图的高级主题选讲

- **负权图最短路径**
- 次小支撑树
- 网络流初探

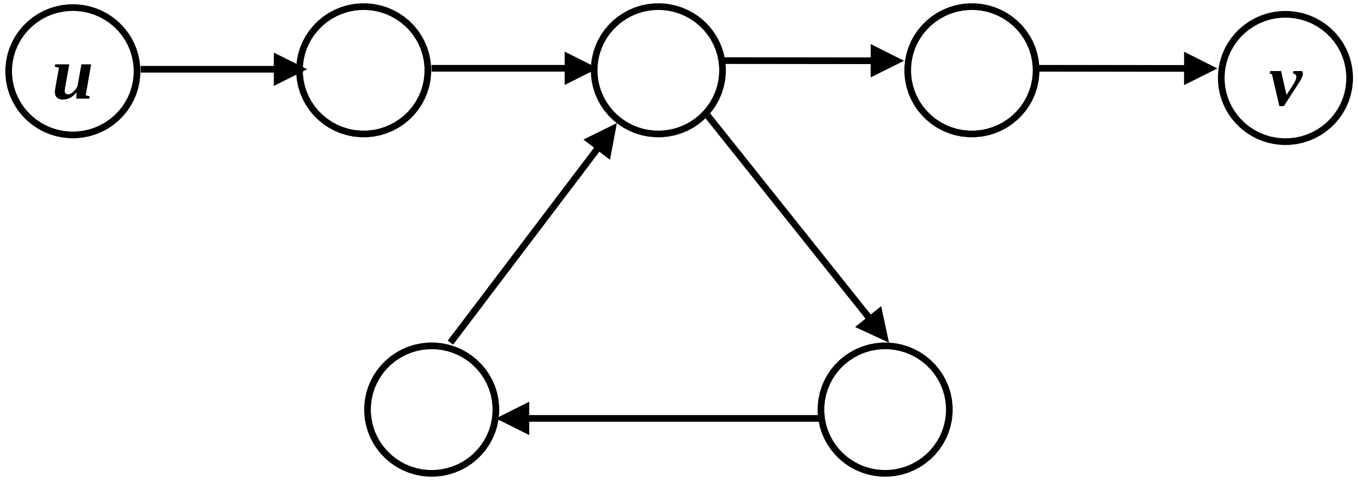
数据之法
结构之美
算法之道

Last updated on 2024.12

zhuyungang@jlu.edu.cn

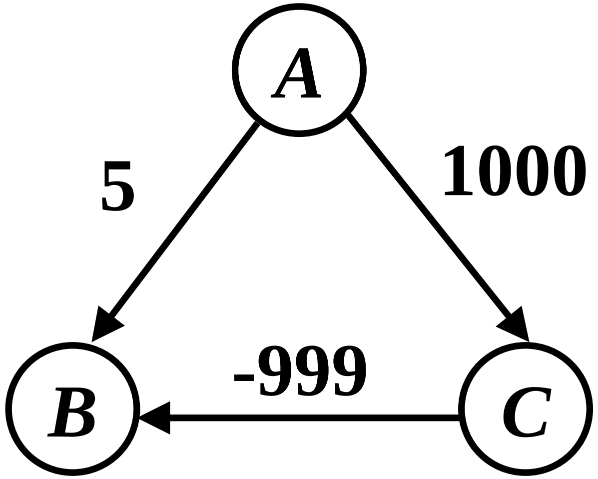
最短路径中是否含有环

- 正权环？
- 负权环？
- 0权环？
- 不失一般性地，我们可以假定最短路径中没有环，最多包含n个顶点，n-1条边。



负权图单源最短路径问题

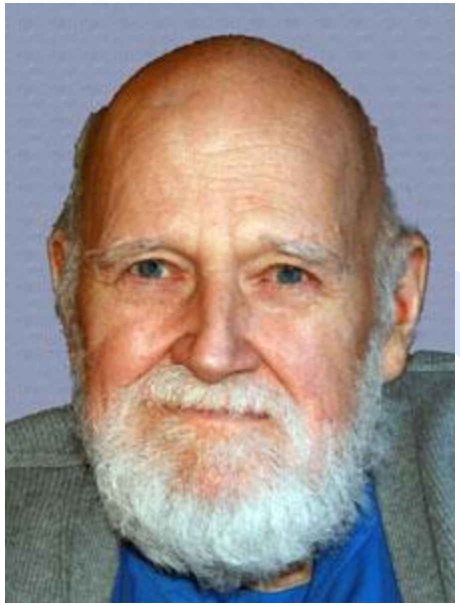
- **Dijkstra算法**：不能处理负权图。当处理正权图时，选出 D_v 值最小的顶点 v 加入集合S后，源点到 v 的最短距离即确定。但在负权图中 v 加入集合S后，源点到 v 的最短距离仍不确定。
- **Bellman-Ford算法**：对于每个点的 D 值，做足够多次计算更新，直至 D 值不再减小。



Bellman-Ford算法

Richard Bellman

南加州大学教授
美国科学院院士
美国工程院院士
动态规划之父



Lester Ford

美国数学家

Bellman-Ford算法

步骤1：初始化

FOR $i = 1$ **TO** n **DO** $D[i] \leftarrow +\infty$.

$D[s] \leftarrow 0$.

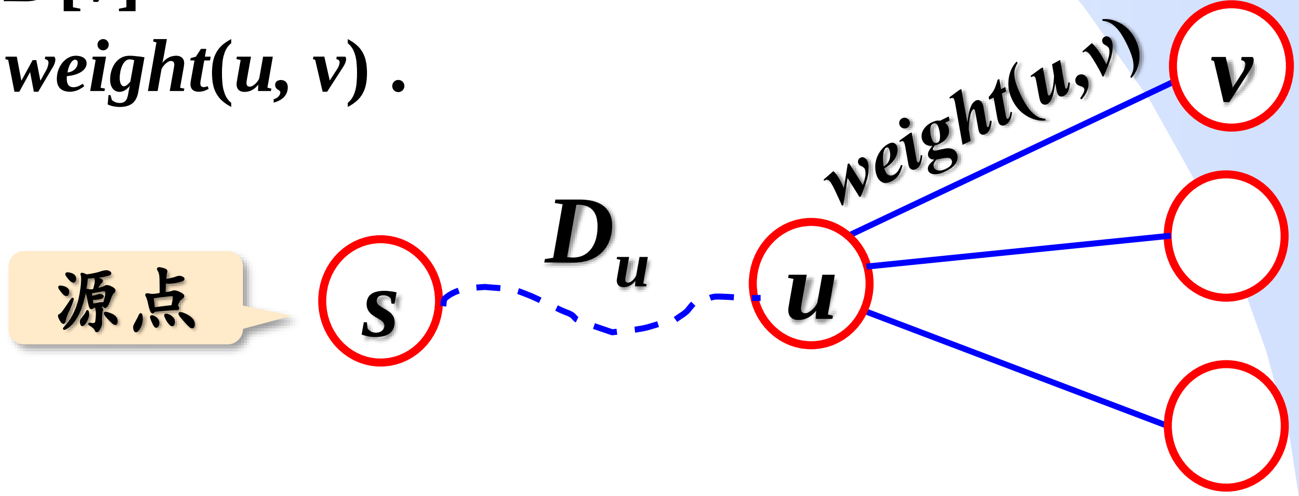
步骤2：迭代求解源点s到各点的最短距离

FOR $i = 1$ **TO** x **DO**

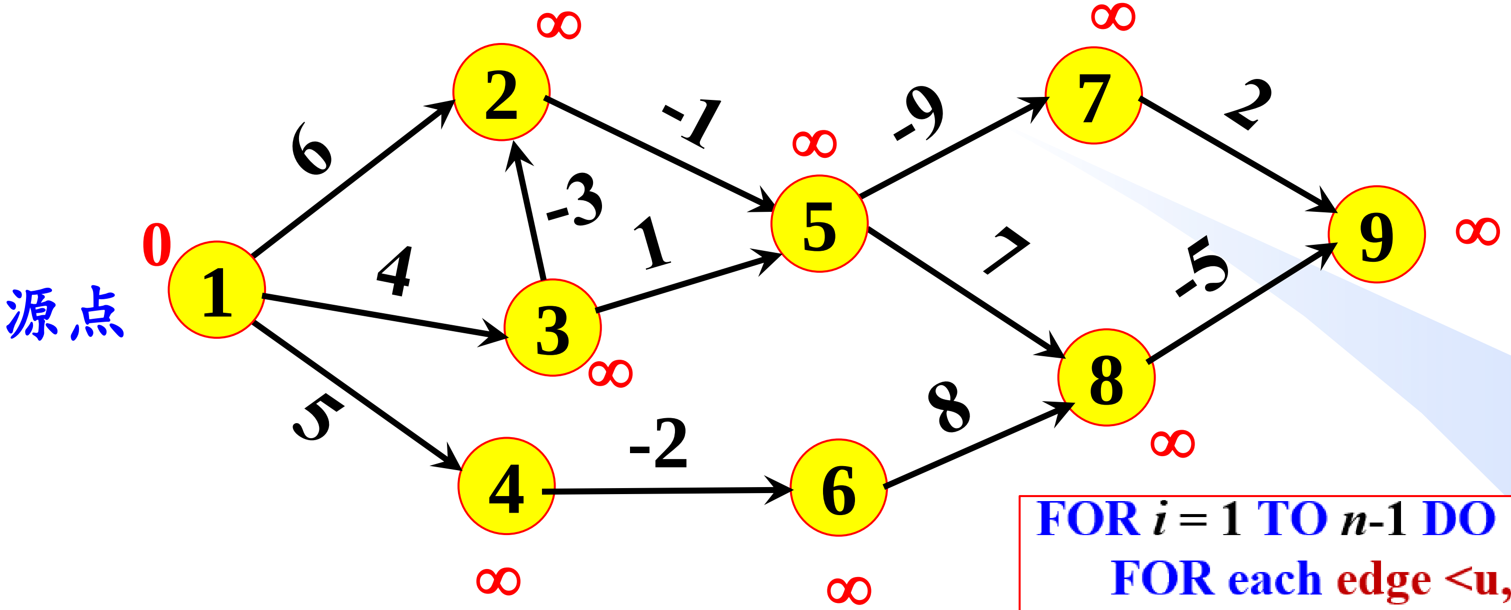
  FOR each edge <u, v> DO

    IF $D[u] + weight(u, v) < D[v]$

     THEN $D[v] \leftarrow D[u] + weight(u, v)$.



