HIT CS&E

第七章 MaxMin方法

骆吉洲
计算机科学与技术学院

HIT CS&E

提纲

HIT CS&E

教学目的

难点： Max-Min关系及其在算法设计和分析中的应用

重点： 基本算法层面： (1)最大流-最小割算法
(2)最大匹配-最小覆盖算法
(3)基本图论算法的总结和复习

算法设计技术层面： (1)Max-min方法
(2)精益求精的算法设计过程

问题特征分析能力层面： (1)准确，渐进

参考书和最新文献:

1. 网络流：理论、算法和应用. 机械工业出版社，2004
2. Incremental graph pattern matching, VLDB, 2011
3. On the complexity of view update and application in annotation propagation. TKDE, 2012

HIT CS&E

7.1 Maximum Flow

HIT CS&E

7.1.1 流网络与流

构建和使用各种网络时，需要考虑网络的通行能力
网络的通行能力受网络结构的限制
- 道路的宽窄
- 管道的粗细
- ……

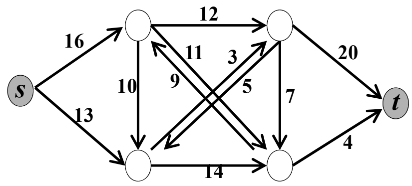
流网络的种类
- 英特网
- 电话网
- 高速路网
- 铁路网
- 电网
- 输气网络
- 排水网络
- 输水网络

能否建立通用模型来研究网络的最大通行能力呢？

HIT CS&E

流网络

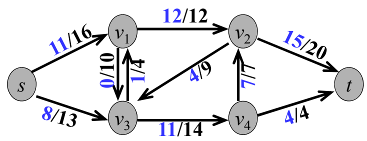
加权有向图G=(V,E)
- 顶点表示网络中的关键节点
- 边 uv表示u和v之间的连接，容量$c(u,v)\geq 0$表示边的通行能力；如果$uv\notin E$，则定义容量$c(u,v)=0$
- 两个特殊顶点s和t，s称为源(source)，t称为汇(sink)
- G中每个顶点均位于某条由s到t的路径上($|E|\geq|V|$-1)



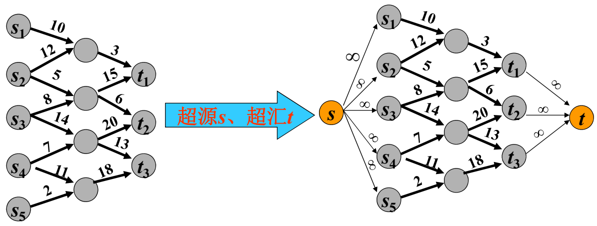
流

流网络G=(*V*,*E*)上的一个流

- 是一个实值函数f:$V\times V\rightarrow R$, 满足
  - 容量约束：$f(u,v)\le c(u,v)$ 对$\forall uv\in E$成立
  - 反对称性：$f(u,v) = -f(v,u)$
  - 守恒约束：$\sum_{v\in V} f(u,v)=0$对任意 $u\in V$-{s,t}成立
- $f(u,v)$称为顶点u到顶点v的流量,可为正、负、零值
- 流f的值定义为$|f|= \sum_{v\in V} f(s,v)$



多源多汇的网络



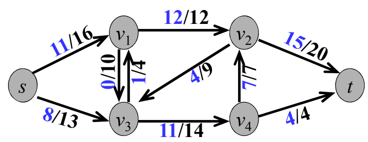
只需讨论单源单汇的网络流

HIT CS&E

最大流问题

给定流网络 $G=(V,E)$,其源为s，汇为t，找出从s 到t 的最大流.

- 怎样高效率地找?

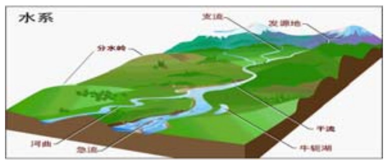


HIT CS&E

直观认识

河水的最大流量取决于

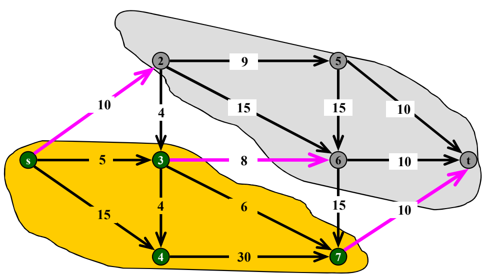
干流中河道狭窄处的通行能力



这种观察能否用于研究最大流问题呢？

HIT CS&E

从s流到t的最大流量不会超过10+8+10=28

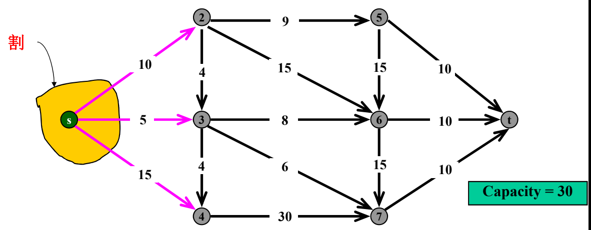


HIT CS&E

割

给定流网络 $G=(V,E)$,其源为s，汇为t

G的一个割(cut)是V的2-集合划分S,T ($T=V-S$)使得$s\in S$, $t\in T$



割的容量定义为 $c(S,T)=\sum_{u\in S,v\in T} c(u,v)$

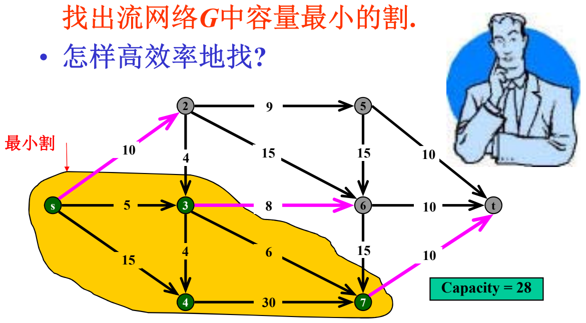
定义

最小割问题

HIT CS&E

给定流网络 $G=(V,E)$,其源为s，汇为t，找出流网络G中容量最小的割.

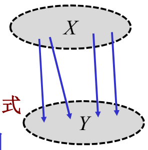
- 怎样高效率地找?



弱对偶关系（预备）

顶点集之间的流量

- 设$X,Y\subseteq V$ 记$f(X,Y)=\sum_{x\in X}\sum_{y\in Y}f(x,y)$
- 称$f(X,Y)$顶点集X和Y之间的流量
- 守恒约束等价于 $f(u,V)=0$对$\forall u\in V\text{-}s\text{-}t$
- 上述表示经常用来简化网络流涉及的等式



引理1 设f是流网络$G=(V,E)$中的一个流，则

- $f(X,X)=0$ 对$\forall X\subseteq V$ 成立
- $f(X,Y)=-f(Y,X)$ $\forall X,Y\subseteq V$ 成立
- $\forall X,Y,Z\subseteq V$ 且$X\cap Y=\phi$，有
  $f(X\cup Y,Z)=f(X,Z)+f(Y,Z)$
  $f(Z,X\cup Y)=f(Z,X)+f(Z,Y)$

故，$|f|=f(s,V)$ $=f(V,V)-f(V\text{-}s,V)$ $=-f(V\text{-}s,V)$ $=f(V,V\text{-}s)$
$=f(V,t)+f(V,V\text{-}s\text{-}t)$
$=f(V,t)$

最大流－最小割间弱对偶关系

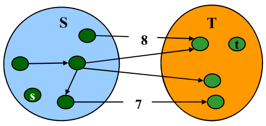
HIT CS&E

引理2 给定流网络 $G=(V,E)$,s是源,t是汇. 设f是G上的一个流，S,T是G的一个割，则$f(S,T)=|f|$.

证明: $f(S,T)=f(S,V)\text{-}f(S,S)$ 引理1第3部分 → $f(S,V)=f(S,T)+f(S,S)$
$=f(S,V)$ 引理1第1部分
$=f(s,V)+f(S\text{-}s,V)$ 引理1第3部分
$=f(s,V)$ $f(S\text{-}s,V)=0$
$=|f|$

引理3 给定流网络 $G=(V,E)$. 设f是G上的一个流，S,T是G的一个割，则$|f|\le c(S,T)$

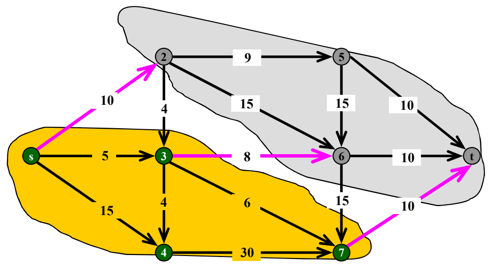
$$|f|=f(S,T)$$
$$=\sum_{u\in S}\sum_{v\in T}f(u,v)$$
$$\le\sum_{u\in S}\sum_{v\in T}c(u,v)$$
$$=c(S,T)$$



最大流－最小割间的弱对偶关系

HIT CS&E

引理 给定流网络 $G=(V,E)$. 设f是G上的任意一个流，S,T 是G的任意一个割，则$|f|\le c(S,T)$



怎么用这个max-min关系来高效地求出最大流（最小割）呢？

利用max-min关系设计最大流算法

HIT CS&E

1.初始化一个可行流f

➢0-流：所有边的流量均等于0的流

2.不断将f增大，直到f不能继续增大为止

3.找出一个割S,T使得$|f|=c(S,T)$

➢由此断言f是最大流，而S,T是最小割

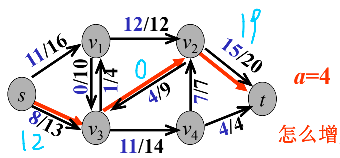
max-min关系提供了高效求解最大流-最小割问题的机制！
关键在于第2步怎么做？

7.1.2 Ford-fulkerson方法

HIT CS&E

算法的基本思想

- 对于当前的流,…
- …找出一条从s到t的路径 p (增广路径)和正数$a>0$, 使得p上的每条边 uv的流量增加a之后仍然满足容量约束，即 $f(u,v)+a\le c(u,v)$
- 则将p上每条边的流量增加a，得到一个更大的流



$a=4$

怎么增大当前的流？

Ford-fulkerson算法概要

算法Ford-Fulkerson(G,s,t)
Input 流网络G，源s，汇t
Output G中从s到t的最大流
1 初始化所有边的流量为0
2 **while** 存在增广路径p **do**
3 沿路径p增大流量得到更大的流f
4 **return** f

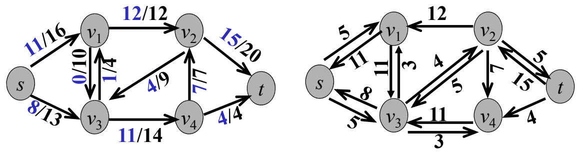
- 增广路径如何找?
- 增广路径上可以增加的流量有多大?
- 该方法总能找到最大流吗?

剩余网络(Residual Network)

增广路径如何找?

我们擅长找路径

寻找增广路径能否转变为在某个图中找路径呢?



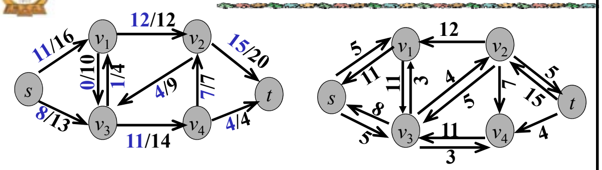
E_f={uv: $c_f(u,v)=c(u,v)-f(u,v)>0$}

剩余网络(形式定义)

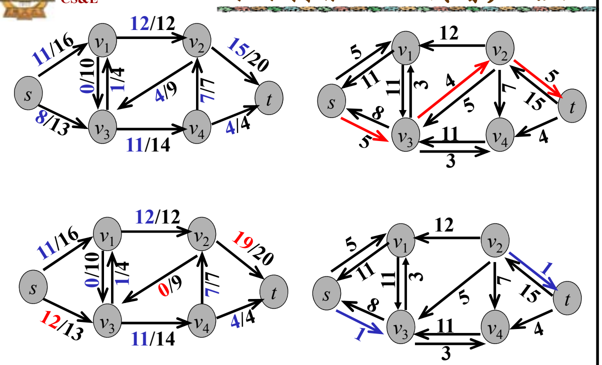
增广路径如何找?

- 增广路径是Residual network中从s到t的路径
  - Residual capacities: $c_f(u,v) = c(u,v) - f(u,v)$
  - Residual network: $G_f=(V,E_f)$, 其中
    $E_f = \{(u,v)\in V\times V \mid c_f(u,v) > 0\}$
  - $f(u,v) < c(u,v)$, 则 $c_f(u,v) = c(u,v) - f(u,v)>0$, $(u,v)\in E_f$
    $f(u,v) >0$ 则 $c_f(v,u) = c(v,u) - f(v,u)>0$, $(v,u)\in E_f$
    $c(u,v)=c(v,u)=0$, 则$f(u,v)=f(v,u)=0$, 进而 $c_f(u,v) = c_f(v,u) =0$
  注意: E_f中的边要么是E中的边，要么是E中边的反向边: $|E_f|\leq 2|E|$
- Residual Network本身也可以看成是流网络

利用剩余网络找增广路径



利用剩余网络找增广路径



增广路径的剩余容量

增广路径上可以增大多少流量?

- P是G_f中的一条增广路径
  - 其剩余容量 $c_f(p) = \min\{c_f(u,v)$: (u,v) 是路径 p上的边}
- 增广过程： 对路径p上的每条边uv
  - 要么在边uv的流量上增加 $c_f(p)$,即 $f(u,v)=f(u,v)+c_f(p)$
  - 要么在边vu的流量上减去$c_f(p)$, 即 $f(v,u)=f(v,u)-c_f(p)$
  - 具体属于哪种情况，根据G
- 增广过程完成后，得到值更大的流

HIT CS&E

Ford-Fulkerson算法

算法Ford-Fulkerson(G,s,t)

Input 流网络G，源s，汇t

Output G中从s到t的最大流

1. For $\forall uv \in E[G]$ do
2. $f(u,v) \leftarrow 0$ → 初始化流
3. $f(v,u) \leftarrow 0$
4. While G_f存在增广路径p do
5. $c_f(p)=\min\{c_f(u,v) \mid uv$是$p$上的边$\}$
6. For p上的每条边uv do
7. If uv是流网络中的边 Then
8. $f(u,v) \leftarrow f(u,v)+c_f(p)$
9. $f(v,u) \leftarrow -f(u,v)$
10. Else
11. $f(v,u) \leftarrow f(v,u)-c_f(p)$
12. $f(u,v) \leftarrow -f(v,u)$
13. 修改剩余中网络G_f相应的边

HIT CS&E

Ford-fulkerson算法的正确性

引理4 给定流网络 $G=(V,E)$,s是源,t是汇; f是G上的一个流. G_f是流f在G上导出的Residual Network，f'是G_f上的一个流. 则$f+f'$是G上值为$|f|+|f'|$的流。

其中，$(f+f')(u,v)= f(u,v)+ f'(u,v)$

证明：(作业)

(1)验证反对称性

(2)验证容量约束

(3)验证守恒约束

(4)验证$|f+f'|= |f|+|f'|$

HIT CS&E

Ford-fulkerson算法的正确性

引理4 (最大流－最小割定理)给定流网络 $G=(V,E)$,s是源,t是汇; f是G上的一个流. 则下列论断等价。

(1)f是最大流;

(2)G_f中不存在增广路径

(3)对G的某个割(S,T), $|f|=c(S,T)$

证明：(1)⇒(2) 反证. 设p是G_f是最大流f对应的增广路径，其剩余容量为f_p。由引理4知道，$f+f_p$是一个值比$|f|$的流. 这与f是最大流矛盾。

(2)⇒(3) 由于G_f中没有从s到t的路径，定义S={v:G_f中存在从s到v的路径}，$T=V-S$。显然$s\in S$且$t\in T$。$\forall$u $\in S$且$v\in T$，$f(u,v)=c(u,v)$，否则uv $\in E_f$进而导致v $\in S$。于是S,T是G的一个割。由引理3知道$|f| = f(S,T)$=c(S,T)

(3)⇒(1) 引理2表明$|f|\le c(S,T)$, 故$|f|=c(S,T)$表明f是最大流。

HIT CS&E

复杂性(1)

