路; 顶点: 城市)
- Pipe network to transport fluid (e.g., water, oil) (管道流体)
  - Edges – pipes, vertices – junctions of pipes (边-管道; 顶点: 管道接头)
- Data communication network (数据通讯网络)
  - Edges – network connections of different capacity, vertices – routers (do not produce or consume data just move it) (边: 网络线; 顶点: 路由器)



Flow networks

Informally Concepts : (一些概念)
- *Source* vertex *s* (where material is produced) (源点)
- *Sink* vertex *t* (where material is consumed) (汇点)
- For all other vertices – what goes in must go out (除源点和汇点外, 其他顶点的流入和流出相等)
- *Goal: maximum rate of material flow from source to sink (目标: 从源点到汇点的最大流量)*



Formalization (正式描述)

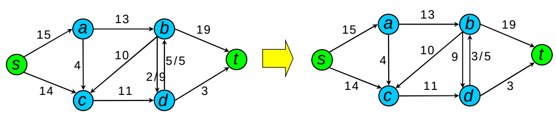
- Graph $G=(V,E)$ – a **flow network** (网络流图)
  - Directed, each edge has **capacity** $c(u,v) \geq 0$ (有向图, 边的容量为$c(u,v) \geq 0$)
  - Two special vertices: ***source*** *s*, and ***sink*** *t* (两个特殊顶点: 源点*s*和汇点*t*)
  - For any other vertex *v*, there is a path $s \to \ldots \to v \to \ldots \to t$ (在源点和汇点之间, 有经过一些中间顶点*v*的通路: $s \to \ldots \to v \to \ldots \to t$)
- **Flow** – a function $f : V \times V \to \mathbf{R}$ (流: 函数: $f : V \times V \to \mathbf{R}$)
  - *Capacity constraint*: For all $u, v \in V$: $f(u,v) \leq c(u,v)$ (容量约束)
  - *Skew symmetry*: For all $u, v \in V$: $f(u,v) = -f(v,u)$ (斜对称)
  - *Flow conservation:* For all $u \in V - \{s, t\}$: (流守恒)

$$\sum_{v \in V} f(u,v) = f(u,V) = 0, \text{ or}$$

$$\sum_{v \in V} f(v,u) = f(V,u) = 0$$

Cancellation of flows(流的抵消)

- *Do we want to have positive flows going in both directions between two vertices? (在两个顶点间, 是否需要相向对流?)*
  - No! such flows *cancel* (maybe partially) each other(不需要, 可抵消)
  - Skew symmetry – notational convenience (斜对称性只是为了符号表示方便性)



Maximum flow(最大流问题)

- *使流f最大化*
  - ***Value*** **of the flow** ***f*****:**

$$|f| = \sum_{v \in V} f(s,v) = f(s,V) = f(V,t)$$

Want to compute max rate that we can ship material from a designated source to a designated sink. (比如, 从源点到汇点的物流最大化)

The Ford-Fulkerson Method

思想:

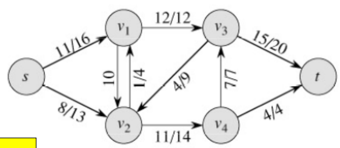
- Try to improve the flow, until we reach the maximum. (不断的增大流, 直到达到流的极大值)
- The residual capacity residual network (通过剩余流和剩余流图实现)

Residual network(剩余网络流图)

- It is any path in *residual network*:
  - Residual capacities: $c_f(u,v) = c(u,v) - f(u,v)$ (剩余流)
  - Residual network: $G_f=(V,E_f)$, where (剩余网络流图) $E_f = \{(u,v) \in V \times V : c_f(u,v) > 0\}$
  - Observation – edges in E_f are either edges in E or their reversals: $|E_f| \leq 2|E|$ ($|E_f| \leq 2|E|$: 因为在剩余网络流图E_f中的边要么在E中, 要么反向, 边数: $|E_f| \leq 2|E|$)

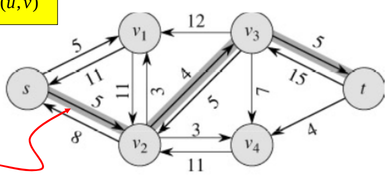
Example of residual capacities(例)

Network:



$\boxed{c_f(u,v) = c(u,v) - f(u,v)}$

Residual Network: (剩余图)



Augmenting path(增流通路)

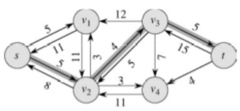
Augmenting Paths(增流通路)