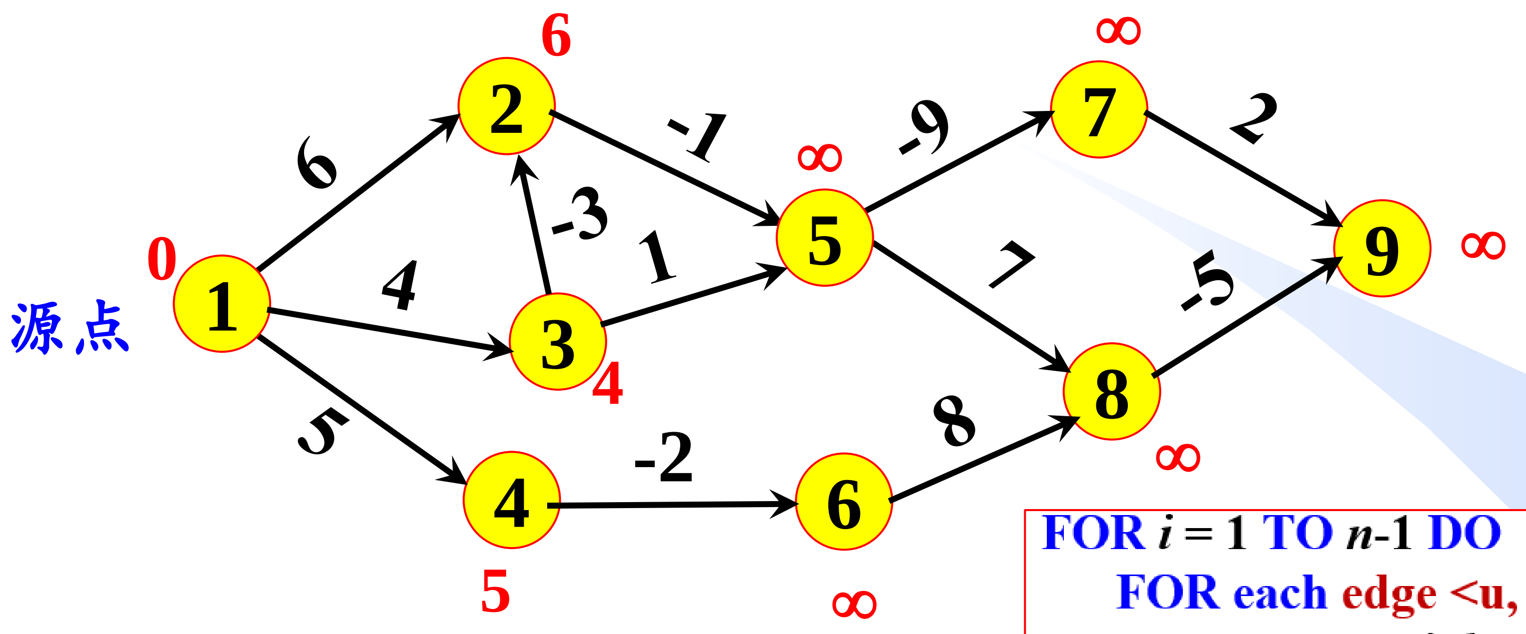
Bellman-Ford算法的正确性




```
FOR i = 1 TO n-1 DO
    FOR each edge <u, v> DO
        IF D[u]+ weight(u, v) < D[v]
            THEN D[v] ← D[u] + weight(u, v)
```

第1次迭代，一定找到V_1到各点的至多包含1条边的最短距离

Bellman-Ford算法的正确性

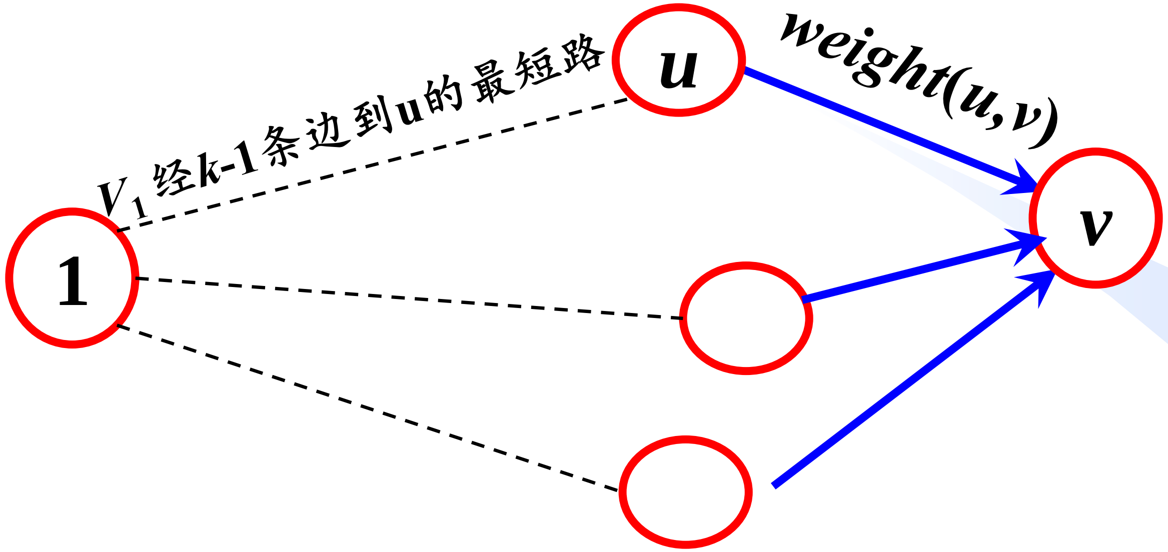


FOR i = 1 **TO** n-1 **DO**
  FOR each edge <u, v> DO
    IF $D[u]+ weight(u, v) < D[v]$
     THEN $D[v] \leftarrow D[u] + weight(u, v)$

第1次迭代，一定找到V_1到各点的至多包含1条边的最短距离
第2次迭代，一定找到V_1到各点的至多包含2条边的最短距离
第3次迭代，一定找到V_1到各点的至多包含3条边的最短距离

Bellman-Ford算法的正确性

第*k*次迭代



第*k*次迭代，一定找到V_1到各点的至多包含*k*条边的最短距离

```
FOR each edge <u, v> DO
   IF D[u]+ weight(u, v) < D[v]
     THEN D[v] ← D[u] + weight(u, v) .
```

Bellman-Ford算法

- 第*k*次迭代，一定找到V_1到各点的至多包含*k*条边的最短距离。
- 一共需要几次迭代？最短路径最多包含全部*n*个顶点，*n*-1条边，即V_1到各点的最短路径至多包含*n*-1条边，故至多迭代*n*-1次。

步骤1：初始化

FOR *i* = 1 **TO** *n* **DO** *D*[*i*] ← +∞.

D[*s*] ← 0.

步骤2：迭代求解源点s到各点的最短距离

FOR *i* = 1 **TO** *n*-1 **DO**

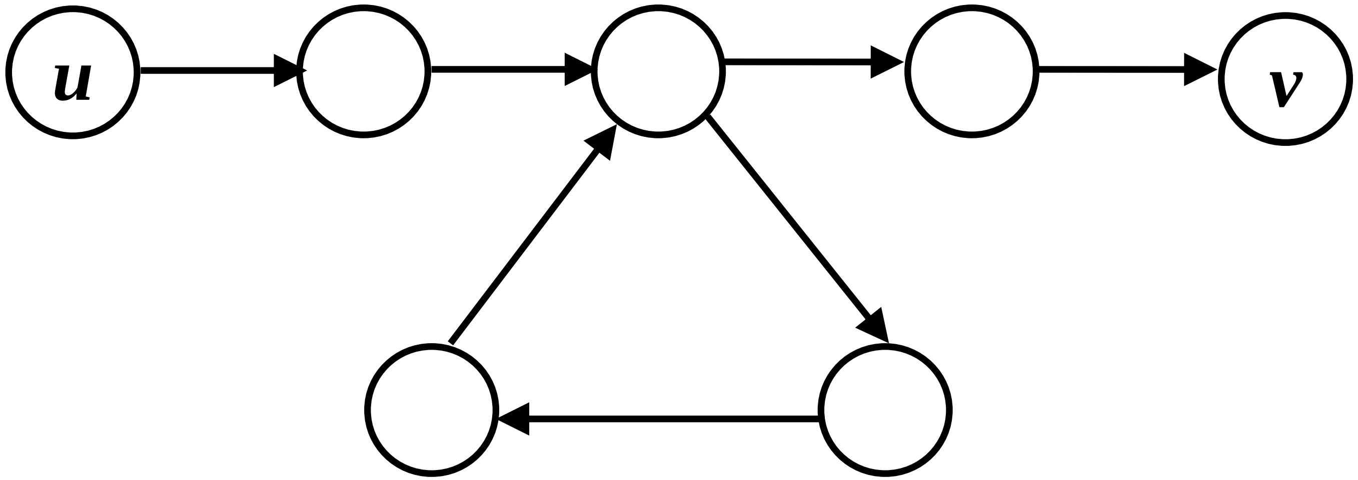
    FOR each edge <u, v> DO

        IF *D*[*u*]+ *weight*(*u*, *v*) < *D*[*v*]

          THEN *D*[*v*] ← *D*[*u*] + *weight*(*u*, *v*) .

