第四次上机 图

时间：12月14日

星期六8:30-11:30

上机地点：基础实验大楼506

上机题目

- 实验目的:
  1. 进一步掌握图的结构及非线性特点，递归特点和动态性。
  2. 进一步巩固图的三种存储结构和二种遍历方法、最小生成树的两种求解算法。

- 实验题目:
  1. 以邻接链表为存储结构，实现图的深度优先和广度优先遍历算法。
  2. 设计合适的存储结构，用普里姆算法和克鲁斯卡尔算法求网的最小生成树。

- 实验性质：设计型实验。

上机报告

- 课后：12月30日24:00前提交上机报告
- 实验报告要求：
  - 实验题目的设计描述
  - 调试程序后得到的结果（截屏）
  - 源程序及程序运行结果打印清单（需要简单注释，说明函数功能、入口和出口参数）
  - 实验结论和结果分析（可选）

- 普里姆算法 (Prim算法,又称边割法)　1957年由Prim提出

假设N=(V, {E})是连通网，TE为最小生成树中边的集合。

Step1: 初始$U=\{u_0\}(u_0 \in V)$, $TE=\varphi$;

Step2: 在所有$u \in U$，　$v \in V-U$的边中选一条代价最小的边(u_0, v_0)并入集合TE，同时将v_0并入U;

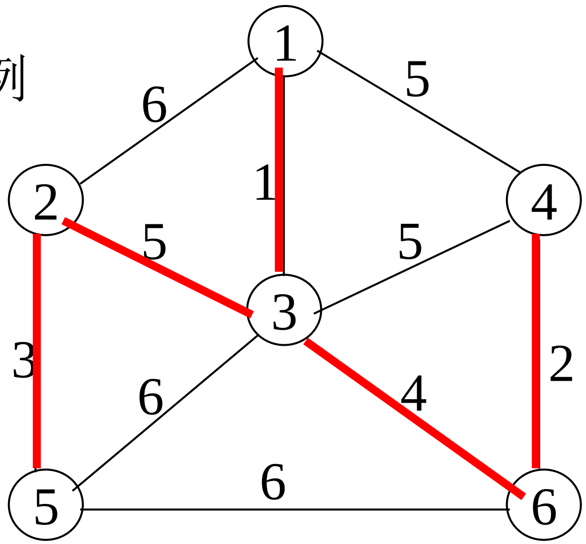
Step3: 重复Step2，直到U=V为止。

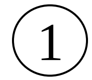
此时，TE中必含有n-1条边，则T=(V, {TE})为N的最小生成树。

从一个平凡图开始，普利姆算法逐步增加U中的顶点，可称为“加点法”。

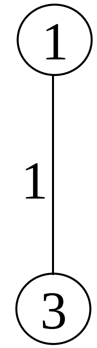
[注]：在最小生成树的生成过程中，所选的边都是一端在U中，另一端在V-U中。选最小边的过程是一个向集合U中添加顶点的过程。

