7.5 生成树和最小生成树

7.5.1 生成树和最小生成树的概念

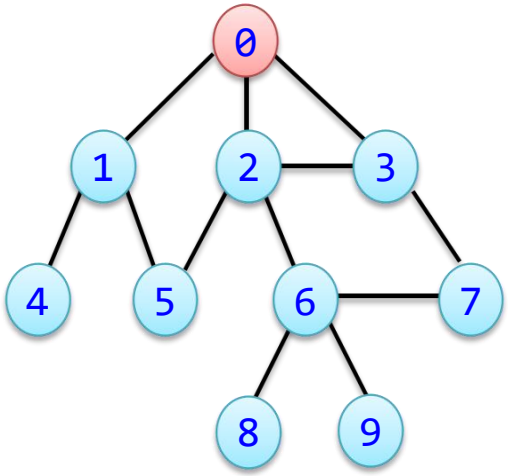
1．什么是生成树

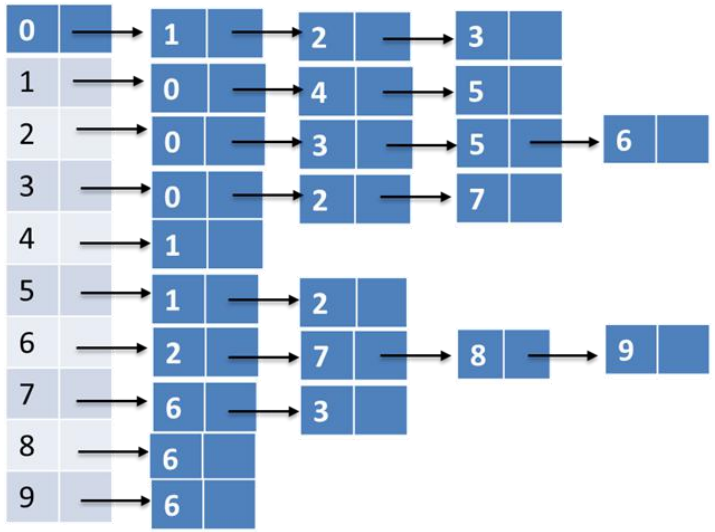
- 树是一个有|V|(用n表示)个顶点，n-1条边的极小连通图。
- 生成树包含图中全部顶点，但只包含其中的*n*-1条边，图的极小连通子图。
- 如果在一棵生成树上添加一条边，必定构成一个环。

2．由两种遍历方法产生的生成树

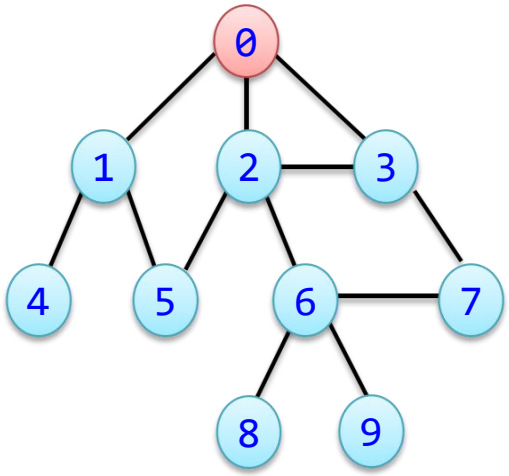
- 由深度优先遍历得到的生成树称为深度优先生成树。
- 由广度优先遍历得到的生成树称为广度优先生成树。

【例7.10】对于如下的无向图，画出其邻接表存储结构，并在该邻接表中，以顶点0为根，画出图G的深度优先生成树和广度优先生成树。





假定邻接表中单链表按顶点编号递增排列！



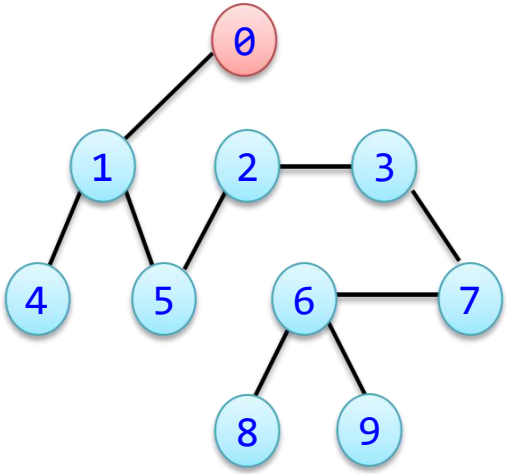
DFS(0)

0 → 1 → 4

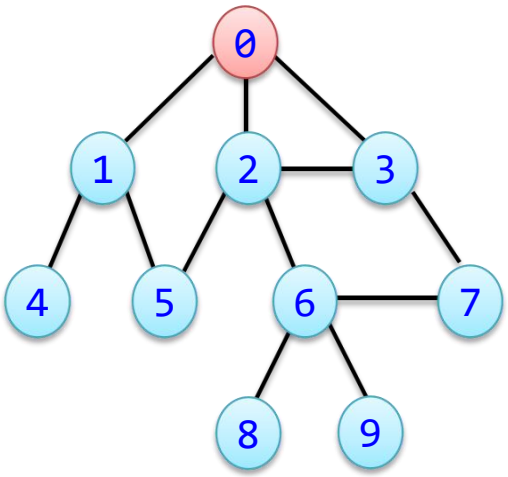
5 → 2 → 3 → 7 → 6 → 8

9

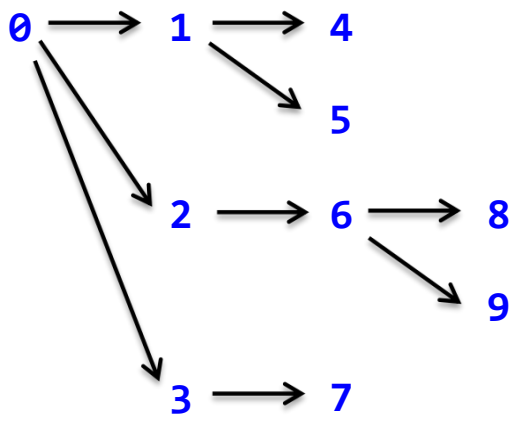
DFS树



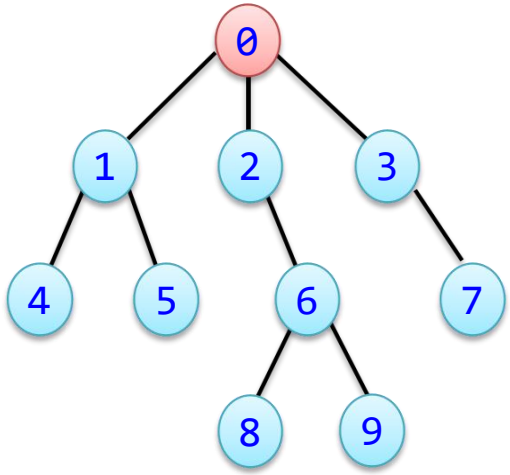
邻接表中单链表按顶点编号递增排列！



BFS(0)