Bellman-Ford算法

- 能处理负权图。但若图中含有负环，则算法不能获得正确结果，但能识别负环情况。
- 如果*n*-1次迭代后，再做1次迭代，源点到某点的最短距离还有变化，则肯定含有负环。



Bellman-Ford算法

步骤1：初始化

FOR $i = 1$ **TO** n **DO** $D[i] \leftarrow +\infty$.

$D[s] \leftarrow 0$.

步骤2：迭代求解源点s到各点的最短距离

```
FOR i = 1 TO n-1 DO
    FOR each edge <u, v> DO
        IF D[u]+ weight(u, v) < D[v]
         THEN D[v] ← D[u] + weight(u, v) .
```

步骤3：检验负权环

```
FOR each edge <u, v> DO
        IF D[u]+ weight(u, v) < D[v]
            THEN RETURN FALSE.
RETURN TRUE.
```

时间复杂度

$O(ne)$

Bellman-Ford算法

```
bool Bellman_Ford(Edge E[],int s,int n,int e,int dist[]){
  for(int i=1;i<=n;i++) dist[i]=(i==s)?0:INF; //初始化
  for(int i=1; i<=n-1; i++)          //n-1轮计算
     for(int k=0; k<e; k++) {        //扫描所有边
          int u=E[k].u; int v=E[k].v; int w=E[k].weight;
          if(dist[u]+w<dist[v]) dist[v]=dist[u]+w;
     }
  for(int k=0; k<e; k++) {              //检测负环
     int u=E[k].u; int v=E[k].v; int w=E[k].weight;
     if(dist[u]+w<dist[v]) return false; //有负环
  }
  return true;
}
```

```
struct Edge{
    int u,v;
    int weight;
};
Edge E[1000];
```

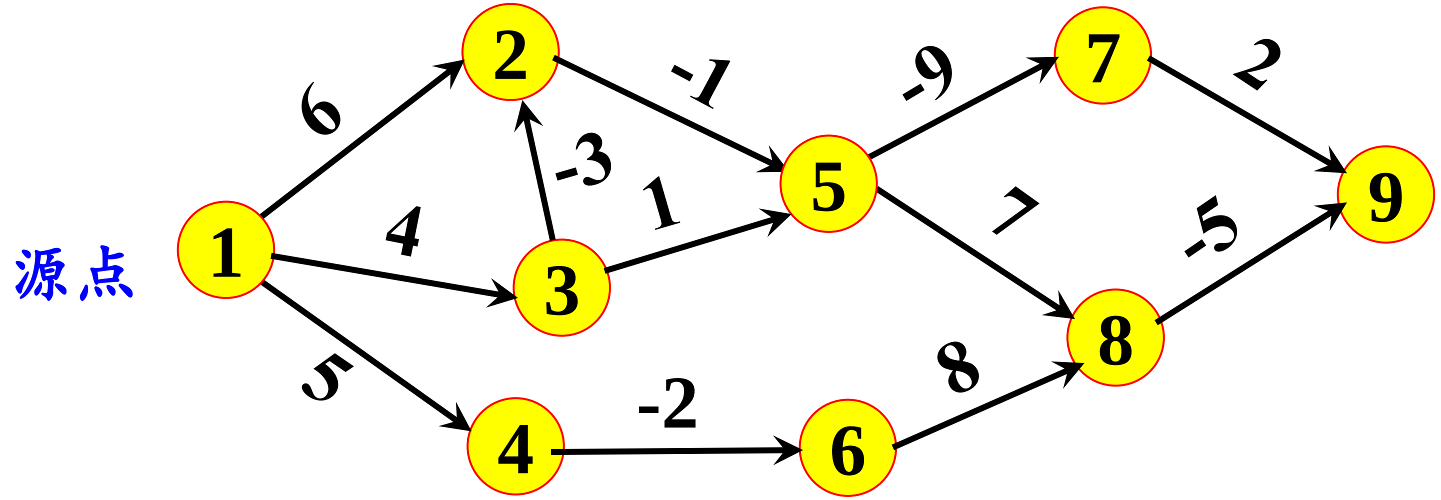
练习

只要在无向有权图中存在1个环（回路）的权值之和为负值，我们就称此无向图存在“负权回路”下面哪个算法可以检验一个无向图是否存在负权回路？ 【搜狗校园招聘笔试题】

- A. 最短路径 Bellman-Ford 算法
- B. 最小生成树 Kruskal 算法
- C. 最小生成树 Prim 算法
- D. 最短路径 Dijkstra 算法

队列优化的Bellman-Ford算法

- 本次迭代中*D*值被更新的顶点，其邻接顶点的*D*值可能在下次迭代时更新。
- 更新*D*[*v*]之后，下一步只计算和调整顶点*v*的邻接顶点，可加快收敛速度。



队列优化的Bellman-Ford算法

- 算法采用一个队列。初始时将源点入队，持续从队列中取出一个顶点，并考察更新其邻居的*D*值，若某个邻居的*D*值被更新，则将其入队。直至队列为空时算法结束。
- 也称为SPFA算法（Shortest Path Faster Algorithm）
- 最坏情况下时间复杂度与Bellman-Ford算法相同，为O(*ne*)。但在稀疏图上运行效率较高，为O(*ke*)，其中*k*为一个较小的常数。

SPFA算法

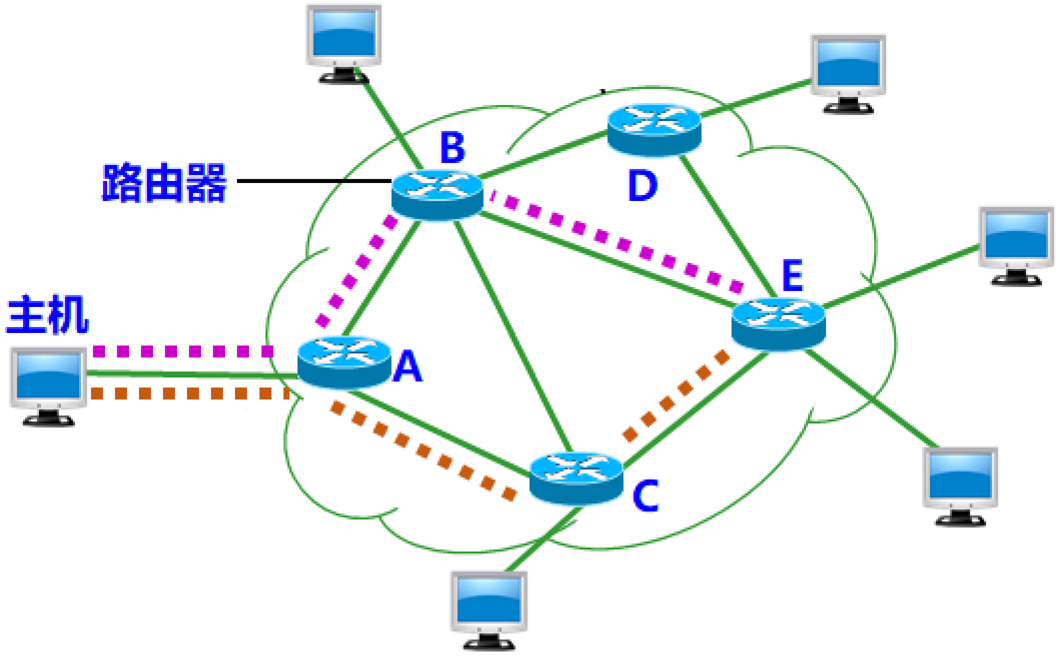
```
void SPFA(Vertex* Head, int n, int u, int dist[]) {
   Queue<int> Q;  int InQueue[N]={0};
   for(int i=1;i<=n;i++) dist[i]=(i==u)?0:INF; //dist初始化
   Q.enQueue(u); InQueue[u]=1;
   while(!Q.Empty()) {
      u = Q.deQueue();  InQueue[u] = 0;
      for(Edge* p=Head[u].adjacent; p; p=p->link) {
         int v=p->VerAdj;
         if(dist[u] + p->cost < dist[v]) {
            dist[v] = dist[u]+p->cost;
            if(!InQueue[v])
               Q.enQueue(v),InQueue[v]=1;
         }
      }
   }
}
```

