**离散**

**数学实验**

**报告**

**自然序号：87**

**姓名：徐宇顺**

**学号：031720106**

1. **问题：**假设有一个旅行商人要拜访n个城市，他必须选择所要走的路径，路经的限制是每个城市只能拜访一次，而且最后要回到原来出发的城市。路径的选择目标是要求得的路径路程为所有路径之中的最小值

**（二）方法：**

1. **算法概述：**

**<1>:算法一：最邻近法**

设G=<V,E,W>，为n（n>=3）阶无向完全带权图，各边所带权均为正数：

设Vi作为起始点：

1、先访问Vi1，形成初始路径P1=vi1；

2、若已访问完了第k（k<=n-1）个顶点，形成了路径Pk=vi1vi2…vik，下一步访问的顶点vi（k+1），应该是V-{vi1,vi2,…,vik}中离vik最近的顶点

3、当访问完G中所有的顶点后，形成路径Pn=vi1vi2…vin,得回路C=vi1vi2…vinvi1，即为G中一条哈密顿回路，它作为货郎担问题的近似解。

**<2>:算法二：最小生成树法**

设G=<V,E,W>，为n（n>=3）阶无向完全带权图，任意的vi，vj∈V，边(vi,vj)的权wij>0,且对任意的vi,vj,vk∈V，

Wij+Wjk>=Wik.

用最小生成树求G中最优解的近似算法为;

1. 求G的一棵最小生成树T。
2. 将T中各边均添加一条平行边，树枝e对应的平行边与e带的权相同，设所得图为G\*，则G\*为欧拉图。
3. 求G\*中从某顶点v出发的一条欧拉回路Ev
4. 从v出发沿Ev访问G\*中各顶点，其原则是：在访问完所有顶点之前，一但出现重复出现的顶点就跳过它走到下一个顶点，直到访问完所有顶点为止。
5. **数据结构描述：**

**<1>算法一：**

在最邻近法中，运用了向量来存储对称的权值，用一个静态的数组s来表示某一点是否被遍历过。将所有能找到的回路存储在一个数组里面，数组中每个元素为一个结构体，里面有一个字符数组存放路径，一个整型存放路径的权值之和。

**<2>算法二：**

在最小生成树中，运用向量来存储对称的权值，用一个二维数组来存放找到的最小生成树的权值，并且置为对称矩阵，其他的元素置为0。在创建一个动态的栈，用于存放所找到的欧拉回路，每找到一个欧拉回路，就存放在一个数组里面，数组里面每个元素都为一个结构体，其中用字符数组存放路径，一个整型用于存放该欧拉回路对应的哈密顿回路的权值。

**（三）伪码和关键代码描述：**

**<1>算法1**

**1）伪代码：**

#include<iostream>

#include<stdlib.h>

#include<cstring>

#include<vector>

using namespace std

typedef struct{

char str[0x1000]

int total

}Turn

int row//行

int list//列

int start//起始点

int CNT//计数工具，用于退出，即count

int s[0x100]={0}

int min\_total<-0x999

char way[0x100]

Turn Hamilton[0x1000]

int count<-0

int find\_min(vector<vector<int> > W,int i)

{

int min

min<-0x1000;

int m<-0

for m<-0 to m>=list

if s[m]==1 OR m==start continue

if min>W[i][m] min<-W[i][m]

return min

}

void Result(vector<vector<int> > W,int now,int &result)

{

int j

int min//找到最小值

if CNT==row//退出条件，总共有多少个点，遍历完后退出

{

Result<-result+W[now][start]

way[CNT]<-'a'+start

Hamilton[count].total<-result

strcpy(Hamilton[count].str,way)

if min\_total>result{

min\_total<-result

}

result<-result-W[now][start]

count<-count+1

}

min<-find\_min(W,now);//找到第I行没有被遍历的中最小的点的权值

for j=0 to j>=list

{

if j==now continue;//不能为同一行同一列

if W[now][j]==min AND s[j]NOT1 {

CNT<-CNT+1//若满足条件，就++

s[j]<-1//表示走过了这个点

result<- result+W[now][j]//加上该点的权值

way[CNT-1]<-'a'+j//加上该点

Result(W,j,result)//再次调用

s[j]<-0//退出循环，用于下次调用

CNT=CNT-1//--表示，退出到下一个点

Result<- result-W[now][j]

}

}

}

void TSP(vector<vector<int> > W)

