实验4  贪心算法实现

一、实验目标：

1． 熟悉贪心算法实现的基本方法和步骤；

2． 学会贪心算法的实现方法和分析方法：

二、实验内容

1. Huffman编码：

测试数据：

X={zhejiang university of technology, computer college}

Y={1310773597218806522025}

...请就上述问题谈谈你对知识产权或法律方面的理解。

知识产权，也称其为"知识所属权"，指"权利人对其智力劳动所创作的成果享有的财产权利"，一般只在有限时间内有效。各种智力创造比如发明、外观设计、文学和艺术作品，以及在商业中使用的标志、名称、图像，都可被认为是某一个人或组织所拥有的知识产权。

知识产权从本质上说是一种无形财产权，他的客体是智力成果或是知识产品，是一种无形财产或者一种没有形体的精神财富，是创造性的智力劳动所创造的劳动成果。它与房屋、汽车等有形财产一样，都受到国家法律的保护，都具有价值和使用价值。有些重大专利、驰名商标或作品的价值也远远高于房屋、汽车等有形财产。

...请你就此，结合你了解的相关知识，谈谈计算机算法对社会和文化所带来的影响。

计算机与我们息息相关，我们利用它办公，学习， 娱乐，这正是我们当今社会生活所必需的，很难想象没有计算机的存在，我们将怎样传递信息、及时了解世界的变化，我们又何谈走向太空，建立空间站。计算机算法使很多生活中的复杂问题简单化、逻辑化、数学化。就拿Haffman编码来说，它使文字转化为机器方便识别的二进制数，在数据量很大时，机器可以快速地编码译码，通过Haffman编码可以隐晦地传递信息，著名的摩尔斯电码和五笔一级简码就是运用了这种原理。

2.最小生成树：

Prim算法。

Kruskal算法。

测试数据：

1.  0 2 8 1 0 0 0 0

2.  2 0 6 0 1 0 0 0

3.  8 6 0 7 5 1 2 0

4.  1 0 7 0 0 0 9 0

5.  0 1 5 0 0 3 0 8

6.  0 0 1 0 3 0 4 6

7.  0 0 2 9 0 4 0 3

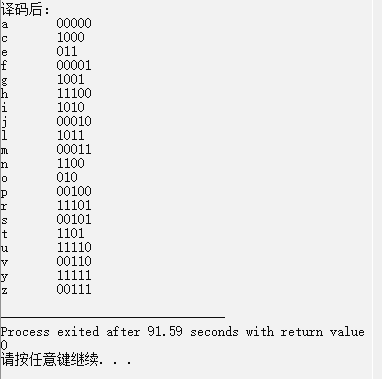
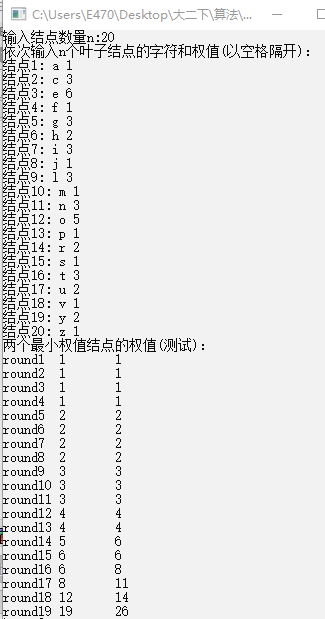
8.  0 0 0 0 8 6 3 0

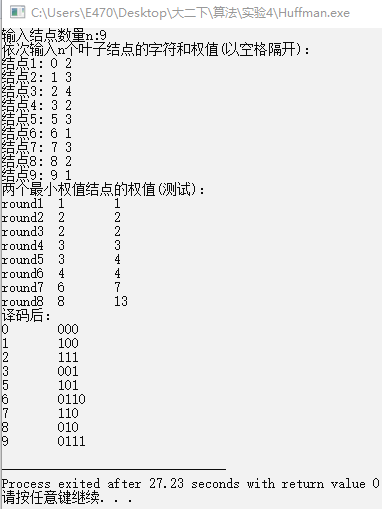
其中矩阵中的数据为节点之间的距离(权值)。

1. 实验结果

1. Huffman编码：

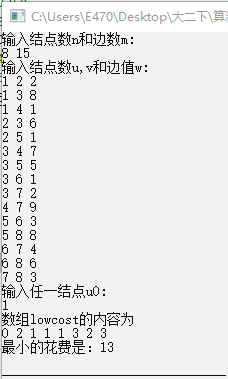
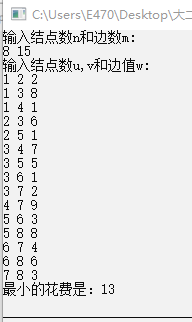
X:



Y:

1. 最小生成树：

Prim算法： Kruskal算法：

1. 源代码

1. Huffman编码：

#include<iostream>

using namespace std;

#define MAXBIT 10

#define MAXVALUE 100

class Node{/\* 结点结构体 \*/

public:

double w;//weight

int p;//parent

int l;//lchild

int r;//rchild

char v;//value

};

class Code{/\* 编码结构体 \*/

public:

int code[MAXBIT];

int start;

};

/\* 构造哈夫曼树 \*/

void HuffmanTree (Node Node[],int n){

/\* w1、w2：构造哈夫曼树不同过程中两个最小权值结点的权值

i1、i2：构造哈夫曼树不同过程中两个最小权值结点在数组中的序号\*/

int i, j, i1, i2;

double w1,w2;

/\* 初始化存Node[]中的结点 \*/

for (i=0; i<2\*n-1;i++){

Node[i].w=0;

Node[i].p=-1;

Node[i].l=-1;

Node[i].r=-1;

}

cout<<"依次输入n个叶子结点的字符和权值(以空格隔开)："<<endl;

for (i=0; i<n; i++){

cout<<"结点"<<i+1<<": ";

cin>>Node[i].v>>Node[i].w;

}

cout<<"两个最小权值结点的权值(测试)："<<endl;

/\* 构造 Huffman 树 \*/

for (i=0; i<n-1; i++){//执行n-1次合并

w1=w2=MAXVALUE;

/\* w1、w2中存放两个无父结点且结点权值最小的两个结点 \*/

i1=i2=0;

/\* 找出所有结点中权值最小、无父结点的两个结点,合并之为一棵二叉树 \*/

for (j=0;j<n+i;j++){

if (Node[j].w<w1&&Node[j].p==-1){

w2=w1;

i2=i1;

w1=Node[j].w;

i1=j;

}

else if(Node[j].w<w2&&Node[j].p==-1){

w2=Node[j].w;

i2=j;

}

}

/\* 设置找到的两个子结点 i1、i2的父结点信息 \*/

Node[i1].p=n+i;

Node[i2].p=n+i;

Node[n+i].w=w1+w2;

Node[n+i].l=i1;

Node[n+i].r=i2;

cout<<"round"<<i+1<<"\t"<<Node[i1].w<<"\t"<<Node[i2].w<<endl; /\* 用于测试 \*/

}

}

/\* 哈夫曼树编码 \*/Code cd;//为什么这个放函数里面就会编译错误 放外面反而对？？

void HuffmanCode(Code Code[],Node Node[],int n){

//Code cd;/\* 定义一个临时变量来存放求解编码时的信息 \*/

int i,j,c,p;

for(i=0;i<n;i++){

cd.start=n-1;

c=i;

p=Node[c].p;

while(p!=-1){

if(c==Node[p].l) cd.code[cd.start]=0;

else cd.code[cd.start]=1;

cd.start--; /\*前移一位 \*/

c=p;

p=Node[c].p; /\* 设置下一循环条件 \*/

}

/\* 把叶子结点的编码信息从临时编码cd中复制出来，放入编码结构体数组 \*/

for (j=cd.start+1; j<n; j++) Code[i].code[j]=cd.code[j];

Code[i].start=cd.start;

}

}

int main(){

int i,j,k,n;

cout<<"输入结点数量n:";

cin>>n;

Node Node[2\*n-1];

Code Code[n];

HuffmanTree(Node,n);//构造哈夫曼树

HuffmanCode(Code,Node,n);//编码

cout<<"译码后："<<endl;

for(i=0;i<n;i++){//译码

cout<<Node[i].v<<"\t";

for(j=Code[i].start+1;j<n;j++) cout<<Code[i].code[j];

cout<<endl;

}

return 0;

}

2.最小生成树：

Prim算法：

#include <iostream>

using namespace std;

const int INF = 0x3fffffff;

const int N = 100;

bool s[N];

int closest[N];

int lowcost[N];

void Prim(int n, int u0, int c[N][N])