算法Ford-Fulkerson(G,s,t)

Input 流网络G，源s，汇t

Output G中从s到t的最大流

1. For $\forall uv \in E[G]$ do
2. $f(u,v) \leftarrow 0$
3. $f(v,u) \leftarrow 0$
4. While G_f存在增广路径p do — $O(|E_f|)=O(|E|)$
5. $c_f(p)=\min\{c_f(u,v) \mid uv$是$p$上的边$\}$
6. For p上的每条边uv do
7. If uv是流网络中的边 Then
8. $f(u,v) \leftarrow f(u,v)+c_f(p)$
9. $f(v,u) \leftarrow -f(u,v)$
10. Else
11. $f(v,u) \leftarrow f(v,u)-c_f(p)$
12. $f(u,v) \leftarrow -f(v,u)$
13. 修改剩余中网络G_f相应的边

$O(|p|)=O(|E|)$

HIT CS&E

复杂性(2)

Ford-Fulkerson算法的时间复杂度$O(|f^*||E|)$

怎么改进Ford-Fulkerson算法呢？

1.选用好的路径计算方法

- 明确选用BFS (Edmonds-Karp算法)

2.计算特殊的增广路径

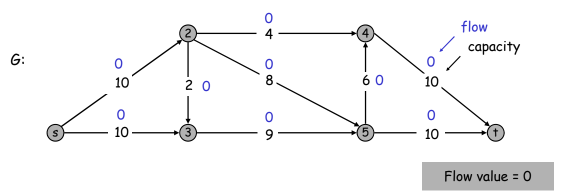
- 第5步：剩余容量 $c_f(p)$达到最大值(习题7.9)

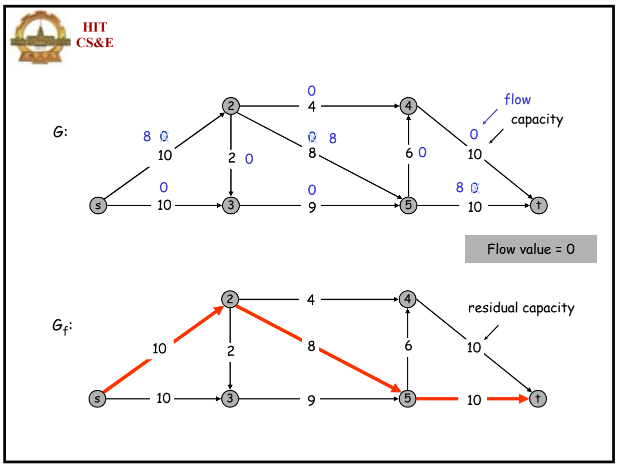
3.利用“最大流等于最小割”这一结论重新设计其他算法

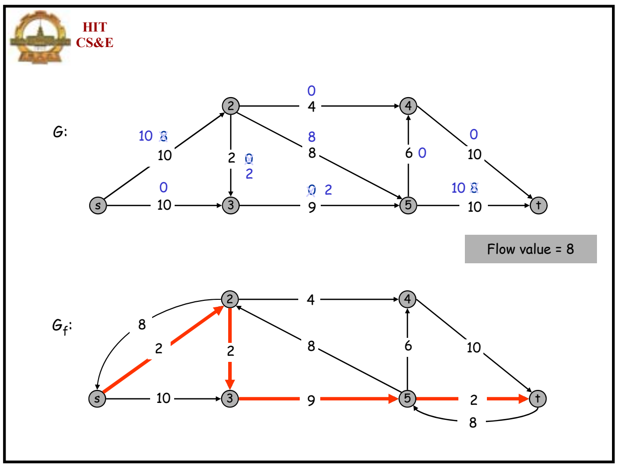
- 7.1.3节和7.1.4节

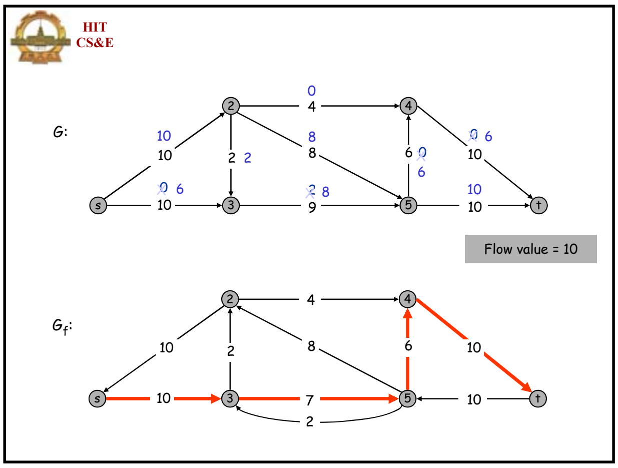
HIT CS&E

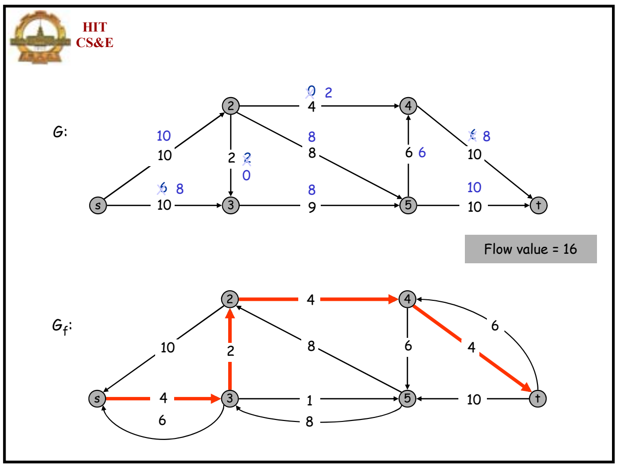
例

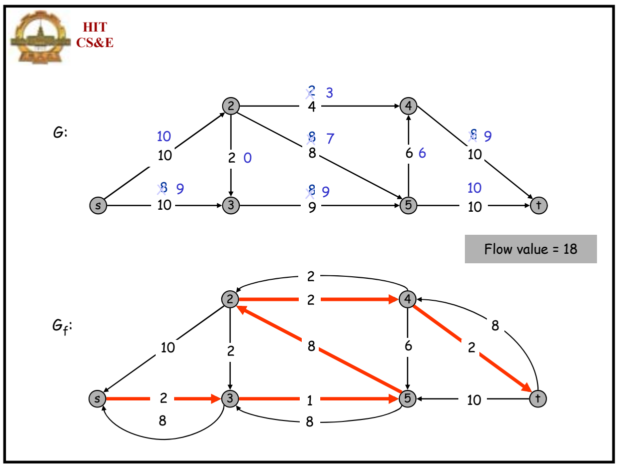


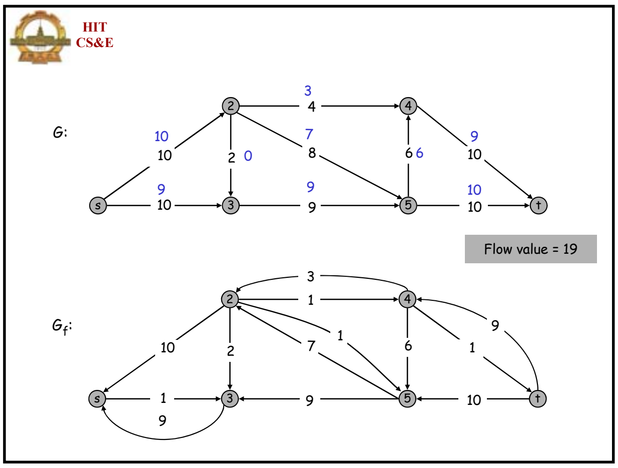

HIT
CS&E
G:
flow
capacity
Flow value = 0
G_f:
residual capacity

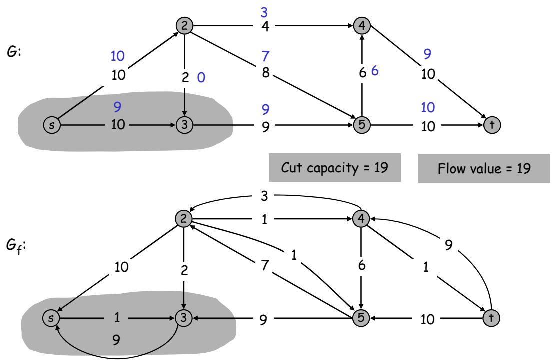

HIT
CS&E
G:
Flow value = 8
G_f:


HIT
CS&E
G:
Flow value = 10
G_f:


HIT
CS&E
G:
Flow value = 16
G_f:


HIT
CS&E
G:
Flow value = 18
G_f:


HIT
CS&E
G:
Flow value = 19
G_f:

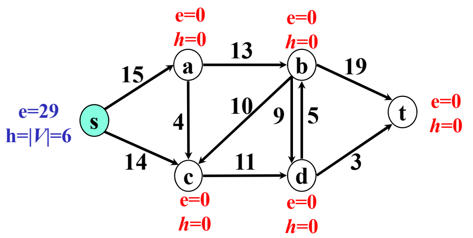


7.1.4 推送复标算法

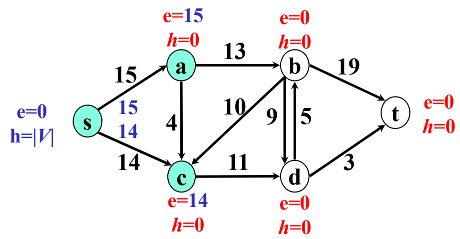
- Ford-Fulkson算法
  - ✓构造剩余网络
  - ✓选取s-t增广路径来增大流值
  - ✓具有全局优化的观点
- 能否只考虑顶点的局部情况
  - ✓逐个顶点查看
  - ✓仅查看其邻接点
  - ✓确定流值的变化

基本思想

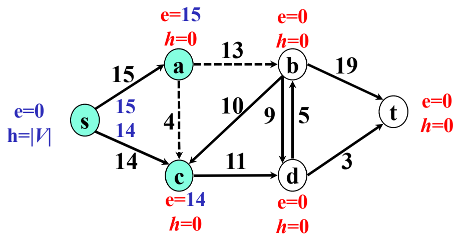
- 将流网络看成管道网络
  - ✓顶点具有库存能力，$s.e=\sum_{su\in E}c(uv)$，$u.e=0$
  - ✓顶点具有不同高度，$s.h=|V|$，$u.h=0$
  - ✓液体将怎么流动？



- 将流网络看成管道网络
  - ✓顶点具有库存能力，$s.e=\sum_{su\in E}c(uv)$，$u.e=0$
  - ✓顶点具有不同高度，$s.h=|V|$，$u.h=0$
  - ✓从高度较高的顶点流向高度较低的顶点



- a有库存，但a的高度可能流动顶点高度一样
  - ✓修改高度(复标高度)
  - ✓确保流可以继续流动
  - ✓如何修改a的高度？

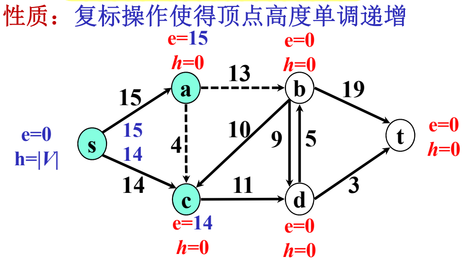


复标操作

- u有库存，但u的高度与可能流动顶点高度一样
  - ✓ $u.e>0$
  - ✓ $uv\in E_f$ ($c(uv)-uv.f>0$,剩余网络边), $u.h\le v.h$

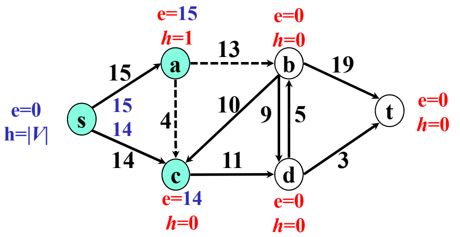
操作：$u.h = 1+\min\{v.h: uv\in E_f\}$

性质：复标操作使得顶点高度单调递增



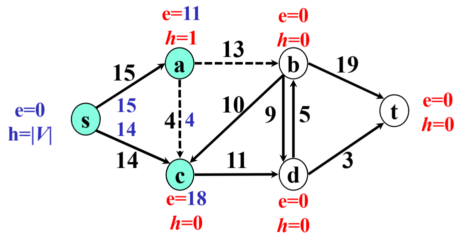
HIT CS&E

- a有库存，但a的高度与其他顶点高度一样
  - ✓ 修改高度(复标高度)
  - ✓ 确保流可以继续流动
  - ✓ a.h = 1+ min {c.h, b.h} =1



HIT CS&E

- a有库存可以向c流动4



HIT CS&E

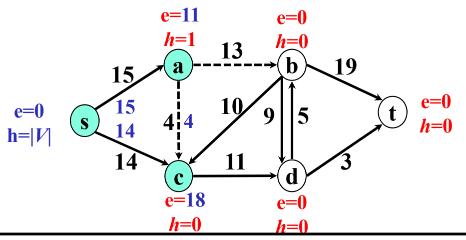
推送操作

- u的库存可向其相邻顶点流动，则将库存推送至此
  - $u.e>0$, $u.h = v.h + 1$ 且 $c(uv)-uv.f>0$

操作：1.If $uv \in E$ Then $uv.f = uv.f + \min\{u.e, c(uv)-uv.f\}$

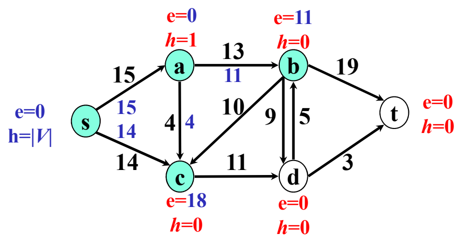
2.Else $vu.f = vu.f - \min\{u.e, c(uv)-uv.f\}$

3.$u.e = u.e - \Delta_f$, $v.e = v.e + \Delta_f$



HIT CS&E

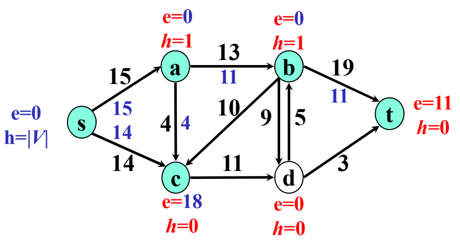
- Push(a,b)还可以将11推送到b
- 推送操作完成后，如果c(uv)-f(uv)=0,则称饱和推送
- 性质：非饱和推送完成后，$u.e=0$



HIT CS&E

Relabel (b)

Push(b,t)还可以将11推送到t



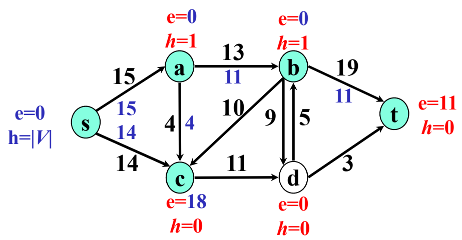
HIT CS&E

推送复标算法

While (推送或复标操作可行)

