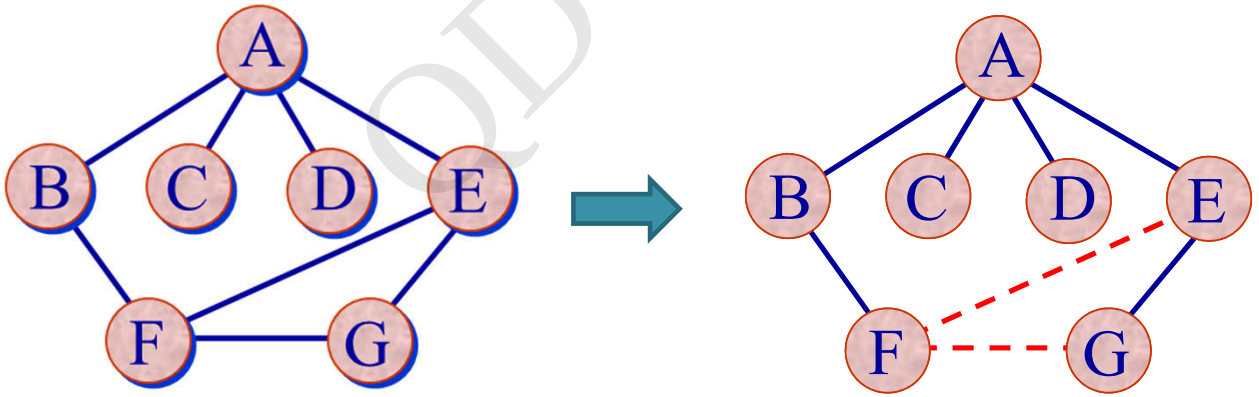
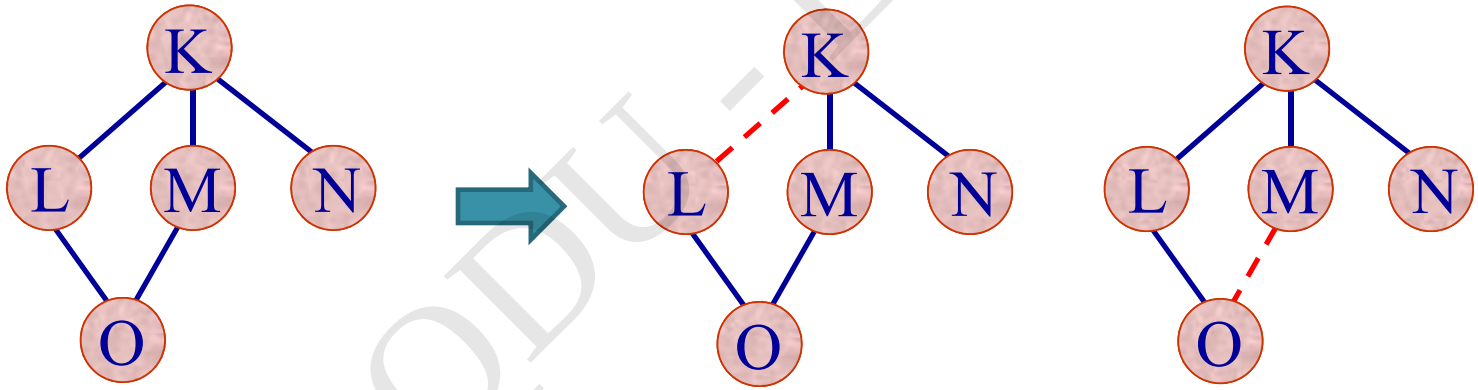
7.4.1 无向图的连通分量和生成树

- 在一个无向连通图G中，如果取它的全部顶点和一部分边构成一个子图G'，即V(G')=V(G) 和 E(G') ⊆ E(G)。
- 若边集E(G')中的边既将图G中的所有顶点连通又不形成回路，则称子图G'是原图G的一棵生成树。



