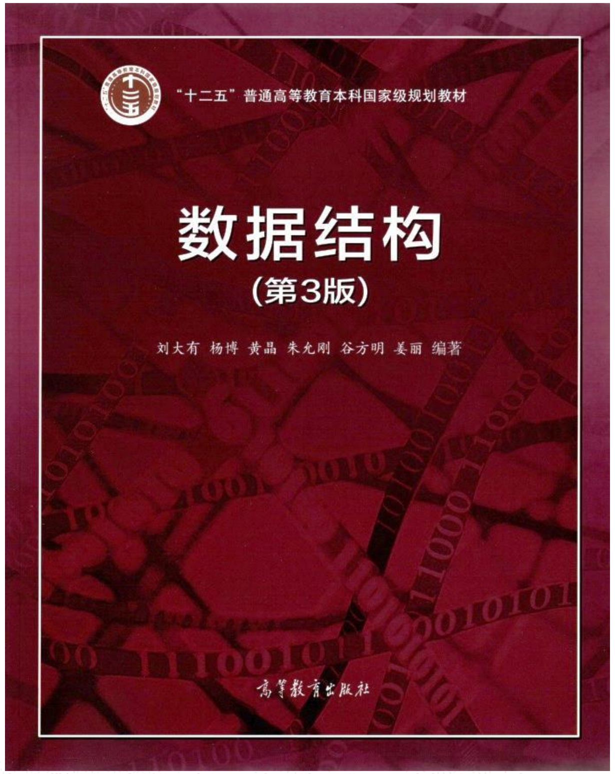
think.create.solve



次小支撑树

- **回顾Prim算法**
- 最小支撑树中的最大边
- 次小支撑树

数据之法
结构之美
算法之道

Last updated on 2023.1

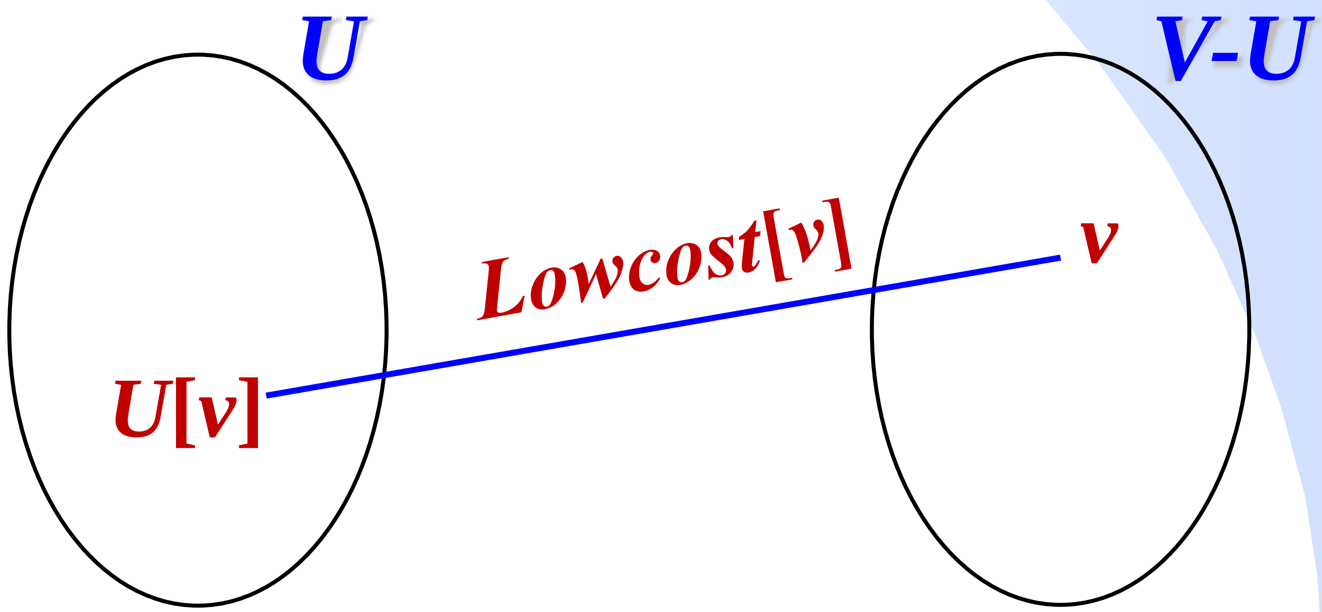
zhuyungang@jlu.edu.cn

Prim算法的实现需要两个辅助数组*Lowcost*[*v*]和 *U*[*v*]，标识顶点v到集合U的最小跨边的信息。定义如下：

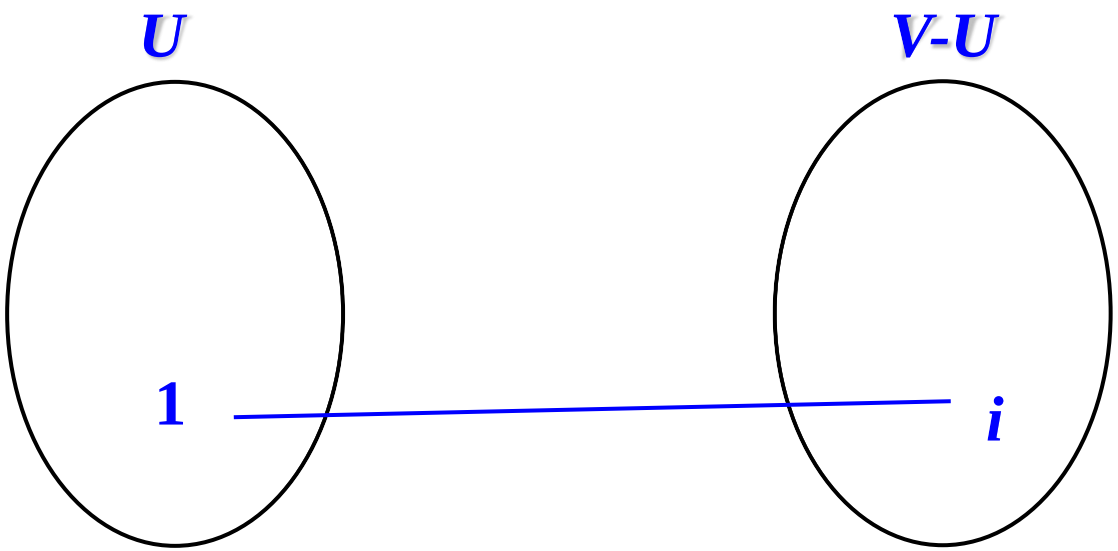
- 若$v \in V-U$，则 $Lowcost[v]=\min\{weight(u, v) \mid u \in U\}$,