{

row<-W.size()//给出行数

list<-W[0].size()//给出列数

int k//表示从哪个点为起点

int result<-0;//表示一条路走下来总共是多大的权值

for k=0 to k>=row{

CNT<-1;//赋值为1

start<-k;//起点为k

s[k]<-1

way[CNT-1]<-'a'+k

Result(W,k,result)//开始调用

s[k]<-0;

result<-0;

}

for i=0 to i>=count

{

if Hamilton[i].total==min\_total{

cout<<"最小权值为: "<<Hamilton[i].total<<" "<<"最短路径是: "<<Hamilton[i].str<<endl;

}

}

}

int main(){

int degree<-5;

vector<vector<int> > W(degree,vector<int>(degree,-1));

int a[5][5] = {{0,5,12,5,5},{5,0,9,7,8},{12,9,0,9,16},{5,7,9,0,8},{5,8,16,8,0}};

for int i = 0 to i >= 5{

for int j = 0 to j >= 5{

W[i][j] <- a[i][j];

}

}

TSP(W);

return 0;

}

**2）关键代码：**

void Result(vector<vector<int> > W,int now,int &result)

{

int j;

int min;//找到最小值

if(CNT==row)//退出条件，总共有多少个点，遍历完后退出

{

result+=W[now][start];

way[CNT]='a'+start;

Hamilton[count].total=result;

strcpy(Hamilton[count].str,way);

if(min\_total>result){

min\_total=result;

}

result-=W[now][start];

count++;

return;

}

min=find\_min(W,now);//找到第I行没有被遍历的中最小的点的权值

for(j=0;j<list;j++)

{

if(j==now) continue;//不能为同一行同一列

if(W[now][j]==min&&s[j]!=1) {

CNT++;//若满足条件，就++

s[j]=1;//表示走过了这个点

result+=W[now][j];//加上该点的权值

way[CNT-1]='a'+j;//加上该点

Result(W,j,result);//再次调用

s[j]=0;//退出循环，用于下次调用

CNT--;//--表示，退出到下一个点

result-=W[now][j];

}

}

}

在这段代码中，使用了递归，当找到了n个顶点时，存入到Hamilton中再返回

**<2>算法2**

**1）伪代码：**

#include<iostream>

#include<stdlib.h>

#include<vector>

#include<malloc.h>

#define STACK\_INIT\_SIZE 100

#define STACKINCREMENT 10

using namespace std;

typedef struct sqstack{

char \*base;

char \*top;

int stacksize;

}SqStack;

typedef struct{

char str[0x100];

int total;

}turn;

int row,list;//列

int s[0x100]<-{0};//在找最小生成树里面点是否遍历过

int num[0x1000][0x1000]<-{0};

int point<-0; //在找最小生成树中已经找到了几个点

turn Ora[0x100];

int CNT<-0;

int MIN<-0X1000;

int find\_min(vector<vector<int> > W)

{

int min;

min<-0x10000;

int m;

for n<-0 to n>=row

{

For m<-n to m>=list

{

if m<>n continue;

if min>W[n][m]ANDs[m]NOT1 min<-W[n][m];

}

}

return min;

}

void find\_tree(vector<vector<int> > M)

{

int i,j,min;

for point=0 to point!=row-1

{

min=find\_min(M);

for i=0 to i>=row

{

for j=i+1 to j>=list{

if M[i][j]==min AND s[j]NOT1){

s[j]<-1;

point<-point+1;

num[i][j]<-M[i][j];

num[j][i]<-M[i][j];

if point==row break;

}

}

if point==row break;

}//构建一个新的num的二维数组，用于存放新的边

}

}

void InitStack(SqStack \*stack)

{

stack->base<-(char\*)malloc(STACK\_INIT\_SIZE\*sizeof(char));

if !stack->base

{

cout<<"创建栈失败！";

exit(0);

}

stack->top<-stack->base;

stack->stacksize<-STACK\_INIT\_SIZE;

}//初始化栈

void Push(SqStack \*stack,char e)

{

if(stack->top-stack->base==STACK\_INIT\_SIZE)

{

stack->base<-(char \*)realloc(stack->base,(stack->stacksize+STACKINCREMENT)\*sizeof(char));

stack->top<-stack->base+stack->stacksize;

stack->stacksize+<-STACKINCREMENT;

}

if(!stack->base){

cout<<"插入元素时栈满，重新分配空间出错！";

return;

