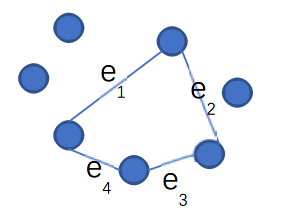
**作业1**

假定G 是一个无向图且此图中所有边的权值各不相同。

假设(e1,e2,…,ek)是G 中的k 条边，并构成一个简单回路。

求证：这个回路中权值最大的那一条边一定不在G 的最小生成树当中。



**证明：**

对于一个无向加权连通图，总是存在一棵或以上的有限课生成树，而这些生成树中肯定存在至少一棵最小生成树。

根据环定理，对于连通图中的任意一个环 C：如果 C中有边 e的权值大于该环中任意一个其它的边的权值，那么这个边必定不是最小生成树中的边。而最小生成树不含回路，由Kruskal算法过程，即可证明权值最大的那一条边，一定不在G 的最小生成树当中。

Kruskal算法步骤：

步骤1：Tree是边的集合，其初始状态为空；

步骤2：从原图剩余边中选取一条最小代价的边；

步骤3：看其是否与当前Tree中其它边构成回路；

步骤4：如果未构成回路，则加入Tree中；否则，丢弃该边；

步骤5：是否还有剩余边，如果有则返回步骤二，否则，程序结束

　　令T为Kruskal算法构造出的生成树，假定U是G的最小生成树。如果T==U那么证明结束。如果T != U，我们就需要证明T和U的构造代价相同。由于T != U，所以一定存在k > 0条边存在于T中，却不在U中。接下来，我们做k次变换，每次从T中取出一条不在U中的边放入U，然后删除U一条不在T中的边，最后使T和U的边集相同。每次变换中，把T中的一条边e加入U，同时删除U中的一条边f。e、f按如下规则选取：a). e是在T中却不在U中的边的最小的一条边；b). e加入U后，肯定构成唯一的一个回路，令f是这个回路中的一条边，但不在T中。f一定存在，因为T中没有回路。

　　这样的一次变换后，U仍然是一棵生成树。

　　我们假设e权值小于f，这样变换后U的代价一定小于变换前U的代价，而这和我们之前假设U是最小生成树矛盾，因此e权值不小于f。

　　再假设e权值大于f。由于f权值小于e，由Kruskal算法知，f在e之前从E中取出，但被舍弃了。一定是由于和权值小于等于f的边构成了回路。但是T中权值小于等于f（小于e）的边一定存在于U中，而f在U中却没有和它们构成回路，又推出矛盾。所以e权值不大于f，于是e权值等于f。

这样，每次变换后U的代价都不变，所以K次变换后，U和T的边集相同，且代价相同，这样就证明了T也是最小生成树。由证明过程可以知道，最小生成树可以不是唯一的。

由此得证，图中的权值最大的边，G的最小生成树一定不包含它；

**作业2**

无向图G 有n 个顶点、n+8 条边、且是连通的（假定所有边的权值各不相同）。请你描述一个算法计算G 的最小生成树，要求时间复杂度为O(n)。（对这道题，你只需要简述算法正确性不必太严谨的证明)

**解答**

可使用Prim算法计算无向图G的最小生成树，伪代码如下：

def MST\_PRIM(G, w, r):

for each :









While 

u=EXTRACT\_MIN(Q)

for each 

if  and :





Prim算法的时间代价为

**具体代码实现：**

#include <stdio.h>

#include <stdlib.h>

#define VertexType int

#define VRType int

#define MAX\_VERtEX\_NUM 20

#define InfoType char

#define INFINITY 65535

typedef struct {

VRType adj; //对于无权图，用 1 或 0 表示是否相邻；对于带权图，直接为权值。

InfoType \* info; //弧额外含有的信息指针

}ArcCell,AdjMatrix[MAX\_VERtEX\_NUM][MAX\_VERtEX\_NUM];

typedef struct {

VertexType vexs[MAX\_VERtEX\_NUM]; //存储图中顶点数据

AdjMatrix arcs; //二维[数组](http://data.biancheng.net/view/181.html)，记录顶点之间的关系

int vexnum,arcnum; //记录图的顶点数和弧（边）数

}MGraph;