  $U[v] = u , u \in U$.
- 若$v \in U$，则$Lowcost[v] = 0$，$U[v] = -1$

✓ U[*v*]=-1表示*v*在集合U中

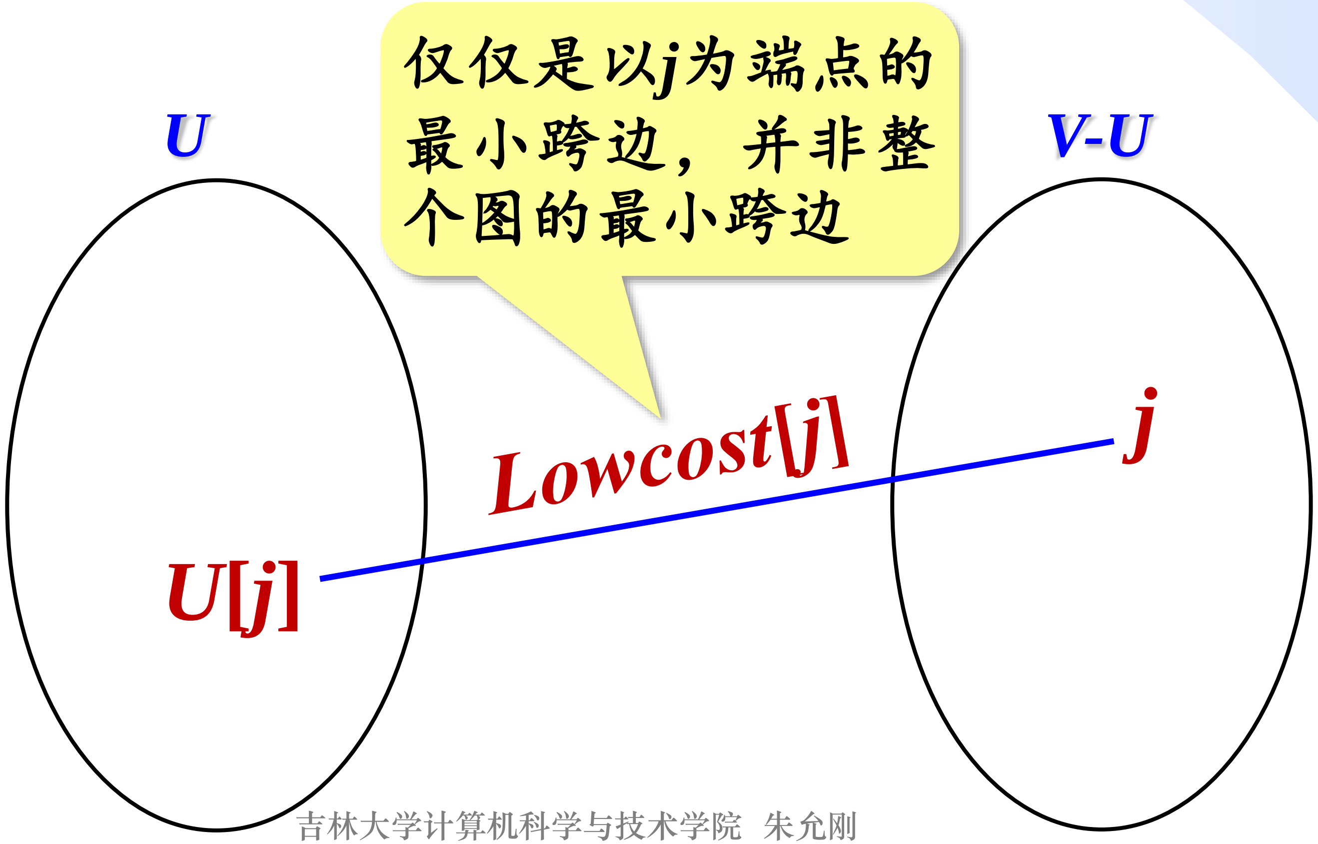
✓ U[*v*]>0表示*v*不在集合U中，U[*v*]的值是*v*到U的最小跨边的端点



```
int Prim(int G[][N],int n,int TE[]){//以点1为起点构造MST,返回权值和
  int Lowcost[N],U[N],ans=0; //ans为mst边权之和
  for(int i=1;i<=n;i++) {Lowcost[i]=G[1][i]; U[i]=1;}
  U[1]=-1;                         //顶点1方入集合U
```