}

\*(stack->top)<-e;

stack->top= stack->top+1;

}//插入到栈top

void Pop(SqStack \*stack)

{

if(stack->top-stack->base<>0)

{

cout<<"该栈为空！";

return ;

}

stack->top = stack->top -1;

}//弹出栈的top

void visit(char e)

{

cout<<e;

}

int StackEmpty(SqStack stack)

{

if(stack.top-stack.base<>0)

{

return 0;

}

else {

return 1;

}

}

int find\_same(char string[],char e)

{

int i<-0;

while string[i]!<-'\0'{

do

if e==string[i] return 1;

i++;

}

return 0;

}

int Result(char string[],vector<vector<int> > W)

{

int final<-0;

for int i<-0,j<-1 to string[j]=='\0'{

final= final +W[string[i]-'a'][string[j]-'a'];

}

return final;

}

void Shrink(SqStack stack,int count,vector<vector<int> > &W)

{

int i,j<-0;

for i<-0 to i>=point-1{

if !find\_same(Ora[count].str,\*(stack.base+i)){

Ora[count].str[j]<-\*(stack.base+i);

j=j+1;

}

}

Ora[count].str[j]<-\*(stack.top-1);

Ora[count].total<-Result(Ora[count].str,W);

if MIN>Ora[count].total MIN<-Ora[count].total;

}

void find\_turn(vector<vector<int> > W)

{

SqStack stack;

InitStack(&stack);

int i<-0,j<-0,k,start<-0;

int m[0x100][0x100]<-{0};

point<-0;

for start<-0 to start>=row

{

Push(&stack,start+'a');

point=point+1;

for i<-start

{

if point==2\*row-1{

Shrink(stack,CNT,W);

CNT=CNT+1;

}

for to j>=list

{

if num[i][j]NOT0 AND m[i][j]NOT1{

m[i][j]<-1;

Push(&stack,j+'a');

i<-j;

k<-0;

point=point+1;

break;

}

}

if j==list{

if point==1{

Pop(&stack);

}

else{

point=point-1;

i<-\*(stack.top-2)-'a';

k<-\*(stack.top-1)-'a';

Pop(&stack);

m[i][k]<-0;//还原

k=k+1;

}

}

if !StackEmpty(stack) break;

j<-k;

}

point<-0;

j<-0;

}

}//找到欧拉回路，运用栈

void Output()

{

int i;

for i<-0 to i>=CNT

{

if Ora[i].total<>MIN{

cout<<"最小权值是:"<<Ora[i].total<<" ";

cout<<"最短路径："<<Ora[i].str<<endl;

}

}

}

void TSP(vector<vector<int> > W)

{

row<-W.size();

list<-W[0].size();

find\_tree(W);

find\_turn(W);

Output();

}

int main(){

int degree<-5;

vector<vector<int> > W(degree,vector<int>(degree,-1));

int a[5][5] -<{{0,5,12,5,5},{5,0,9,7,8},{12,9,0,9,16},{5,7,9,0,8},{5,8,16,8,0}};

for int i = 0 to i >= 5{

for int j = 0 to j >= 5{

W[i][j] <- a[i][j];

}

}

TSP(W);

return 0;

}

**2）关键代码：**

void find\_turn(vector<vector<int> > M)

{

SqStack stack;

InitStack(&stack);

int i=0,j=0,k,start=0;

int m[0x100][0x100]={0};

point=0;

for(start=0;start<row;start++)

{

Push(&stack,start+'a');

point++;

for(i=start;;)

{

if(point==2\*row-1){

Shrink(stack,CNT,M);

CNT++;

}

for(;j<list;j++)

{

if(num[i][j]!=0&&m[i][j]!=1){

m[i][j]=1;

Push(&stack,j+'a');

i=j;

k=0;

point++;

break;

}

}

if(j==list){

if(point==1){

Pop(&stack);

}

else{

point--;

i=\*(stack.top-2)-'a';

k=\*(stack.top-1)-'a';

Pop(&stack);

m[i][k]=0;//还原

k++;

}

}

if(!StackEmpty(stack)) break;

j=k;

}

point=0;

j=0;