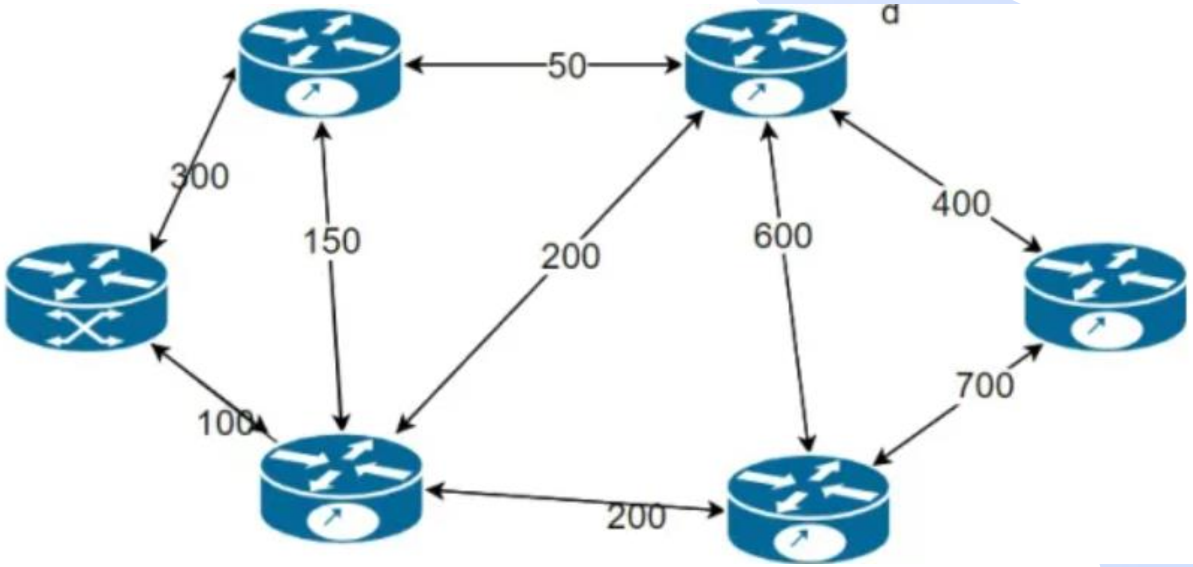
✓顶点出队后可以再次入队

✓若某点入队次数大于n次，则存在负环

计算机网络的路由算法

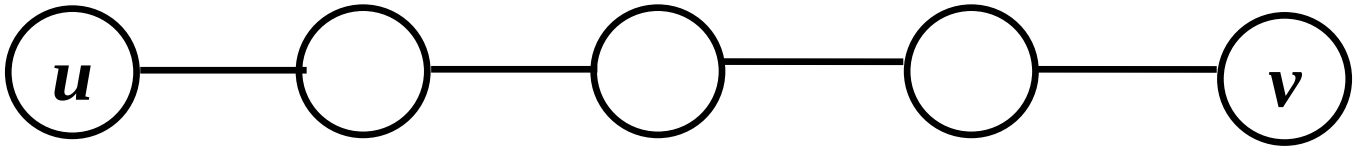
- OSPF路由协议：基于Dijkstra算法
- RIP、BGP路由协议：基于Bellman-Ford算法





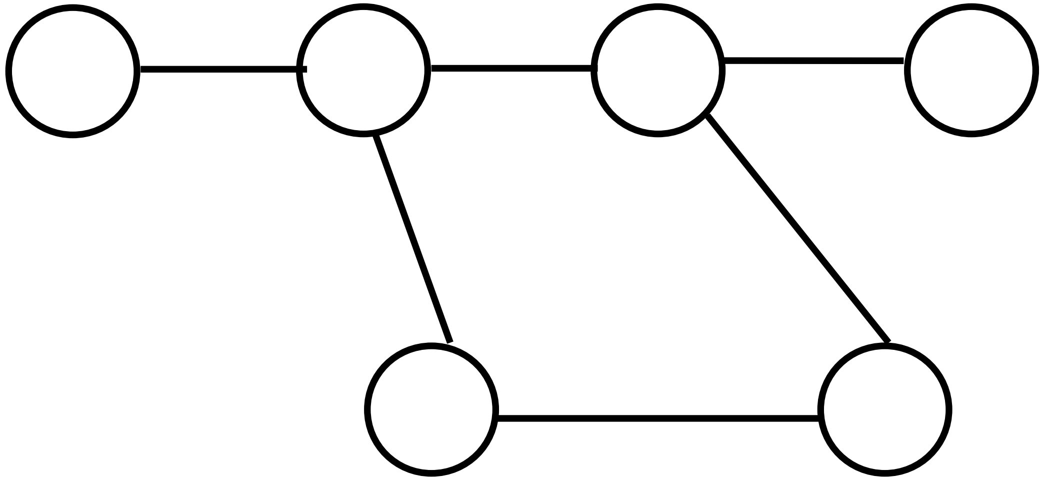
练习

A国有n个城市，编号为1至n。小明需要从城市1到城市n出差。城市间航线有m条，已知每条航线的航行时间。另外因为疫情，小明到达城市i后需要进行c_i时间的隔离。编写程序帮助小明规划一条路线，能够在最短时间到达城市n。【2022年蓝桥杯国赛真题】



练习

小明的实验室有n台计算机，编号为1至n。原本这n台计算机之间恰有n-1条数据链路，构成一个树形网络。不过再最近一次维护中时，管理员误操作使某两台计算机之间增加了一条数据链路。使网络中出现环路，为恢复正常，请帮助小明找到环路上的计算机。【2017年蓝桥杯国赛真题】



图的高级主题选讲

- 负权图最短路径
- 次小支撑树
- 网络流初探

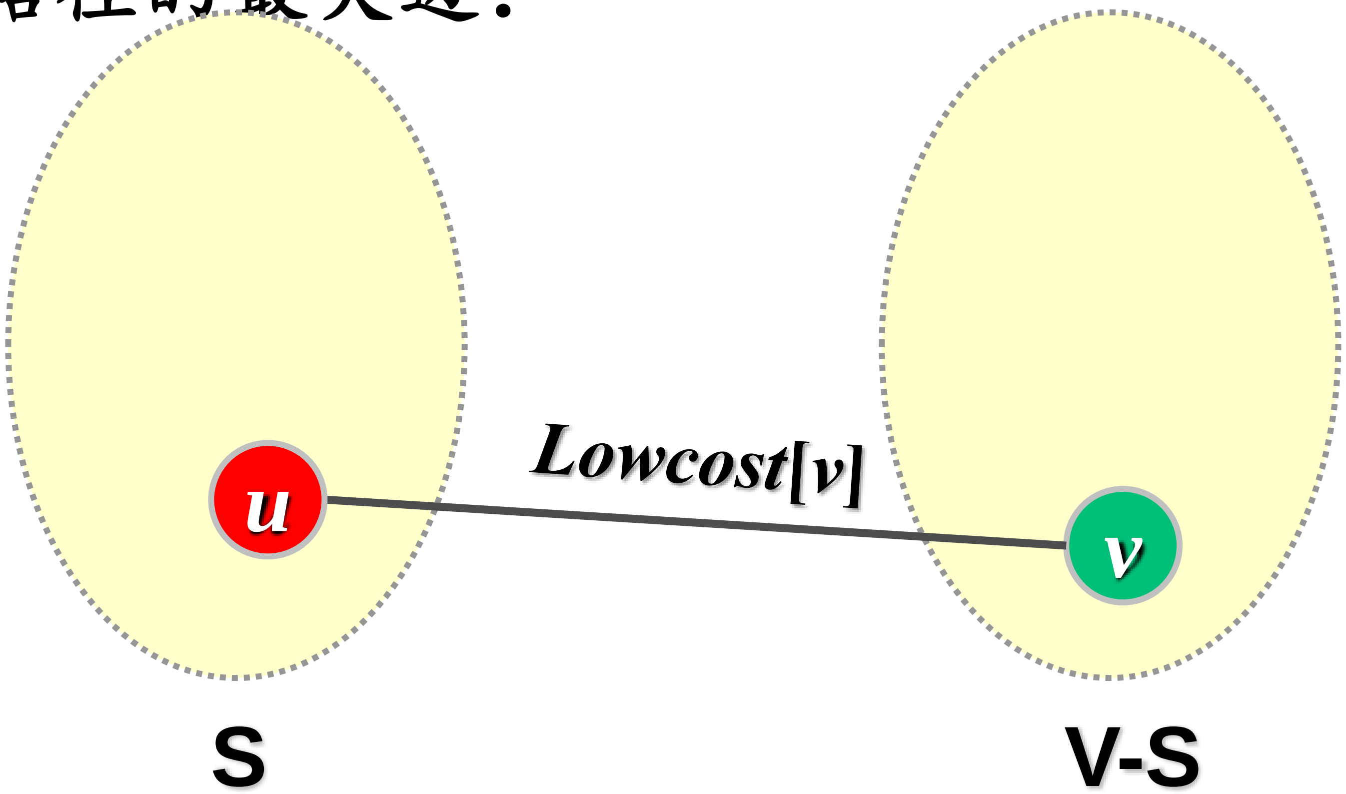
数据之法
结构之美
算法之道

Last updated on 2024.12

zhuyungang@jlu.edu.cn

最小支撑树中的最大边

- maxcost[u][v]：最小支撑树中点u到点v的路径中的最大边的权值
- 每选出一条最小跨边(u,v)，把v加入集合S时，计算v到S中各点间路径的最大边：



最小支撑树中的最大边(动态规划)

- **maxcost[u][v]：最小支撑树中点u到点v的路径中的最大边的权值**
- **每选出一条最小跨边(u,v)，把v加入集合S时，计算v到S中各点间路径的最大边：**