//根据顶点本身数据，判断出顶点在二维数组中的位置

int LocateVex(MGraph G,VertexType v){

int i=0;

//遍历一维数组，找到变量v

for (; i<G.vexnum; i++) {

if (G.vexs[i]==v) {

return i;

}

}

return -1;

}

//构造无向网

void CreateUDN(MGraph\* G){

scanf("%d,%d",&(G->vexnum),&(G->arcnum));

for (int i=0; i<G->vexnum; i++) {

scanf("%d",&(G->vexs[i]));

}

for (int i=0; i<G->vexnum; i++) {

for (int j=0; j<G->vexnum; j++) {

G->arcs[i][j].adj=INFINITY;

G->arcs[i][j].info=NULL;

}

}

for (int i=0; i<G->arcnum; i++) {

int v1,v2,w;

scanf("%d,%d,%d",&v1,&v2,&w);

int m=LocateVex(\*G, v1);

int n=LocateVex(\*G, v2);

if (m==-1 ||n==-1) {

printf("no this vertex\n");

return;

}

G->arcs[n][m].adj=w;

G->arcs[m][n].adj=w;

}

}

//辅助数组，用于每次筛选出权值最小的边的邻接点

typedef struct {

VertexType adjvex;//记录权值最小的边的起始点

VRType lowcost;//记录该边的权值

}closedge[MAX\_VERtEX\_NUM];

closedge theclose;//创建一个全局数组，因为每个函数中都会使用到

//在辅助数组中找出权值最小的边的数组下标，就可以间接找到此边的终点顶点。

int minimun(MGraph G,closedge close){

int min=INFINITY;

int min\_i=-1;

for (int i=0; i<G.vexnum; i++) {

//权值为0，说明顶点已经归入最小生成树中；然后每次和min变量进行比较，最后找出最小的。

if (close[i].lowcost>0 && close[i].lowcost < min) {

min=close[i].lowcost;

min\_i=i;

}

}

//返回最小权值所在的数组下标

return min\_i;

}

//普里姆算法函数，G为无向网，u为在网中选择的任意顶点作为起始点

void miniSpanTreePrim(MGraph G,VertexType u){

//找到该起始点在顶点数组中的位置下标

int k=LocateVex(G, u);

//首先将与该起始点相关的所有边的信息：边的起始点和权值，存入辅助数组中相应的位置，例如（1，2）边，adjvex为0，lowcost为6，存入theclose[1]中，辅助数组的下标表示该边的顶点2

for (int i=0; i<G.vexnum; i++) {

if (i !=k) {

theclose[i].adjvex=k;

theclose[i].lowcost=G.arcs[k][i].adj;

}

}

//由于起始点已经归为最小生成树，所以辅助数组对应位置的权值为0，这样，遍历时就不会被选中

theclose[k].lowcost=0;

//选择下一个点，并更新辅助数组中的信息

for (int i=1; i<G.vexnum; i++) {

//找出权值最小的边所在数组下标

k=minimun(G, theclose);

//输出选择的路径

printf("v%d v%d\n",G.vexs[theclose[k].adjvex],G.vexs[k]);

//归入最小生成树的顶点的辅助数组中的权值设为0

theclose[k].lowcost=0;

//信息辅助数组中存储的信息，由于此时树中新加入了一个顶点，需要判断，由此顶点出发，到达其它各顶点的权值是否比之前记录的权值还要小，如果还小，则更新

for (int j=0; j<G.vexnum; j++) {

if (G.arcs[k][j].adj<theclose[j].lowcost) {

theclose[j].adjvex=k;

theclose[j].lowcost=G.arcs[k][j].adj;