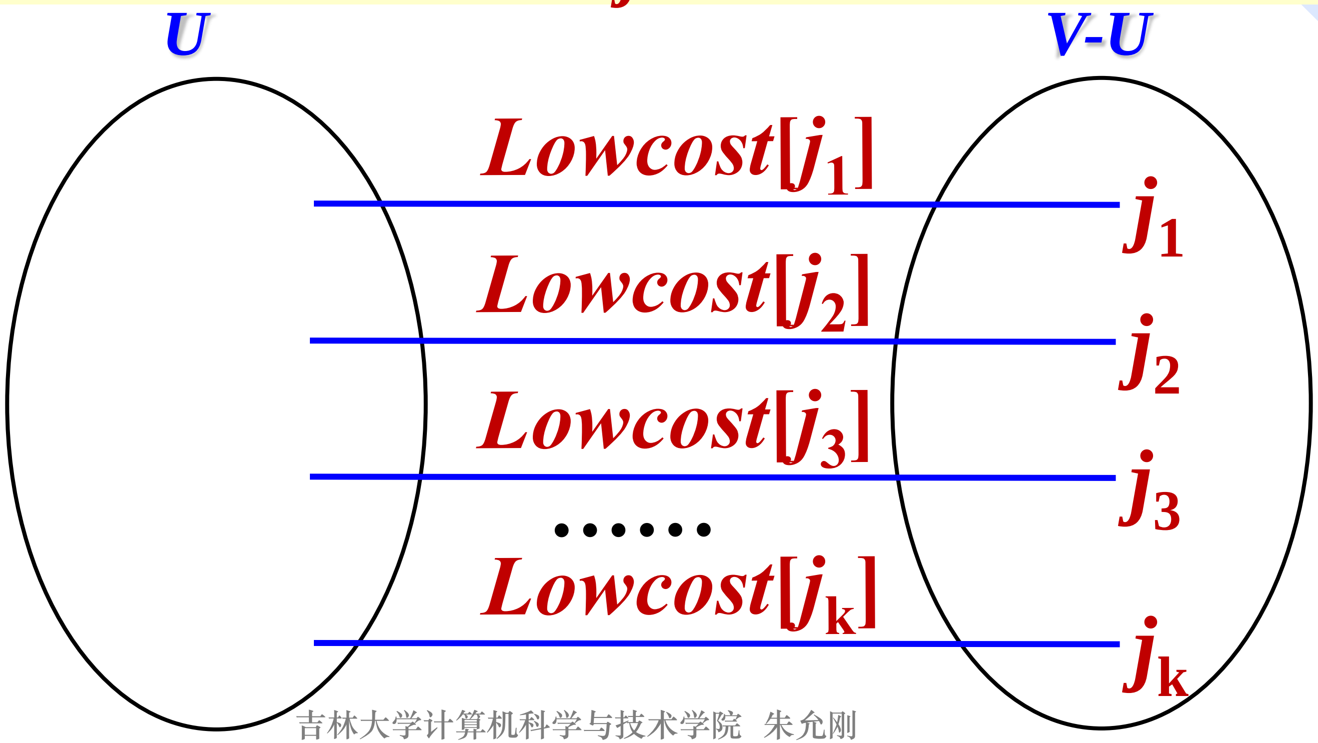


```
int Prim(int G[][N],int n,int TE[]){//以点1为起点构造MST,返回权值和
  int Lowcost[N],U[N],ans=0; //ans为mst边权之和
  for(int i=1;i<=n;i++) {Lowcost[i]=G[1][i]; U[i]=1;}
  U[1]=-1;                         //顶点1方入集合U
  for(int i=1;i<n;i++){            //找n-1条边，循环n-1次
                                   //找最小跨边
```



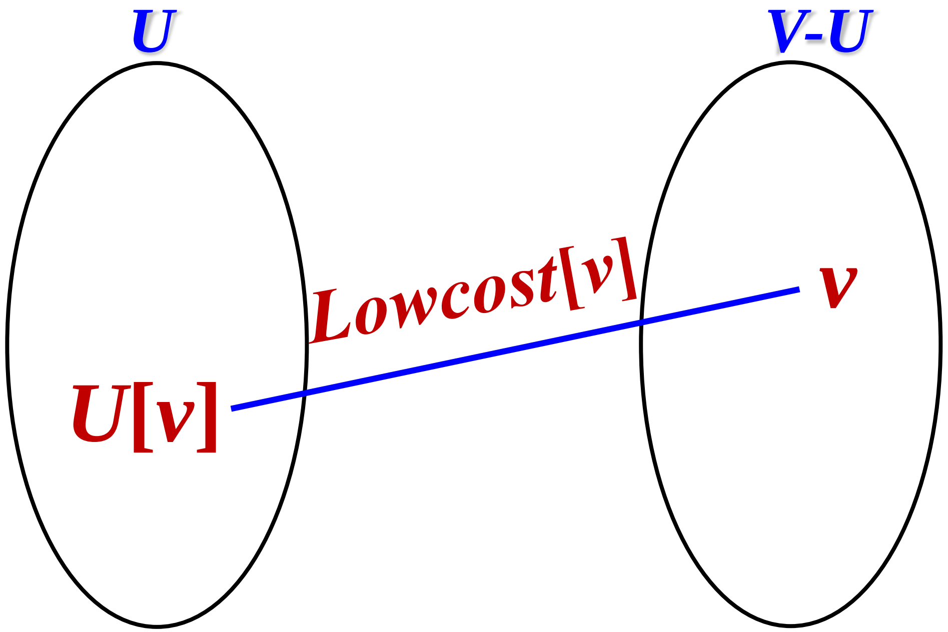
```
int Prim(int G[][N],int n,int TE[]){//以点1为起点构造MST,返回权值和
  int Lowcost[N],U[N],ans=0; //ans为mst边权之和
  for(int i=1;i<=n;i++) {Lowcost[i]=G[1][i]; U[i]=1;}
  U[1]=-1;                    //顶点1方入集合U
  for(int i=1;i<n;i++){       //找n-1条边，循环n-1次
                              //找最小跨边
```