BFS树



- 连通图：仅需调用遍历过程（DFS或BFS）一次，从图中任一顶点出发，便可以遍历图中的各个顶点，产生相应的生成树。
- 非连通图：需多次调用遍历过程。每个连通分量中的顶点集和遍历时走过的边一起构成一棵生成树。所有连通分量的生成树组成非连通图的生成森林。

3．什么是最小生成树

- 一个带权连通图G（假定每条边上的权值均大于零）可能有多棵生成树.
- 每棵生成树中所有边上的权值之和可能不同.
- 其中边上的权值之和最小的生成树称为图的最小生成树。

重点：求带权连通图的最小生成树

普里姆算法—让一棵小树长大

普里姆（Prim）算法是一种构造性算法，是一种贪心算法。

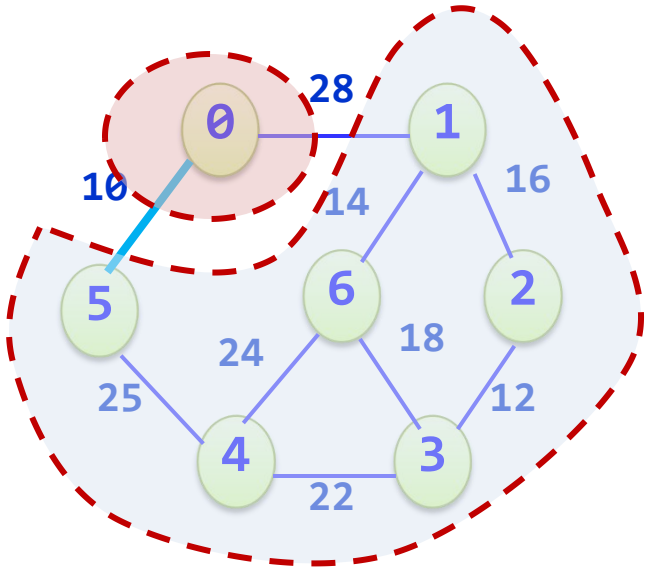
贪心算法，也称为贪婪算法：每一步都选择当前最好的，即局部最优。

好：权重最小的边

需要约束：

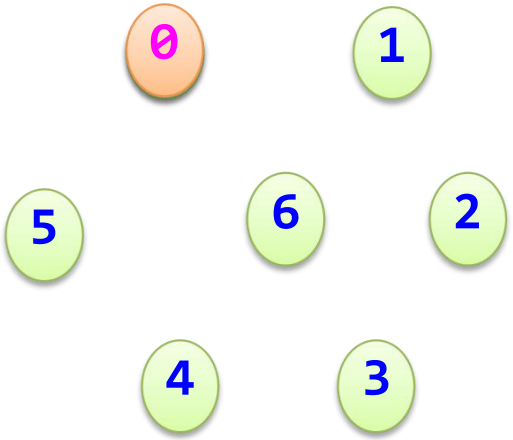
- 只能用图中有的边；
- 只能正好用掉n-1条边；
- 不能有回路

Prim算法示例演示（起点0）



图G

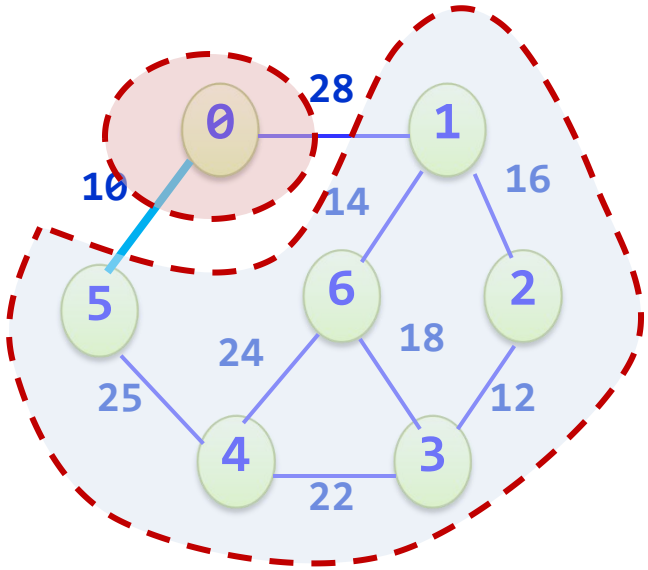
让一棵小树逐步长大



U={0 }

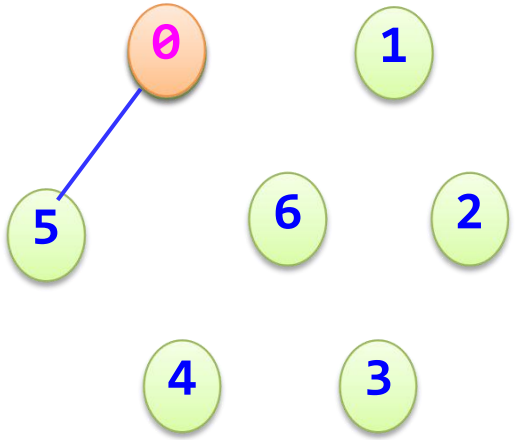
普里姆算法求解最小生成树的过程

Prim算法示例演示（起点0）



图G

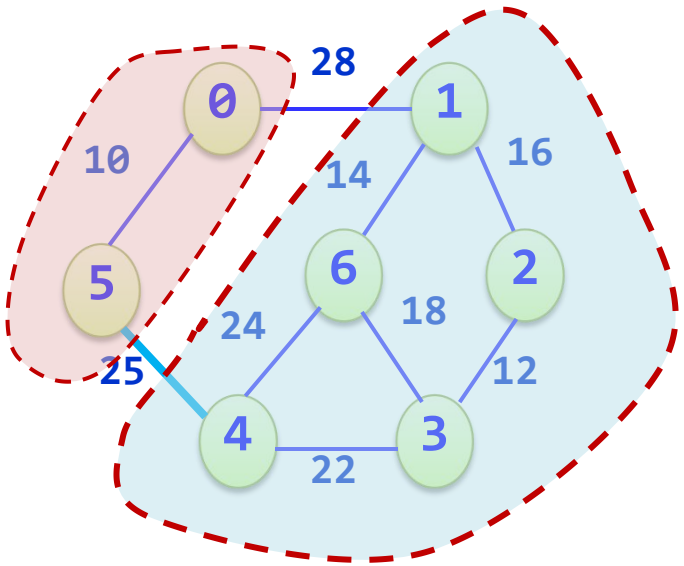
让一棵小树逐步长大



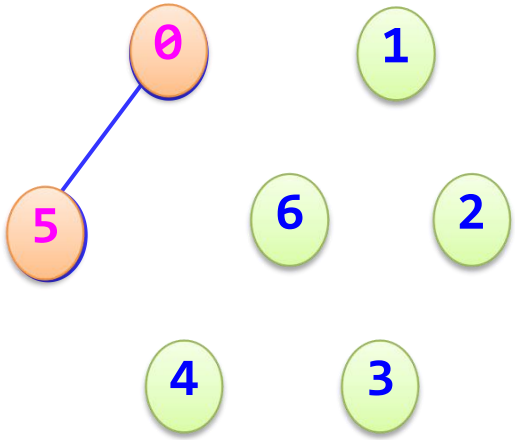
U={0，5}

普里姆算法求解最小生成树的过程

Prim算法示例演示（起点0）