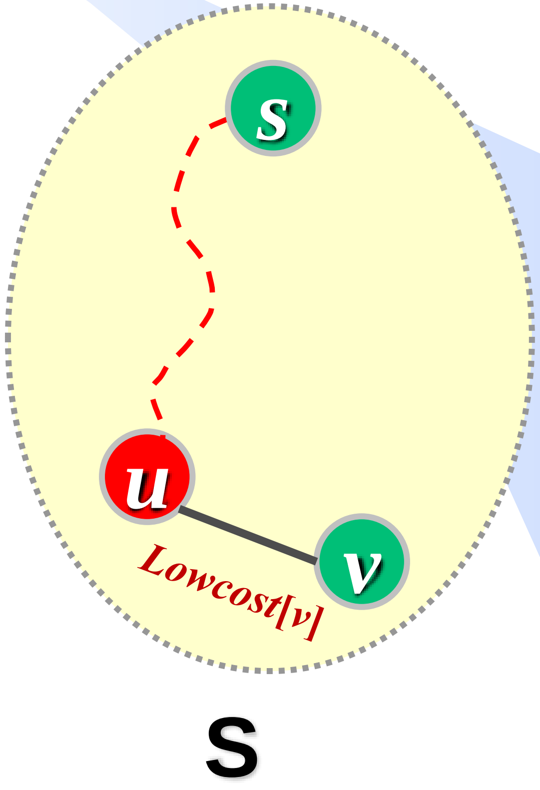
对于u=pre[v]，有：

maxcost[u][v]=maxcost[v][u]=Lowcost[v];

对于集合S中的其他点*s*，有：

maxcost[s][v]=maxcost[v][s]

=max(maxcost[u][v], maxcost[s][u]);



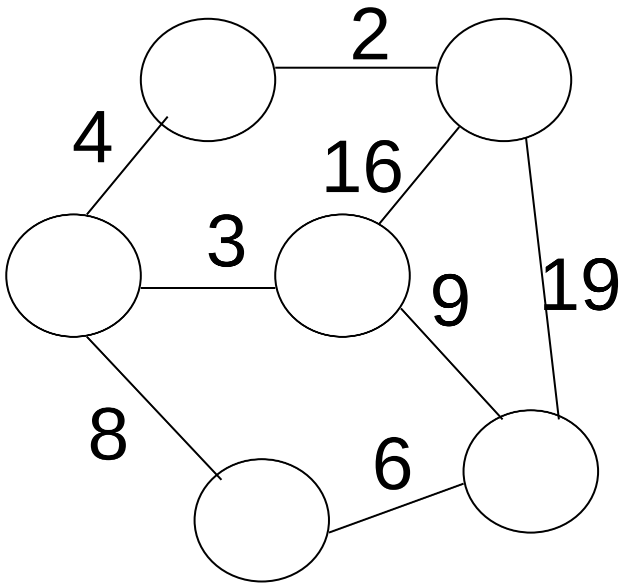
Prim算法的实现

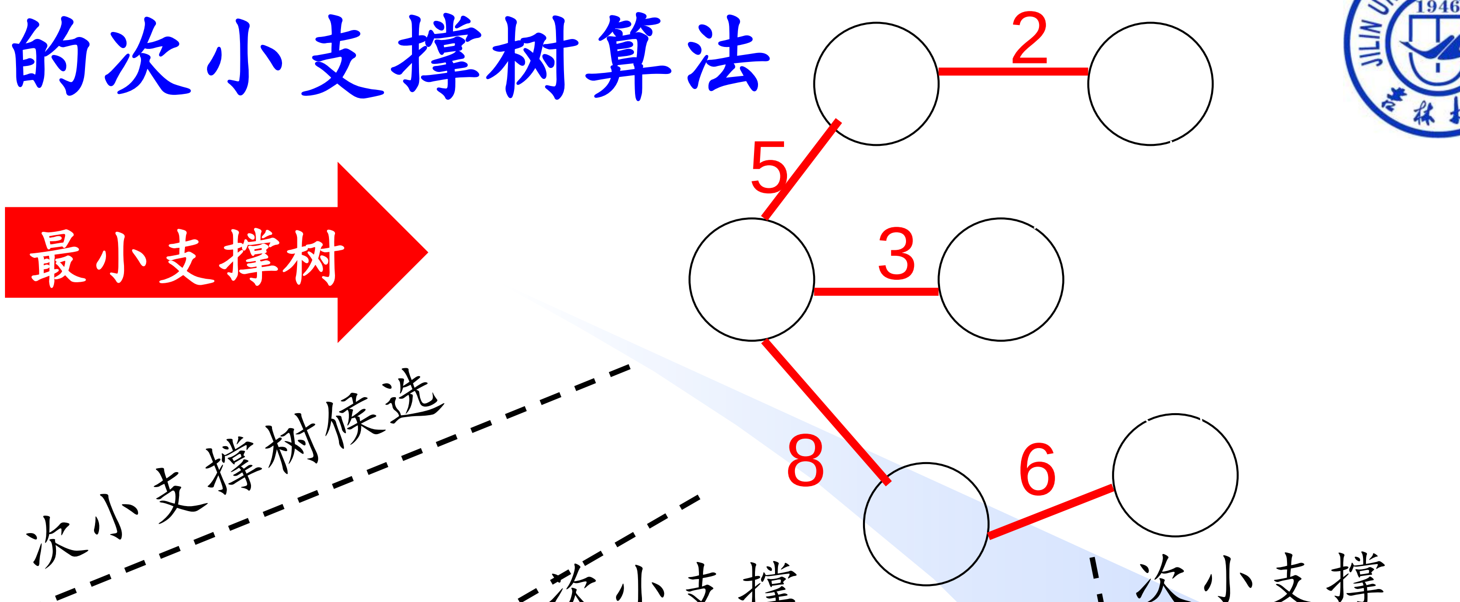
B

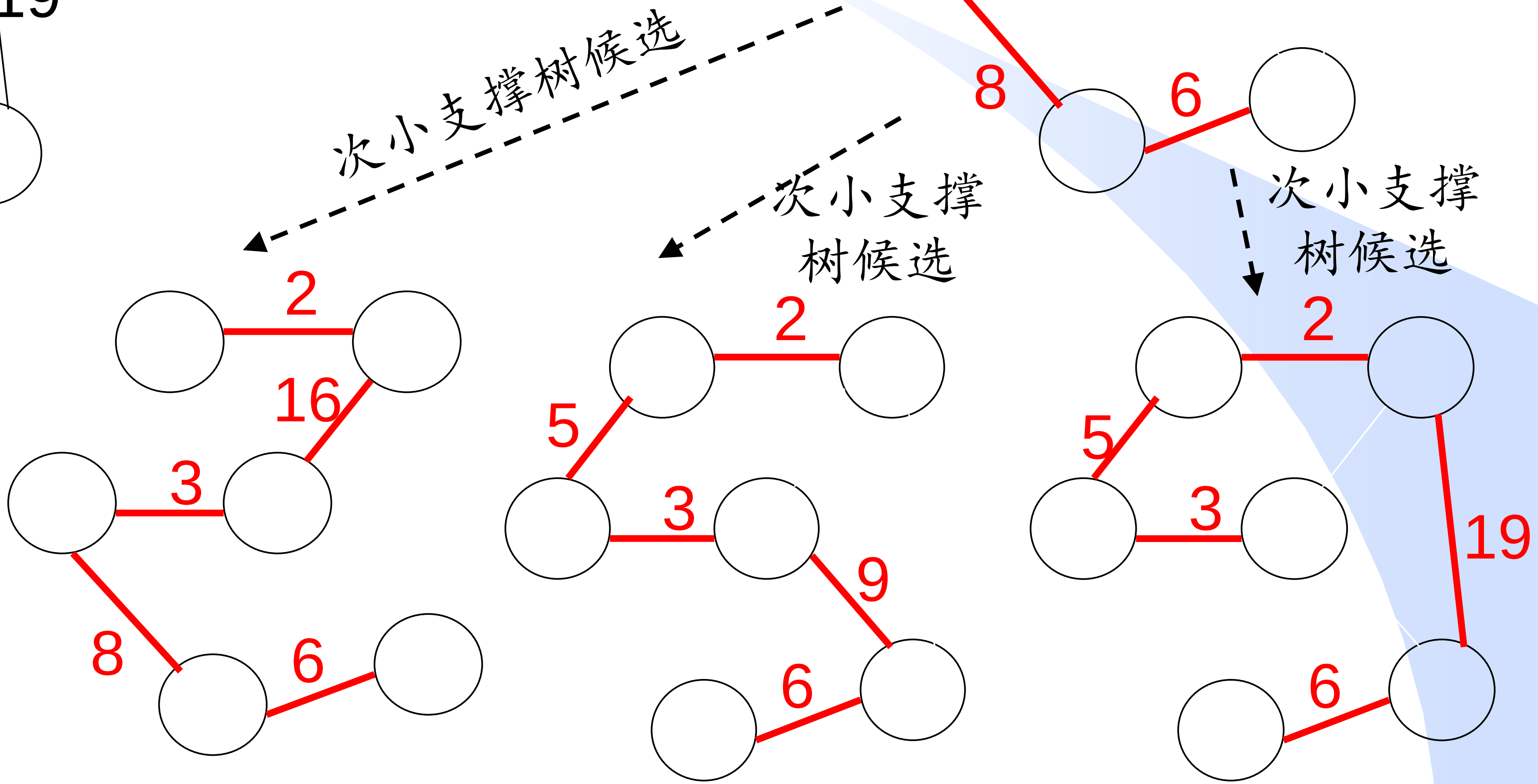
```
int Prim(int G[N][N],int n,int u,int Lowcost[],int pre[]){
  int S[N]={0}, sum=0; // sum存MST边权之和，作为函数返回值
  for(int i=1;i<=n;i++){pre[i]=-1; Lowcost[i]=(i==u)? 0:INF;}
  for(int i=1;i<=n;i++){            //把n个点逐个加入集合S
    int v=FindMin(S,Lowcost,n); //从不在S中的点里选Lowcost值最小的点
    if(v==-1) return sum;       //不存在跨边，图不连通
    sum+=Lowcost[v];        //找到一条MST的边 (pre[v],v), v加入集合S
    对于maxcost的处理
    S[v]=1;
    for(int w=1; w<=n; w++)    //更新v邻接顶点的Lowcost和pre值
      if(S[w]==0 && G[v][w]<Lowcost[w]){
          Lowcost[w]=G[v][w]; pre[w]=v;
      }
  }
  return sum;
}
```