执行推送或复标操作

/*算法停止吗，怎么办？*/



HIT CS&E

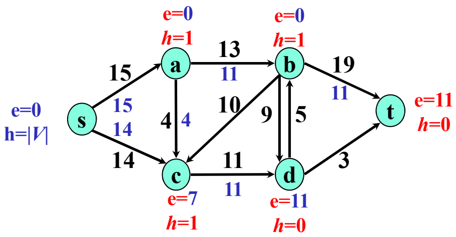
推送复标算法

预流

(1) 除了在顶点可能不满足守恒约束之外

(2) 满足流的其他所有性质

(3) 推送操作将一个预流变成另一个预流



高度

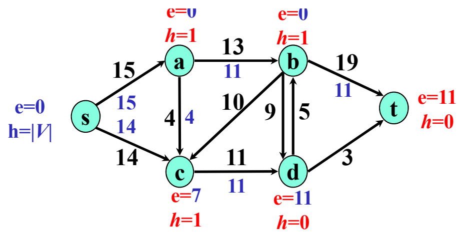
(1) 对任意顶点u赋予一个非负整数值$u.h$

(2)$s.h = |V|$ $t.h$=0保持不变

(3)若$uv \in E_f$, 则 $u.h \le v.h+1$

复标操作：维持高度满足上述三条性质，且高度递增

性质：如果$u.h>v.h+1$,则uv不是剩余边



HIT CS&E

推送复标算法

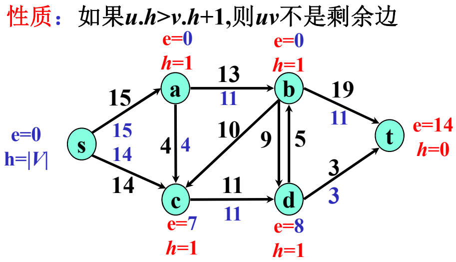
高度

(1) 对任意顶点u赋予一个非负整数值$u.h$

(2)$s.h = |V|$ $t.h$=0保持不变

(3)若$uv \in E_f$, 则 $u.h \le v.h+1$

性质：如果$u.h>v.h+1$,则uv不是剩余边



HIT CS&E

推送复标算法

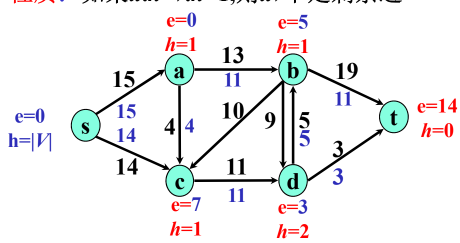
高度

(1) 对任意顶点u赋予一个非负整数值$u.h$

(2)$s.h = |V|$ $t.h$=0保持不变

(3)若$uv \in E_f$, 则 $u.h \le v.h+1$

性质：如果$u.h>v.h+1$,则uv不是剩余边



HIT CS&E

推送复标算法

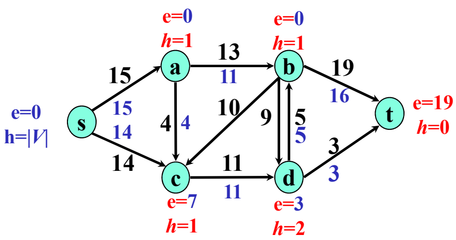
高度

(1) 对任意顶点u赋予一个非负整数值$u.h$

(2)$s.h = |V|$ $t.h$=0保持不变

(3)若$uv \in E_f$, 则 $u.h \le v.h+1$

性质：如果$u.h>v.h+1$,则uv不是剩余边



HIT CS&E

推送复标算法

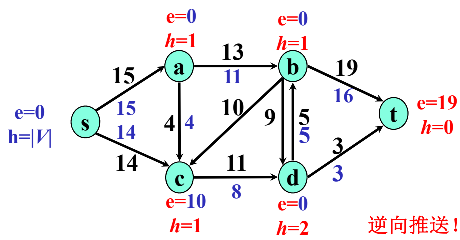
高度

(1) 对任意顶点u赋予一个非负整数值$u.h$

(2)$s.h = |V|$ $t.h$=0保持不变

(3)若$uv \in E_f$, 则 $u.h \le v.h+1$

性质：如果$u.h>v.h+1$,则uv不是剩余边



逆向推送！

HIT CS&E

推送复标算法

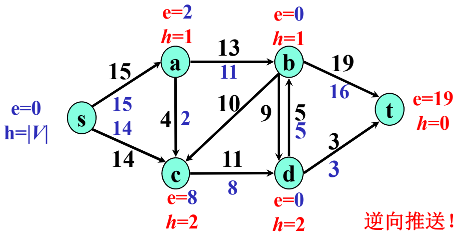
性质：剩余网络中不存在从s到t的路径

事实：(1)若$uv\in E_f$, 则 $u.h\le v.h+1$) (2)$s.h=|V|$, $t.h=0$

不然，$s=v_0,v_1,\ldots,v_k=t$是剩余网络中简单路径 ($k<|V|-1$)

$s.h\le v_1.h+1\le v_2.h+2\le\ldots\le v_k.h+k$

$|V|\le k$　　矛盾



逆向推送！

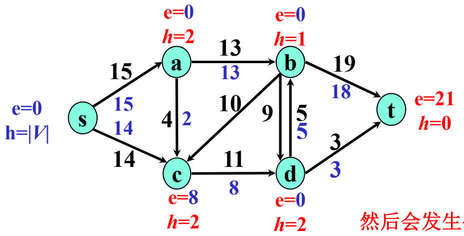
HIT CS&E

u.e>0

推送操作沿uv(某个v)可行 OR 复标操作在u上可行

存在$uv\in E_f$使$u.h=v.h+1$　　$\forall uv\in E_f$满足$u.h<v.h+1$

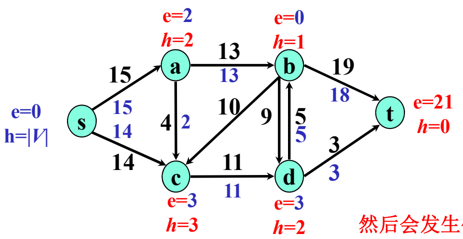
（$uv\in E_f\Rightarrow$ u.h≤v.h+1）



然后会发生什么！

HIT CS&E

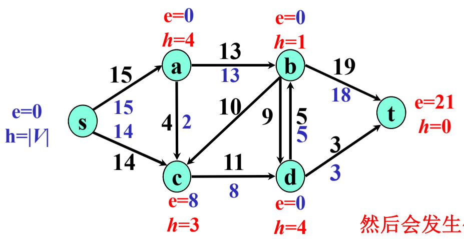
推送复标算法



然后会发生什么！

HIT CS&E

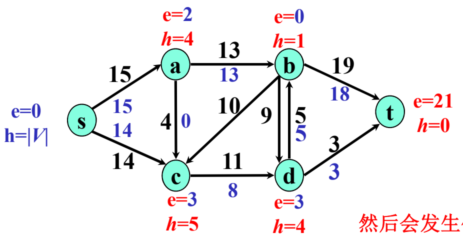
推送复标算法



然后会发生什么！

HIT CS&E

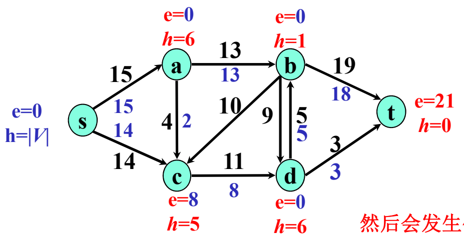
推送复标算法



然后会发生什么！

HIT CS&E

推送复标算法



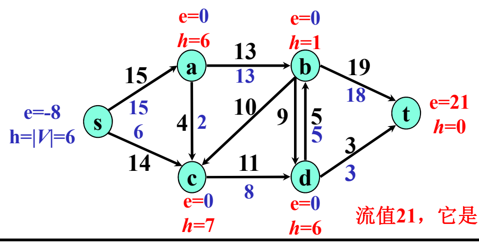
然后会发生什么！

HIT
CS&E

得到了一个流

所有顶点上推送复标操作都不可行

e.u=0　对$\forall u \in V$-{s,t}



流值21，它是最大流吗？

HIT
CS&E

推送复标算法

初始化：　u.e=0;　u.h=0;　uv.f=0　　$\forall$u$\in$V, uv $\in$E

　　s.e= - $\sum_{sv\in E} c(sv)$，$s.h=|V|$

　　$sv.f=c(sv)$，　v.e = c(sv)　　$\forall sv\in E$

While (推送或复标操作可行)

　　执行推送或复标操作

HIT
CS&E

算法正确性

定理.如果推送算法终止，则最后的预流是最大流。

初始化：算法初始化是预流

循环　：每次推送操作或复标操作后，仍是预流

终止　：算法结束后得到一个预流f

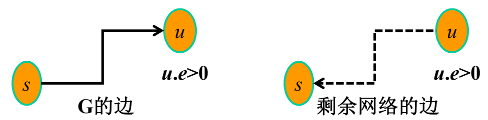
此时$u.e$=0对任意u≠s,t成立，　故f是流

剩余网络中不存在s-t路径，　由最大流最小割定理，f是最大流

HIT
CS&E

算法复杂性

引理5.给定s-t流网络G=(V,E), 对于任意预流f及其上的高度函数h。如果$u.e$>0，则在剩余网络中存在u-s路径。



HIT
CS&E

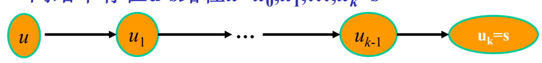
引理6.给定s-t流网络G=(V,E), 则对于任意顶点u，在推送复标算法运行过程的任意时刻，$h(u)\le 2|V|-1$.

若u=s, 则$h(s)$=|V|不变，结论成立

若u=t, 则$h(t)$=0不变，结论成立

若u≠s,t,初始时，$h(u)=0$

对u的每次复标操作时(u.e>0)，由引理5，剩余网络中存在u-s路径$u=u_0,u_1,\ldots,u_k=s$



$h(u)\le h(u_1)+1$　$h(u_1)\le h(u_2)+1$　…　$h(u_{k-1})\le h(u_k)+1$　$h(u_k)$=|V|

$h(u)\le h(s)+k \le 2|V|-1$,　　　$k \le |V|-1$

HIT
CS&E

引理7.给定s-t流网络G=(V,E), 则推送复标算法在G上至多执行$(2|V|-1)(|V|-1) \le 2|V|^2$ 次复标操作。

在s,t上永远不执行复标操作

对于其余每个顶点u（共|V|-1个顶点）

$h(u)$的初始值为0

每次复标使得$h(u)$增大，且引理6表明$h(u) \le 2|V|-1$

故在u上至多执行2|V|-1次复标操作

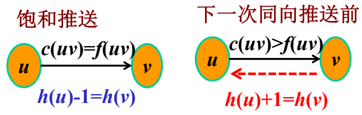
HIT CS&E

引理8.给定*s-t*流网络*G*=(*V*,*E*), 则推送复标算法在G上至多执行2|V||E| 次饱和推送操作。

$\forall u,v\in V$，考察*u*和*v*之间饱和推送的总次数

- 从*u*到*v*的饱和推送
- 从*v*到*u*的饱和推送
- $uv\in E$或$vu\in E$　　顶点对的个数≤|E|

饱和推送　　下一次同向推送前



h(*u*)至少增大2
0≤ *h*(*u*)≤2|V|-1

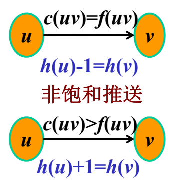
•从*u*到*v*的饱和推送至多|V|次
•同理，从*v*到*u*的饱和推送至多|V|次

引理9.给定*s-t*流网络*G*=(*V*,*E*), 则推送复标算法在G上至多执行4|V|²(|V|+|E|) 次非饱和推送操作。

定义$\varphi=\sum_{u.e>0}h(u)$　初始时，φ=0且φ≥0恒成立

复标顶点u使得其高度增加，导致φ增大
顶点u上的所有复标操作，导致φ增大总量≤2|V|-1
所有复标操作导致φ的总增量≤2|V|²

饱和推送



推送前：u.e>0,　　v.e=0或v.e>0
推送后：u.e=0或u.e>0　v.e>0
每次饱和推送导致φ的增量≤2|V|
所有饱和推送导致φ的增量≤2|V|.2|V||E|

推送前：u.e>0,　　v.e=0或v.e>0
推送后：u.e=0　　v.e>0
每次非饱和推送导致φ至少减小1

由于φ≥0恒成立，总增量≥总减量

HIT CS&E

定理10.给定*s-t*流网络*G*=(*V*,*E*), 则推送复标算法在G上至多执行O(|V|²|E|) 次基本操作后终止。

由引理7，算法至多执行2|V|²次复标操作
由引理8，算法至多执行2|V||E|次饱和推送操作
由引理9，算法至多执行4|V|²(|V|+|E|)次非饱和推送操作

HIT CS&E

7.1.5前置复标算法

推送复标算法

- 以不确定的顺序选择推送、复标顶点
- 对每个顶点的处理不彻底
  - ✓ 对u的推送或复标操作后，u.e>0
  - ✓ 可能转而处理其他顶点
- 时间复杂性O(V^2E)

提高算法性能的着手点

- 如果精细选择推送、复标操作顺序
- 对每个顶点进行彻底处理，处理后u.e=0

HIT CS&E

DisCharge操作

彻底处理顶点*u*

- *u.e*>0, 则要么推送操作可行，要么复标操作可行
- 重复在*u*顶点处进行推送或复标操作，直到*u.e*=0
- 仅需考察*u*的相邻顶点
- 对*u*维护邻接链表*L*(*u*)　$v\in L(u) \Leftrightarrow uv\in E$或$vu\in E$

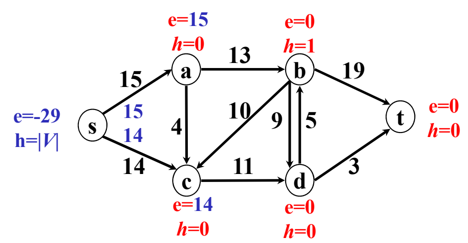
```
DisCharge(u)
  While  u.e>0
    v ← L(u).current          /*考察当前处理的顶点*/
    If  v = Null  Then  ReLabel(u), L(u).current=L(u).head
    ElseIf c(uv)-f(uv)>0 且 u.h=v.h+1  Then Push(u)
    Else L(u).current ← L(u).next
```

考虑初始化操作之后DisCharge(*c*)的执行过程

L(*c*):

s	a	b	d	Null

v=s≠Null，　　　不执行复标操作
c(cs)-f(cs) = 0-(-14) =14 但c.h≠s.h+1 不执行Push



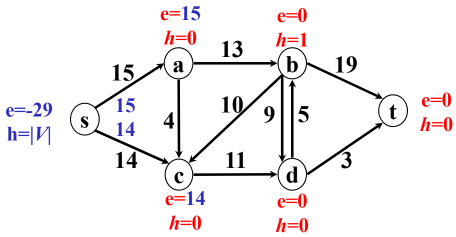
考虑初始化操作之后**DisCharge(*c*)**的执行过程

L(*c*):

s	a	b	d	Null

v=a≠Null， 不执行复标操作

c(ca)-f(ca) = 0-0 =0 不执行推送操作



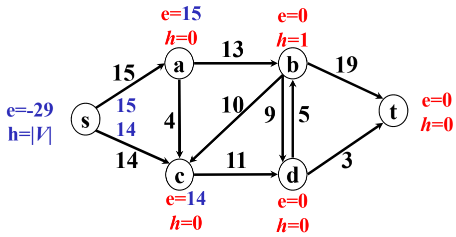
考虑初始化操作之后**DisCharge(*c*)**的执行过程

L(*c*):

s	a	b	d	Null

v=b≠Null， 不执行复标操作

c(cb)-f(cb) = 0-0 =0 不执行推送操作



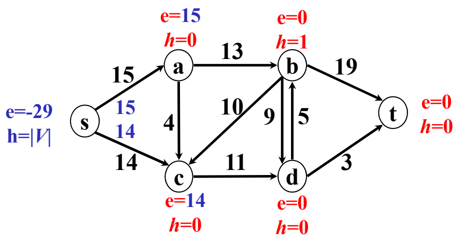
考虑初始化操作之后**DisCharge(*c*)**的执行过程

L(*c*):

s	a	b	d	Null

v=d≠Null， 不执行复标操作

c(cd)-f(cd) =11 但 c.h≠d.h+1 不执行推送操作

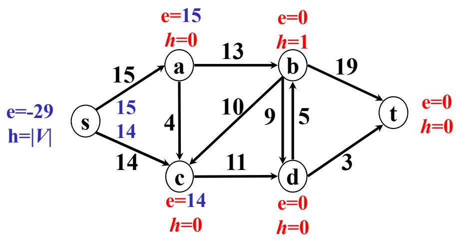


考虑初始化操作之后**DisCharge(*c*)**的执行过程

L(*c*):

s	a	b	d	Null

v=Null， 执行复标操作

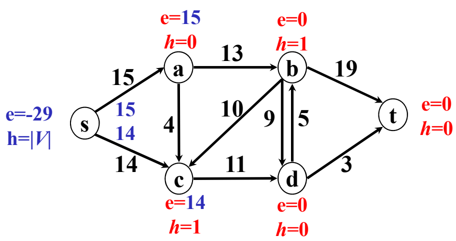


考虑初始化操作之后**DisCharge(*c*)**的执行过程

L(*c*):