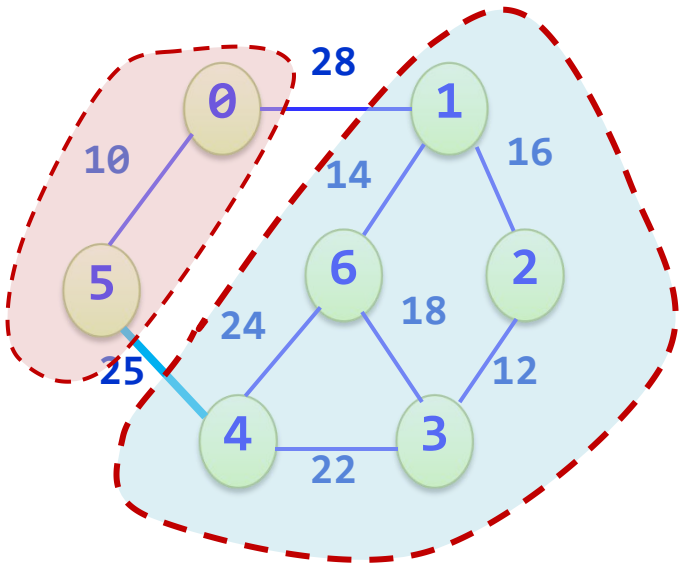
图G



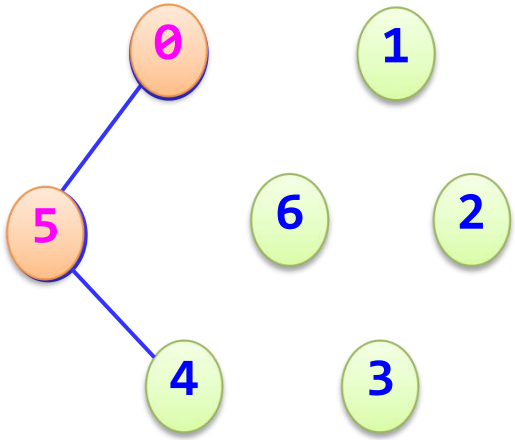
U={0，5}

普里姆算法求解最小生成树的过程

Prim算法示例演示（起点0）



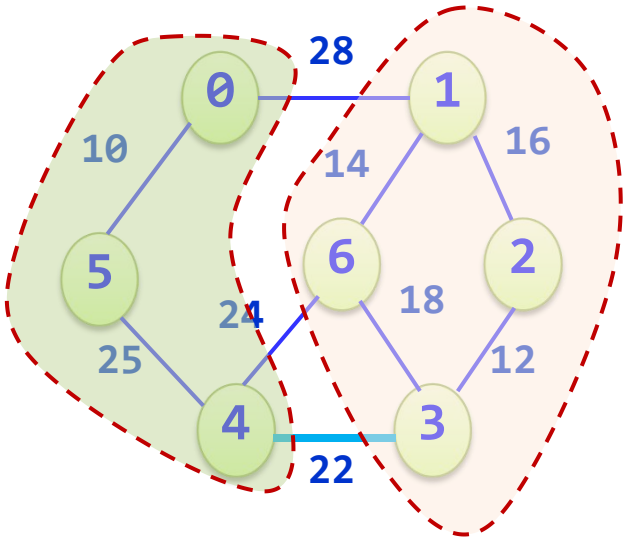
图G



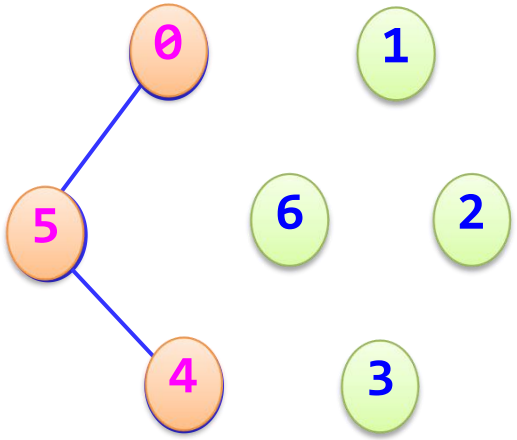
U={0，5，4}

普里姆算法求解最小生成树的过程

Prim算法示例演示（起点0）



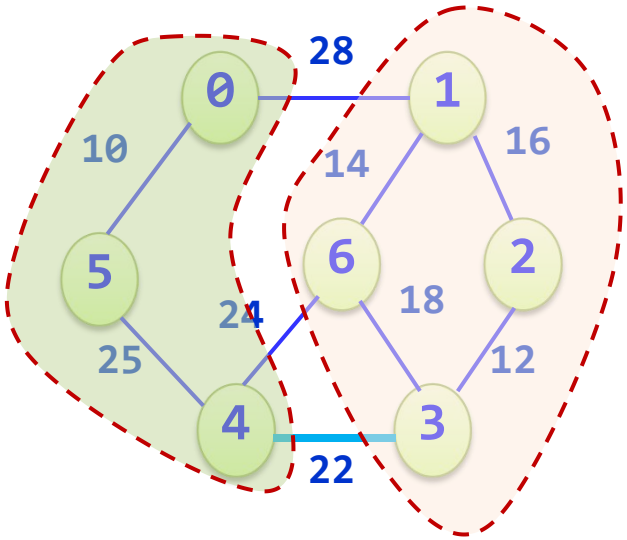
图G



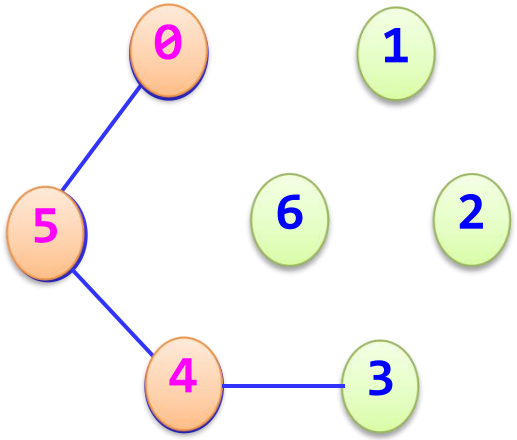
U={0，5，4}

普里姆算法求解最小生成树的过程

Prim算法示例演示（起点0）



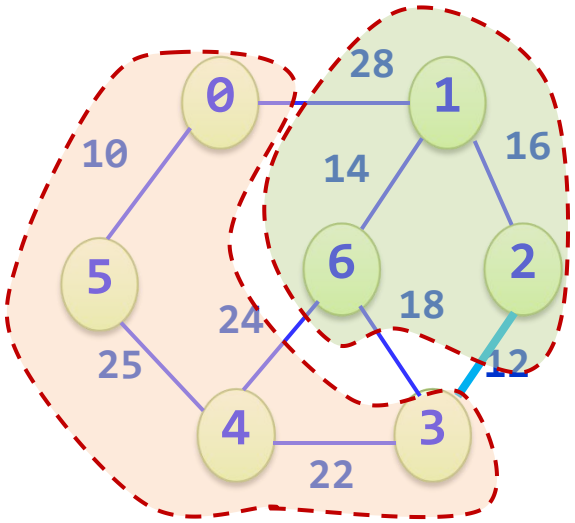
图G



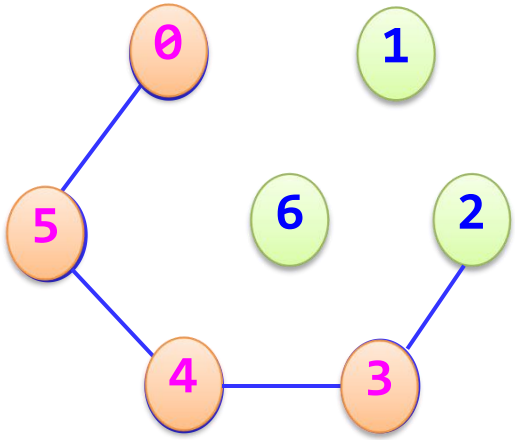
U={0，5，4，3}

普里姆算法求解最小生成树的过程

Prim算法示例演示（起点0）



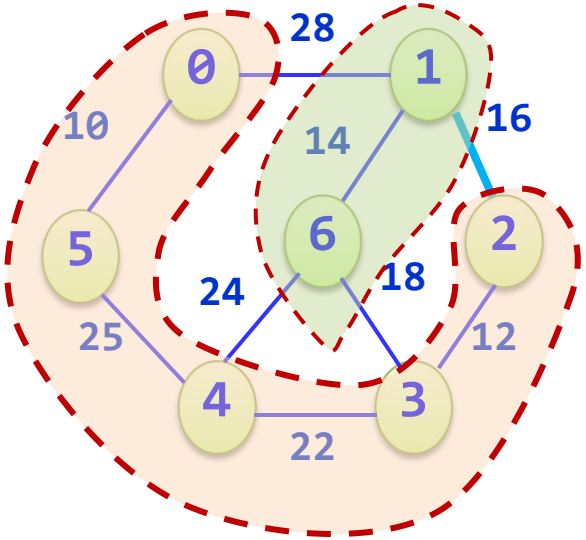
图G



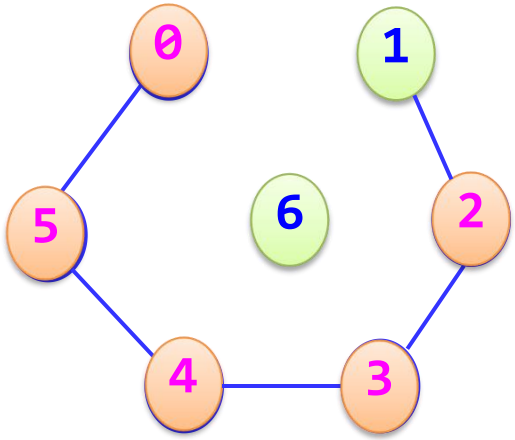
U={0，5，4，3，2}

普里姆算法求解最小生成树的过程

Prim算法示例演示（起点0）



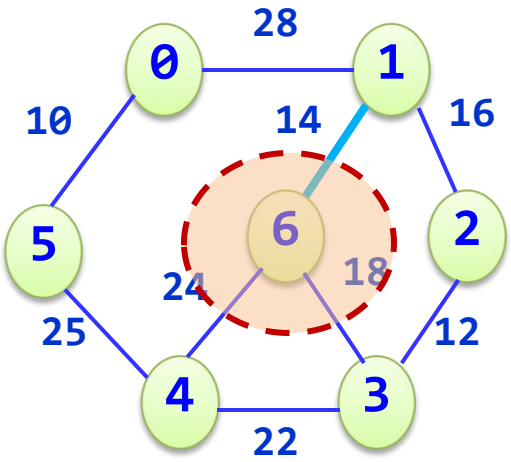
图G



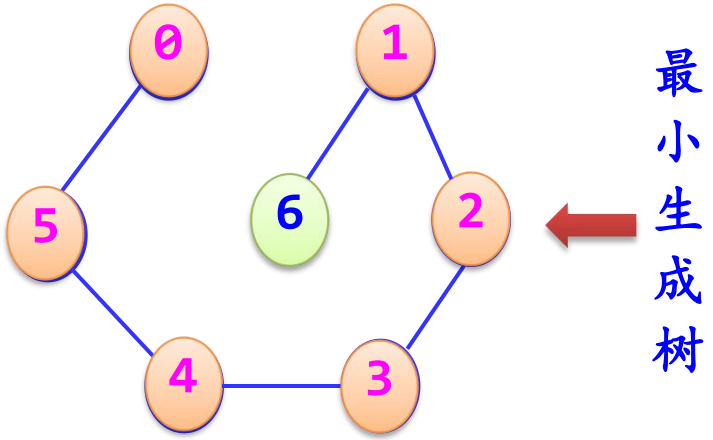