- An augmenting path p is a simple path from s to t on the residual network. (增流通路p是剩余图中从源点s到汇点t的一条通路)
- We can put more flow from s to t through p. (通过增流通路p, 可以提高网络中的流, 即存在$a > 0$, 对于p上的边(u,v), 有$f(u,v) + a \leq c(u,v)$)
- We call the maximum capacity by which we can increase the flow on p the residual capacity of p. (增流通路p的剩余容量是通过该通过可以增加的最大的流容量)

$$c_f(p) = \min\{c_f(u,v) : (u,v) \text{ is on } p\}$$

Residual capacity of a path

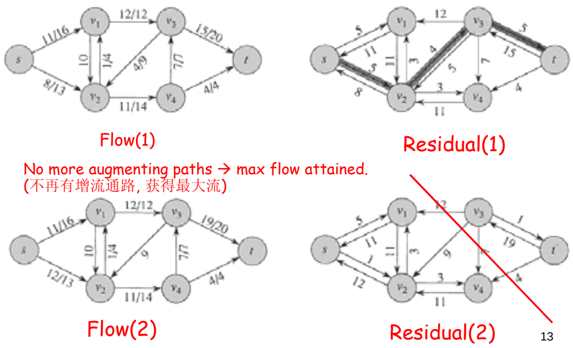
- *Residual capacity* of a path *p* in G_f: (增流通路p的剩余容量) $c_f(p) = \min\{c_f(u,v): (u,v) \text{ is in } p\}$
- Doing augmentation: for all (u,v) in p, we just add this $c_f(p)$ to $f(u,v)$ (and subtract it from $f(v,u)$) (增流操作: 对于p上的边(u,v), 把$c_f(p)$加到$f(u,v)$)
- Resulting flow is a valid flow with a larger value. (得到的是增流后的网络流图)
- *What is the residual capacity of the path* (s,v2,v3,t)?



Ford-Fulkerson Method

```
FORD-FULKERSON-METHOD(G, s, t)
initialize flow f to 0
while there exists an augmenting path p
      do augment flow f along p
return f
```

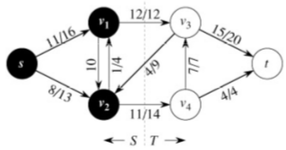
Example



Cuts(切割)

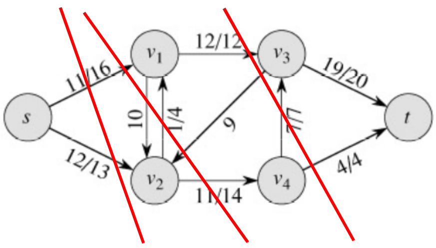
- A *cut* is a partition of *V* into *S* and *T* = *V* – *S*, such that *s* ∈ *S and t* ∈ *T* (一个切割将*V*分割成*S*和*T* = *V* – *S*, 使得*s* ∈ *S*和*t* ∈ *T*)
- The *net flow* (*f*(*S,T*)) through the cut is the sum of flows *f*(*u,v*), where *u* ∈ *S and v* ∈ *T* (流经切割的网络流*f*(*S,T*)是所有边(*u,v*), *u* ∈ *S*, *v* ∈ *T*的流*f*(*u,v*)之和)
- The *capacity* (*c*(*S,T*)) of the cut sum of capacities *c*(*u,v*), where *u* ∈ *S and v* ∈ *T* (切割的网络流容量*c*(*S,T*)是所有边(*u,v*), *u* ∈ *S*, *v* ∈ *T*的流*c*(*u,v*)之和)
- *Minimum cut* – a cut with the smallest capacity of all cuts(最小切割即为具有最小容量的切割)

- *How to determine the maximum flow?*



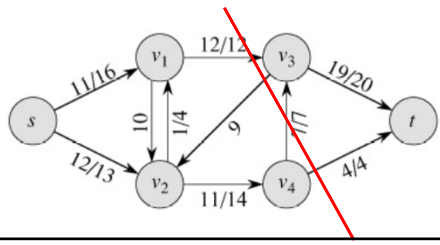
Lemma 引理(7.6)

- For any cut (S,T), the net flow across (S,T) is f(S,T)=|f|. (对于任何切割(S,T)和流f, 有 f(S,T)=|f|)



Corollary推论

- **The value of any flow f in a flow network G is bounded from above by the capacity of any cut of G. (网络流图G中的任意流f以任何切割的流容量为上限)**



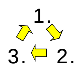
Theorem (7.9) 最大流最小切割定理 (Max-flow min-cut theorem)

- **If f is a flow in a flow network G=(V,E), with source s and sink t, then the following conditions are equivalent: (对于网络流图G=(V,E), s是源点, t是汇点, f是G中的流, 下面三个语句是等价的)**