s	a	b	d	Null

v=Null， 执行复标操作



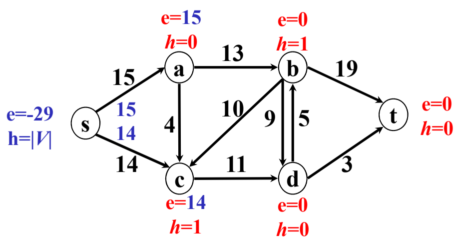
考虑初始化操作之后**DisCharge(*c*)**的执行过程

L(*c*):

s	a	b	d	Null

v=s≠Null， 不执行复标操作

c(cs)-f(cs) = 0-(-14) =14 但c.h≠s.h+1 不执行**Push**



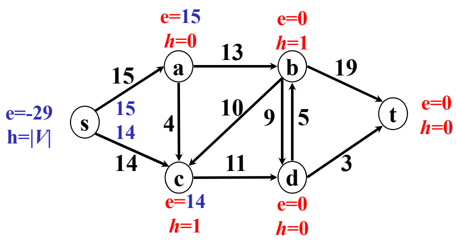
考虑初始化操作之后DisCharge(*c*)的执行过程

L(*c*):

s	a	b	d	Null

v=a≠Null， 不执行复标操作

c(ca)-f(ca) = 0-0 =0 不执行推送操作



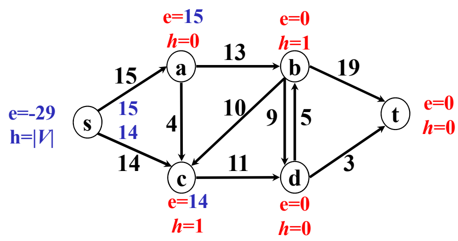
考虑初始化操作之后DisCharge(*c*)的执行过程

L(*c*):

s	a	b	d	Null

v=b≠Null， 不执行复标操作

c(cb)-f(cb) = 0-0 =0 不执行推送操作



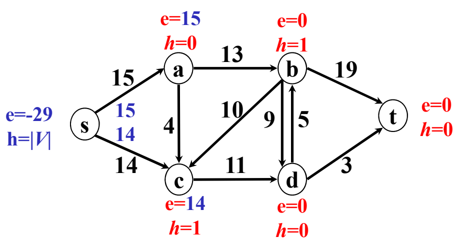
考虑初始化操作之后DisCharge(*c*)的执行过程

L(*c*):

s	a	b	d	Null

v=d≠Null， 不执行复标操作

c(cd)-f(cd) =11 且 c.h=d.h+1 执行Push操作



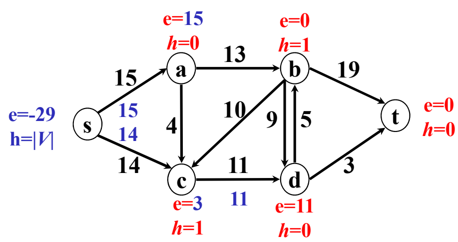
考虑初始化操作之后DisCharge(*c*)的执行过程

L(*c*):

s	a	b	d	Null

v=d≠Null， 不执行复标操作

c(cd)-f(cd) =11 且 c.h=d.h+1 执行Push操作



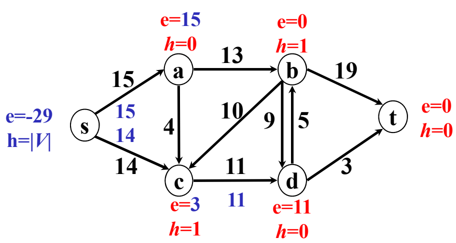
考虑初始化操作之后DisCharge(*c*)的执行过程

L(*c*):

s	a	b	d	Null

v=Null， 执行复标操作

c.h = 1+ min {v.h | c(cv)-f(cv)>0 } = |V|+1



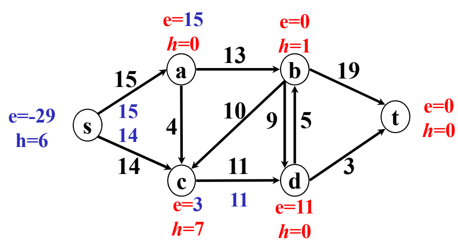
考虑初始化操作之后DisCharge(*c*)的执行过程

L(*c*):

s	a	b	d	Null

v=Null， 执行复标操作

c.h = 1+ min {v.h | c(cv)-f(cv)>0 } = |V|+1

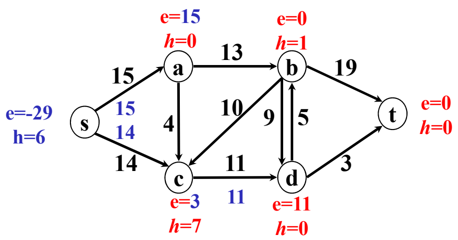


考虑初始化操作之后DisCharge(*c*)的执行过程

L(*c*): | s | a | b | d | Null |

v=s≠Null， 不执行复标操作

c(cs)-f(cs)=14且 c.h = c.s+ 1 执行Push

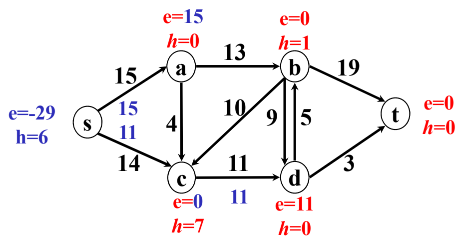


考虑初始化操作之后DisCharge(*c*)的执行过程

L(*c*): | s | a | b | d | Null |

v=s≠Null， 不执行复标操作

c(cs)-f(cs)=14且 c.h = c.s+ 1 执行Push



前置复标算法

顶点链表B $u_1 \rightarrow u_2 \rightarrow \cdots \rightarrow u_{n-2} \rightarrow$ *Null*

邻接链表L $L(u_1)$ $L(u_2)$ $L(u_{n-2})$

依次处理*B*中的每个顶点*u*

•处理过程即调用DisCharge操作使得*u.e*=0

•处理后将*u*前置于*B*的开始位置

•然后顺序处理下一个顶点

到达B的结束位置，则u.e=0对所有顶点成立

即得到最大流

$u_1.e=0$ $u_2.e=0$ $u_{n-2}.e=0$

$u_1 \rightarrow u_2 \rightarrow \cdots \rightarrow u_{n-2} \rightarrow$ *Null*

$L(u_1)$ $L(u_2)$ $L(u_{n-2})$

前置复标算法

初始化： u.e=0; u.h=0; uv.f=0 ∀u∈V, uv ∈E

$s.e= - \sum_{sv\in E} c(sv)$，$s.h=|V|$

$sv.f=c(sv)$， v.e = c(sv) ∀$sv\in E$

创建链表B管理V-{s,t}的所有顶点

L(v)←v的相邻顶点链表

L(v).current ← L(v).head

1. *u* ←*B*.head
2. While *u*≠NULL
3. oldHeight ←u.h
4. DisCharge(u);
5. If u.h>oldHeight then 将*u*前置到*B*的前端
6. u ←u.next

前置复标算法分析

引理1.前置复标算法终止后，则最后的预流是最大流。

初始化：算法初始化是预流

循环 ：每次推送操作或复标操作后，仍是预流

终止 ：算法结束后得到一个预流*f*

算法结束后*u.e*=0对任意u≠s,t成立， 故*f*是流

类似与推送复标算法，可以证明剩余网络中不存在s-t路径， 由最大流最小割定理，*f*是最大流

前置复标算法分析

引理2.前置复标算法在O(V^3)个基本操作之后必然终止。

前置复标算法是推送复标操作的特例

• 至多执行$2V^2$个复标操作

• 至多执行$2VE$个饱和推送操作

•只需限定非饱和推送的个数

非饱和推送操作 个数 ≤ DisCharge执行次数

每次非饱和推送执行后，u.e=0, DisCharge操作结束

考察执行复标操作的两个DisCharge操作之间

• 算法不改变任意顶点的高度,故算法顺序扫描B中顶点

• 算法顺序处理链表B中的一个连续区段

• 该区段的长度不超过B的总长度V-2

非饱和推送至多执行$2V^2\cdot(V-2) < 2V^3$个

HIT CS&E

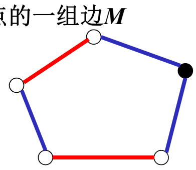
7.2 匹配算法

HIT CS&E

7.2.1 覆盖与匹配

匹配——图$G=(V,E)$中没有公共端点的一组边M

- 匹配边——M中的边
- 自由边——E/M中的边
- 被浸润的顶点——M中边的端点
- 未被浸润的顶点——其他顶点



完美匹配——浸润G的每个顶点的匹配

最大匹配——边的条数达到最大值的匹配

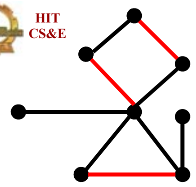
性质：完美匹配是最大匹配，反之不然

最大匹配问题

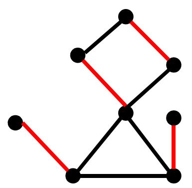
输入：图$G=(V,E)$

输出：G的最大匹配M

HIT CS&E



最大匹配
非完美匹配

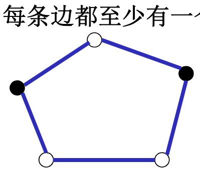


最大匹配
完美匹配

HIT CS&E

顶点覆盖——图$G=(V,E)$中的一个顶点子集C

E中每条边都至少有一个端点在C中



最小顶点覆盖——G的顶点个数最少的覆盖

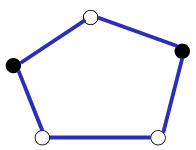
最小顶点覆盖问题

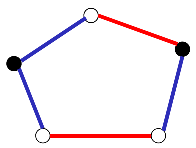
输入：图$G=(V,E)$

输出：G的最小顶点覆盖

HIT CS&E

弱对偶性





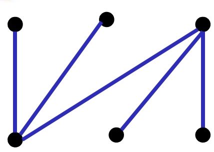
若C是图G的任意顶点覆盖，M是图G的任意匹配

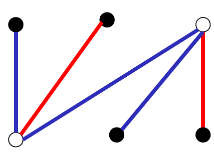
- M中每条边都至少有一个端点在C中
- M中任意两条边不存在公共端点

故$|M|\le|C|$

定理:图G的最小顶点覆盖C和最大匹配M满足$|M| \le|C|$
在二分图G中,$|M|=|C|$

HIT CS&E





若C是图G的任意顶点覆盖，M是图G的任意匹配

- M中每条边都至少有一个端点在C中
- M中任意两条边不存在公共端点

故$|M|\le|C|$

定理:图G的最小顶点覆盖C和最大匹配M满足$|M| \le|C|$
在二分图G中,$|M|=|C|$

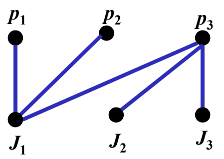
HIT CS&E

研究匹配算法的意义

工作分配

输入：n个人$p_1,\ldots,p_n$，n项工作$J_1,\ldots,J_n$，第i个人胜任其中k项工作

输出：是否存在工作分配方案使得每个人完成1项自己胜任的工作



计算节点　实习单位

计算任务　实习人员

目前，大规模图数据管理已经非常盛行

很多高效算法都以匹配算法或覆盖算法为基础！

HIT CS&E

7.2.2 最大二分匹配算法

定理：图G的最小顶点覆盖C和最大匹配M满足$|M| \le |C|$

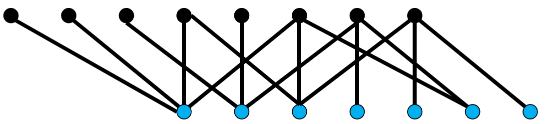
在二分图G中，$|M|=|C|$

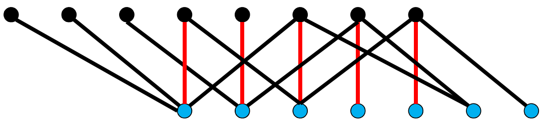
这意味着二分图上的最大匹配可以这样求解

- ◆初始化一个匹配M
- ◆不断地增大M
- ◆M无法增大时，找出一个顶点覆盖C使得$|M|=|C|$
- ◆M是最大匹配，C是最小覆盖

HIT CS&E

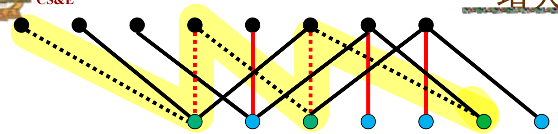
初始化





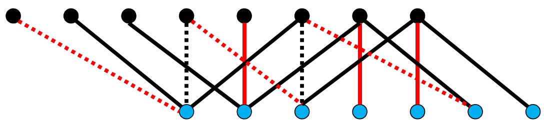
HIT CS&E

增大



M交错路径：边交替出现在M和E-M的路径

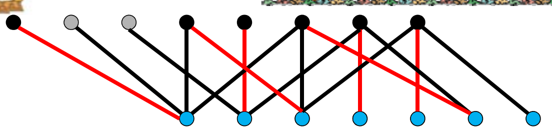
M增广路径：端点未被M浸润的交错路径



找出增广路径，就能增大M！怎么找呢？

HIT CS&E

通过BFS找增广路径

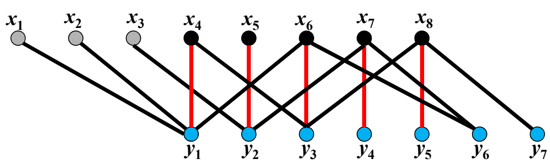


1. $U \leftarrow X$中未被M浸润的所有顶点
2. For $\forall x \in U$ do
   如果存在从x出发的M增广路径，则增大M
   Goto 1
3. M是最大匹配