U={0，5，4，3，2，1}

普里姆算法求解最小生成树的过程

Prim算法示例演示（起点0）



图G



U={0，5，4，3，2，1，6}

普里姆算法求解最小生成树的过程

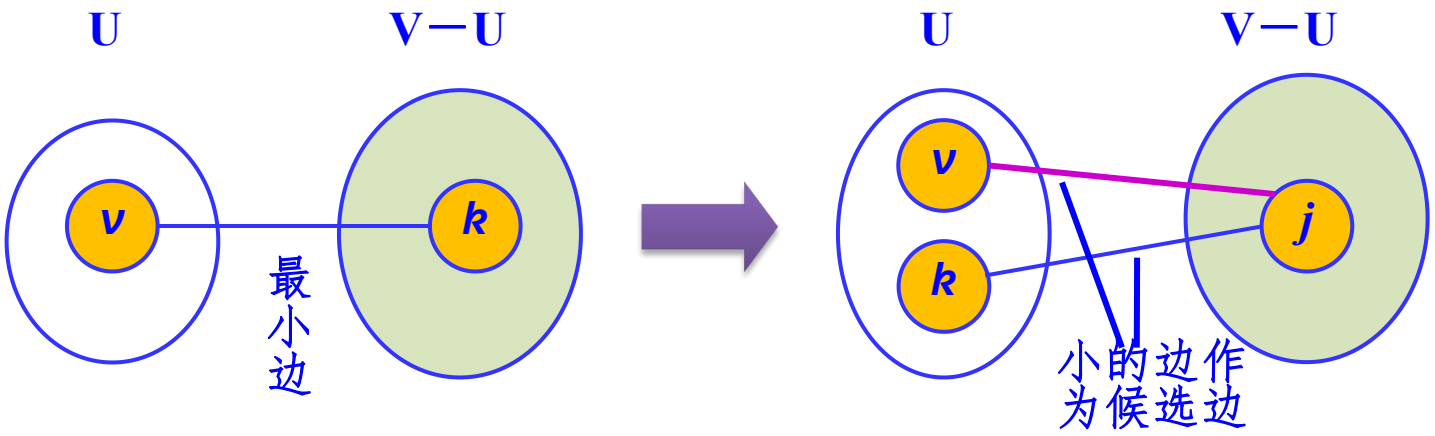
Prim算法过程如下：

（1）初始化U={*v*}。*v*到其他顶点的所有边为候选边；

（2）重复以下步骤*n*-1次，使得其他*n*-1个顶点被加入到U中：

❶ 从候选边中挑选权值最小的边输出，设该边在V-U中的顶点是*k*，将*k*加入U中；

❷ 考察当前V-U中的所有顶点*j*，修改候选边：若(*j*，*k*)的权值小于原来和顶点*j*关联的候选边，则用(*k*，*j*)取代后者作为候选边。

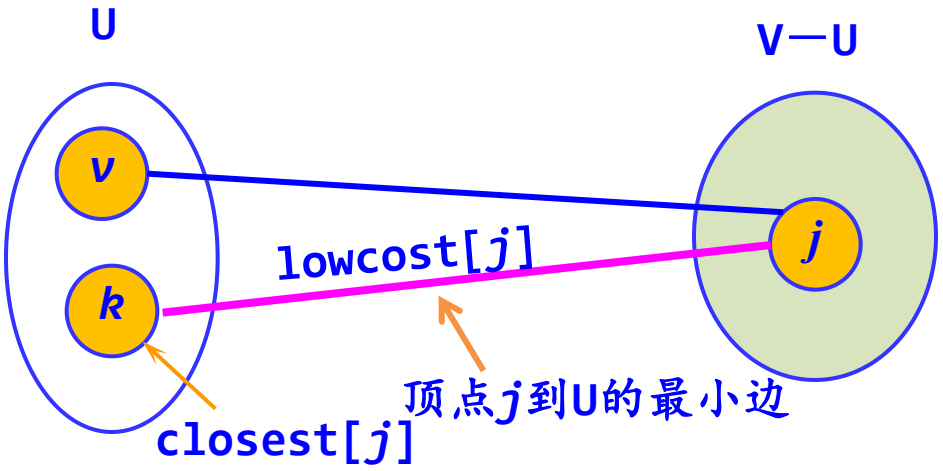


算法设计（解决4个问题）：

- 如何求U、V-U两个顶点集之间的最小边？（只求一条）

  只考虑V-U中顶点j到U顶点集的最小边（无向图），比较来找最小边

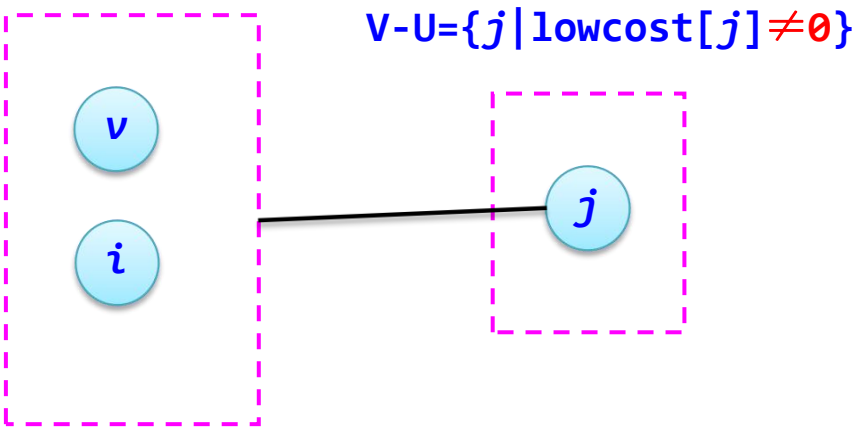
- 如何存储顶点j到U顶点集的最小边？



- 一个顶点属于哪个集合？
- 图采用哪种存储结构更合适？ 邻接矩阵

➢ 对于任意顶点i，如何知道它属于集合U还是集合V-U呢？

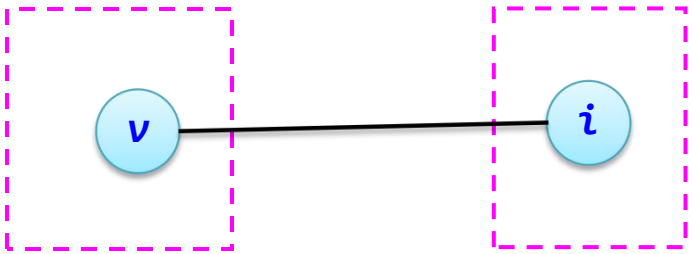
U={i|lowcost[i]=0}



初始化

初始时，U中只有一个顶点v，其他顶点i均在V-U中。

U={i|lowcost[i]=0}　V-U={j|lowcost[j]≠0}



- 如果（v，i）有一条边，它就是i到U的最小边，置closest[i]=v，lowcost[i]=g.edges[v][i]。
