**Kruskal最小生成树算法实验报告**

1. **算法原理**

Kruskal算法是贪心法的典例，简单个描述为：给定一组边，将它们按照权重排序，每次挑选权重最小的边做试探，如果选择的边的两个端点不出现在同一个集合里，那么这个边就是最小生成树的一条边。

1. **问题描述**

本实验完成如下的输入输出过程：用邻接矩阵表示的无向图为随机生成的矩阵，直接将此图作为Kruskal算法的输入，输出结果为边的选择和最小生成树的代价和。

1. **算法过程分析**

算法核心部分由两个步骤完成：第一个是排序步，需要将所有的边按照权值从小到大排序，不存在的边用某一个最大值表示。

第二步是选择步，选择一条当前权值最小的边加入到生成树当中，如果加入的这条边使得当前的图成环，那么这条边被抛弃。

算法流程图可以简单的表示如下：

1. **算法复杂度分析**

Kruskal的算法时间复杂度非常容易分析，由于主要的时间耗费在边权值的排序上，因此对于排序步的时间复杂度为O（eloge）。

对于选择步，并查集的操作均可在O（n）的过程中完成，其中n为顶点数。

1. **可能的改进**
2. 对于排序过程，使用了qsort库函数，不能保证是最优的排序方法。
3. 本实验具有一定的局限性，仅对带权简单无向图进行讨论。对于多重图，Kruskal存在一个广义的方法同样适用，本实验为了突出重点，仅考虑前一种情况。
4. **算法实现**
5. **数据结构和函数**

typedef struct MGraph{

int vexNum;

int arcNum;

int adjM[MAX\_VEXNUM][MAX\_VEXNUM];

GraphType type;

}MGraph;

typedef struct{

int fromVex;

int toVex;

int weight;

}Edge;

Edge结构负责了组织边的选择问题，而MGraph为抽象的带权无向简单图。

bool createMG(MGraph &MG,int vexs)方法提供了随机生成矩阵的可能，每次输入均不同，可以有效地检验算法的正确性。

1. **算法核心流程**

qsort(arr\_Edge,p,sizeof(Edge),cmp);

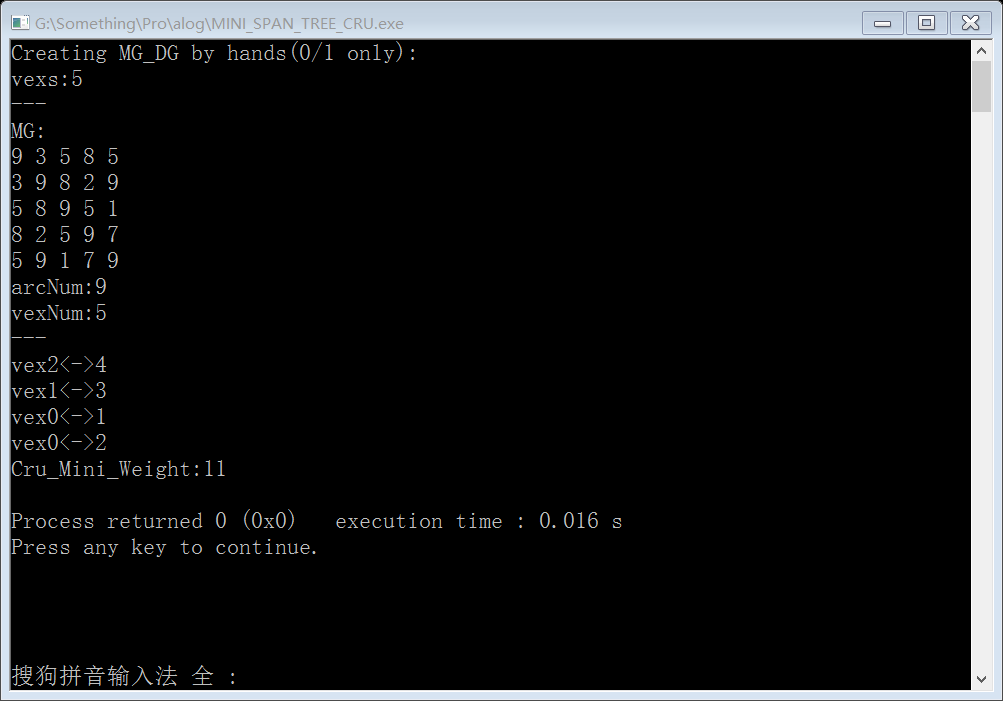
比较函数给出了按照权值排序的过程。

for( m = arr\_Edge[i].fromVex; m != check[m]; m = check[m]) check[m] = check[check[m]];

for( n = arr\_Edge[i].toVex; n != check[n]; n = check[n]) check[n] = check[check[n]];