能够以更高效的方式进行广度优先搜索呢？

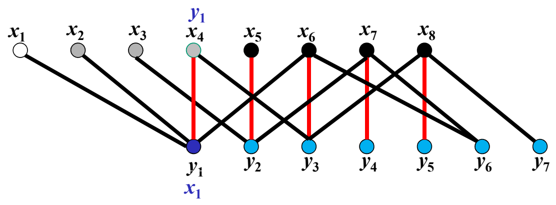
HIT CS&E

同时进行BFS



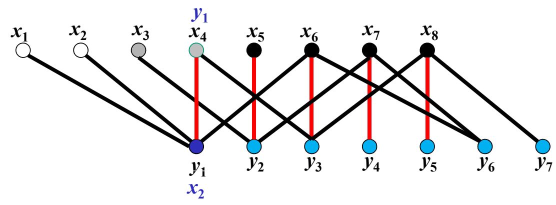
HIT CS&E

同时进行BFS



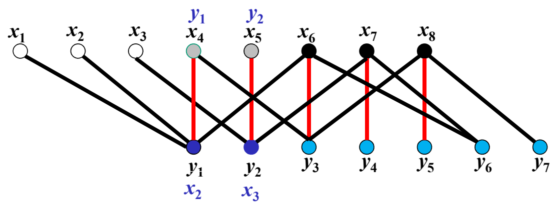
HIT CS&E

同时进行BFS



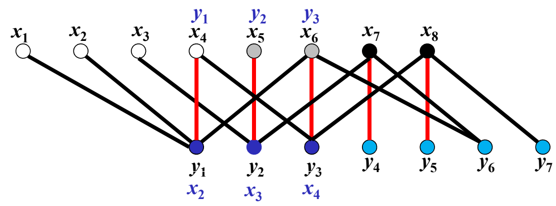
HIT CS&E

同时进行BFS



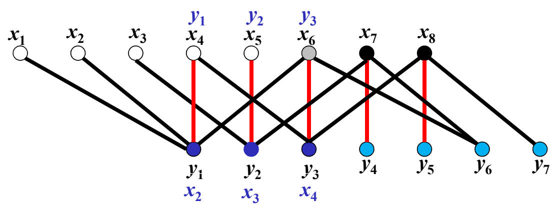
HIT CS&E

同时进行BFS



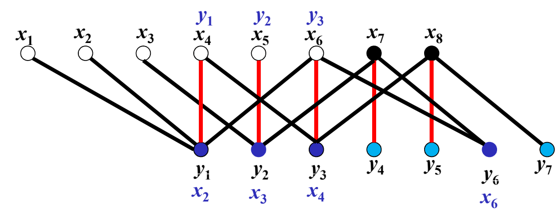
HIT CS&E

同时进行BFS



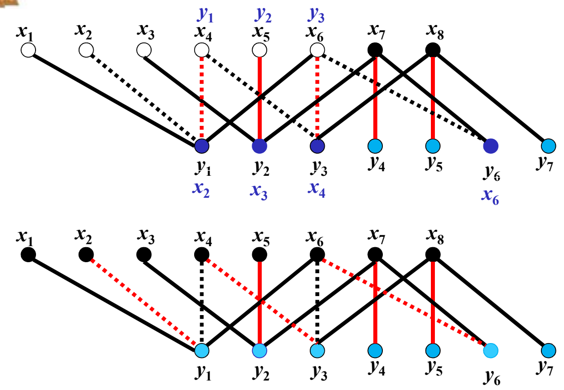
HIT CS&E

同时进行BFS



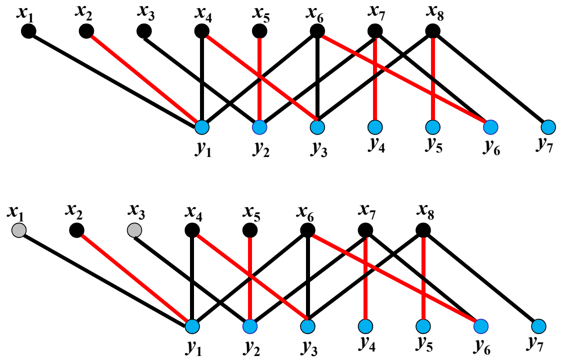
HIT CS&E

同时进行BFS



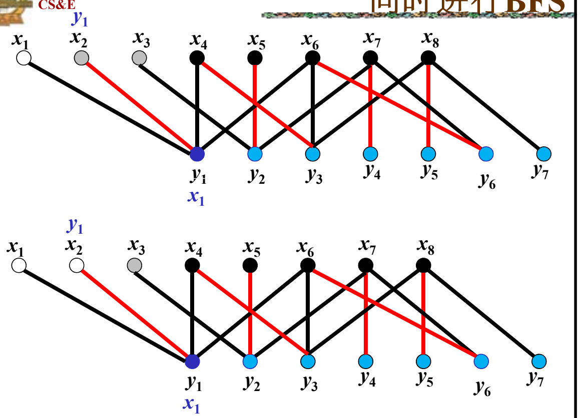
HIT CS&E

同时进行BFS



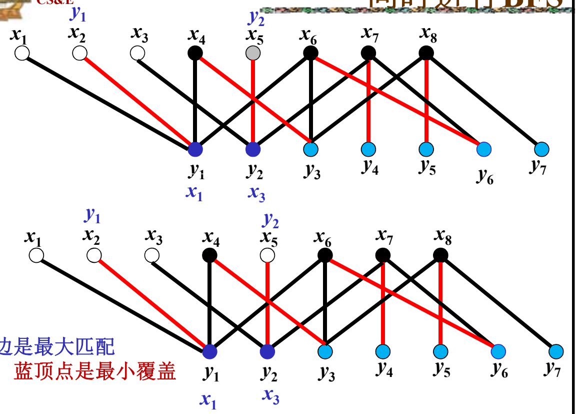
HIT CS&E

同时进行BFS



HIT CS&E

同时进行BFS



红边是最大匹配
黑、蓝顶点是最小覆盖

HIT CS&E

二分匹配的增广路径算法

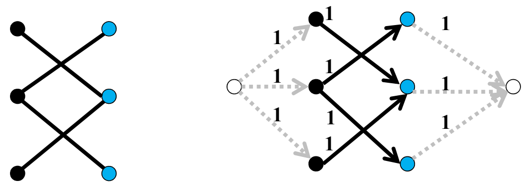
输入：二分图$G=(X\cup Y,E)$
输出：G的最大匹配和最小覆盖

时间复杂度$O(|V||E|)$

1.$M\leftarrow\varnothing$;　IsInc←true
2.While　IsInc　do
3.U←X中未被M浸润的所有顶点；　IsInc←false;
4.S←U，$T\leftarrow\varnothing$；　　//*S*记录灰、白顶点，*T*记录蓝顶点
5.While　S中存在灰顶点x　do
6.　　For　$\forall(x,y)\in E-M$　do
7.　　　　If　$\exists(w,y)\in M$　Then
8.　　　　　$T\leftarrow T\cup\{y\}$,记录y由x到达
9.　　　　　$S\leftarrow S\cup\{w\}$，记录w由y到达,w标记为灰色
10.　　　　else
11.　　　　　由y回溯得到一条增广路径，并用它增大M
12.　　　　　IsInc ←true; Goto第2步；
13.　　标记x为白色;　　　　　　//从x的BFS结束
14.输出M为最大匹配，输出T ∪(X-S)为最小覆盖

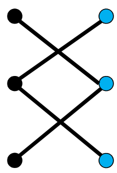
HIT CS&E

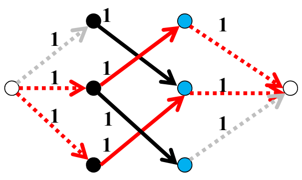
最大二分匹配的另一种算法



将$G=(V,E)$中每条边定向为从X到Y并定义容量1
添加虚拟源s,和s到X中各个顶点的有向边，容量为1
添加虚拟汇t,和Y中各顶点到t的有向边，容量为1

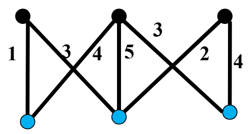
基于最大流的最大二分匹配算法





f是新图上的最大流
当且仅当
E中流量大于0的边构成G的最大匹配M

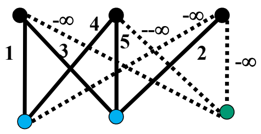
7.2.3最大权值二分匹配



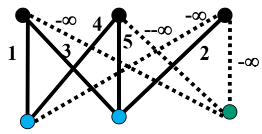
计算结点
◆边上的权值表示计算效率
◆最大匹配就是计算效率最高的计算任务分配方案
计算任务

匹配M的权值$c(M)$:M中各条边的权值之和

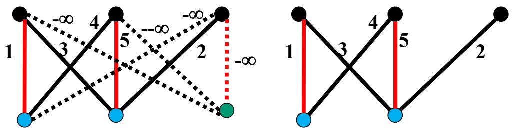
最大权值二分匹配问题
输入：加权二分图$G=(X\cup Y,E)$, $w:E\rightarrow R$
输出：G上的最大权值二分匹配M



最大权值二分匹配问题
输入：加权完全二分图$G=(X\cup Y,E)$, $w:E\rightarrow R$
输出：G上的最大权值二分匹配M

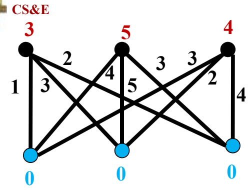


最大权值二分匹配问题
输入：加权完全二分图$K_{n,n}$, $w:E\rightarrow R$
输出：G上的最大权值二分匹配M



最大权值二分匹配问题
输入：加权完全二分图$K_{n,n}$, $w:E\rightarrow R$
输出：G上的最大权值二分匹配M

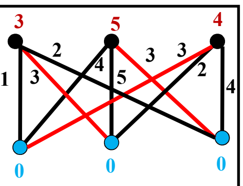
最大权值顶点覆盖



加权$K_{n,n}$的顶点覆盖
◆两个n维实值向量u,v
◆$u_i+v_j\geq w_{ij}$

最大权值顶点覆盖问题
输入：加权二分图$K_{n,n}$, $w:E\rightarrow R$
输出：分量之和达到最小值的顶点覆盖

对偶关系



任意完美匹配M，任意覆盖u,v

$$u_i+v_j\geq w_{ij}$$

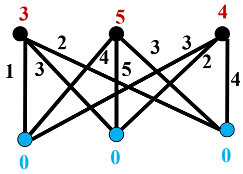
$$\Sigma u_i+\Sigma v_j\geq \Sigma w_{ij}$$

$$c(u,v)\geq c(M)$$

定理：在加权$K_{n,n}$上，任意完美匹配M和任意顶点覆盖u,v比满足$c(M)\leq c(u,v)$，并且M是最大权值匹配当且仅当$c(M)=c(u,v)$，此时M中的每条边ij均满足$u_i+v_j=w_{ij}$

对偶关系给出了问题的求解思路

初始化顶点覆盖u,v



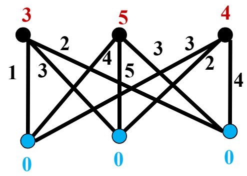
判断$\{ij: u_i+v_j=w_{ij}\}$中能否找出完美匹配M

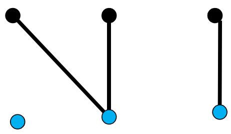
若是，则M是最大匹配，u,v是最小覆盖

否则，调整u,v

HIT CS&E

初始化





$u_i\leftarrow\max\{w_{ij}\}$

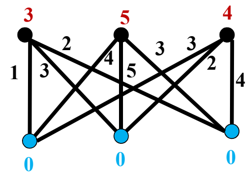
$v_j\leftarrow 0$

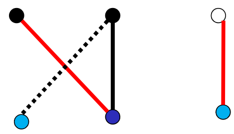
相等子图

仅含$u_i+v_j=w_{ij}$的所有边

HIT CS&E

初始化





$u_i\leftarrow\max\{w_{ij}\}$

$v_j\leftarrow 0$

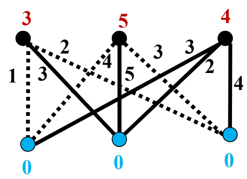
红边：最大匹配

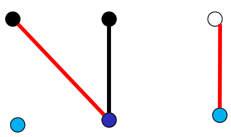
蓝、白顶点：最小覆盖

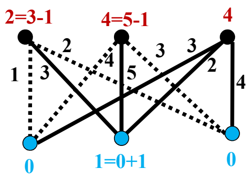
调整端点不在最小覆盖中的边，让它出现在相等子图中

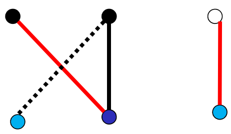
HIT CS&E

调整覆盖u,v





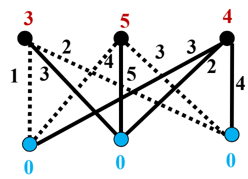


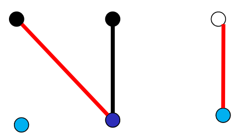


HIT CS&E

调整覆盖u,v

$\varepsilon=1$





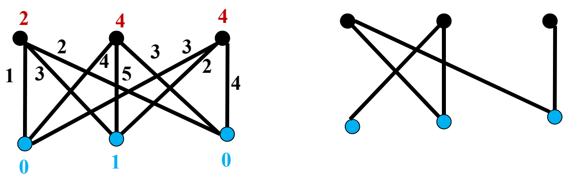
$\varepsilon\leftarrow\min\{u_i+v_j-w_{ij}\}$　i,j不在当前最小覆盖中

$u_i\leftarrow u_i-\varepsilon$　i不在当前最小覆盖中

$v_j\leftarrow v_j+\varepsilon$　j在当前最小覆盖中

HIT CS&E

$\varepsilon=1$

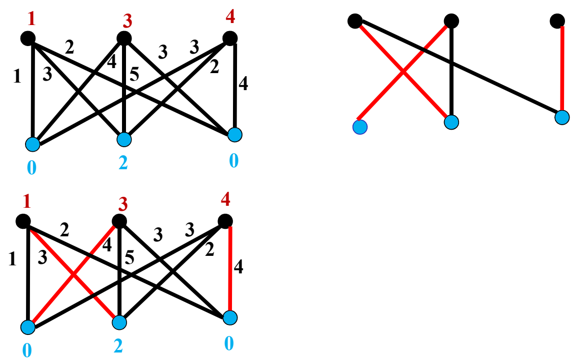


$\varepsilon \leftarrow \min\{u_i+v_j-w_{ij}\}$ i,j不在当前最小覆盖中

$u_i \leftarrow u_i-\varepsilon$ i不在当前最小覆盖中

$v_j \leftarrow v_j+\varepsilon$ j在当前最小覆盖中

HIT CS&E



HIT CS&E

加权最大二分匹配算法

输入：加权完全二分图$K_{n,n}$, 边ij的权值为w_{ij}

输出：$K_{n,n}$加权最大匹配和最小覆盖

时间复杂度$O(n^5)$

1.For $i \leftarrow 1$ To n

2. $u_i \leftarrow \max\{w_{ij}\}$; $v_i \leftarrow 0$;

3.While true do

至多循环n^2次

4. 构建由$\{ij: u_i+v_j=w_{ij}\}$构成的相等子图$G_{u,v}$;

5. 找到$G_{u,v}$的最大匹配M和最小顶点覆盖C;

6. If $|M|=n$ Then 输出M，算法结束

7. Else

8. $\varepsilon \leftarrow \min\{u_i+v_j-w_{ij}$: x_i, y_j不属于C}

9. $u_i \leftarrow u_i-\varepsilon$ 任意x_i不属于C

10. $v_j \leftarrow v_j+\varepsilon$ 任意y_j属于C

$O(n^3)$

循环不变量：u,v是加权顶点覆盖

第3-10步每执行n遍，$|M|$至少增大1

HIT CS&E

补充阅读材料

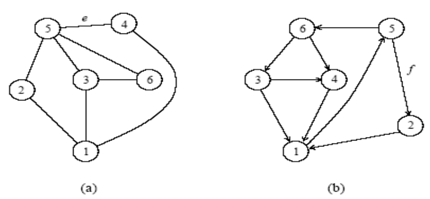
HIT CS&E

7.3 图及其表示法

- 无向图G是二元组(V,E)，其中V是顶点集合，E是边集
- 边$e \in E$是一个无序顶点对(u,v),其中$u,v \in V$
- 在有向图中，边$e \in E$是一个有序顶点对(u,v)，并称边(u,v)是由顶点u指向顶点
- 顶点u到顶点v的一条路径是一个顶点序列$<v_0,v_1,v_2,\ldots,v_k>$,其中$v_0=v$, $v_k=u$且$(v_i, v_{i+1}) \in E$对$i=0, 1,\ldots, k$-1成立
- 路径的长度定义为路径中边的条数

HIT CS&E

例



a) 无向图 (b) 有向图

HIT CS&E

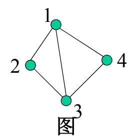
- 无向图是**连通**的，如果其任意两个顶点之间均有一条路径
- **森林**指的是无环图，**树**是连通的森林.
- 如果图的每条边均有一个权值与之关联，则称该图为**加权图**.

HIT CS&E

图的表示

给定图 $G = (V,E)$ 如何在计算机内存储它?

1. 邻接矩阵:顶点i表示为矩阵的第i行和第i列; 矩阵第i行第j列的元素被置为1(或者相应边的权值)，如果$(i,j)\in E$;否则置为0
2. 邻接表: 顶点作为数组的索引，每个顶点用一个链表存储与之邻接的所有顶点



图

	1	2	3	4
1	0	1	1	1
2	1	0	1	0
3	1	1	0	1
4	1	0	1	0

图的邻接矩阵

1 → 2 | 3 | 4
2 → 1 | 3
3 → 1 | 2 | 4
4 → 1 | 3

图的邻接表

HIT CS&E

表示方法的比较

给定图 $G = (V,E)$, $|V|=n$,$|E|=m$?