可以通过遍历方式产生一个无向图的生成树。

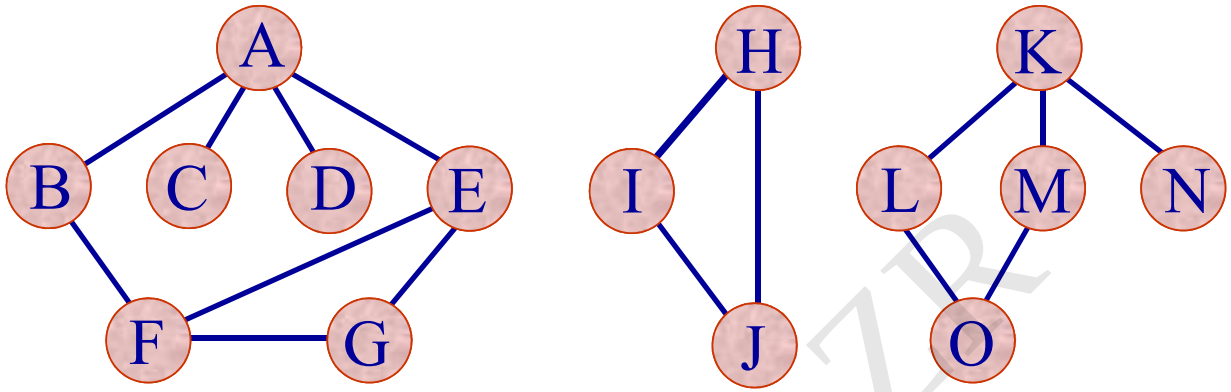
- 通过深度优先遍历产生的生成树称为深度优先生成树。
- 通过广度优先遍历产生的生成树称为广度优先生成树。



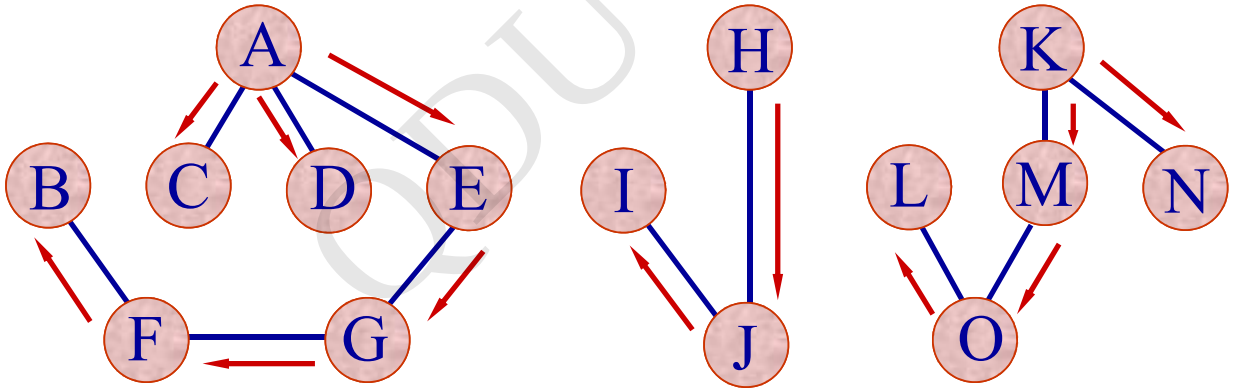
DFS生成树　　BFS生成树

无向图进行遍历时：

- 连通图：仅需要从图中任一顶点出发，进行深度优先遍历或广度优先遍历便可以访问到图中所有顶点，因此连通图的一次遍历所经过的边的集合及图中所有顶点的集合就构成了该图的一棵生成树。
- 非连通图：它由多个连通分量构成的，则需要从每个连通分量的任一顶点出发进行遍历，每次从一个新起点出发进行遍历过程得到的顶点访问序列恰为各个连通分量中的顶点集。每个连通分量产生的生成树合起来构成整个非连通图的生成森林。