时间复杂度
$O(n^2)$

求图G的次小支撑树算法







枚举G中不在MST中的每条边(*u*,*v*)：删去MST中u到v路径中的最大边，加入边(*u*,*v*)，得到一棵可能的次小支撑树

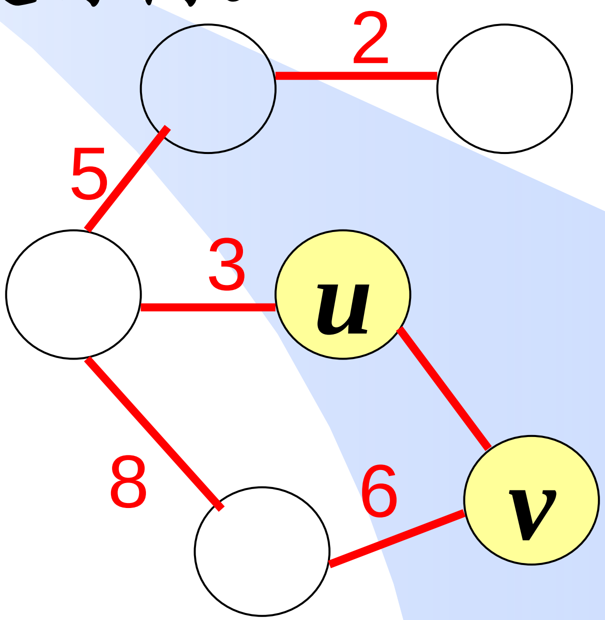
求图G的次小支撑树算法

（1）求出G的最小支撑树MST和maxcost数组；

（2）枚举G中不在MST中的每条边(*u*,*v*)：删去MST中u到v路径中的最大边，加入边(*u*,*v*)，得到一棵可能的次小支撑树；

（3）所有可能的次小支撑树中的最小者即次小支撑树。

```
int cost=Prim(G,n); //求G的mst, 边权和为cost
int ans, min = INF;
for(each edge(u,v) ∈ G)
    if(edge(u,v) ∉ mst){
        ans = cost + G[u][v] - maxcost[u][v];
        if(ans < min) min = ans;
    }
return min;
```



时间复杂度$O(n^2)$

延伸

- 问题：给定图G，判断图G的最小支撑树是否唯一?
- 方案：用Prim算法求最小支撑树和次小支撑树，看二者边权之和是否相等。时间复杂度O(n^2)

图的高级主题选讲

- 负权图最短路径
- 次小支撑树
- **网络流初探**

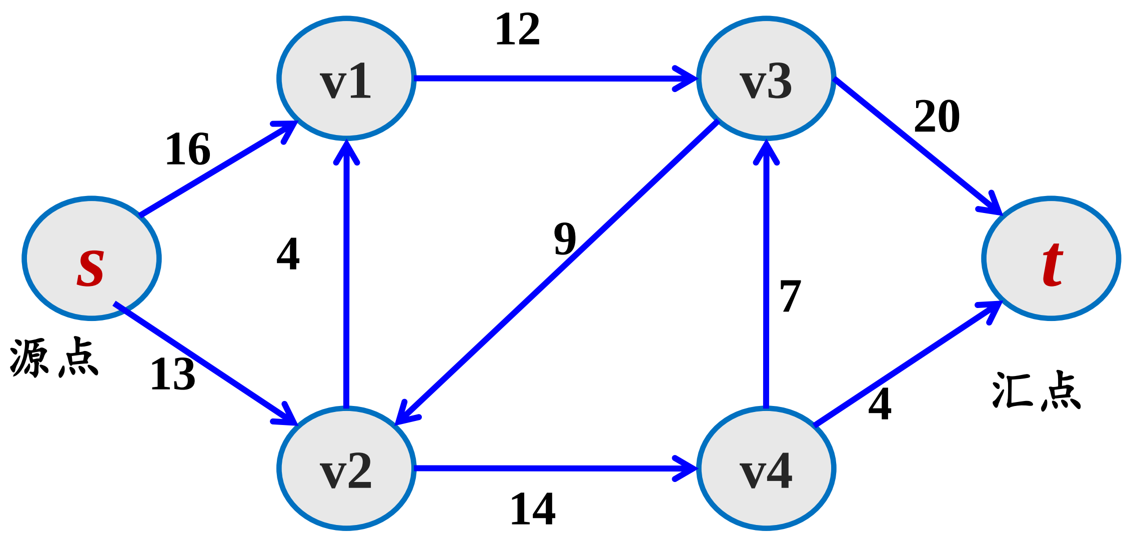
数据之法
结构之美
算法之道

Last updated on 2024.12

zhuyungang@jlu.edu.cn

网络流问题

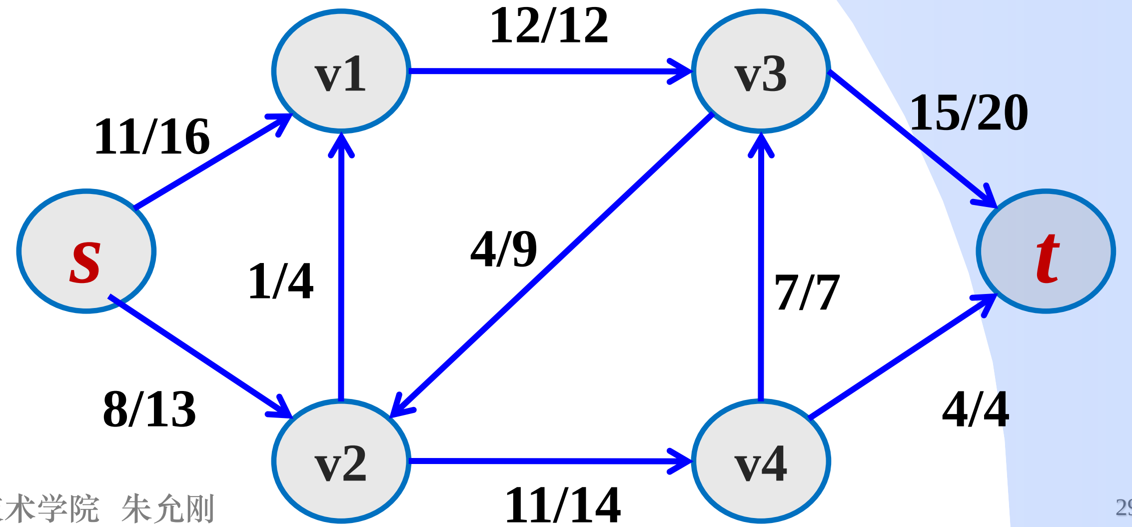
- 将有向图看做一个“流网络”，以建模和优化物品流动方面的问题，如物流运输、石油等管道运输、车流量等。
- 运输的对象称为流，承载流的运输网络称为流网络。
- 流网络中的有向边看做是物品流通的通道，每条通道有限定的容量，即边的权值，表示运输最大承载量，例如公路的最大货运能力、石油管道的输送能力、网络带宽等。



道路上的车流
网络中的数据流（带宽）
管道中液体的流动
电网中电流的流动
通信网络中的信息流动
运输路线上货物的流动

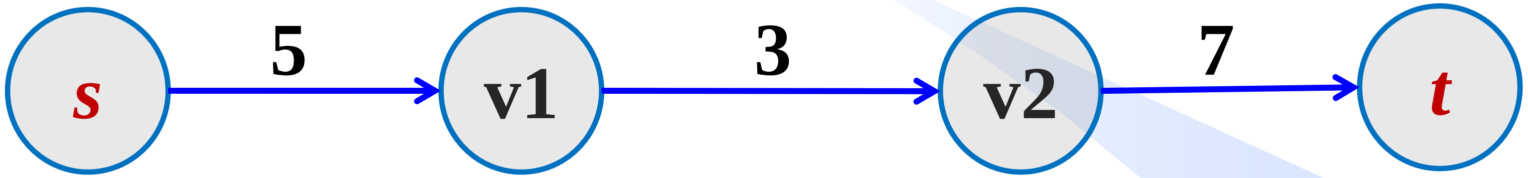
网络流的概念

- 流网络是一个有向图G=(V, *E*)，有一个源点*s*和汇点*t*，边<*u* , *v*>有一个非负容量 $c(u,v)$。
- G上的流 $f(u, v)$ 定义为满足如下条件的函数：
  - ✓容量限制：$0 \le f(u,v) \le c(u,v)$ 。
  - ✓流量守恒：对于除*s*和*t*之外的任意顶点，流入流量之和=流出流量之和。
- **最大流问题**：给定流网络G，源点*s*，汇点*t*，问从*s*到*t*最多能输送多大的流？【[洛谷P3376]】