- 如果（v，i）没有边，不妨认为有一条权为∞的边，同样置closest[i]=v，lowcost[i]=g.edges[v][i]

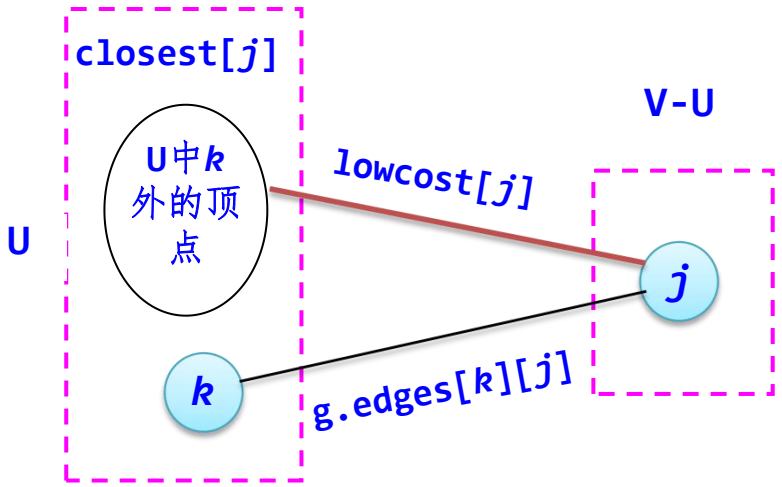
↑

此时恰好有g.edges[v][i]为∞，看出邻接矩阵为什么这样表示的原因。

```
def Prim(g,v):                          #求最小生成树
  lowcost=[0]*MAXV                      #建立数组lowcost
  closest=[0]*MAXV                      #建立数组closest
  for i in range(g.n):                  #给lowcost[]和closest[]置初值
     lowcost[i]=g.edges[v][i]
     closest[i]=v
```

初始化阶段

```
for i in range(1,g.n):                          #找出最小生成树的n-1条边
  min=INF
  k=-1
  for j in range(g.n):                          #在(V-U)中找出离U最近的顶点k
    if lowcost[j]!=0 and lowcost[j]<min:
       min=lowcost[j]
       k=j                                      #k记录最小顶点的编号
  print("(%d,%d):%d" %(closest[k],k,+min),end=' ')    #输出最小生成树的边