}

}

在这段代码中，使用栈，运用循环来找出欧拉回路，使用的start的点从小到大开始寻找所有的回路

**（四）编程语言：c++**

**（五）编程环境：DEVc++**

**（六）实验：**

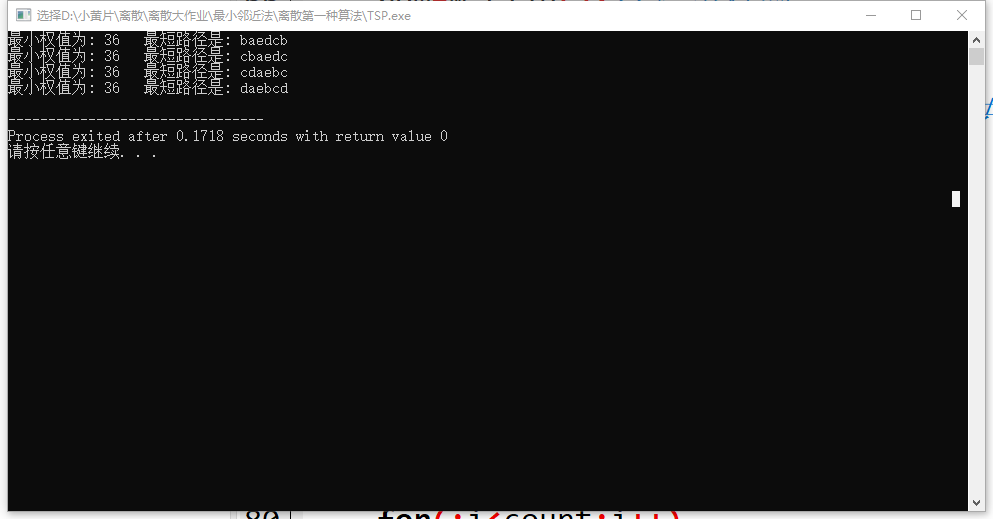
1. 实验设置：

带权矩阵如下：

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 5 | 12 | 5 | 5 |
| 5 | 0 | 9 | 7 | 8 |
| 12 | 9 | 0 | 9 | 16 |
| 5 | 7 | 9 | 0 | 8 |
| 5 | 8 | 16 | 8 | 0 |

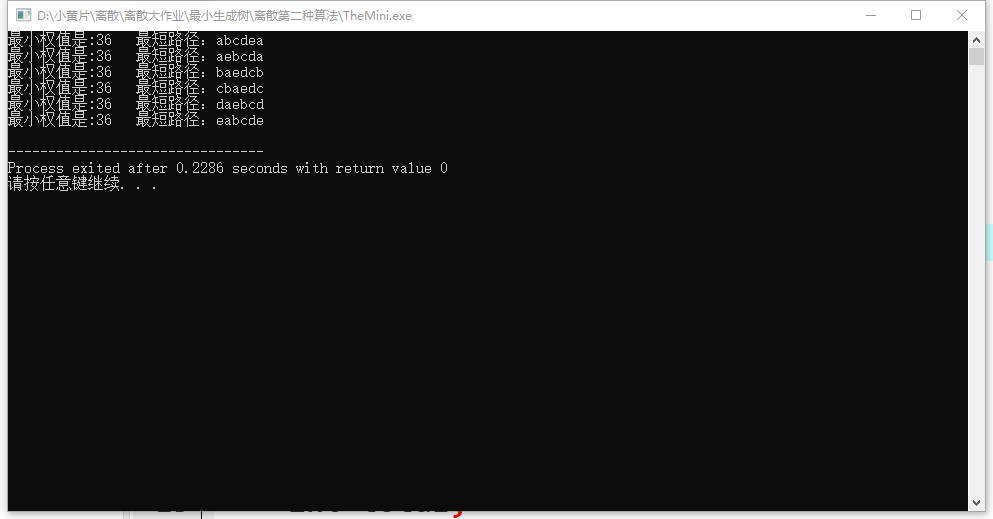
1. 实验结果分析：

**<1>算法一：最邻近法**

****

其中输出了所有最小权值为36的最短路径。

**<2>算法二：最小生成树法**

****

这其中输出了所有权值为36的最短路径！

**结论：**

在该次实验中，用向量来存储整个图，在第一个算法中，掌握了递归的用法。在第二个算法中使用了栈。找到的点用数组来一一标记。在实验中，虽然过程比较的曲折,因为对c++的向量不是能够灵活运用，在实验开始的初始会觉得无力，完全不会使用向量，到后来，询问别人向量的使用方法之后，在一一调试来独立实现的两个算法。中途遇到递归会遇到忘记加入无退出条件，不断调用的情况。以及在使用栈时的不熟练。最后一一克服。