1. 存储空间
   邻接矩阵:$\Theta(n^2)$ 邻接表: $\Theta(m+n)$
2. 查看给定顶点的所有相邻顶点
   邻接矩阵:$\Theta(n)$ 邻接表: $\Theta(m/n)$
- 检查给定的顶点对是否是G的一条边
  邻接矩阵:$\Theta(1)$ 邻接表: $\Theta(m/n)$

–具体采用何种数据结构，看具体的任务主要涉及何种操作

–稀疏图往往采用邻接表存储

HIT CS&E

7.4 基本图论算法

7.2.1 广度优先搜索

7.2.2 深度优先搜索

7.2.3 拓扑排序

7.2.4 连通分支

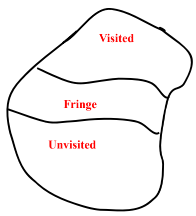
HIT CS&E

搜索算法

- 访问图中边访问所有顶点的系统化方法
- 可以发现图的很多结构信息
- 很多其他图论算法均是这些基本搜索算法的精细化结果
- 图的搜索算法是图论算法的核心

HIT CS&E

图搜索算法中的三类顶点



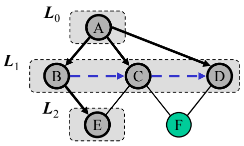
搜索过程中，顶点集可以看成如下3个部分的并集

- ❑ 已访问顶点：已被算法处理过的顶点.
- ❑ 边缘顶点 ：马上将被访问的顶点.
- ❑ 未访问顶点: 仍未被算法处理过的顶点

❖DFS算法用**栈**存储边缘顶点，而BFS算法用**队列**存储边缘顶点

HIT
CS&E

7.4.1 广度优先搜索BFS



HIT
CS&E

- Breadth-first search (BFS)是遍历图的一个一般技术
- BFS遍历图 *G*，可以
  - 访问*G*的所有边和顶点
  - 确定*G*是否连通
  - 计算*G*的所有连通分支
  - 计算*G*的生成森林
- BFS遍历具有*n*个顶点和*m*条边的图*G*的时间复杂性为O(*n*+*m*)
- BFS可以扩展以解决其他图论问题
  - 找出两个顶点间具有最少边数的路径
  - 找出图中的简单环，如果存在的话

HIT

BFS算法

- 该算法通过不断标记顶点和边，访问所有顶点和边，并划分边集合

算法 ***BFS*(*G*)**
Input 图 *G*
Output 标记并划分G的边

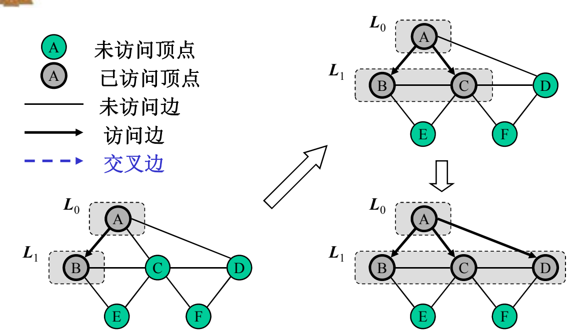
```
1.for each  u ∈ V_G
      将u标记为unvisited
2. for each e ∈ E_G
      将e标记为UNEXPLORED
3. for each v ∈ V_G
      if  v的标签是unvisited
          BFS(G, v)
```

算法*BFS*(*G*, *s*)
Input 图*G*，出发顶点*s*
Output 将*s*所在连通分支的边标记为三类

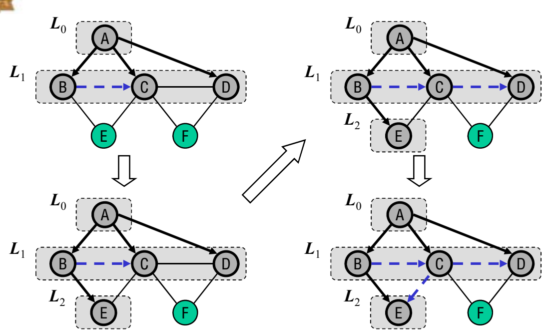
```
L_0 ← 空队列
将s插入L_0
将s标记为 VISITED
i ← 0
while  L_i不空
    L_{i+1} ← 空队列
    for each  v ∈ L_i
        for  each (v,u) ∈ E_G
            if (v,u) 的标签是UNEXPLORED
               if  u的标签是 unvisited
                   将(v,u)标记是DISCOVERY
                   将u标记为 visited
                   将u插入L_{i+1}
               else
                   将(v,u)标记为 CROSS
      i ← i +1
```

HIT
CS&E

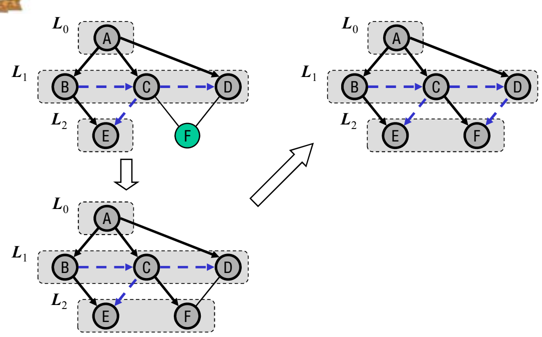
例



HIT
CS&E



HIT
CS&E



HIT CS&E

BFS算法的性质

记号

G_s: 顶点s所在的连通分支

Property 1

$BFS(G, s)$ 访问 G_s中的所有边和顶点

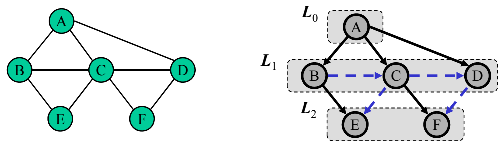
Property 2

$BFS(G, s)$ 找出的所有发现边构成 G_s的一棵生成树T_s

Property 3

对任意 $v \in L_i$

- 在T_s 中，s到 v的路径有 i条边
- 在G_s中， s 到 v的任意路径至少有 i 条边



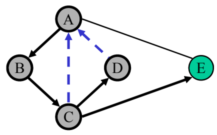
HIT CS&E

BFS算法分析

- 标记和获取顶点或边的时间开销为 $O(1)$
- 每个顶点被标记两次
  - 一次将其标记为unvisited
  - 一次将其标记为visited
- 每条边被标记两次
  - 一次将其标记为UNEXPLORED
  - 一次将其标记为 DISCOVERY 或 CROSS
- 每个顶点仅被唯一地插入某个 L_i
- 查找顶点的邻边仅被每个顶点调用一次 第8步
- 只要图是以邻接表存储的，BFS的运行时间为 $O(n + m)$
  - 注意 $\sum_v \deg(v) = 2m$

HIT CS&E

7.4.2 深度优先搜索DFS



HIT CS&E

- Depth-first search (DFS)是遍历图的一个一般技术
- DFS遍历图 G，可以
  - 访问G的所有边和顶点
  - 确定G是否连通
  - 计算G的所有连通分支
  - 计算G的生成森林
- DFS遍历具有n个顶点和m条边的图G的时间复杂性为$O(n+m)$
- DFS可以扩展以解决其他图论问题
  - 找出两个顶点间的路径
  - 找出图中的简单环，如果存在的话

HIT

DFS算法

- 该算法通过不断标记顶点和边，访问所有顶点和边，并划分边集合

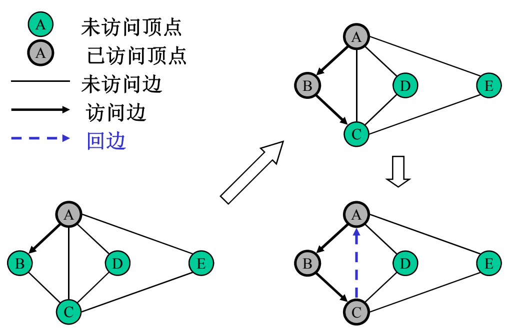
```
算法 DFS(G)
Input   图 G
Output 标记并划分G的边
  1.for each  u ∈ V_G
      将u标记为unvisited
  2. for each e ∈ E_G
      将e标记为UNEXPLORED
  3. for each v ∈ V_G
       if  v的标签是unvisited
          Recursive_DFS(G, v)
```

```
算法Recursive_DFS(G, s)
Input    图G，出发顶点s
Output 将s所在连通分支的边标记为两类
    将s标记为 VISITED
     for   each (s,u) ∈ E_G
        if (s,u) 的标签是UNEXPLORED
          if  u的标签是 unvisited
             将(s,u)标记是DISCOVERY
             将u标记为 visited
             Recursive_DFS(G, u)
          else
             将(s,u)标记为 Back
```

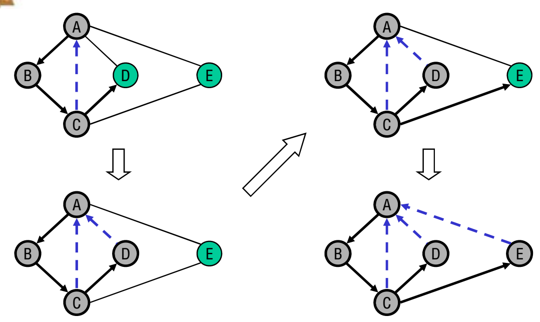
很容易将上述算法改为栈的形式

HIT CS&E

例

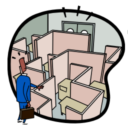


HIT CS&E

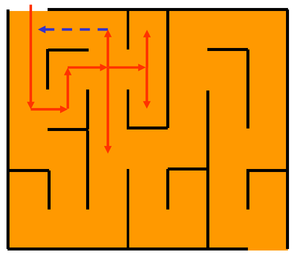


HIT CS&E

DFS与迷宫



- DFS算法很像走迷宫的经典策略
  - We mark each intersection, corner and dead end (vertex) visited
  - We mark each corridor (edge) traversed
  - We keep track of the path back to the entrance (start vertex) by means of a rope (recursion stack)



HIT CS&E

DFS算法的性质

Property 1

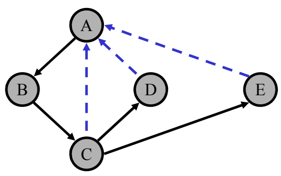
DFS(*G*, *v*) 访问 *v*所在连通分支的所有顶点和边

Property 2

DFS(*G*, *v*)找出的所有发现边构成 *v*所在连通分支的生成树

Property 3

有向图*G*是一个无环图当且仅当*G*上的DFS过程没有回边



HIT CS&E

DFS算法分析

- 标记和获取顶点或边的时间开销为 $O(1)$
- 每个顶点被标记两次
  - 一次将其标记为unvisited
  - 一次将其标记为visited
- 每条边被标记两次
  - 一次将其标记为UNEXPLORED
  - 一次将其标记为 DISCOVERY 或 BACK
- 查找顶点的邻边仅被每个顶点调用一次　第2步
- 只要图是以邻接表存储的，BFS的运行时间为 $O(n + m)$
  - 注意 $\sum_v \deg(v) = 2m$

HIT CS&E

作业

分别修改BFS和DFS以完成以下任务

1. 确定*G*是否连通
2. 计算*G*的所有连通分支
3. 计算*G*的生成森林
4. 找出两个顶点间的路径
5. 找出图中的简单环，如果存在的话

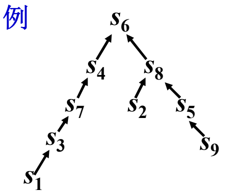
HIT CS&E

7.4.3 拓扑排序

HIT CS&E

偏序集合的拓朴排序

- 拓朴排序
  - 输入: 偏序集合$(S, \leq)$
  - 输出: S的拓朴序列是$<s_1, s_2, \ldots, s_n>$,
    满足: 如果$s_i \leq s_j$, 则s_i排在s_j的前面.
  - 例



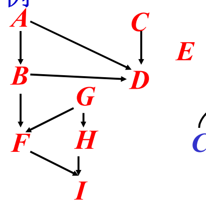
拓朴排序:

s_1 s_3 s_7 s_4 s_9 s_5 s_2 s_8 s_6

HIT CS&E

有向图的拓扑排序

- 拓朴排序
  - 输入: 无环有向图$G=(V,E)$
  - 输出:G的所有顶点的一个拓扑排序
    即,如果$uv \in E$,则u在该排序中位于v之前
  - 例



拓朴排序:

C $A \to B \to D$ E $G \to F$ $H \to I$

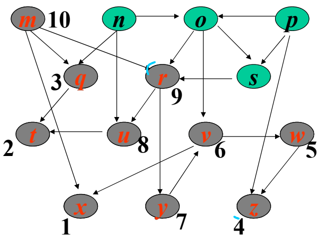
HIT CS&E

有向图的拓扑排序算法

算法 TOPOLOGICAL_SORT(G)
Input 无环有向图$G=(V,E)$
Output G的s所有顶点的拓朴排序
1.初始化空链表L
2.调用DFS算法计算每个顶点v的结束时间$f(v)$
3. 当v的结束时间$f(v)$被计算出来时，将v插入L的最前端
3. 链表L中的顶点顺序即为一个拓朴序列

HIT CS&E

例



m r u y v w z q t x

HIT CS&E

算法分析

定理 对无环有向图G, TOPOLOGICAL_SORT(G)
得到G的一个拓朴排序

证明：仅需证明，如果$uv \in E$则$f(v)<f(u)$。

考虑uv被DFS访问的时刻，v要么已被访问完（$f(v)$已被计算出来），要么v仍未被访问过，否则将出现环。

对于第一种情况，显然有$f(v)<f(u)$。

对于第二种情况，v是u的后代，DFS算法必然会先结束对v的访问，故$f(v)<f(u)$。

时间复杂度分析

即DFS的时间复杂度，O(|V|+|E|)

HIT CS&E

7.4.4 有向图的强连通分枝分解

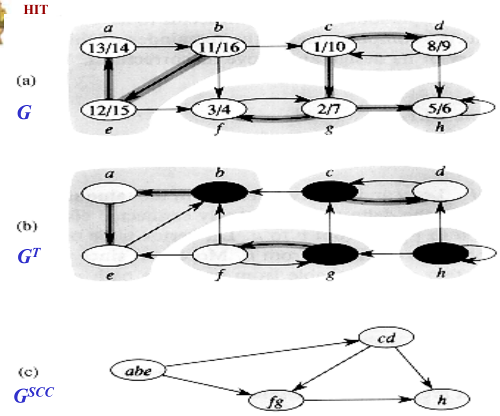
- 将有向图分解为强连通分枝是DFS的一个经典应用
- 许多应用需要将有向图分解为强连通分枝，然后再对每个强连通分枝应用某种操作或算法
- 给定有向图$G=(V,E)$, G的一个强连通分枝指的是一个极大集$C \subseteq V$使得$\forall u,v \in E$均有$u \leftrightarrow v$。
- 将有向图$G=(V,E)$的所有边反向后得到的图成为G的转置，记为G^T
  - •若G以邻接矩阵A给出，G^T的邻接矩阵即为A^T
  - •若G以邻接表给出，请给出一个算法计算G^T的邻接表

HIT
CS&E

- 注意，G的转置和G具有相同的连通分枝
- G的连通分枝图$G^{SCC}=(V^{SCC},E^{SCC})$定义如下:
  - G的每个连通分枝对应V^{SCC}中的一个顶点
  - $x,y\in V^{SCC}$, 如果x对应连通分枝到y对应的连通分枝在G中有一条有向边，则$(x,y)\in E^{SCC}$
- 连通分枝分解算法的分析依赖于G^{SCC}的一些性质

HIT

例



HIT
CS&E

强连通分枝分解算法

算法 STONGLY_CONNECT_COMPONENTS(G)

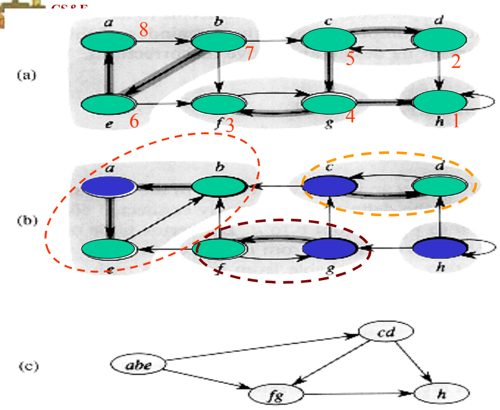
Input 无环有向图$G=(V,E)$

Output G的强连通分枝分解

1. 在G上调用DFS计算每个顶点v的结束时间$f(v)$
2. 计算G^T
3. do
4. 在G^T的剩余顶点中，从最大$f(v)$对应的顶点开始进行DFS访问
5. 如果当前访问顶点没有未被访问过的边，则输出当前访问过的顶点 作为一个连通分枝
6. while(G^T仍有顶点未被访问)

HIT

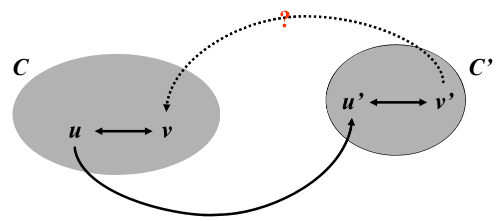
例



HIT
CS&E

算法分析

引理1 设 C和 C'是有向图$G=(V, E)$中两个不同的连通分枝，且 $u, v\in C$, $u', v'\in C'$。如果$u\to u'$ 则$v'\to v$不成立



HIT
CS&E

- 将DFS算法首次进入顶点u的时间的$d(u)$和结束对u的访问的时间$f(u)$扩展到集合上，有

$$d[U]=\min_{u\in U}\{d[u]\} \qquad f[U]=\max_{u\in U}\{f[u]\}$$

- 引理2 设 C和 C'是有向图 $G=(V, E)$的两个不同连通分枝且存在$(u, v)\in E$,其中 $u\in C$ 且$v\in C'$. 则 $f(C)>f(C')$.