非连通无向图G



DFS生成森林

7.4.3 最小生成树

- 由一个带权无向图可能产生多棵生成树， 把具有权之和最小的生成树称为图的最小生成树（Minimum Cost Spanning Tree，简称MCST）。
- 构造一个图的最小生成树主要有两个算法，即普里姆算法和克鲁斯卡尔算法。

7.4.3.1 普里姆算法

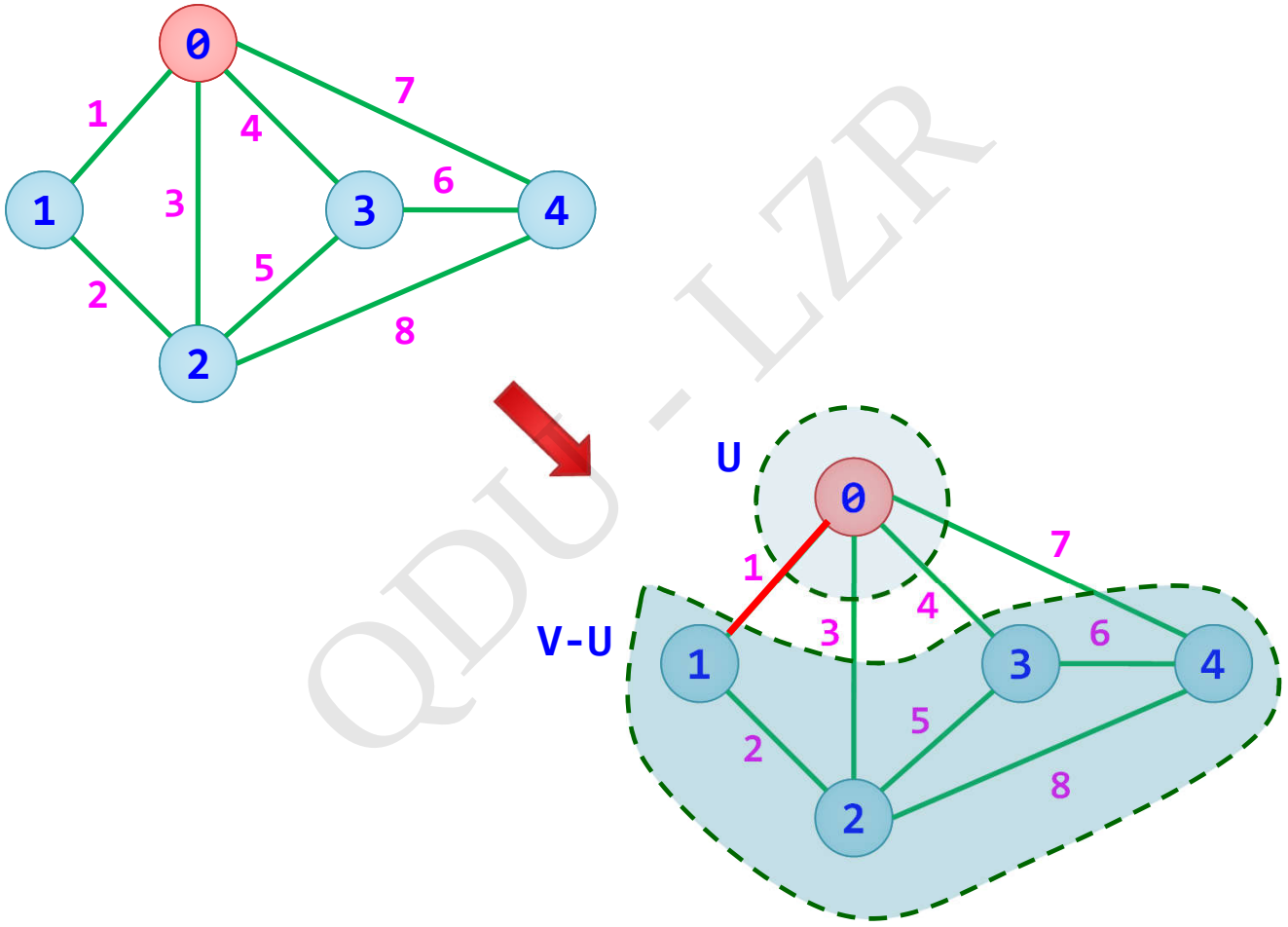
普里姆（Prim）算法是一种构造性算法。

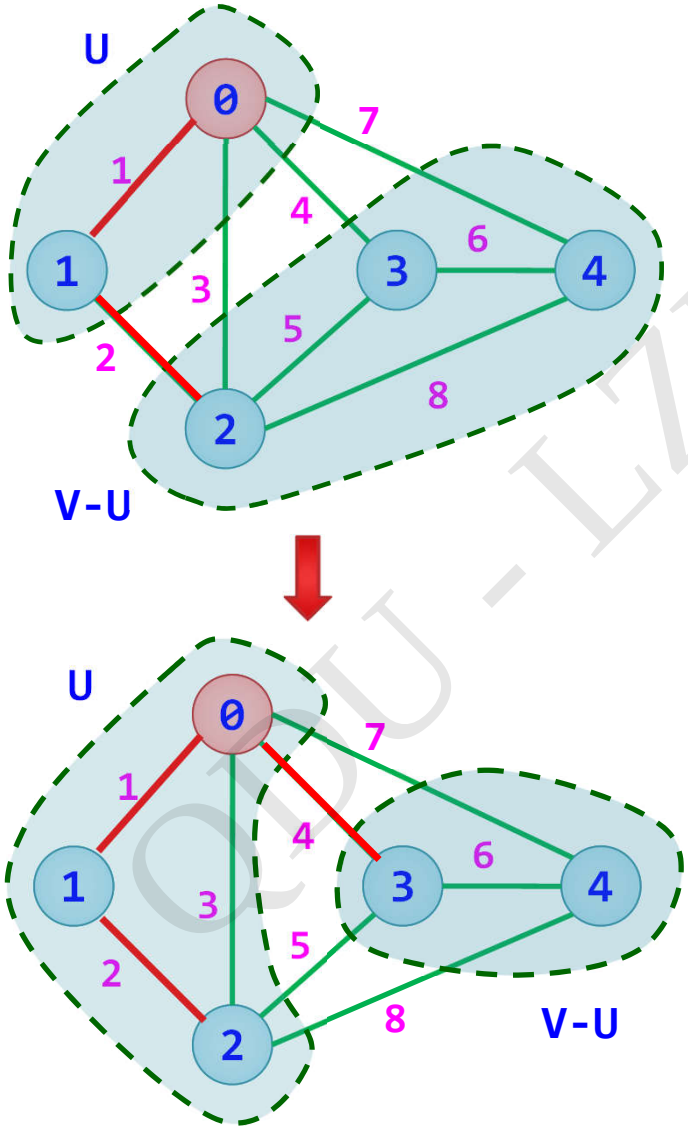