整个图的最小跨边：$\min_{j\in V-U}\{Lowcost[j]\}$



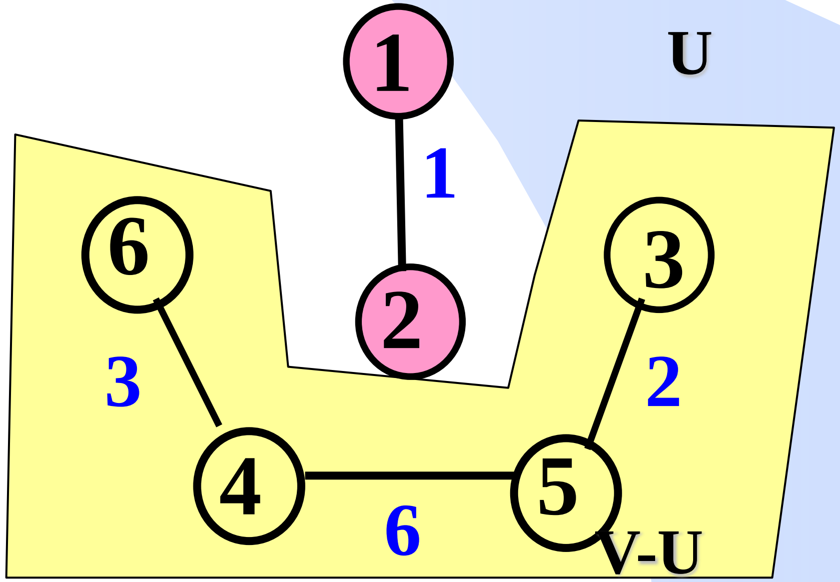
```
int Prim(int G[][N],int n,int TE[]){//以点1为起点构造MST,返回权值和
  int Lowcost[N],U[N],ans=0; //ans为mst边权之和
  for(int i=1;i<=n;i++) {Lowcost[i]=G[1][i]; U[i]=1;}
  U[1]=-1;                     //顶点1方入集合U
  for(int i=1;i<n;i++){        //找n-1条边，循环n-1次
     int v=SelectMin(Lowcost,U,n);    //找最小跨边
```