1. **f is a maximum flow in G. (f是G中的最大流)**
2. **The residual network G_f contains no augmented paths. (剩余流图G_f中不再有增流通路)**
3. **|f| = c(S,T) for some cut (S,T) (a min-cut).** **(存在一个切割(S,T), |f| = c(S,T))**

Correctness of Ford-Fulkerson

- We have to prove three parts: (证明方式, 阅读教材)

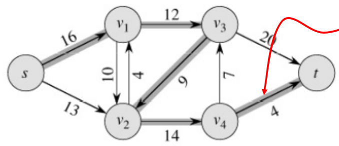


- From this we have 1.⇔2., which means that the Ford-Fulkerson method always correctly finds a maximum flow (该定理保障了Ford-Fulkerson方法的正确性)

The Basic Ford-Fulkerson Algorithm (基本F-F算法)

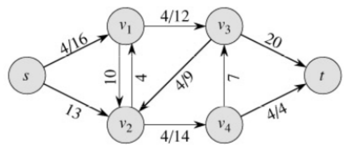
```
FORD-FULKERSON(G, s, t)
  for each edge (u, v) ∈ E[G]
      do f[u, v] ← 0
         f[v, u] ← 0
  while there exists a path p from s to t in the residual network G_f
      do c_f(p) ← min {c_f(u, v) : (u, v) is in p}      增流通路的容量
         for each edge (u, v) in p
             do f[u, v] ← f[u, v] + c_f(p)
                f[v, u] ← −f[u, v]
```