从连通网G=(V，E)中找最小生成树T=(V，TE)。

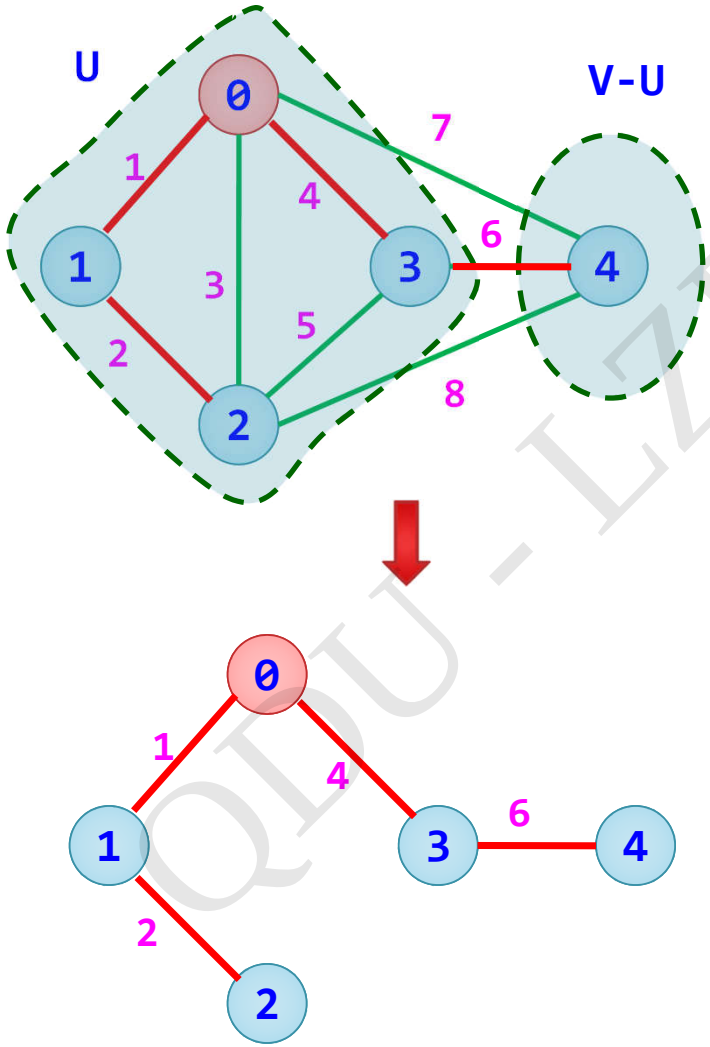
普里姆算法的思想：

- (1) 若从顶点v_0出发构造，U={v_0}，TE={}；
- (2) 先找权值最小的边(u，v)，其中u∈U且v∈V-U，并且子图不构成环，则U= U∪{v}，TE=TE∪{(u，v)}；
- (3) 重复(2)，直到U=V为止。则TE中必有n-1条边，T=(U，TE)就是最小生成树。

采用普里姆算法求最小生成树的过程



