

## 回溯法解决旅行售货员(TSP)问题

班级：2015211312

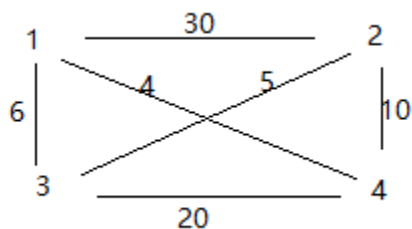
学号：2015211484

姓名：刘佳鑫

### ● 问题描述：

某个售货员要到若干城市去推销商品，已知各城市之间的路程（或旅费）。他要选定一条从驻地城市出发，经过每个城市一遍，最后回到驻地的路线，使总的路程（或总旅费）最小。

如：正确答案应该是 1->3->2->4->1，最少路费为 25.



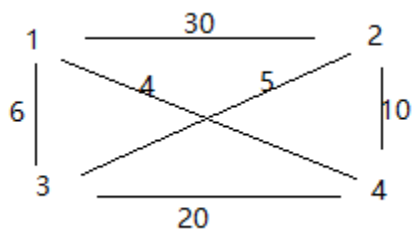
### ● 算法描述及思考过程：

用回溯法解决，解空间是排列树。

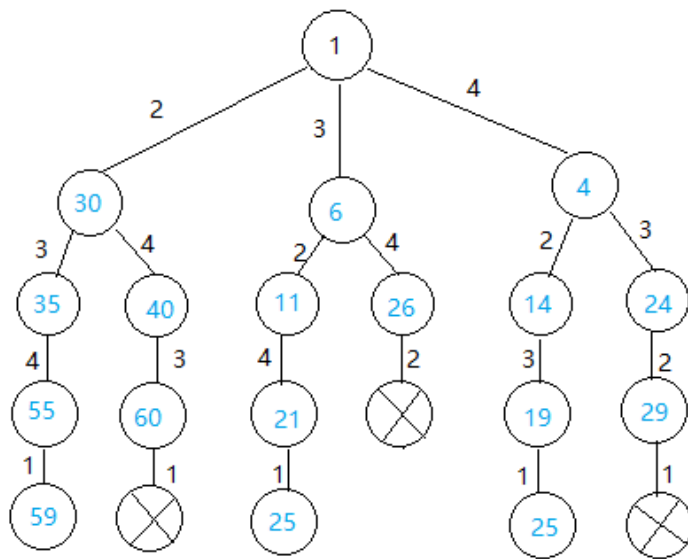
而剪枝的条件是，如果不存在这条路径（从  $x[n-1]$  到  $x[n]$  或是从  $x[1:n]$ ），或者当前  $x[1:n]$  的费用大于了当前最优值，则剪枝。

所以当路径存在，且费用小于最优值时，选取这条路径。

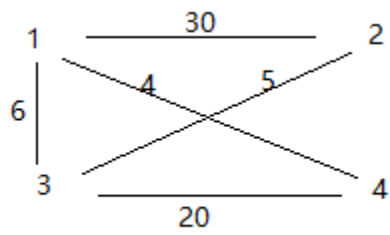
例 1：



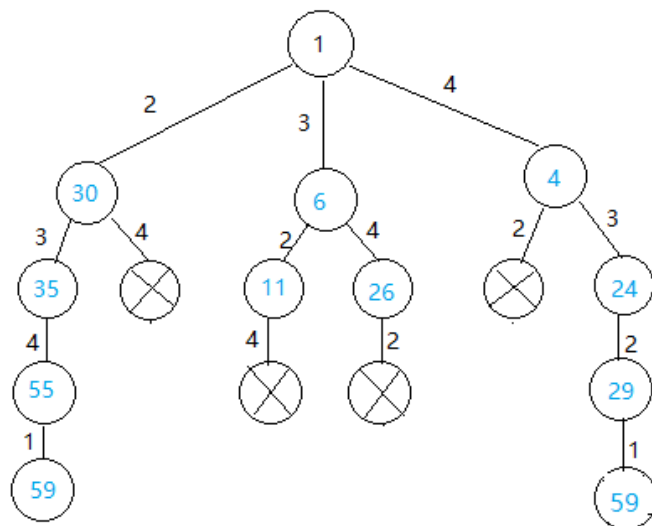
剪枝后的解空间树为：



例 2：



剪枝后的解空间树为：



● 源代码：

```
//解空间是排列树
#include<iostream>
#include<limits>//为了声明无穷
using namespace std;
float floatmax=numeric_limits<float>::max();//floatmax 则代表无穷

class BTTSP
{
    friend void TSP(float **a,int n);

    float **a; //图 G 的邻接矩阵
    int n;//图 G 的顶点数
    int *x;//当前解
    int *bestx;//当前最优解
    float bestc;//当前最优值
    float cc;//当前费用

private:
    void backtrack(int i);

};

void BTTSP::backtrack(int i)
{
    if(i==n)//是叶子结点
    {
        if (a[x[n-1]][x[n]]<floatmax&& a[x[n]][1]<floatmax)//存在路径
        {
            if(cc+a[x[n-1]][x[n]]+a[x[n]][1]<bestc)//加上该条，值小于最优解
            {
                bestc=cc+a[x[n-1]][x[n]]+a[x[n]][1];//最优解更新
                for(int j=1;j<=n;j++)//最优路径更新
                {
                    bestx[j]=x[j];
                }
            }
        }
    }
    else//不是叶子结点，回溯（排列树）
```

```

{
    for(int j=i;j<=n;j++)
    {
        if (a[x[i-1]][x[j]]<floatmax&&cc+a[x[i-1]][x[j]]<bestc)
        {
            swap(x[i],x[j]); //swap 是系统函数 !
            cc+=a[x[i-1]][x[i]];
            backtrack(i+1);
            cc-=a[x[i-1]][x[i]];
            swap(x[i],x[j]);
        }
    }
}
}
}