  lowcost[k]=0                                  #将顶点k加入U中
  for j in range(g.n):                          #修改数组lowcost和closest
    if lowcost[j]!=0 and g.edges[k][j]<lowcost[j]:
       lowcost[j]=g.edges[k][j]
       closest[j]=k
```



算法特点

- 上述普里姆算法中有两重for循环，所以时间复杂度为$O(n^2)$，其中n为图的顶点个数。
- 由于与e无关，所以普里姆算法特别适合于稠密图求最小生成树。

7.5.3. 克鲁斯卡尔算法

1. 克鲁斯卡尔算法过程

克鲁斯卡尔（Kruskal）算法是一种按权值的递增次序选择合适的边来构造最小生成树的方法。假设G=(V，E)是一个具有*n*个顶点的带权连通图，T=(U，TE)是G的最小生成树，则构造最小生成树的步骤如下：

（1）置U的初值等于V（即包含有G中的全部顶点），TE的初值为空集（即图T中每一个顶点都构成一个分量）。

（2）将图G中的边按权值从小到大的顺序依次选取：若选取的边未使生成树T形成回路，则加入TE；否则舍弃，直到TE中包含*n*-1条边为止。

2．克鲁斯卡尔算法设计

关键是如何判断选择的边是否与生成树中已有边形成回路？

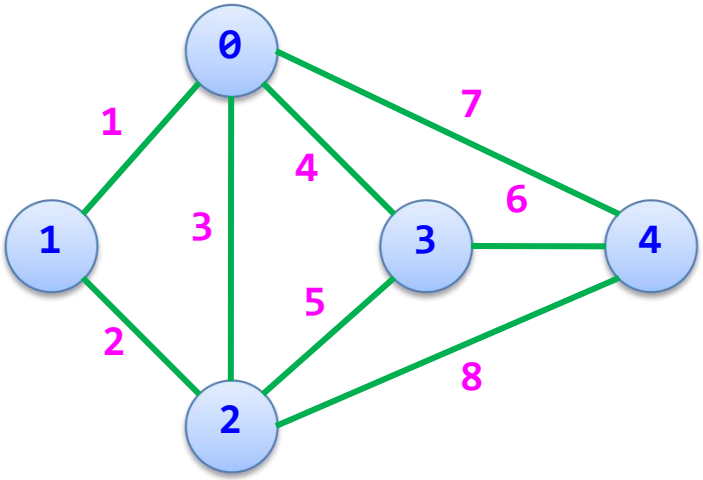
设置一个辅助数组vset[0..*n*-1]，其元素vset[*i*]代表顶点*i*所属的连通分量的编号（同一个连通分量中所有顶点的vset值相同）。