例

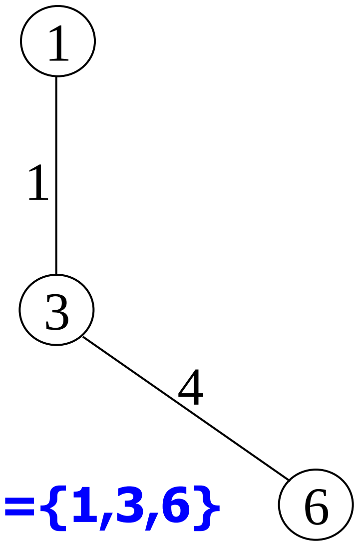




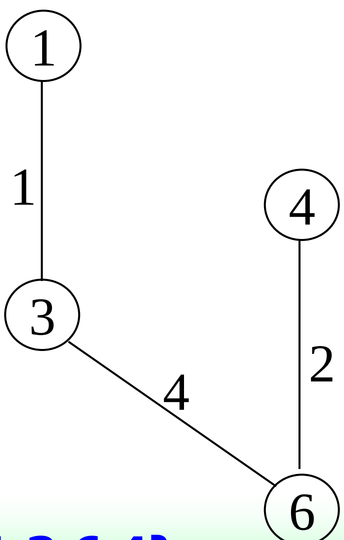
U={1}



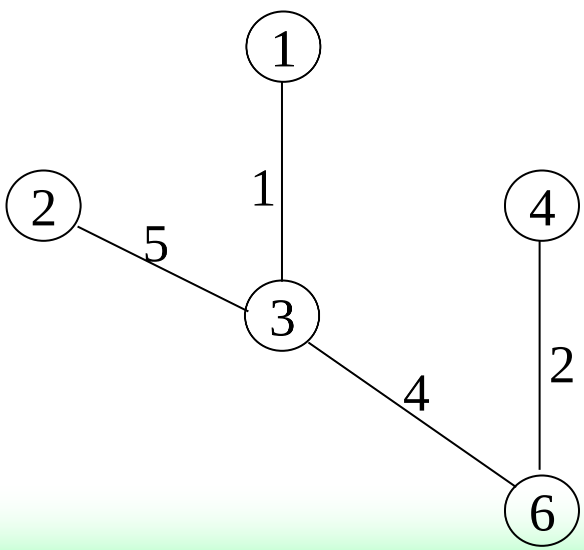
U={1,3}



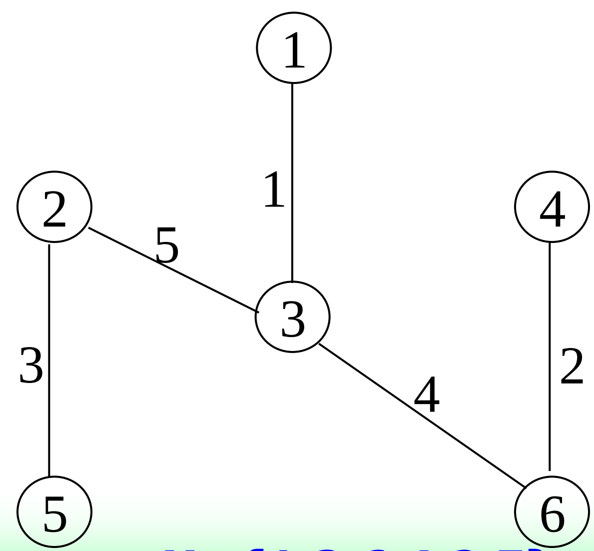
U={1,3,6}



U={1,3,6,4}



U={1,3,6,4,2}



U={1,3,6,4,2,5}

计算机内怎样实现Prim（普里姆）算法?

Prime算法特点: 将顶点归并，与边数无关，适于稠密网。
故采用邻接矩阵作为图的存储表示。

设计思路:

① 增设一辅助数组Closedge[n]，以记录从U到V-U具有最小代价的边， 每个数组分量都有两个域:

Colsedge[i]