证明：分两种情况讨论

(1)$d(C)<d(C')$。设$x\in C$为第一个被发现的顶点，则$d[x]=d(C)$。$d[x]$时，C中其他顶点和C'中所有顶点都是白色的。对任意的$w\in C'$，由$x\to u\to v\to w$，知$f(x)>f(w)$。所以，$f(C)>f(C')$。

(2)$d(C)>d(C')$。设$y\in C'$为第一个被发现的顶点，则$d[y]=d(C')$。$d[y]$时，C'中其他顶点都是白色的，因此$f(C')=f[y]$。由于C'中任一点都不可达C（否则的话$C\cup C'$强连通）。故$f(C)>f(C')$。

推论3 设 C 和C'是有向图 $G=(V, E)$中的两个不同的连通分枝.如果$(u, v)\in E^T$,其中$u\in C$且 $v\in C'$.则 $f(C)<f(C')$.

证明: $(u, v)\in E^T \Leftrightarrow (v, u)\in E$; 由引理2即可得到结论。

推论3意味着什么?

如果u的f(u)最大且u∈C 则从u出发DFS不能访问到其他分枝

除非结束对C的访问后再重新指定DFS的出发点

定理 算法STONGLY_CONNECTED_COMPONENTS(G)能够正确计算有向图$G=(V, E)$ 所有连通分枝.

证明: 根据推论3对连通分枝数量做数学归纳法。自己下来书写!

时间复杂度 $O(|V|+|E|)$

两遍DFS，外加一个计算G^T的时间开销

7.5 最小生成树算法

参见第五章，贪心算法

7.6 单源最短路径算法

- 问题的定义
- 单源最短路径的子结构性质
- Bellman-Ford算法
- Dijkstra算法

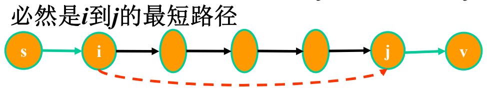
单源最短路径问题

- 给定加权图或不加权的图 G, 找出给定的源顶点s到目标顶点v的最短路径
  - 在给定的网络拓扑下，最小化路由代价.
  - 最小化基因-基因反应中的能量开销.
  - 最小化代价是许多实际问题中的基本要素
- 在加权图中
  - "最短路径" = 权值最小的路径
  - 路径的权值等于路径上所有边的权值之和
  - 不能用BFS来求解该问题
- 在不加权图中
  - "最短路径" = 边数最少的路径
  - 可以用BFS在 O(V+E)的时间内找出源顶点到目标顶点的最短路径.

最短路径的特征-优化子结构

优化子结构:最短路径包含了最短子路径

设s到v的最短路径P经过顶点i和j，则P上从i到j的部分必然是i到j的最短路径



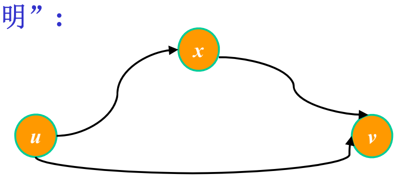
证明: 如果子路径 (i, j) 不是顶点i到顶点j的最短路径，则
- 在i和j之间必然存在一条更短的路径(红色路径)
- 用这条更短的子路径替换原来的子路径
- 得到一条比原路径更短的路径.
- 矛盾.

最短路径的特征-三角不等式

- 定义$\delta(u,v)$为从u到v的最短路径的代价(长度)
- 最短路径代价满足三角不等式

  $\delta(u,v) \le \delta(u,x) + \delta(x,v)$

- "证明"：

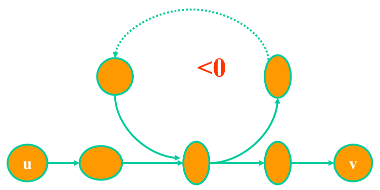


最短路径不比其他任何路径长

HIT CS&E

最短路径的特征-环

- u到v的最短路径上能存在环吗?



- 负环

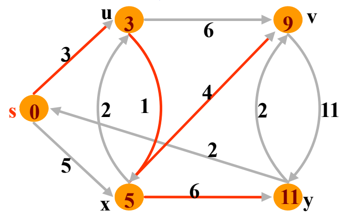
  如果u到v的最短路径上能存在负环，则定义$\delta(u,v)=-\infty$

HIT CS&E

最短路径的特征-最短路径树

最短路径树包含图G中从s出发可达的所有顶点

- 形成一棵以 s为根的树.
- 在这棵树中从s 到v的唯一路径,即为G中从s到v的最短路径.



最短路径的存储

$\pi[v]$存储从s到v的最短路径中顶点v的前驱

HIT CS&E

最短路径的特征-松弛技术

松弛技术是最短路径算法的关键技术

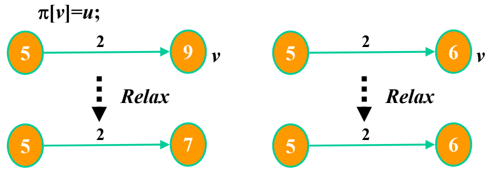
- 对所有 v, 维护$\delta(s,v)$的一个上界 $d[v]$

算法Relax(u,v,w)

Input 顶点u和v，图的加权函数w

Output 松弛后的$d[v]$

```
if (d[v] > d[u]+w(u,v)) then
    d[v]=d[u]+w(u,v);
    π[v]=u;
```



HIT CS&E

Bellman-Ford算法

算法 Bellman-Ford(G,w,s)

Input 图$G=(V,E)$,边加权函数w,源顶点s

Output s到其所有可达顶点的最短路径

```
For ∀v∈V do
    d[v]←∞;
    π[v]←null;
d[s] ←0;
For i←1 to |V|-1 do
    For ∀uv ∈E do
        Relax(u,v,w);
For ∀uv ∈E do
    If d(v)>d(u)+w(u,v) then
        return FALSE
Return TRUE
```

初始化

求解

执行|V|-1遍，松弛每条边

检查解的合理性，环，负环

请你确定Bellman-Ford算法的时间复杂性？$O(VE)$

HIT CS&E

Bellman-Ford算法的分析

算法 Bellman-Ford(G,w,s)

Input 图$G=(V,E)$,边加权函数w,源顶点s

Output s到其所有可达顶点的最短路径

```
For ∀v∈V do
    d[v]←∞;
    π[v]←null;
d[s] ←0;
For i←1 to |V|-1 do
    For ∀uv ∈E do
        Relax(u,v,w);
For ∀uv ∈E do
    If d(v)>d(u)+w(u,v) then
        return FALSE
Return TRUE
```

为什么算法运行|V|-1遍就足够了？

HIT CS&E

- $d[v] = \delta(s,v)$必然在|V|-1遍运行后成立
  - 设$P = (v_0, \dots, v_k)$是的从s到 v最短路径，其中$v_0 = s$ 且 $v_k = v$.
  - 由于 P简单路径, 故 $k < |V|-1$.
- 下面通过对最短路径的长度做归纳，证明$d[v_k] = \delta(s,v_k)$ 在 k遍之后成立, 其中k是 $(v_0, v_1 \dots, v_k)$，即s到v_k的最短路径的长度
  - 注意到最短k-路径可以通过最短 (k-1)-路径来构造

HIT CS&E

证明：考虑从s到v的最短路径

$s=v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_{k-1} \rightarrow v_k=v$

- 最初, $d[v_0]=d[s]=0=\delta(s,s)$且以后不再变化.
- 一遍之后, $d[v_1]=\delta(s,v_1)$ 是s到v_1的最短路径，且$d[v_1]$以后不再变化.
- 设k-1边后有$d[v_{k-1}]=\delta(s,v_{k-1})$，则第$k$遍过程中,
  - 如果($d[v_k] > d[v_{k-1}]+w$) 则$d[v_k]=d[v_{k-1}]+w$
  - $d[v_k] =\delta(s,v_k)$, 因为每条最短(k-1)-路径均会被松弛过程检查和扩展.
- $d[v] =\delta(s,v)$在$|V|$-1遍后成立.

Bellman-Ford算法具有负环检测能力

CS&E

```
算法 Bellman-Ford(G,w,s)
Input   图G=(V,E),边加权函数w,源顶点s
Output s到其所有可达顶点的最短路径
For ∀v∈V do
    d[v]←∞;
    π[v]←null;
d[s] ←0;
For  i←1  to |V|-1  do
    For ∀uv ∈E  do
        Relax(u,v,w);
For ∀uv ∈E  do
    If  d(v)>d(u)+w(u,v)  then
        return FALSE
Return TRUE
```

为什么算法能够发现负环?

- 在G中,如果从s出发不能到达任何负环，则
  - Bellman-Ford算法返回 TRUE,
  - $d[v]=\delta(s,v)$ 对任意顶点v成立.
- 在 G中如果从s出发能够到达某个负环, 则
  - 算法返回 FALSE.
  - Bellman-Ford 能够检测负环的存在性.

证明:(1)设G中没有负环.

- |V|-1遍后, 我们有$d[v]=\delta(s,v)$对任意顶点v成立.
- 由三角不等式,

  $d[v]=\delta(s,v)\leq \delta(s,u)+w(u,v)=d[u]+w(u,v)$,对任意$(u,v)\in E$成立.

证明:(2)反证法

- 设G中从s可以到达负环 $(v_0,v_1, \dots,v_k)$其中$v_k=v_0$.

$$\sum_{i=1}^{k} w(v_{i-1},v_i)<0$$

- 但，算法返回TRUE
  - 没有负环被检测到
  - 检测负环$(v_0,v_1, \dots,v_k)$上的任意一条边时，均有

$$d[v_i]\leq d[v_{i-1}]+w(v_{i-1},v_i) \quad \text{for } i=1,2,\dots,k.$$

$$\sum_{i=1}^{k} d[v_i]\leq \sum_{i=1}^{k} d[v_{i-1}]+\sum_{i=1}^{k} w(v_{i-1},v_i) .$$

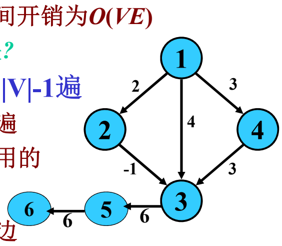
$$\text{Since } \sum_{i=1}^{k} d[v_i]=\sum_{i=1}^{k} d[v_{i-1}], \quad \sum_{i=1}^{k} w(v_{i-1},v_i)\geq 0$$

  - 矛盾

HIT CS&E

DAG上的单源最短路径

- 问题: 在无环有向图(Directed Acyclic Graph-DAG中如何高效地解决单源最短路径问题
  - Bellman-Ford算法的时间开销为$O(VE)$
  - 在 DAG中我们能否更快?
- Bellman-Ford算法执行|V|-1遍
  - 每遍均需扫描所有边一遍
  - 对许多边的扫描均是无用的
- 事实上
  - 无需扫描不影响结果的边
  - 对于已经找到的最短路径，其上的边无需再扫描



HIT CS&E

DAG上的单源最短路径

- 问题: 在无环有向图(Directed Acyclic Graph-DAG中如何高效地解决单源最短路径问题
  - Bellman-Ford算法的时间开销为$O(VE)$
  - 在 DAG中我们能否更快?
  - 基本想法—利用拓扑排序
    - DAG中每条路径均是拓扑序顶点序列的子序列
    - 能够容易识别从s可达的顶点，避免无用边扫描
    - 按照拓扑序处理顶点，将始终是前向地处理每条路径，避免重复扫描已知最短路径上的边
    - 仅需要一遍扫描

HIT CS&E

DAG单源最短路径算法

```
算法 DAG-Shortest-Paths(G,w,s)
Input   无环有向图G=(V,E),边加权函数w,源顶点s
Output s到其所有可达顶点的最短路径
   将V中顶点进行拓扑排序
   For ∀v∈V do
       d[v]←∞;
       π[v]←null;
   d[s] ←0;
   For each u ∈V (按拓扑序考虑) do
       For ∀v ∈Adj[u]   do
           Relax(u,v,w);
```

大家尝试自己去分析该算法

HIT CS&E

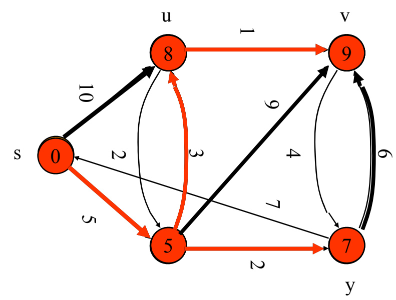
Dijkstra算法

- Dijkstra算法假设$w(uv)\geq 0$ 对$\forall uv\in E$成立
- 始终维护顶点集 S 使得
  - $\forall v\in S$, $d[v]=\delta(s, v)$,即, s到v的最短路径已经找到.
  - 初始值: $S=\varnothing$, $d[s]=0$ 且$d[v]=+\infty$
- 算法运行过程中
  - (a) 选择 $u\in V\text{-}S$ 使得
    $d[u]=min\ \{d[x]|x \in V\text{-}S\}$. 令$S=S\cup\{u\}$
    此时$d[u]=\delta(s, u)$!为什么?
  - (b)对于u的每个相邻顶点 v执行 RELAX(u, v, w)
- 重复上述步骤(a)和 (b) 直到 $S=V$.
- 该算法类似与Prim算法，属于贪心算法

HIT CS&E

```
算法 Dijkstra(G,w,s)
Input   图G=(V,E),边加权函数w,源顶点s
Output s到其所有可达顶点的最短路径
   For ∀v∈V do
       d[v]←∞;
       π[v]←null;
   d[s] ←0;
   S ←∅
   Q ←V
   while Q≠∅ do
       u ← EXTRACT -MIN(Q);
       S ←S∪{u}
       For ∀v ∈Adj[u]   do
           Relax(u,v,w);
```

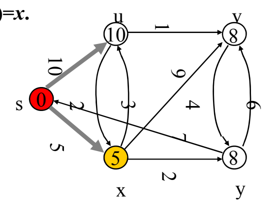
HIT CS&E



HIT CS&E

第一步：假设EXTRACT -MIN(Q)=x.

- sx是仅含一条边的最短路径
  - 为什么?
  - 因为sx是从s出发的最短的边.
- 它也是s到x的最短路径



证明:

(1) 设P: $s\to u\ldots\to x$ 是s到x的最短路径，则$w(s,u)\geq w(s,x)$.

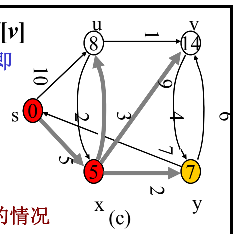
(2) 由于图中没有负权值边, 路径P的总权值至少为

$w(s,u)\geq w(s,x)$.

(3) 故,边sx是s到x的最短路径.

第二步：S={s,x}　　$d[y]$= $\min_{v\in V\text{-}S}d[v]$

- 论断:$d[y]$是从s到y的最短路径代价，即
  - 要么sy是最短路径
  - 要么$s\to x\to$y是最短路径.
- 为什么?
  - 如果sy是最短路径，论断成立
  - 考察$s\to x\to$y是从s到y的最短路径的情况



证明: (反证法)设$s\to x\to$y不是从s到y的最短路径

(1) 设P_1: $s\to y'\to\ldots\to x$ 是s到y的最短路径，其中$y'\notin S$. (注意此时，我们已经考察了y'=x和y'=s的情形).

(2) 因此，$w(P_1)$<w($s\to x\to$y).

(3) 由于$w(uv)\geq 0$对任意边成立，故$w(sy')$<$w(P_1)$< w($s\to x\to$y).