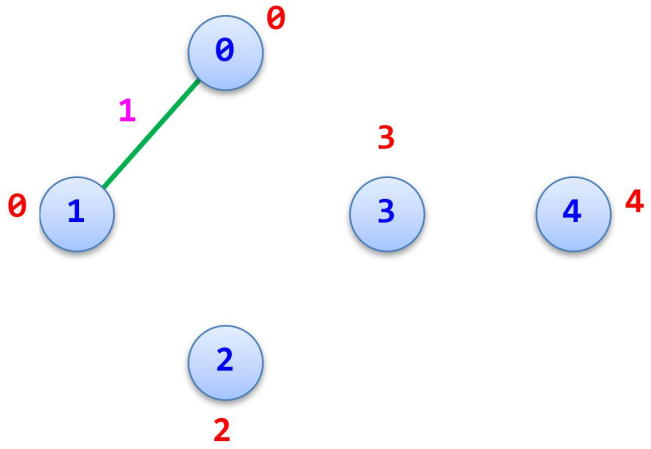
- 初始时T中只有n个顶点，没有任何边，每个顶点i看成一个连通分量，该连通分量的编号就是i，即vset[i]=i。
- 将图中所有边按权值递增排序，从前向后选边（保证总是选择权值最小的边），当选择一条边（u_1，v_1），求出这两个顶点所属连通分量的编号分别为sn_1和sn_2：

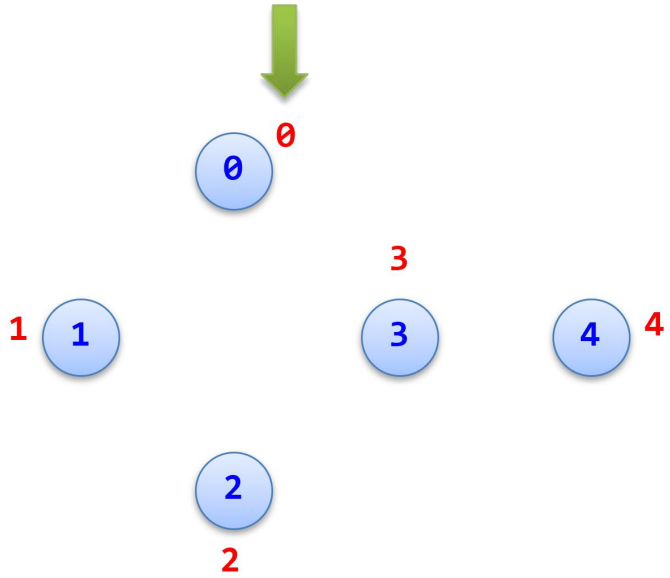
  - 若$sn_1=sn_2$，说明顶点u_1和v_1属于同一个连通分量 ⇨ 不能添加该边。
  - 若$sn_1\neq sn_2$，说明顶点u_1和v_1属于不同连通分量 ⇨ 添加该边。添加后原来的两个连通分量需要合并，即将两个连通分量中所有顶点的vset值改为相同（改为sn_1或者sn_2均可）。