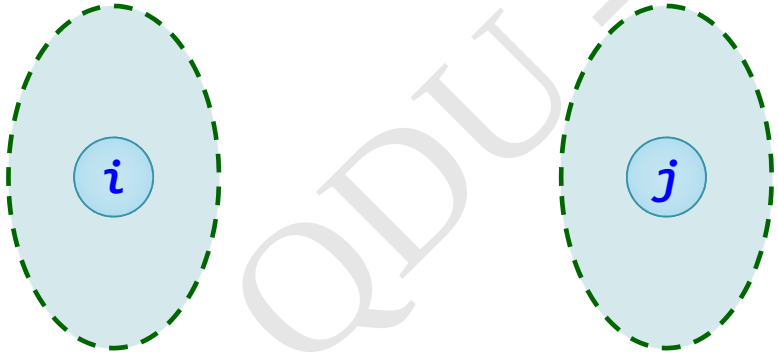

U
0
1
2
3
4
7
1
4
6
3
5
2
8
V-U
U
0
1
2
3
4
7
1
4
6
3
5
2
8
V-U



最小生成树

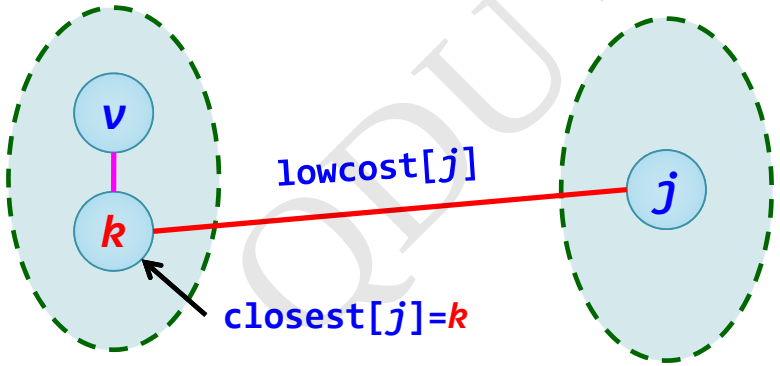
实现普里姆算法：

- 建立了两个辅助数组closest[]和lowcost[]。
- 所有顶点分为U和V-U两个顶点集。
- U中的顶点i：lowcost[i]=0;
- V-U中的顶点j：lowcost[j]>0。