进而d[y']<d[y], 这样算法第二步不可能选中y，矛盾!

后续步骤：设S是算法维护的集合,令$d[y]= \min_{v\in V-S} d[v]$

- 定理:$d[y]$是从s到y的最短路径代价(正确性分析中最难的部分)

证明: (归纳法+反证)

归纳假设:设对$\forall v\in S$, $d[v]$是从s到v的最短路径的代价，往证本次操作完成后$d[y]$将是s到y的最短路径的代价

若不然, $d[y]$ 不是从s到y的最短路径的代价。 设P_1: $s\rightarrow\ldots\rightarrow y'\rightarrow\ldots\rightarrow y$是从$s$到$y$的最短路径，其中$y'\notin S$是 P_1上第一个不属于S的顶点. 这意味着$y\neq y'$且$w(P_1)<d[y]$.

因此,$w(s\rightarrow\ldots\rightarrow y')<w(P_1)$. (每条边的权值均非负)
进而$w(s\rightarrow\ldots\rightarrow y')<w(P_1)<d[y]$.
据此, $d[y']\leq w(s\rightarrow\ldots\rightarrow y')<w(P_1)<d[y]$.

因此，算法在本次操作中不会选中y，矛盾!

HIT CS&E

Dijkstra算法的时间复杂度

- 时间复杂度依赖于优先队列Q的实现
- 模型1: 利用数组存储Q
  - EXTRACT -MIN(Q) —需要 $O(|V|)$ 时间.
    - 总共需要执行 $|V|$次 EXTRACT -MIN(Q).
    - $|V|$次 EXTRACT -MIN(Q)操作的总时间为$O(|V|^2)$.
  - RELAX(u,v,w) —需要 $O(1)$时间.
    - 总共需要执行$|E|$次 RELAX(u, v, w)操作.
    - $|E|$次 RELAX(u,v,w)操作的总时间为 $O(|E|)$.
  - 总时间开销为$O(|V|^2+|E|)=O(|V|^2)$
- 模型2:Q用堆实现.
  - 需要$O(\log|V|)$时间.
  - 总时间开销为 $O(|V|\log |V|+E)$.

HIT CS&E

7.7 all-pairs shortest paths

- 问题的定义
- 单源最短路径的子结构性质
- Bellman-Ford算法
- Dijkstra算法

HIT CS&E

7.7.1问题定义及求解方法

- 给定加权图或不加权的图 $G=(V,E)$, 我们希望对任意$u,v\in V$计算出从u到v的最短路径
- 用Bellman-Ford算法或Dijkstra算法解决
  - 直接调用Bellman-Ford或Dijkstra算法$|V|$遍
  - Dijkstra算法$O(V\log V+E)\Rightarrow O(V^3)$
  - Bellman-Ford算法$O(VE)\Rightarrow O(V^4)$
- Faster-All-Pairs-Shortest-Paths
  - $O(V^3\lg V)$

HIT

7.7.2 基于矩阵乘法的算法

最短路径的结构

- 设p是顶点i到顶点j的最短路径
- 如果$i=j$, 则p中不含任何边，路径的权值为0
- 如果$i\neq j$, 设j在路径p中的前驱为k，则p可以分解为 $i \xrightarrow{p'} k \rightarrow j$
  - 由最短路径的优化子结构知道$i \xrightarrow{p'} k$是i到k的最短路径
  - 从而 $\delta(i,j)=\delta(i,k)+w(k,j)$
  - 如果p有m条边，则p'有m-1条边
  - 提示我们，最短路径可以存为前驱矩阵$\pi[i,j]$
  - $\pi(i,j)$表示从i到j的最短路径中j的前驱
  - 根据前驱矩阵，可以打印所有的最短路径(自己写个算法)
- 如果从路径的长度入手可能建立递归过程

HIT CS&E

递归计算

- 定义$l_{ij}^{(m)}$= 从i到j的至多仅含m条边的最短路径的代价
- 显然 $l_{ij}^{(0)}= 0$ if $i=j$ 或 $l_{ij}^{(0)}= \infty$ if $i\neq j$
- 因此 $l_{ij}^{(m)}= \min\{l_{ij}^{(m-1)}, \min_{1\leq k\leq n} \{l_{ik}^{(m-1)}+w(k,j)\}\}$
  $= \min_{1\leq k\leq n} \{l_{ik}^{(m-1)}+w(k,j)\}$ 因为$w(j,j)=0$
- 由于从i到j的最短路径最多含有n-1条边，故
  $\delta(i,j)= l_{ij}^{(n-1)}= l_{ij}^{(n)}= l_{ij}^{(n+1)}=\ldots$
- 自底向上计算
  - $L^{(1)}, L^{(2)},\ldots, L^{(n-1)}$, 其中 $L^{(m)}= (l_{ij}^{(m)})_{n\times n}$
  - 注意 $L^{(1)}=W$是权值矩阵

HIT CS&E

算法 Extened_Shortest_path(L,w)
Input 当前代价矩阵L,边加权函数w
Output 经一步扩展后的代价矩阵
1. n←$row[L]$;
2. L'←$(l'_{ij})_{n\times n}$
3. For i←1 to n do
4. For j←1 to n do
5. l'_{ij}←∞
6. For k←1 to n do
7. l'_{ij}←min(l_{ij}, l_{ik}+$w(k,j)$)
8. Return L'

矩阵乘法LW ⇒ 替换 +→min *→+ ⇒ Extened_Shortest_Patth(L,W)

$$L^{(1)} = L^{(0)}\cdot W = W$$
$$L^{(2)} = L^{(1)}\cdot W = W^{2}$$
$$L^{(3)} = L^{(2)}\cdot W = W^{3}$$
M
$$L^{(n-1)} = L^{(n-2)}\cdot W = W^{n-1}$$

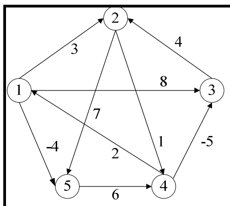
时间复杂性$O(n^4)$

修改算法完成对前驱矩阵的计算?

算法 Slow_All_Pairs_Shortest_path(W)
Input 图的边加权函数矩阵w
Output all_pair_Shortest_Paths代价
1. n←$row[L]$;
2. $L^{(1)}$←W
3. For m←2 to n-1 do
4. $L^{(m)}$←Extened_Shortest_path($L^{(m)}$,W)
5. Return $L^{(n-1)}$

- 该算法能否检测到图中的负环?
- 如何降低该算法的时间复杂性?

例



$$L^{(1)}=\begin{pmatrix}0&3&8&\infty&-4\\ \infty&0&\infty&1&7\\ \infty&4&0&\infty&\infty\\ 2&\infty&-5&0&\infty\\ \infty&\infty&\infty&6&0\end{pmatrix}$$

$$L^{(2)}=\begin{pmatrix}0&3&8&2&-4\\ 3&0&-4&1&7\\ \infty&4&0&5&11\\ 2&-1&-5&0&-2\\ &&&&0\end{pmatrix}$$

$$L^{(3)}=\begin{pmatrix}0&3&-3&2&-4\\ 3&0&-4&1&-1\\ 7&4&0&5&11\\ 2&-1&-5&0&-2\\ 8&5&1&6&0\end{pmatrix}$$

$$L^{(4)}=\begin{pmatrix}0&1&-3&2&-4\\ 3&0&-4&1&-1\\ 7&4&0&5&3\\ 2&-1&-5&0&-2\\ 8&5&1&6&0\end{pmatrix}$$

© DB-L

HIT CS&E

降低算法的时间复杂性

Improving the running time:

$$L^{(1)} = W$$
$$L^{(2)} = W^2 = W\cdot W$$
$$L^{(4)} = W^4 = W^2\cdot W^2$$
M
$$L^{(2^{\lceil\log(n-1)\rceil})} = W^{2^{\lceil\log(n-1)\rceil}}$$

i.e., using repeating squaring!

Time complexity: $O(n^3\log n)$.

HIT CS&E

算法 Faster_All_Pairs_Shortest_path(W)
Input 图的边加权函数矩阵w
Output all_pair_Shortest_Paths代价
1. n←$row[L]$;
2. $L^{(1)}$←W
3. while m < n-1 do
4. $L^{(m)}$←Extened_Shortest_path($L^{(m)}$, $L^{(m)}$)
5. m←2m
5. Return $L^{(m)}$

注意: $L^{(n-1)}$=$L^{(n)}$=…

HIT CS&E

7.7.3 Floyd-Warshall算法

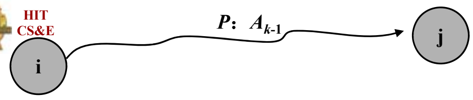
- 动态规划算法求解all-pairs最短路径
  - 遵循动态规划算法设计的一般过程
  - 运行时间为$O(V^3)$
  - 允许有负权值的边
  - 但不允许有负环
- 给出一个类似的算法寻找有向图的传递闭包

HIT CS&E

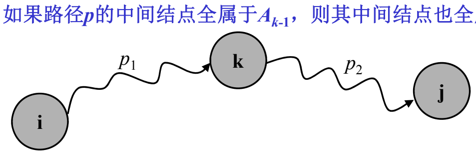
优化子结构

- 设i到j的最短路径为p:$i\to\ldots\to k\to\ldots\to j$
  - 矩阵算法的优化子结构考虑j在p上的前驱
  - Floyd-Warshall算法的优化子结构将考虑路径p上经过的中间结点集
- 设$V=\{1,2,\ldots,n\}$　　$A_k=\{1,2,\ldots,k\}\subseteq V$
  - p是从i到j的中间结点全属于A_k的最短路径
  - P'是从i到j的中间结点全属于A_{k-1}的最短路径
  - Floyd-Warshall算法通过考察p和p'之间的关系建立优化子结构

HIT CS&E



如果路径p的中间结点全属于A_{k-1}，则其中间结点也全属于A_k



如果路径p的中间结点全属于A_k且包含了结点k

- p可以从k处断开成两条路径
- p_1:$i\to\ldots\to k$是从i到k的中间结点全属于A_{k-1}的最短路径
- p_2:$k\to\ldots\to j$是从i到k的中间结点全属于A_{k-1}的最短路径

HIT CS&E

例

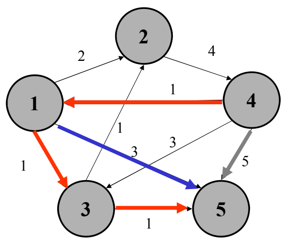
i=4 ， j=5

k=0

k=1

k=2

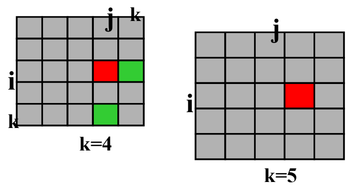
k=3



HIT CS&E

递归关系的建立

- 令$d_{ij}^{(k)}$＝从i到j的中间结点全属于A_k的最短路径的代价
  - $d_{ij}^{(k)}=w(i,j)$　if　k=0　(此时路径至多一条边)
  - $d_{ij}^{(k)}=\min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)}+d_{kj}^{(k-1)}\}$　if $k\geq 1$
  - $D^{(n)}=(d_{ij}^{(n)})_{n\times n}$给出最终答案，$d_{ij}^{(n)}=\delta(i,j)$



HIT CS&E

Floyd-Warshall算法

算法Floyd_Warshall(W)
Input　图的边加权函数矩阵W
Output all_pair_Shortest_Paths代价
1. $n\leftarrow row[L]$;
2. $D^{(0)}\leftarrow W$
3. For $k\leftarrow 1$ to n do
4. 　For $i\leftarrow 1$ to n do
5. 　　For $j\leftarrow 1$ to n do
6. 　　　$d_{ij}^{(k)}=\min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)}+d_{kj}^{(k-1)}\}$
5. Return $D^{(n)}$

定义前驱矩阵$\pi^{(k)}=(\pi_{ij}^{(k)})_{n\times n}$

–如何在计算$D^{(k)}$的过程中，完成$\pi^{(k)}$的计算

–如何由$\pi^{(n)}$给出最短路径

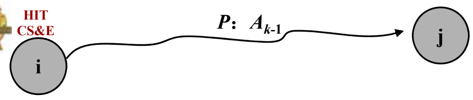
HIT CS&E

有向图的传递闭包

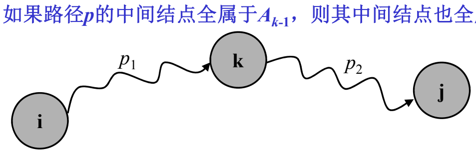
- 给定有向图$G=(V,E)$,其中$V=\{1,2,\ldots,n\}$
  - $\forall i,j\in V$, 在G中从i到j是否有有向路径可达
  - G的传递闭包是有向图$G^*=(V,E^*)$，其中$E^*=\{(i,j)|$在G中存在从i到j的有向路径$\}$
  - 令G中每条边的权值为1，调用Floyd-Warshall算法可以计算传递闭包

优化子结构

- 设i到j的最短路径为$p:i\rightarrow\dots\rightarrow k\rightarrow\dots\rightarrow j$
  - 矩阵算法的优化子结构考虑j在p上的前驱
  - Floyd-Warshall算法的优化子结构将考虑路径p上经过的中间结点集
- 设$V=\{1,2,\dots,n\}$　　$A_k=\{1,2,\dots,k\}\subseteq V$
  - p是从i到j的中间结点全属于A_k的最短路径
  - P'是从i到j的中间结点全属于A_{k-1}的最短路径
  - Floyd-Warshall算法通过考察p和p'之间的关系建立优化子结构



如果路径p的中间结点全属于A_{k-1}，则其中间结点也全属于A_k



如果路径p的中间结点全属于A_k且包含了结点k

- p可以从k处断开成两条路径
- $p_1:i\rightarrow\dots\rightarrow k$是从$i$到$k$的中间结点全属于$A_{k-1}$的最短路径
- $p_2:k\rightarrow\dots\rightarrow j$是从$i$到$k$的中间结点全属于$A_{k-1}$的最短路径

例

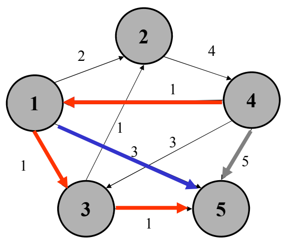
i=4 ， j=5

k=0

k=1

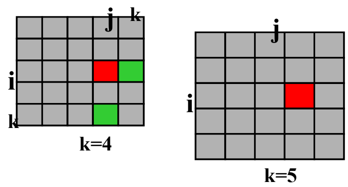
k=2

k=3



递归关系的建立

- 令$d_{ij}^{(k)}$＝从i到j的中间结点全属于A_k的最短路径的代价
  - $d_{ij}^{(k)}=w(i,j)$　if　k=0　(此时路径至多一条边)
  - $d_{ij}^{(k)}=\min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)}+d_{kj}^{(k-1)}\}$　if $k\geq 1$
  - $D^{(n)}=(d_{ij}^{(n)})_{n\times n}$给出最终答案，$d_{ij}^{(n)}=\delta(i,j)$



Floyd-Warshall算法

算法Floyd_Warshall(W)
Input　图的边加权函数矩阵W
Output all_pair_Shortest_Paths代价
1. $n\leftarrow row[L]$;
2. $D^{(0)}\leftarrow W$
3. For $k\leftarrow 1$ to n do
4. 　For $i\leftarrow 1$ to n do
5. 　　For $j\leftarrow 1$ to n do
6. 　　　$d_{ij}^{(k)}=\min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)}+d_{kj}^{(k-1)}\}$
5. Return $D^{(n)}$

定义前驱矩阵$\pi^{(k)}=(\pi_{ij}^{(k)})_{n\times n}$

–如何在计算$D^{(k)}$的过程中，完成$\pi^{(k)}$的计算

–如何由$\pi^{(n)}$给出最短路径

有向图的传递闭包

- 给定有向图$G=(V,E)$,其中$V=\{1,2,\dots,n\}$
  - $\forall i,j\in V$, 在G中从i到j是否有有向路径可达
  - G的传递闭包是有向图$G^*=(V,E^*)$，其中E*＝{(i,j)| 在G中存在从i到j的有向路径}
  - 令G中每条边的权值为1，调用Floyd-Warshall算法可以计算传递闭包