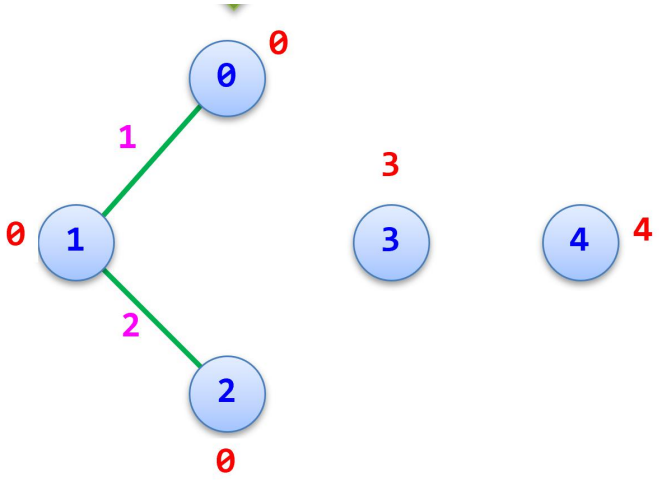
如此这样直到在T中添加n-1条边为止。

示例



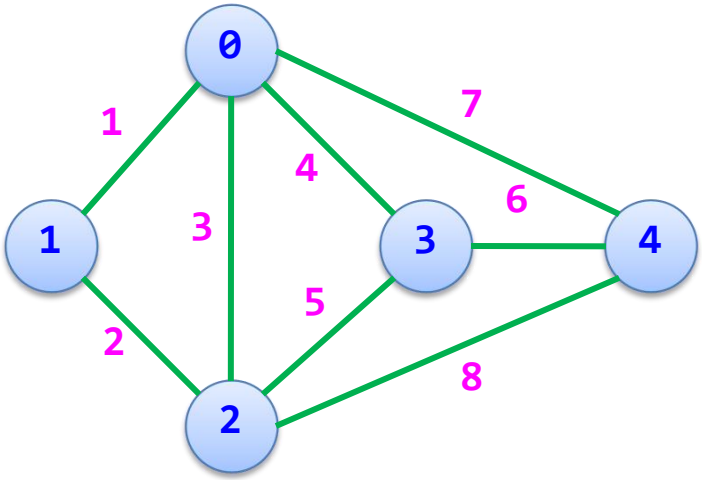


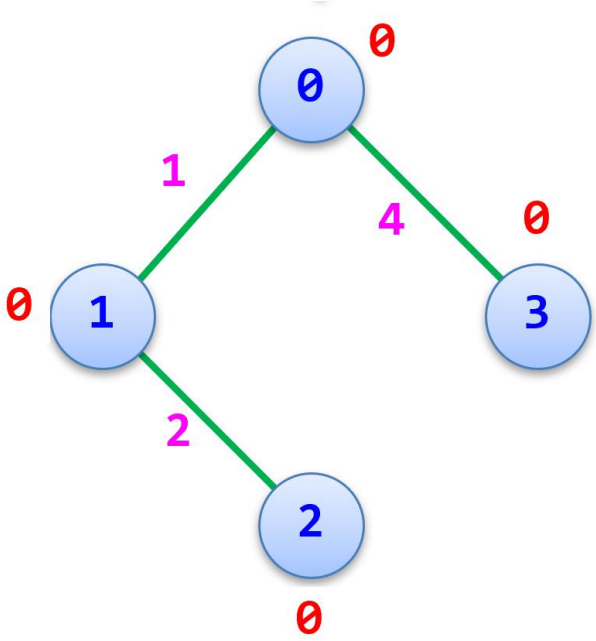


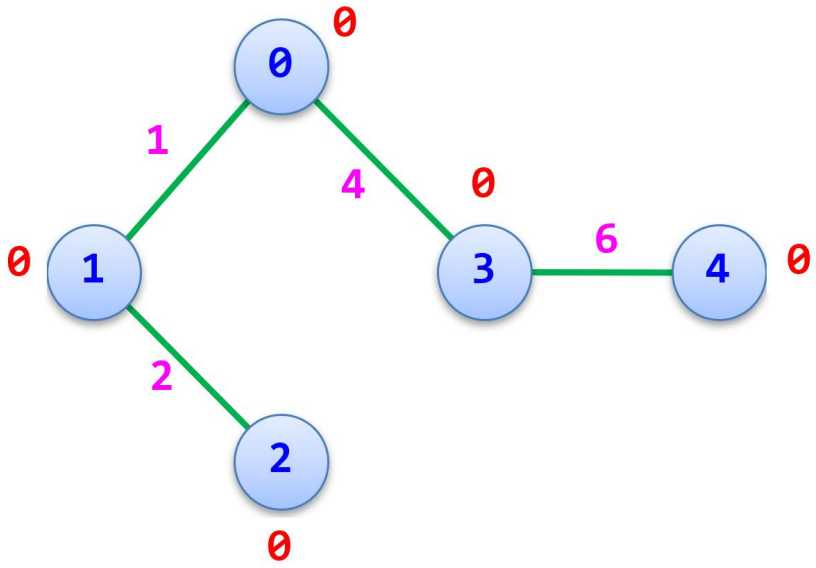


构造的最小生成树

示例







从图G的邻接矩阵中获取所有边集数组E：

- 由于是无向图，将邻接矩阵上三角部分的所有边存放在列表E中，每一条边对应的列表为[*u*，*v*，*w*]，其中*u*、*v*分别为边的头尾顶点，*w*为边的权值。
- 对E按权值*w*递增排序后做上述操作。

克鲁斯卡尔算法

```
def Kruskal1(g):                                    #求最小生成树:基本的Kruskal算法
  vset=[-1]*MAXV                                    #建立数组vset
  E=[]                                              #建立存放所有边的列表E
  for i in range(g.n):                              #由邻接矩阵g产生的边集数组E
     for j in range(i+1,g.n):                       #对于无向图仅考虑上三角部分的边
        if g.edges[i][j]!=0 and g.edges[i][j]!=INF:
           E.append([i,j,g.edges[i][j]]) #添加[i,j,w]元素
  E.sort(key=itemgetter(2))                         #按权值递增排序
```

```
for i in range(g.n):vset[i]=i                #初始化辅助数组
cnt=1                                        #cnt为最小生成树的第几条边,初值为1
j=0                                          #取E中边的下标,初值为0
while cnt<g.n:                               #生成的边数小于n时循环
    u1,v1=E[j][0],E[j][1]                    #取一条边的头尾顶点
    sn1=vset[u1]
    sn2=vset[v1]                             #分别得到两个顶点所属的集合编号
    if sn1!=sn2:                             #两顶点属于不同的集合,则加入
        print("(%d,%d):%d" %(u1,v1,E[j][2]),end=' ')  #输出最小生成树的边
        cnt+=1                               #生成边数增1
        for i in range(g.n):                 #两个集合统一编号
            if vset[i]==sn2:                 #集合编号为sn2的改为sn1
                vset[i]=sn1
    j+=1                                     #继续取E的下一条边
```

3*．改进的克鲁斯卡尔算法设计

连通分量

并查集中的一个子集

利用并查集的克鲁斯卡尔算法

```
parent=[-1]*MAXV                    #并查集存储结构
rank=[0]*MAXV                       #存储结点的秩
```

并查集初始化、查找和合并算法

```
def Kruskal2(g):                                    #求最小生成树：改进的Kruskal算法
  E=[]                                              #建立存放所有边的列表E
  for i in range(g.n):                              #由邻接矩阵g产生的边集数组E
    for j in range(i+1,g.n):                        #对于无向图仅考虑上三角部分的边
        if g.edges[i][j]!=0 and g.edges[i][j]!=INF:
           E.append([i,j,g.edges[i][j]])                #添加[i,j,w]元素
  E.sort(key=itemgetter(2))                         #按权值递增排序
```

```
Init(g.n)                                  #并查集初始化
cnt=1                                      #cnt表示最小生成树的第几条边,初值为1
j=0                                        #取E中边的下标,初值为0
while cnt<g.n:                             #生成的边数小于n时循环
    u1,v1=E[j][0],E[j][1]                  #取一条边的头尾顶点
    sn1=Find(u1)
    sn2=Find(v1)                           #分别得到两个顶点所属连通分量编号
    if sn1!=sn2:                           #两顶点属于不同的集合,则加入
        print("(%d,%d):%d" %(u1,v1,E[j][2]),end=' ')  #输出最小生成树的边
        cnt+=1                             #生成边数增1
        Union(u1,v1);                      #合并
    j+=1                                   #继续取E的下一条边
```

改进克鲁斯卡尔算法的时间复杂度为$O(e\log_2 e)$。由于与n无关，所以克鲁斯卡尔算法特别适合于稀疏图求最小生成树。