整个图的最小跨边：$\min_{j \in V-U}\{Lowcost[j]\}$

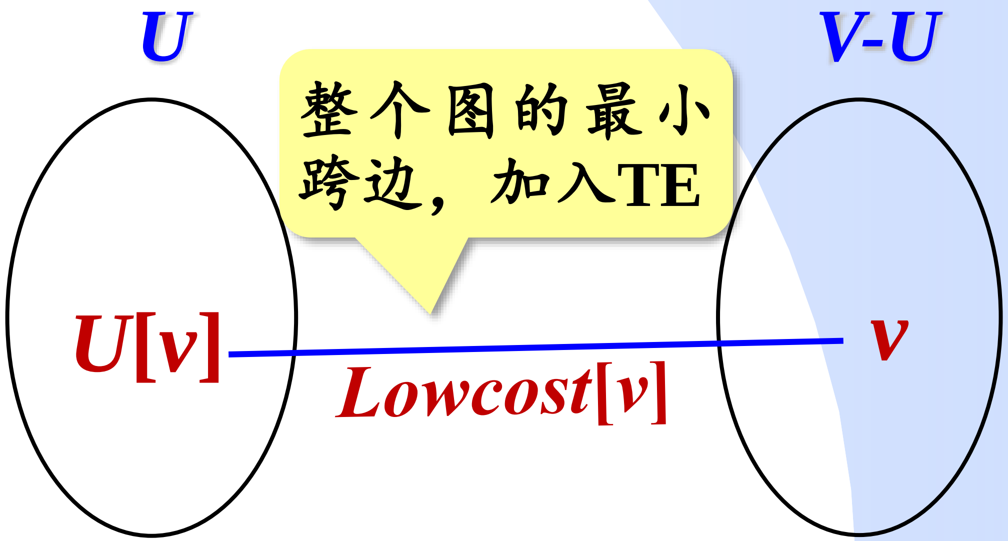


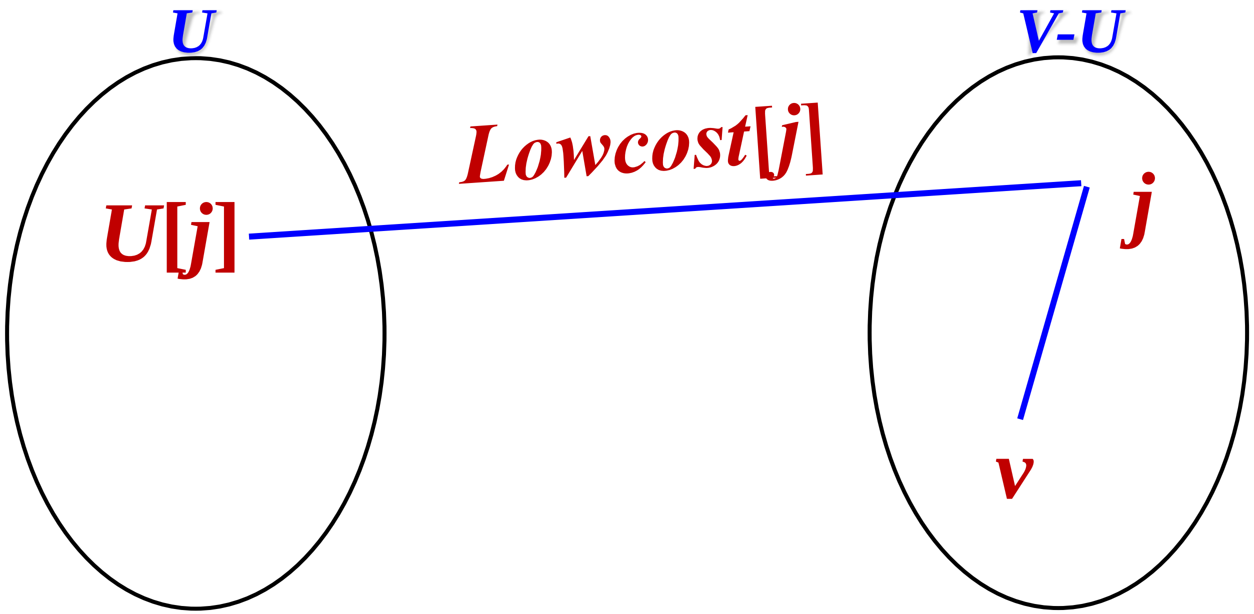

```
int SelectMin(int Lowcost[],int U[],int n){
   int v=0, min=INF;
   for(int j=1; j<=n; j++)
      if(U[j]!=-1 && Lowcost[j]<min){
         min=Lowcost[j];
         v=j;
      }
   return v;
}
```

```
int Prim(int G[][N],int n,int TE[]){//以点1为起点构造MST,返回权值和
  int Lowcost[N],U[N],ans=0; //ans为mst边权之和
  for(int i=1;i<=n;i++) {Lowcost[i]=G[1][i]; U[i]=1;}
  U[1]=-1;                      //顶点1方入集合U
  for(int i=1;i<n;i++){         //找n-1条边，循环n-1次
     int v=SelectMin(Lowcost,U,n); //找最小跨边
     if(v==0) return INF;          //不存在跨边，图不连通
```

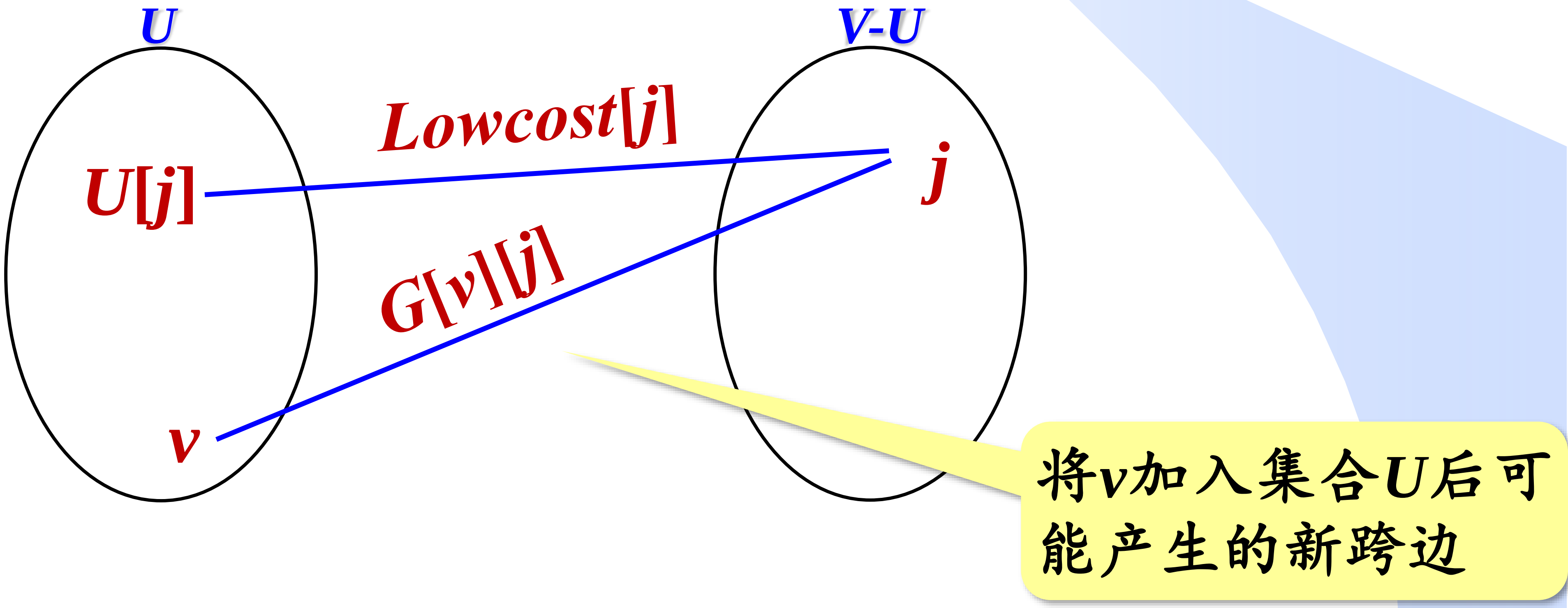


```
int Prim(int G[][N],int n,int TE[]){//以点1为起点构造MST,返回权值和
  int Lowcost[N],U[N],ans=0; //ans为mst边权之和
  for(int i=1;i<=n;i++) {Lowcost[i]=G[1][i]; U[i]=1;}
  U[1]=-1;                   //顶点1方入集合U
  for(int i=1;i<n;i++){      //找n-1条边，循环n-1次
     int v=SelectMin(Lowcost,U,n); //找最小跨边
     if(v==0) return INF;          //不存在跨边，图不连通
     TE[i].head=U[v]; TE[i].tail=v; TE[i].cost=Lowcost[v];
     ans+=Lowcost[v]; U[v]=-1;   //累加边权，v加入U中
```