{ //顶点个数n、开始顶点u0、带权邻接矩阵C[n][n]

//如果s[i]=true,说明顶点i已加入最小生成树

//的顶点集合U；否则顶点i属于集合V-U

//将最后的相关的最小权值传递到数组lowcost

s[u0]=true; //初始时，集合中U只有一个元素，即顶点u0

int i;

int j;

for(i=1; i<=n; i++){

if(i!=u0){

lowcost[i]=c[u0][i];

closest[i]=u0;

s[i]=false;

}

Else lowcost[i]=0;

}

for(i=1; i<=n;i++){ //在集合中V-u中寻找距离集合U最近的顶点t

int temp=INF;

int t=u0;

for(j=1;j<=n;j++)

{

if((!s[j])&&(lowcost[j]<temp))

{

t=j;

temp=lowcost[j];

}

}

if(t==u0)

break; //找不到t，跳出循环

s[t]=true; //否则，讲t加入集合U

for(j=1; j<=n;j++) //更新lowcost和closest

{

if((!s[j])&&(c[t][j]<lowcost[j]))

{

lowcost[j]=c[t][j];

closest[j]=t;

}

}

}

}

int main(){

int n, c[N][N], m, u, v, w;

int u0;

cout<<"输入结点数n和边数m:"<<endl;

cin>>n>>m;

int sumcost=0;

for(int i=1; i<=n; i++)

for(int j=1; j<=n; j++)

c[i][j]=INF;

cout <<"输入结点数u,v和边值w:"<<endl;

for(int i=1; i<=m; i++)

{

cin>>u>>v>>w;

c[u][v]=c[v][u]=w;

}

cout <<"输入任一结点u0:"<<endl;

cin >> u0 ;

//计算最后的lowcos的总和，即为最后要求的最小的费用之和

Prim(n, u0, c);

cout <<"数组lowcost的内容为"<<endl;

for(int i = 1; i <= n; i++)

cout << lowcost[i] << " ";

cout << endl;

for(int i = 1; i <= n; i++)

sumcost += lowcost[i];

cout << "最小的花费是："<<sumcost<<endl;

return 0;

}

Kruskal算法：

#include <iostream>

#include <algorithm>

using namespace std;

const int N = 100;

int father[N];

int n, m;

struct Edge {

int u;

int v;

int w;

}e[N\*N];

bool comp(Edge x, Edge y) {

return x.w < y.w;

}

void Init(int n){

for(int i = 1; i <= n; i++)

father[i] = i;

}

int Find(int x){

if(x != father[x])

father[x] = Find(father[x]);

return father[x];

}

int Merge(int a, int b){

int p = Find(a);

int q = Find(b);

if(p==q) return 0;

if(p > q)

father[p] = q;//小的赋值给大的集合号

else

father[q] = p;

return 1;

}

int Kruskal(int n){

int ans = 0;

for(int i=0;i<m;i++)

if(Merge(e[i].u, e[i].v))

{

ans += e[i].w;

n--;

if(n==1)

return ans;

}

return 0;

}

int main() {

cout <<"输入结点数n和边数m:"<<endl;

cin >> n >> m;

Init(n);

cout <<"输入结点数u,v和边值w:"<<endl;

for(int i=1;i<=m;i++)

cin>>e[i].u>>e[i].v>>e[i].w;

sort(e, e+m, comp);

int ans = Kruskal(n);

cout << "最小的花费是：" << ans << endl;

return 0;

}

1. 实验体会

Haffman编码

我的代码算法时间复杂度为O(n2),空间复杂度为O(n\*MAXBIT).

算法优化可从2方面入手:

①HaffmanTree()中找2个权值最小的结点时使用优先队列,时间复杂度变为O(nlogn)；

②HaffmanCode()中Node[]定义一个动态分配空间的链表来存储编码,节省空间.

最小生成树

Prim算法(破圈法)：从任意结点出发；当前形成的集合始终是一棵树；时间复杂度与图的边数无关，适用于稠密图。(不断去掉回路中权最大的边)

Kruskal算法(避圈法)：从权最小的边出发；当前形成的集合除了结果外通常不是一棵树；时间复杂度取决于图的边数，适用于稀疏图。(n个点 n-1个边)

①时间：Prim-O(n2)、 Kruskal-O(eloge)

②空间：Prim每次只扫描与当前结点集到U集合的最小边，空间小；Kruskal需对所有边排序，对于大型图占用空间较大.