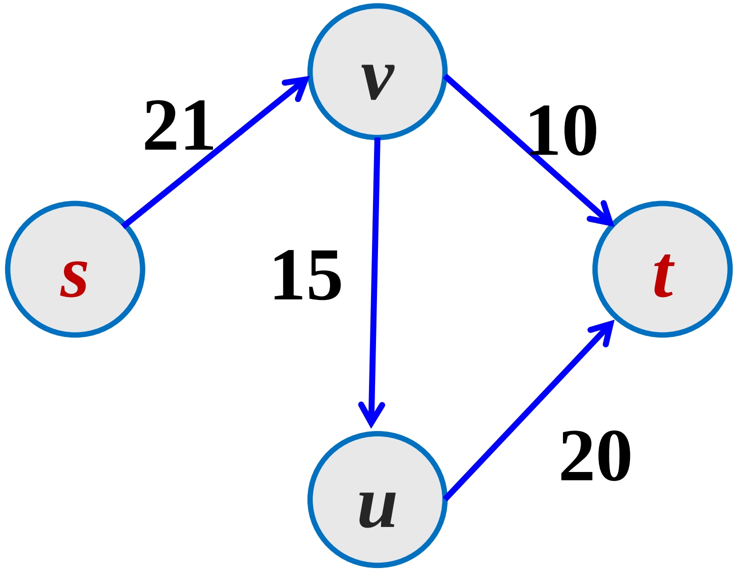


如何求最大流

- 残存边:流网络中仍有流量调整空间的边。
- 残存容量:边的容量-该边当前流量，若残存容量等于0，则不存在残存边。



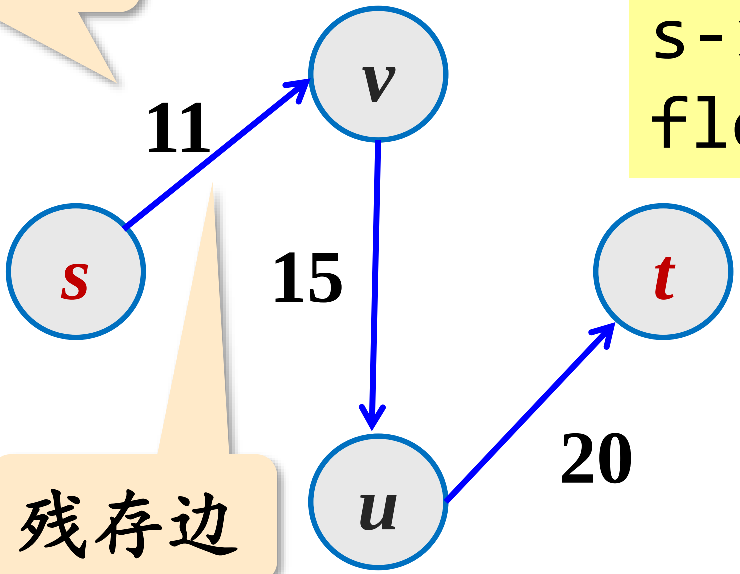
最大流：min{5, 3, 7}=3



s-v-t

flow=10

残存容量

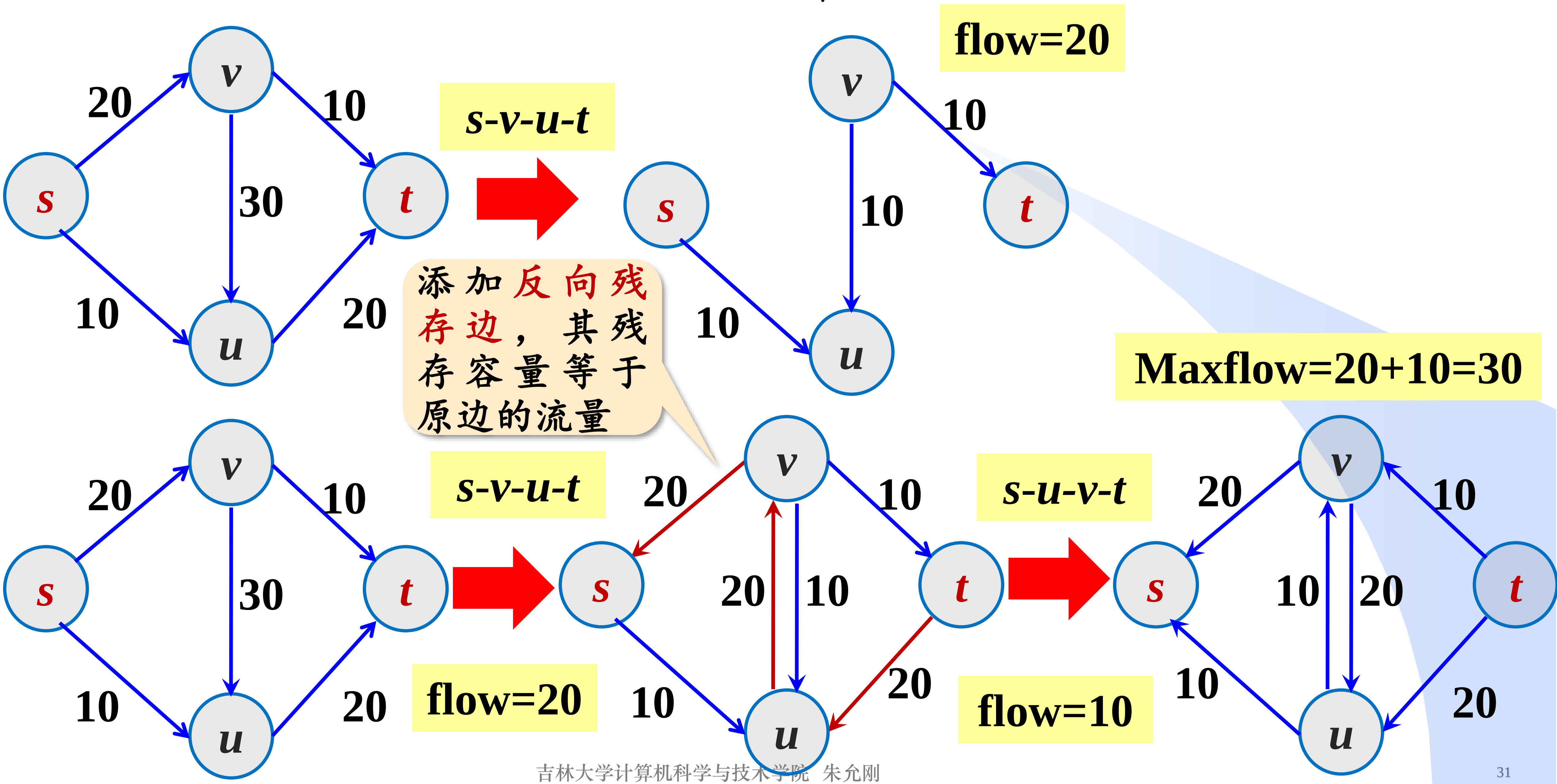


残存边

s->v->u->t
flow=10+11=21

残存网络:包含残存边的网络，表示当前流网络中还有多少流量可以增加

如何求最大流

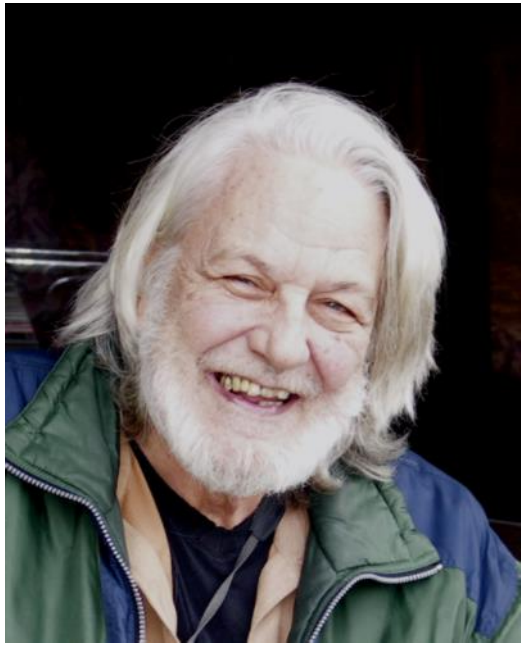


s-v-u-t
flow=20
添加反向残存边，其残存容量等于原边的流量
s-v-u-t
flow=20
s-u-v-t
flow=10
Maxflow=20+10=30

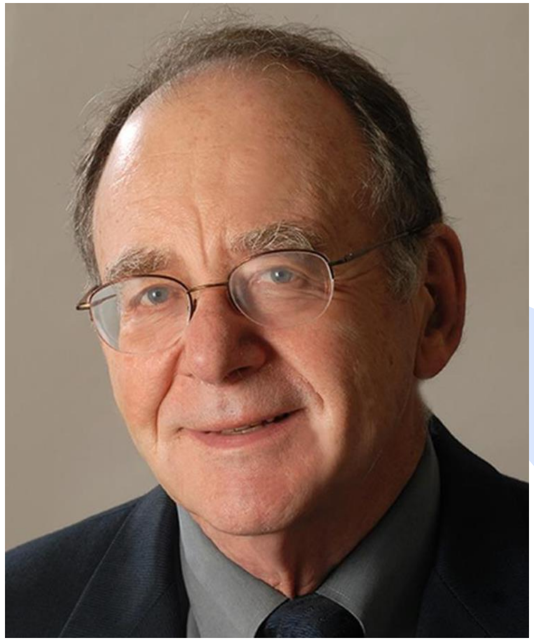
Ford-Fulkerson方法

- 增广路:残存网络G_f中一条从*s*到*t*的路径。增广路上可增加的最大流量为该路径上各边残存容量的最小值。
- 基本思想：在残存网络中不断寻找增广路，使网络总流量增加，直至残留网络找不到增广路。
- 最大流最小割定理：当在残存网络中找不到增广路时，获得最大流。