U
V-U
Lowcost[j]
U[j]
j
v

```
//更新v在V-U中邻接顶点的Lowcost和U值
for(int j=1; j<=n; j++)
   if(U[j]!=-1 && G[v][j]<Lowcost[j]){
       Lowcost[j]=G[v][j];
       U[j]=v;
   }
```

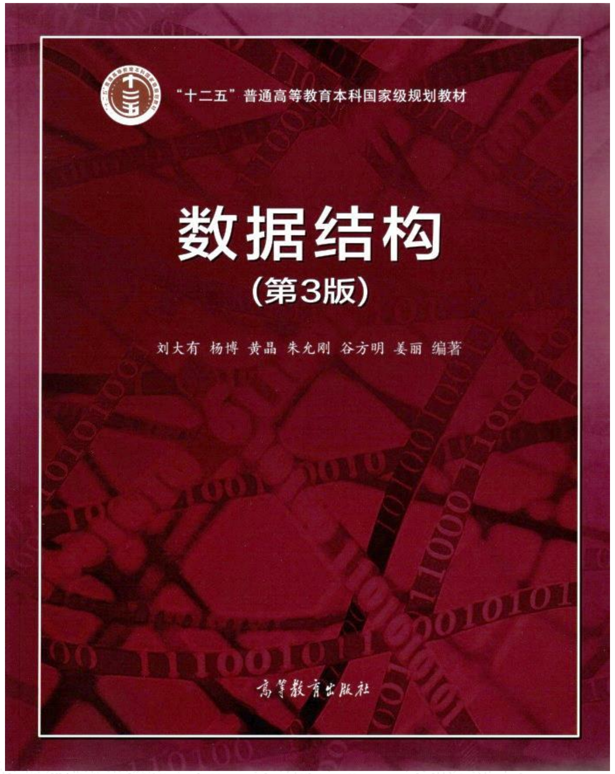


```
int Prim(int G[][N],int n,int TE[]){//以点1为起点构造MST,返回权值和
  int Lowcost[N],U[N],ans=0;         //ans为mst边权之和
  for(int i=1;i<=n;i++) {Lowcost[i]=G[1][i]; U[i]=1;}
  U[1]=-1;                           //顶点1方入集合U
  for(int i=1;i<n;i++){              //找n-1条边，循环n-1次
     int v=SelectMin(Lowcost,U,n);//找最小跨边
     if(v==0) return INF;            //不存在跨边，图不连通
     TE[i].head=U[v]; TE[i].tail=v; TE[i].cost=Lowcost[v];
     ans+=Lowcost[v]; U[v]=-1;       //累加边权，v加入U中
     for(int j=1; j<=n; j++)         //更新v邻接顶点的Lowcost和U值
        if(U[j]!=-1 && G[v][j]<Lowcost[j]){
            Lowcost[j]=G[v][j]; U[j]=v;
        }
  }
  return ans;
}
```