}

}

}

printf("\n");

}

int main(){

MGraph G;

CreateUDN(&G);

miniSpanTreePrim(G, 1);

}

**作业3**

给定字符串T=T1…Tn。定义π(j) = max{k<j | T[1,j]的长为k 的前、后缀相等}。严格的证明：{k| T[1,j]的长为k 的前、后缀相等} = {j, π[j], π[π[j]], π[π[π[j]]], …, 0}。

**证明：**

采用数学归纳法

由，可知。根据定义可知：。

假设对于，有。

对于,

在字符串T的

因此

即:当时，

同理可证

由数学归纳法可总结得到相邻的有如下关系：

因此得证：

**作业4**

假设x=(x1,…,xn）是一个实数序列，并且任意两个元素互不相同。Slide 7 中给出了求x 的最长递增子序列的O(n2 )的算法。在slide 20 中，利用AVL 树，我们给出了一个更快的O(n log n)算法。实际上，对于最长递增子序列问题，存在一个更加简单的O(n log n)算法，它不需要用到诸如AVL 树、Splay 树这样的高级数据结构，而是只要用到数组。请设计出这个终极的O(n log n)算法。要求认真描述清楚算法，并简单扼要的写出证明。

**算法：**

1. 设置一个数组 ，初始时为空；

**注意：**数组  虽然是有序数组，但它不是问题中的「最长上升子序列」（下文还会强调），不能命名为 LIS。有序数组只是用于求解 LIS 问题的辅助数组。

1. 在遍历实数序列 的过程中，每来一个新的实数，如果这个数严格大于有序数组的最后一个元素，就把放在有序数组的最后，否则进入步骤3；

**注意：**这里的大于是「严格大于」，不包括等于的情况。

1. 在有序数组中查找第 1 个大于等于的那个数，试图让它变小：

* 如果有序数组中存在等于的元素，什么都不做，因为以结尾的最短的「上升子序列」已经存在；
* 如果有序数组中存在大于的元素，找到第 1 个，让它变小，这样我们就找到了一个结尾更小的相同长度的上升子序列。

**证明：**

定义数组，表示长度为的上升子序列中最后一个元素的值，若有多个长度为的上升子序列，则取所有子序列最后一个元素的最小值。

那么显然数组的长度即为最长递增子序列的长度，换而言之，只要维护数组即可。下面证明数组是单调不减的，即对于，则有。

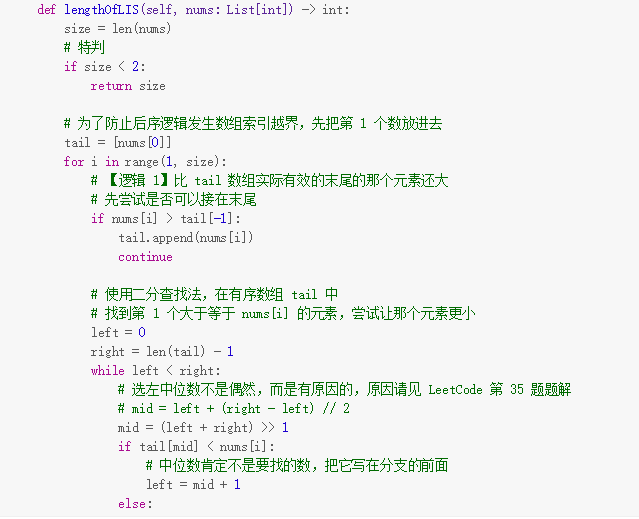
假设存在，；

设长度为的上升子序列是

另设长度为的上升子序列是

则有，那么子序列的最后一个元素作为比要小，与数组的定义相违背，故而假设不成立，证毕。

**Python代码：**



**复杂度分析：**

时间复杂度：，遍历数组使用了，二分查找法使用了。

空间复杂度：，开辟有序数组的空间至多和原始数组一样。