adjvex	lowcost

V-U中顶点v_i

v_i 在U中的邻接点u（u，vi）

u与v_i之间对应的边权

要求：使Colsedge[i].lowcost = min((u ,vi))　u ∈ U

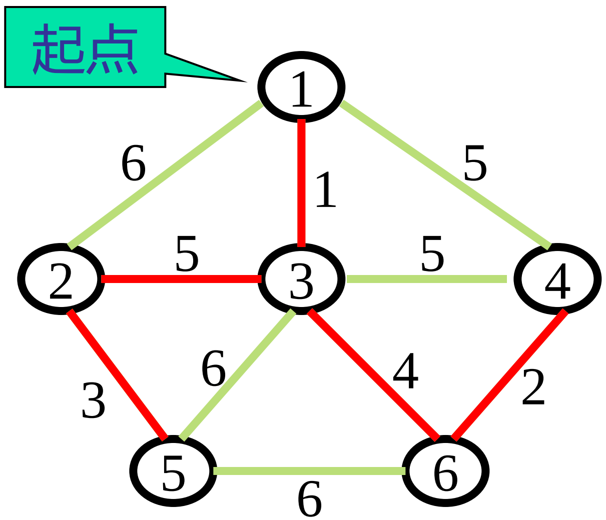
u从U（u_1~u_n）中挑选，选择U中到顶点vi边的权值最小的顶点

```
struct {
    VertexType  adjvex;  // u在U集中的顶点序号
    VRType     lowcost;  // 边的权值
} closedge[MAX_VERTEX_NUM];
```

具体示例：

怎样完成这个算法？

可以采用双层for循环控制



$$
\begin{array}{c}
1\\2\\3\\4\\5\\6
\end{array}
\begin{bmatrix}
0 & 6 & 1 & 5 & \infty & \infty \\
6 & 0 & 5 & \infty & 3 & \infty \\
1 & 5 & 0 & 5 & 6 & 4 \\
5 & \infty & 5 & 0 & \infty & 2 \\
\infty & 3 & 6 & \infty & 0 & 6 \\
\infty & \infty & 4 & 2 & 6 & 0
\end{bmatrix}
$$

closedge \ V	2	3	4	5	6	U	V-U
adjvex lowcost	1 6	1 1	1 5	1 ∞	1 ∞	{1}	{2,3,4,5,6}
adjvex lowcost	3 5	0	1 5	3 6	3 4	{1, 3}	{2, 4, 5, 6}
adjvex lowcost	3 5	0	6 2	3 6	0	{1,3,6}	{2, 4,5}
adjvex lowcost	3 5	0	0	3 6	0	{1,3,6,4}	{2, 5}
adjvex lowcost	0	0	0	2 3	0	{1,3,6,4,2}	{5}
adjvex lowcost	0	0	0	0	0	{1,3,6,4,2,5}	{}

算法复杂度：$O(n^2)$

```
void MiniSpanTree_PRIM( MGraph G, VertexType u) {
  //用Prim算法生成最小生成树
  k = LocateVex(G, u);
   for(j=0; j<G.vexnum; ++j) //辅助化数组初始化
      if(j!=k)    closedge[j] = { u, G.arcs[k][j].adj };
   closedge[k].lowcost = 0; //初始化U，即U={u}
   for(i=1; i<G.vexnum; ++i){ //选择其余G.vexnum-1个顶点
      k = minimun(closedge);  //求出T的下一个节点：第k个结点
      printf(colsedge[k].adjvex, G.vexs[k]); //输出边，也可以选择保存边
      closedge[k].lowcost = 0; //将第k个结点并入U集
      for(j=0; j<G.vexnum; ++j)
        if(G.arcs[k][j].adj < closedge[j].lowcost)
              //新顶点并入U后重新选择最小边
           colsedge[j] = { G.vexs[k], G.arcs[k][j].adj };
   }
}
```

适于稠密网，对边较少的图复杂度比较高

适用于稀疏网的最小生成树，算法复杂度为：O(nlogn)

- 克鲁斯卡尔算法（避圈法）**Kruskal于1956年提出**

假设G=(V,E)是一个具有n个顶点的带权连通无向图, T=(U,TE)是G的最小生成树, 则构造最小生成树的步骤如下：

Step1：**T=(V,{})，有n个顶点而无边的非连通图；**

Step2：在E中选择代价最小的边，若该边依附的顶点落在T中不同的连通分量上，则将此边加入**TE**；否则舍弃，而选择下一条代价最小的边；

Step3：**若T中有n-1条边，结束；否则转step2.**

从一个零图开始，克鲁斯卡尔算法逐步增加生成树的边， 与普里姆算法相比，可称为“加边法”

例