时间复杂度$O(n^2)$

若存在MST, 返回MST边权之和,否则（图不连通）返回∞

think.create.solve



次小支撑树

- 回顾Prim算法
- **最小支撑树中的最大边**
- 次小支撑树

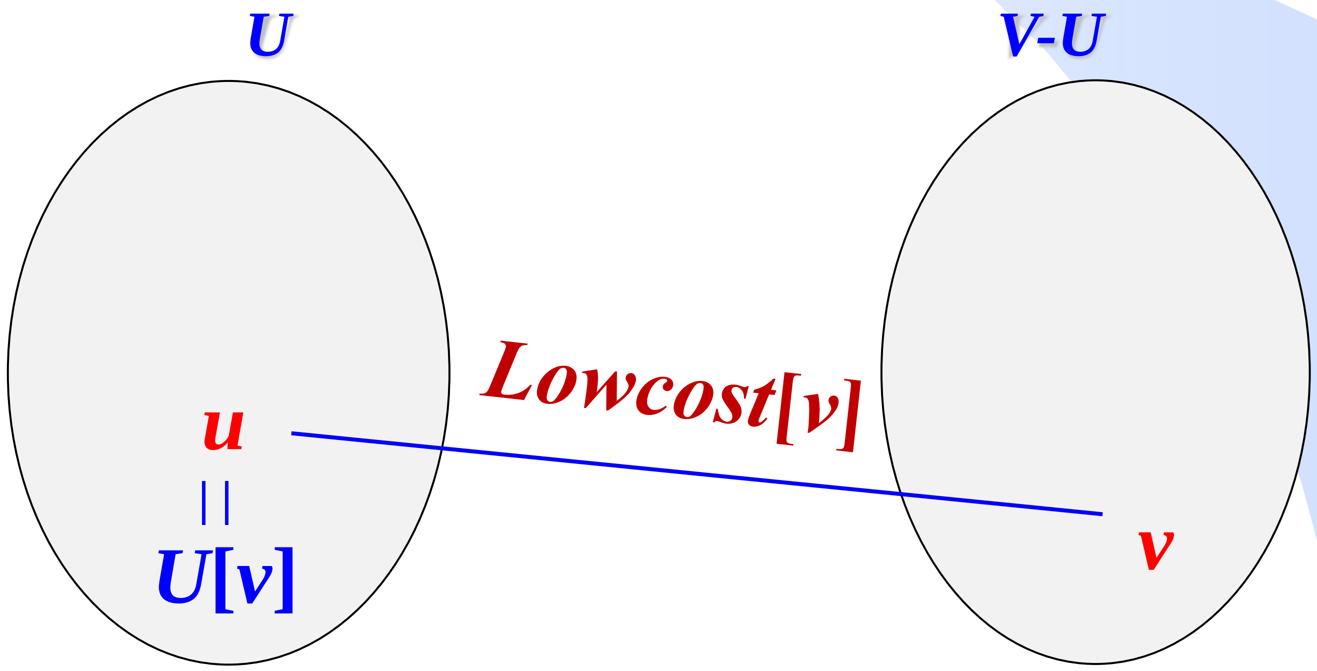
数据之法
结构之美
算法之道

Last updated on 2023.1

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U时，计算v到U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U时，计算v到U中各点间路径的最大边：

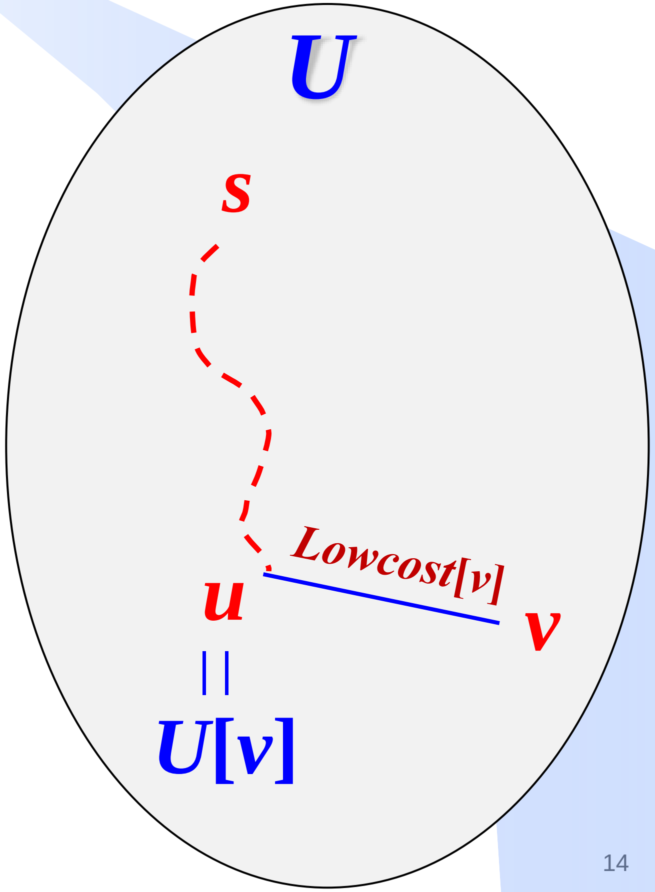
对于u=U[v]，有：

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

对于集合U中的其他点s，有：

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

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



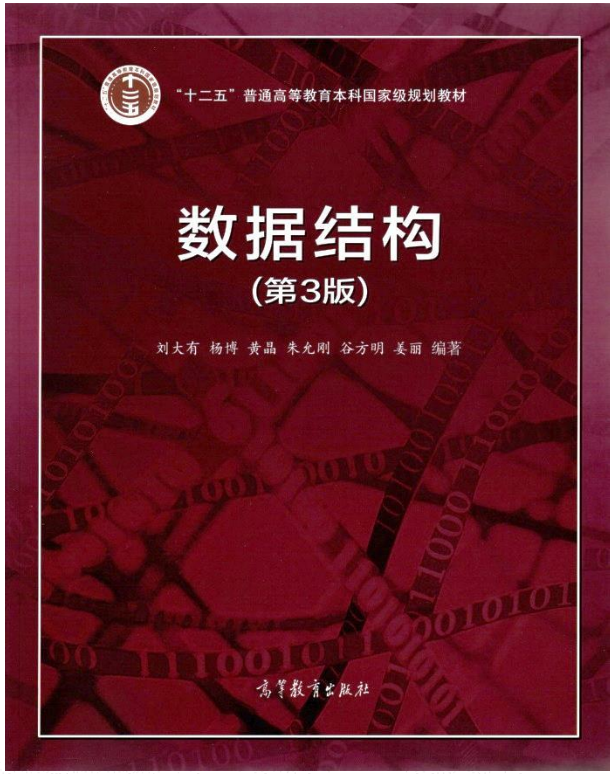
```
int Prim(int G[][N],int n,int TE[]){//以点1为起点构造MST,返回权值和
  int Lowcost[N],U[N],ans=0;        //ans为mst边权之和
  for(int i=1;i<=n;i++) {Lowcost[i]=G[1][i]; U[i]=1;}
  U[1]=-1;                          //顶点1方入集合U
  for(int i=1;i<n;i++){             //找n-1条边，循环n-1次
      int v=SelectMin(Lowcost,U,n);//找最小跨边
      if(v==0) return INF;          //不存在跨边，图不连通
      TE[i].head=U[v]; TE[i].tail=v; TE[i].cost=Lowcost[v];
      ans+=Lowcost[v];              //累加边权
```

对于maxcost的处理

```
      U[v]=-1;                      //v加入U中
      for(int j=1; j<=n; j++)       //更新v邻接顶点的Lowcost和U值
         if(U[j]!=-1 && G[v][j]<Lowcost[j]){
               Lowcost[j]=G[v][j];   U[j]=v;
         }
  }
  return ans;
}
```