两个for循环为查找某一条边的始点和中点是否属于同一个集合，实质上是并查集操作。

1. **输出结果示例与说明**



给出了一个节点数为5的图的例子。这里可以看到这个图的节点数为5，边数为9，是随机生成的。vex序列给出了边的选择，最小代价总和为11，经过检验可知其正确性。

1. **总结和反思**

Kruskal在学习数据结构的时候已经有了一定的了解，因此本次编码过程很顺利。本次编码还采用了随机输入的方法，可以有效地、快速地检验算法的正确性。Kruskal算法体现了贪心算法的本质：算法本身需要一个排序过程，从最值元素开始，通过规则筛选最后选出结果的集合，在本问题里就是最后的最小代价生成树。

#include<iostream>

#include<stdlib.h>

#include<queue>

#include<ctime>

#include<cstring>

using namespace std;

#define MAX\_VEXNUM 5

#define MAX\_WEIGHT 9

typedef enum{

DG,DN,UDG,UDN

}GraphType;

typedef struct MGraph{

int vexNum;

int arcNum;

int adjM[MAX\_VEXNUM][MAX\_VEXNUM];

GraphType type;

}MGraph;

typedef struct{

int fromVex;

int toVex;

int weight;

}Edge;

bool initGraph(MGraph &MG)

{

MG.vexNum=MG.arcNum=0;

for(int i=0;i<=MAX\_VEXNUM-1;i++)

{

for(int j=0;j<=MAX\_VEXNUM-1;j++)

MG.adjM[i][j]=0;

}

}

bool createMG(MGraph &MG,int vexs)

{

int arc\_count=0;

srand(time(NULL));

cout<<"Creating MG\_DG by hands(0/1 only):"<<endl;

cout<<"vexs:"<<vexs<<endl;

MG.type = UDG;

MG.vexNum = vexs;

MG.arcNum = 0;

for(int row = 0; row <=MAX\_VEXNUM -1 ;row++)

{

for(int col = row;col <=MAX\_VEXNUM - 1;col++)

{

if(row == col)

MG.adjM[row][col] = MAX\_WEIGHT;

else

{

MG.adjM[row][col] = rand()%MAX\_WEIGHT + 1; // 随机数创建矩阵

MG.adjM[col][row] = MG.adjM[row][col];

if(MG.adjM[row][col]!=MAX\_WEIGHT)

MG.arcNum++;

}

}

}

}

int cmp(const void\*p1,const void\*p2)

{

Edge \*a=(Edge\*) p1;

Edge \*b=(Edge\*) p2;

int result=0;

if(a->weight>b->weight)

return 1;

return -1;

}

void MiniSpanTree\_Crus(MGraph &MG)

{

//MG is UDN

Edge arr\_Edge[MG.vexNum\*(MG.vexNum-1)/2];

int countWeight=0;

int p=0;

for(int row=0;row<=MG.vexNum-1;row++)

{

for(int col=row+1;col<=MG.vexNum-1;col++)

{

arr\_Edge[p].weight=MG.adjM[row][col];

arr\_Edge[p].fromVex=row;

arr\_Edge[p].toVex=col;

p++;

}

}

int check[MG.vexNum]; // check数组负责统计每个顶点所在的集合

for(int i =0 ;i<=MG.vexNum-1;i++)

check[i] = i;

qsort(arr\_Edge,p,sizeof(Edge),cmp);

int count\_arc=0;

int m,n;

for(int i=0;i<=p-1;i++)

{

if(arr\_Edge[i].weight == MAX\_WEIGHT) continue;

for( m = arr\_Edge[i].fromVex; m != check[m]; m = check[m]) check[m] = check[check[m]];

for( n = arr\_Edge[i].toVex; n != check[n]; n = check[n]) check[n] = check[check[n]];

if( m == n ) continue; // 如果始点所在和集合和终点所在集合不同，那么可以增加此边

else

{

check[n] = m;

count\_arc++;

countWeight+=arr\_Edge[i].weight;

cout<<"vex"<<arr\_Edge[i].fromVex<<"<->"<<arr\_Edge[i].toVex<<endl;

if(count\_arc==MG.vexNum-1)

{

cout<<"Cru\_Mini\_Weight:"<<countWeight<<endl;

return ;

}

}

}

cout<<"Error\_Cru"<<endl;

return ;

}

void printGraph(MGraph &MG)

{

cout<<"---"<<endl;

cout<<"MG:"<<endl;

for(int row=0;row<=MG.vexNum-1;row++)

{

for(int col=0;col<=MG.vexNum-1;col++)

{

cout<<MG.adjM[row][col]<<" ";

}

cout<<endl;

}

cout<<"arcNum:"<<MG.arcNum<<endl;

cout<<"vexNum:"<<MG.vexNum<<endl;

cout<<"---"<<endl;

}

int main(void)

{

MGraph MG;

initGraph(MG);

createMG(MG,MAX\_VEXNUM); //可通过修改宏来改变顶点个数

printGraph(MG);

MiniSpanTree\_Crus(MG);

return 0;

}