Example



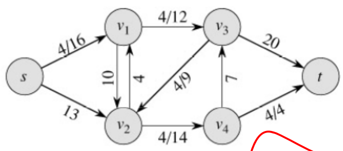
augmenting path

Original Network

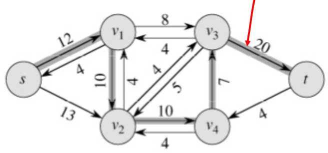


Resulting Flow = 4

Example



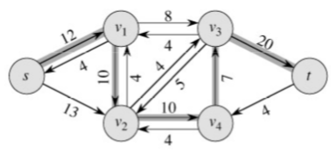
Resulting Flow = 4



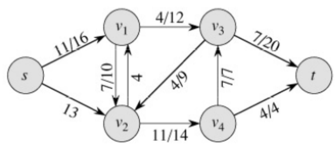
augmenting path

Residual Network

Example

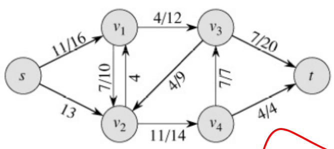


Residual Network

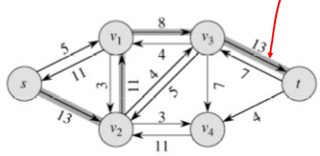


Resulting Flow = 11

Example



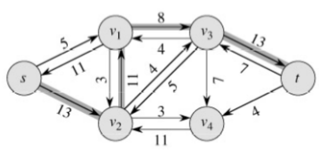
Resulting Flow = 11



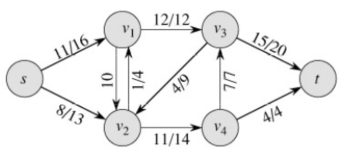
augmenting path

Residual Network

Example

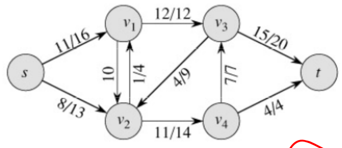


Residual Network



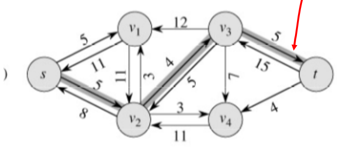
Resulting Flow = 19

Example



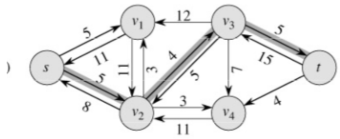
Resulting
Flow = 19

augmenting path

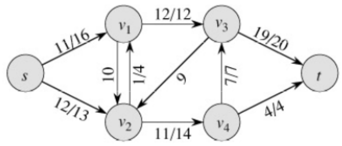


Residual Network

Example

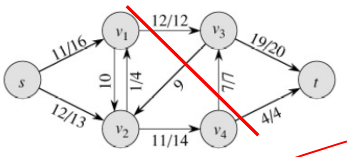


Residual Network



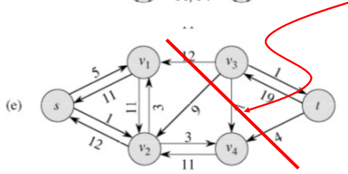
Resulting
Flow = 23

Example



Resulting
Flow = 23

No augmenting path:
Maxflow=23



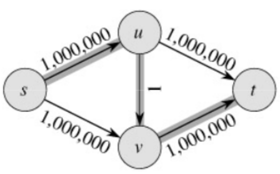
Residual Network

Analysis

- If capacities are all integer, then each augmenting path raises |f| by ≥ 1. (如果容量为整数, |f|每次增流≥ 1)
- If max flow is f*, then need ≤ |f*| iterations, so the time is O(E|f*|). (如果最大流为f*, 所以增流次数≤ |f*| ,时间开销O(E|f*|))
- Note that this running time is not polynomial in input size. It depends on |f*|, which is not a function of |V| or |E|. (时间开销不是输入规模的多项式, 与|f*|有关, 不是|V|或|E|的函数)
- If capacities are rational, can scale them to integers. (如果容量是有理数, 可以换算为整数)
- If irrational, FORD-FULKERSON might never terminate! (如果是无理数, 有可能不会终止)

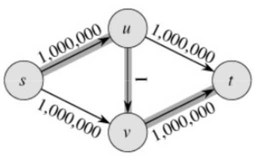
The basic Ford-Fulkerson Algorithm

- This problem is real: Ford-Fulkerson may perform very badly if we are unlucky: (增流通路选择十分关键, 否则运行情况将十分的糟糕)

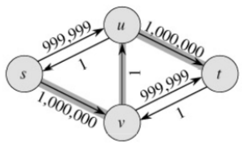


|f*|=2,000,000

Run Ford-Fulkerson on this example

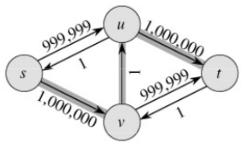


Augmenting Path

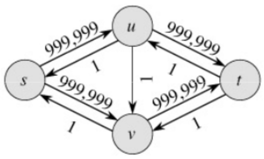


Residual Network

Run Ford-Fulkerson on this example

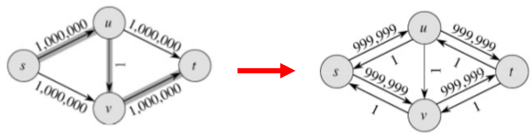


Augmenting Path



Residual Network

Run Ford-Fulkerson on this example



- Repeat 999,999 more times… (重复次数)

The Edmonds-Karp Algorithm (E-K算法)

- **A small fix to the Ford-Fulkerson algorithm makes it work in polynomial time. (对F-F算法的修改, 多项式开销)**
- **Specify how to compute the path in line 4. (第4行确定方法)**

FORD-FULKERSON(G, s, t)

1 **for** each edge $(u, v) \in E[G]$
2   **do** $f[u, v] \leftarrow 0$
3         $f[v, u] \leftarrow 0$
4 **while** there exists a path p from s to t in the residual network G_f
5   **do** $c_f(p) \leftarrow \min\{c_f(u, v) : (u, v) \text{ is in } p\}$
6   **for** each edge (u, v) in p
7   **do** $f[u, v] \leftarrow f[u, v] + c_f(p)$
8                 $f[v, u] \leftarrow -f[u, v]$

The Edmonds-Karp Algorithm (E-K算法)

- **Compute the path in line 4 using breadth-first search on residual network. (在剩余图中用广度优先搜索来计算第4行)**
- **The augmenting path p is the shortest path from s to t in the residual network (treating all edge weights as 1). (增流通路p是剩余图中从s到t的最短路通路)**
- **Runs in time O(V E²). (开销)**

Edmonds-Karp algorithm

- *Take **shortest path** (in terms of number of edges) as an augmenting path – Edmonds-Karp algorithm*
  - *How do we find such a shortest path?*
  - Running time $O(VE^2)$, because the number of augmentations is $O(VE)$
  - To prove this we need to prove that:
    - *The length of the shortest path does not decrease*
    - *Each edge can become **critical** at most ~ V/2 times.* Edge (u,v) on an augmenting path p is critical if it has the minimum residual capacity in the path:
      $c_f(u,v) = c_f(p)$

Non-decreasing shortest paths

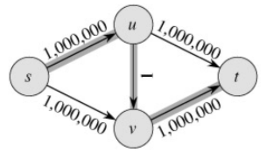
- *Why does the length of a shortest path from s to any v does not decrease?*
  - Observation: Augmentation may add some edges to residual network or remove some.
  - Only the added edges (“shortcuts”) may potentially decrease the length of a shortest path.
  - Let’s supose $(s,\ldots,v)$ – the shortest decreased-length path and let’s derive a contradiction

Number of augmentations

- *Why each edge can become critical at most ~V/2 times?*
  - Scenario for edge (u,v):
    - Critical the first time: (u,v) on an augmenting path
    - Disappears from the network
    - Reappears on the network: (v,u) has to be on an augmenting path
    - We can show that in-between these events the distance from s to u increased by at least 2.
    - This can happen at most V/2 times
- We have proved that the running time of Edmonds-Karp is $O(VE^2)$.