```

void TSP(float \*\*A,int m)//初始化以及调用回溯

```

{
    BTTSP p;

    p.n=m;
    p.x= new int[m+1];
    p.bestx=new int[m+1];

    for(int j=1;j<=m;j++)
    {
        p.x[j]=j;
    }

    p.a=A;
    p.cc=0;
    p.bestc=floatmax;
    //搜索 x[2:n]的全排列
    p.backtrack(2);

    cout<<"应走路径为："<<endl;
    for(int j=1;j<=m;j++)
    {
        cout<<p.bestx[j]<<"\t";
    }
    cout<<p.bestx[1]<<endl;//回路又回到起点

    cout<<"最少路费为："<<endl;
}

```

```

        cout<<p.bestc<<endl;

        delete []p.x;
        p.x=0;
        delete []p.bestx;
        p.bestx=0;

    }

    main()
    {
        cout<<"请输入城市个数"<<endl;
        int n;
        cin>>n;
        float **A=new float *[n+1];
        for(int j=0;j<=n;j++)
        {
            A[j]=new float[n+1];
        }

        cout<<"请按顺序输入邻接矩阵,同城市请输入-1, 不可达也请输入-1"<<endl;
        for(int i=1;i<=n;i++)
        {
            for(int j=1;j<=n;j++)
            {
                cin>>A[i][j];
            }
        }
        cout<<"邻接矩阵为"<<endl;
        for(int i=1;i<=n;i++)
        {
            for(int j=1;j<=n;j++)
            {
                cout<<A[i][j]<<" ";
            }
            cout<<endl;
        }

        for(int i=1;i<=n;i++)
        {
            for(int j=1;j<=n;j++)
            {
                if (A[i][j]==-1&&i!=j)

```

```

        {
            A[i][j]=floatmax;//把不可达设为距离为无穷
        }
    }

}

TSP(A,n);

for(int j=0;j<=n;j++)
{
    delete []A[j];
}
delete []A;

A=0;
system("pause");
return 0;
}

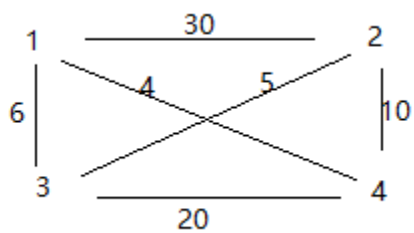
```

## ● 运行结果及实验结果说明：

例子 1.

四个城市间都可以相互到达，如图。

正确答案应该是 1->3->2->4->1，最少路费为 25.



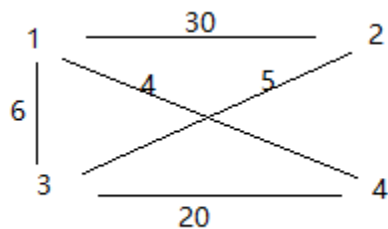
运行结果：

```
E:\大二\大二下\算法设计与分析 代码\回溯\TSP.exe
4
5 请输入城市个数
6 4
7 请按顺序输入邻接矩阵, 同城市请输入-1, 不可达也请输入-1
8 -1 30 6 4
9 30 -1 5 10
0 6 5 -1 20
1 4 10 20 -1
2 邻接矩阵为
3 -1 30 6 4
4 30 -1 5 10
5 6 5 -1 20
6 4 10 20 -1
7 应走路径为:
8 1 3 2 4 1
9 最少路费为:
0 25
  请按任意键继续. . .
```

### 例子 2

若城市 2,4 不通, 如图。

正确结果应为 1->2->3->4->1,最少费用为 59.



运行结果如图：

```
E:\大二\大二下\算法设计与分析 代码\回溯\TSP.exe
请输入城市个数
4
请按顺序输入邻接矩阵, 同城市请输入-1, 不可达也请输入-1
-1 30 6 4
30 -1 5 -1
6 5 -1 20
4 -1 20 -1
邻接矩阵为
-1 30 6 4
30 -1 5 -1
6 5 -1 20
4 -1 20 -1
应走路径为:
1      2      3      4      1
最少路费为:
59
请按任意键继续. . .
```

## ● 遇到的问题及解决方法：

在思考算法时，路径不存在则是他们之中距离为无穷，但是如何表示无穷则出现问题。  
解决方法：

c++ 的 float 类型包含的最值问题.... - chenyu964877814 的专栏 - 博客频道 - CSDN.NET <http://blog.csdn.net/chenyu964877814/article/details/7457123>





方法二：

头文件：`#include <limits>`

定义方式：`float floatMax = numeric_limits<float>::max();`

如下例：（借用网友的.....）

```
[html]    
01. #include <iostream>  
02. #include <limits>  
03.  
04. using namespace std;  
05.  
06. int main()  
07. {  
08.     int intMax = numeric_limits<int>::max();  
09.     int intMin = numeric_limits<int>::min();  
10.  
11.     float floatMax = numeric_limits<float>::max();  
12.     float floatMin = numeric_limits<float>::min();  
13. }
```

## ● 实验总结：

因为有了上课给的排列树的基本模板，所以只用自己考虑剪枝条件便能很快编出程序。自己的程序参考的书上给的程序基本成型后，我又从网上看了下别人编的程序，大概思路是一样的，但是我发现我漏了好多 delete，这还是属于 c++ 掌握的不扎实。以及友元函数的使用，也是又重新了解了下。同时还纠结了下用不用起点也要分情况讨论下，因为老师的 PPT 上给的解空间树把起点也分情况讨论了，但是后来发现，因为最终又会回到起点，所以类似一个循环，所以起点是哪个无所谓，所以就把起点固定为 1 了。