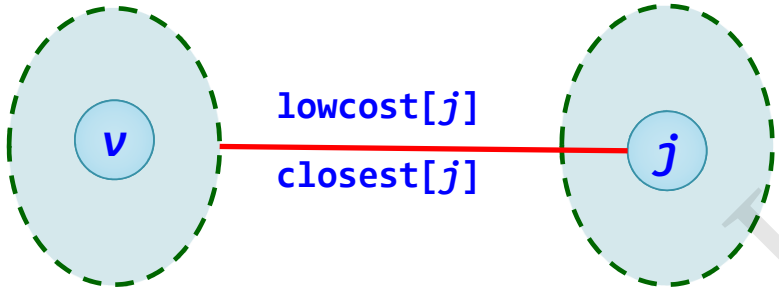
U中i：lowcost[i]=0　　V-U中j：lowcost[j]>0

- 由于是无向图，U到V-U的边与V-U到U的边相同，这里考虑V-U到U的边。
- 对于V-U中的每个顶点j，记录它到U中的一条最小边：
  closest[j]存储该边依附的在U中的顶点编号，
  lowcost[j]存储该边的权值。

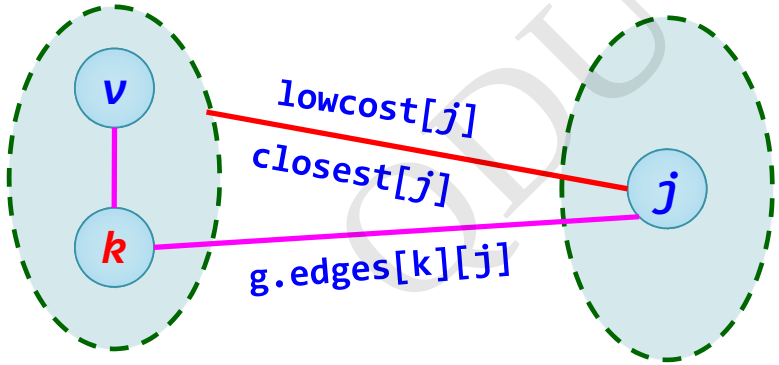


U中k：lowcost[k]=0　　V-U中j：lowcost[j]>0

- U中的顶点是逐个添加的。
- U中添加顶点k之前的情况：



- U中添加顶点k之后的情况：



- 对于j：若g.edges[k][j]<lowcost[j]⇨调整；否则不变

为了方便，假设图G采用邻接矩阵g存储，对应的Prim(g,v)算法如下：

```
void Prim(MGraph g,int v)     //输出求得的最小生树的所有边
{
   int lowcost[MAXVEX];       //建立数组lowcost
   int closest[MAXVEX];       //建立数组closest
   int min,i,j,k;
   for (i=0;i<g.n;i++)        //给lowcost[]和closest[]置初值
   {   lowcost[i]=g.edges[v][i];
       closest[i]=v;
   }
```

```
  for (i=1;i<g.n;i++)          //构造n-1条边
  {  min=INF; k=-1;
     for (j=0;j<g.n;j++)       //在(V-U)中找出离U最近的顶点k
       if (lowcost[j]!=0 && lowcost[j]<min)
       {  min=lowcost[j];
          k=j;                 //k为最近顶点的编号
       }
     printf("  边(%d,%d),权值为%d\n",closest[k],k,min);
     lowcost[k]=0;             //标记k已经加入U
     for (j=0;j<g.n;j++)       //修正数组lowcost和closest
       if (lowcost[j]!=0 && g.edges[k][j]<lowcost[j])
       {  lowcost[j]=g.edges[k][j];
          closest[j]=k;
       }
   }
}
```

- 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。

END