37

The Edmonds-Karp Algorithm - example



- Edmonds-Karp's algorithm runs only 2 iterations on this graph. (2次重复)

38

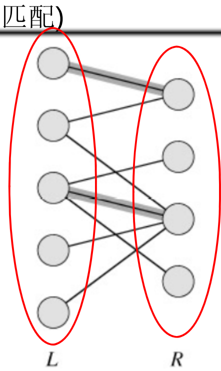
Further Improvements (更多改进算法)

- Push-relabel algorithm ([CLRS, 26.4]) – $O(V^2 E)$.
- The relabel-to-front algorithm ([CLRS, 26.5) – $O(V^3)$.
- The scaling Max-Flow algorithm (section 7.3) – $O(E^2 \log C)$, where C is the maximum integer capacity.

39

Maximum Bipartite Matching (最大二分图匹配)

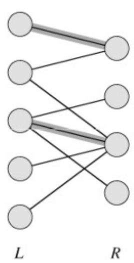
- **A bipartite graph is a graph G=(V,E) in which V can be divided into two parts L and R such that every edge in E is between a vertex in L and a vertex in R. (二分图G=(V,E), V分成L和R, E中任意边的顶点分别在L和R中)**
- e.g. vertices in L represent skilled workers and vertices in R represent jobs. An edge connects workers to jobs they can perform. (L: 工人, R: 工作)

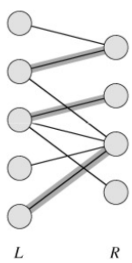


40

•A matching in a graph is a subset M of E, such that for all vertices v in V, at most one edge of M is incident on v.

匹配是E的一个子集M, 对于V中的任何顶点v, M中至多有一条边以它为顶点

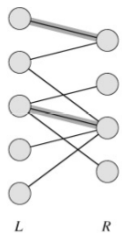




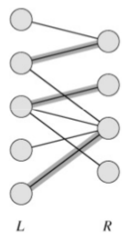
41

•A maximum matching is a matching of maximum cardinality. (最大匹配即具有最大的势的匹配)

not maximum



maximum

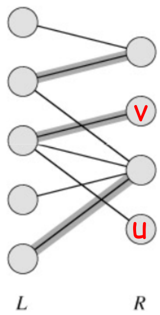


42

A Maximum Matching(最大匹配)

- **No matching of cardinality 4, because only one of v and u can be matched. (不存在势为4的匹配, 因为v和u只有一个可能匹配)**

- **In the workers-jobs example a max-matching provides work for as many people as possible. (如果L为工人, R为工作, 最大匹配意味为最多人提供工作)**



43

Solving the Maximum Bipartite Matching Problem(最大匹配求解)

- Reduce an instance of the maximum bipartite matching problem on graph G to an instance of the max-flow problem on a corresponding flow network G'. (转换为最大流问题)
- Solve using Ford-Fulkerson method. (用F-F方法求解)

44

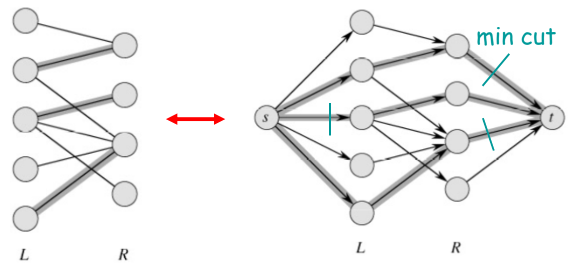
Corresponding Flow Network (相应的网络流图)

- **To form the corresponding flow network G' of the bipartite graph G: (从二分图G生成相应的网络流图G')**
  - **Add a source vertex s and edges from s to L. (增添源点s和从s到L的边)**
  - **Direct the edges in E from L to R. (保持原来的E)**
  - **Add a target vertex t and edges from R to t. (增添汇点t和从R到t的边)**
  - **Assign a capacity of 1 to all edges. (边的容量为1)**

•Claim: max-flow in G' corresponds to a max-bipartite-matching on G. (G'的最大流对应G的最大匹配)

45

Example



|M| = 3 ↔ max flow = 3

46