时间复杂度$O(n^2)$

think.create.solve



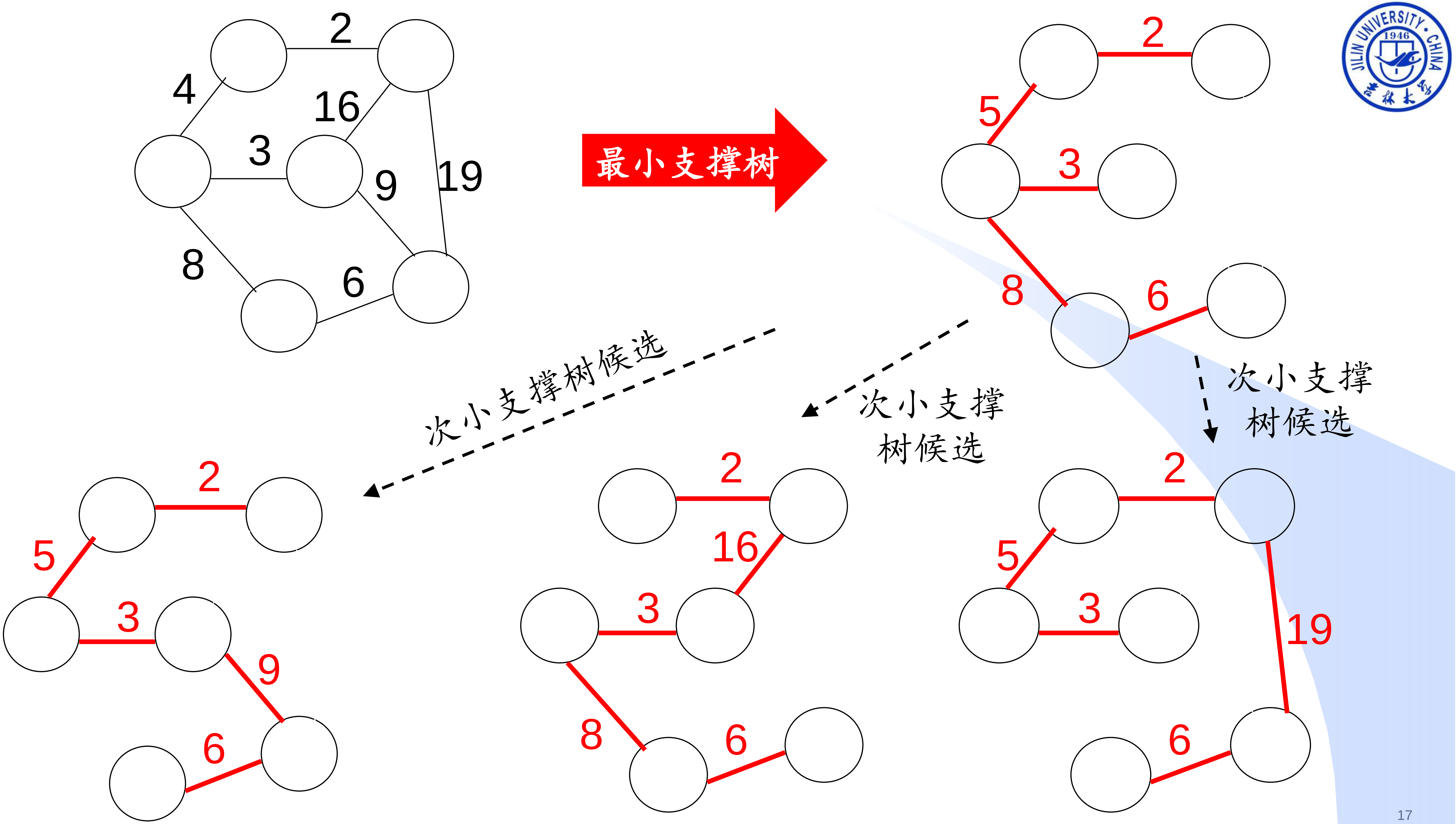
次小支撑树

- 回顾Prim算法
- 最小支撑树中的最大边
- **次小支撑树**

数据之法
结构之美
算法之道

Last updated on 2023.1

zhuyungang@jlu.edu.cn



2
4
16
3
9
19
8
6
最小支撑树
2
5
3
8
6
次小支撑树候选
次小支撑
树候选
次小支撑
树候选
2
5
3
9
6
2
16
3
8
6
2
5
3
19
6

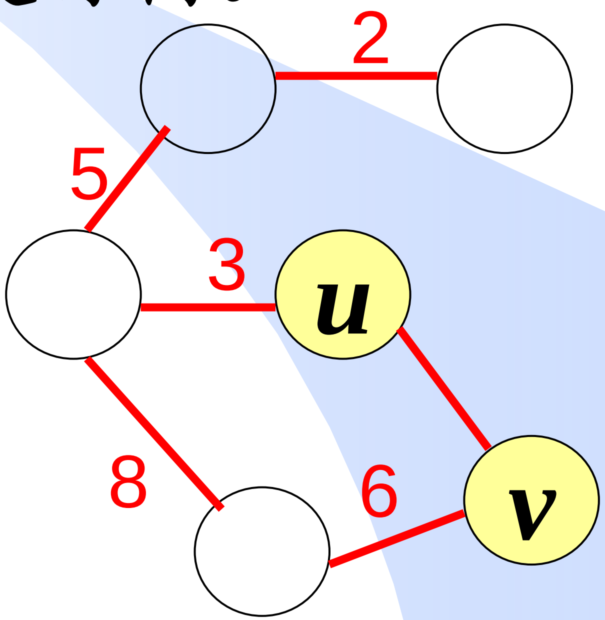
求图G的次小支撑树算法

（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)