Edmonds-Karp算法



Jack Edmonds
滑铁卢大学教授



Richard Karp
加州大学伯克利分校教授
图灵奖获得者
美国科学院院士
美国工程院院士
欧洲科学院院士

Edmonds-Karp算法

是Ford–Fulkerson方法的具体实现，在残存网络中通过BFS寻找增广路，时间复杂度O(ne^2)。

```
int BFS(int G[N][N], int pre[], int n, int s, int t){/*执行BFS找到一条路，返回路径的流量(最小边权),G记录图和残存网络,pre[i]表示路径上i的前驱*/
    for(int i=1; i<=n; i++) pre[i]=-1;
    int flow[N]={0};
    flow[s] = INF; Queue<int> q; q.enQueue(s);
    while(!Q.Empty()){
        int u = q.deQueue();
        if(u == t) break;          //找到一条s到t的增广路，结束
        for(int v=1; v<=n; v++){   //考察u的邻接顶点
            if(v!=s && G[u][v]>0 && pre[v]==-1){
                pre[v]=u; q.enQueue(v);
                flow[v]=min(flow[u],G[u][v]);
            }
        }
    }
    if(pre[t]==-1) return -1;     //未找到s到t的增广路
    return flow[t];
}
```

Edmonds-Karp算法

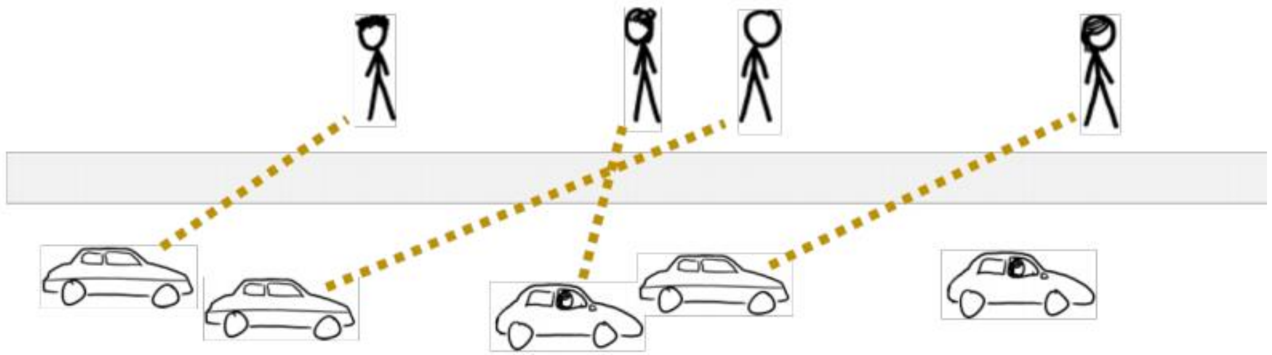
```
int Edmonds_Karp(int G[N][N], int n, int s, int t){
   int pre[N], Maxflow = 0;
   while(true){
     int flow=BFS(G,pre,n,s,t); //执行BFS找到一条增广路,返回路径的流量
     if(flow == -1) return Maxflow;      //没有找到新的增广路，结束
     int v = t;                     //更新路径上的残存网络
     while(v!=s){                   //一直沿路径回溯到起点
        int u = pre[v];             //pre[]记录路径上的前一个点
        G[u][v] -= flow;            //更新残存网络：正向边减flow
        G[v][u] += flow;            //更新残存网络：反向边加flow
        v = u;
     }
     Maxflow += flow;
   }
}
```

最小费用最大流问题

- 每条边除了容量之外，再增加一个限制：费用。
- 目标：求最小费用的最大流，即在满足流量最大的前提下，希望流的费用最小。
- 每次在残存网络上选增广路时，采用最短路径算法，选费用最小的增广路。

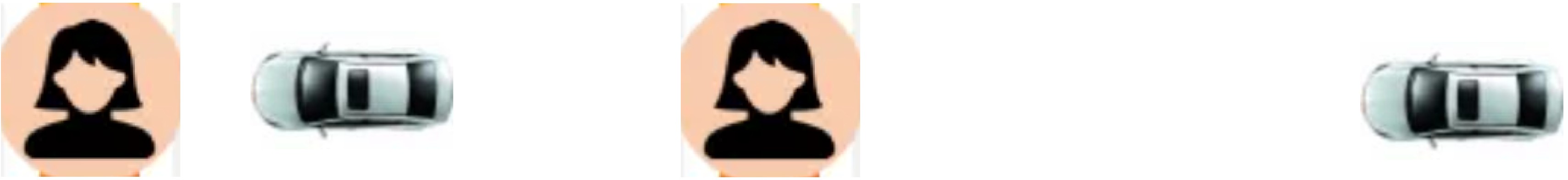
打车软件中的派单算法

- 订单分配：在派单系统中将乘客的订单分配给在线司机。
- 滴滴快车：采用全局最优的派单模式，通过搜索1.5-2秒内所有可能的司机乘客匹配，由算法综合考虑接驾距离、道路拥堵情况等因素，自动将订单匹配给最合适的司机接单，让全局乘客接驾时间最短。



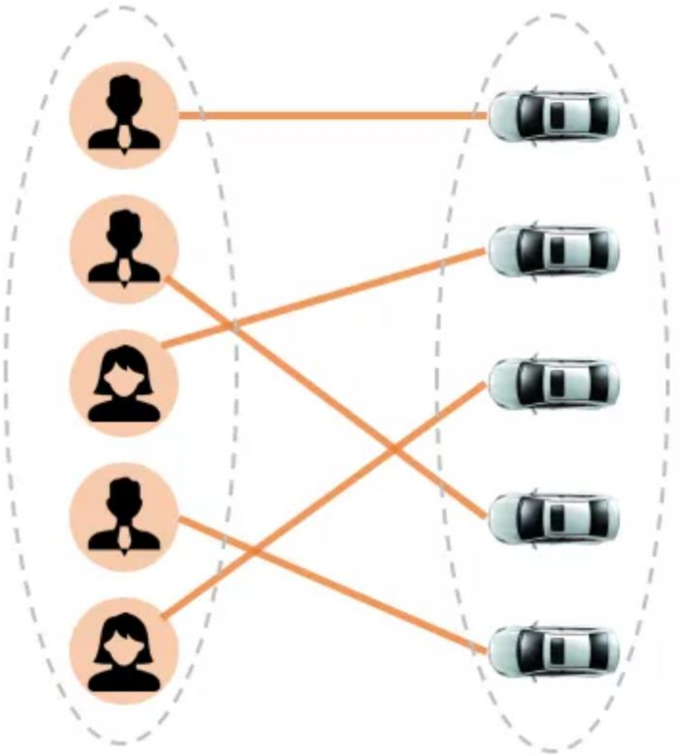
打车软件中的派单算法

- 订单分配：在派单系统中将乘客的订单分配给在线司机。
- 滴滴快车：采用全局最优的派单模式，通过搜索1.5-2秒内所有可能的司机乘客匹配，由算法综合考虑接驾距离、道路拥堵情况等因素，自动将订单匹配给最合适的司机接单，让全局乘客接驾时间最短。



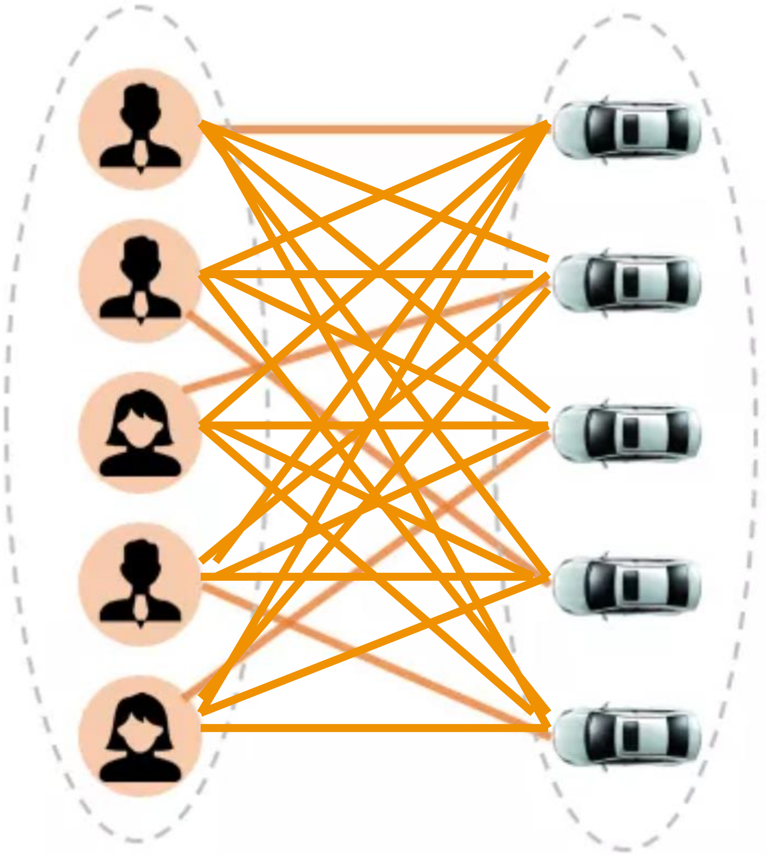
打车软件中的派单算法

- 暴力方案：O(n!)
- 滴滴数据：高峰期每分钟接收超过6万乘车需求，平均每2秒就需要匹配几百到上千的乘客和司机。



带权二分图的最大匹配

- 二分图
- 任意乘客和车之间都有边，边的权值为乘客与车的距离或时间



带权二分图的最大匹配

- 可转换为最小费用最大流问题
- 费用：乘客与车的距离或等待时间
- 改成有向图，边容量：1
- 美团外卖