旅行售货员

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。

实验要求

基于回溯法实现旅行售货员问题。（选做，结合贪心策略，实现最优剪枝）
要求：画出解空间树，写出剪枝策略并再树上标出剪枝点，编程实现算法。

算法设计

回溯思想

以深度优先的方式，从树根结点开始，依次扩展树结点，直到达到叶结点——搜索过程中动态产生解空间。在生成解空间树时，有以下三种节点：

- 扩展结点:一个正在产生儿子的结点
- 活结点:一个自身已生成但其儿子还没有全部生成的结点
- 死结点:一个所有儿子已经产生的结点

回溯法从根结点出发，按照深度优先策略遍历解空间树，搜索满足约束条件的解。

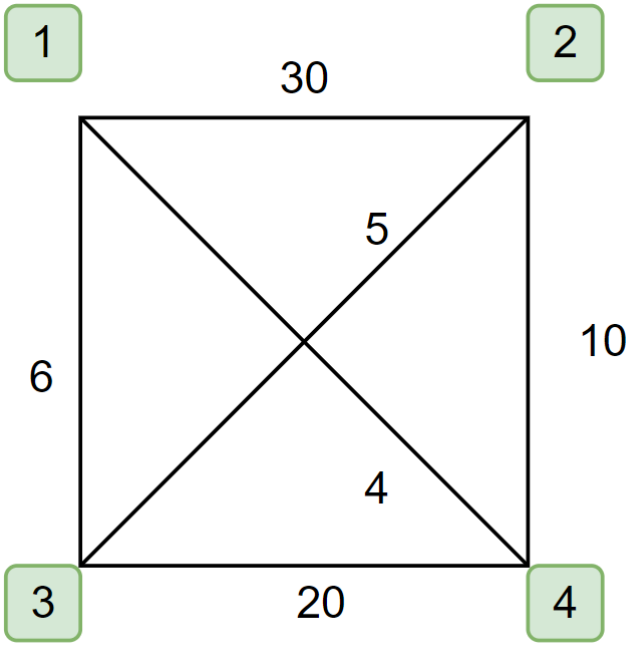
初始时，根结点成为一个活结点，同时也称为当前的扩展结点。

在当前扩展结点处，搜索向**纵深**方向移至一个新结点。这个新结点成为一个新的活结点，并成为当前的扩展结点。

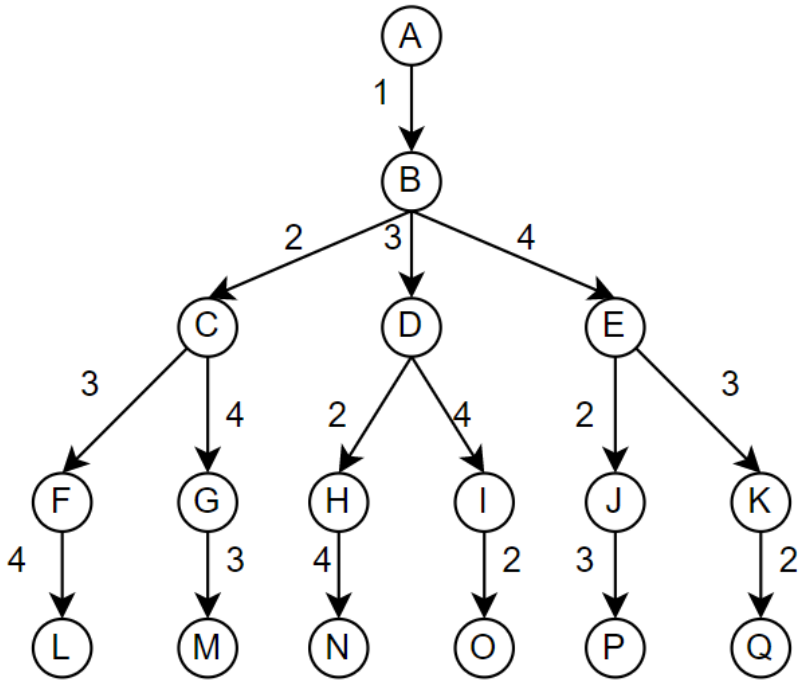
如果在当前的扩展结点处不能再向深方向移动，则当前的扩展结点就成为一个死结点。此时，应往回移动回溯至最近的一个活结点处，并使这个活结点成为当前的扩展结点。

回溯法以这种工作方式递归地在解空间中搜索，直至找到所要求的解或解空间中已无活结点时为止。

使用“旅行售货员.pptx”中的四城市图为例：



画出解空间树：



共有6条周游路线：

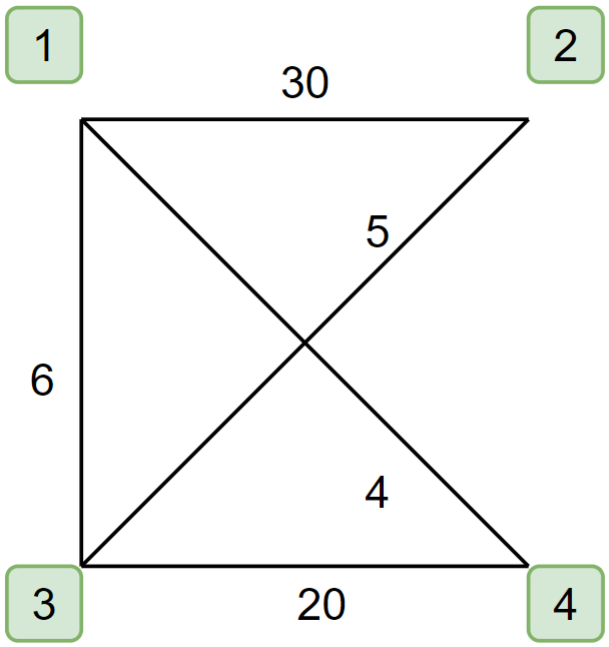
路线	开销
(1,2,4,3,1)	66
(1,2,3,4,1)	59
(1,3,2,4,1)	25
(1,3,4,2,1)	66
(1,4,2,3,1)	25
(1,4,3,2,1)	59

剪枝策略

剪枝策略1

如果当前正在考虑的顶点 j 与当前路径中的末端结点 i 没有边相连，即 $w(i,j)=\infty$, 则不必搜索 j 所在分支

若去除路线图中连接2、4的路径，即令 $w(2,4)=w(4,2)=\infty$



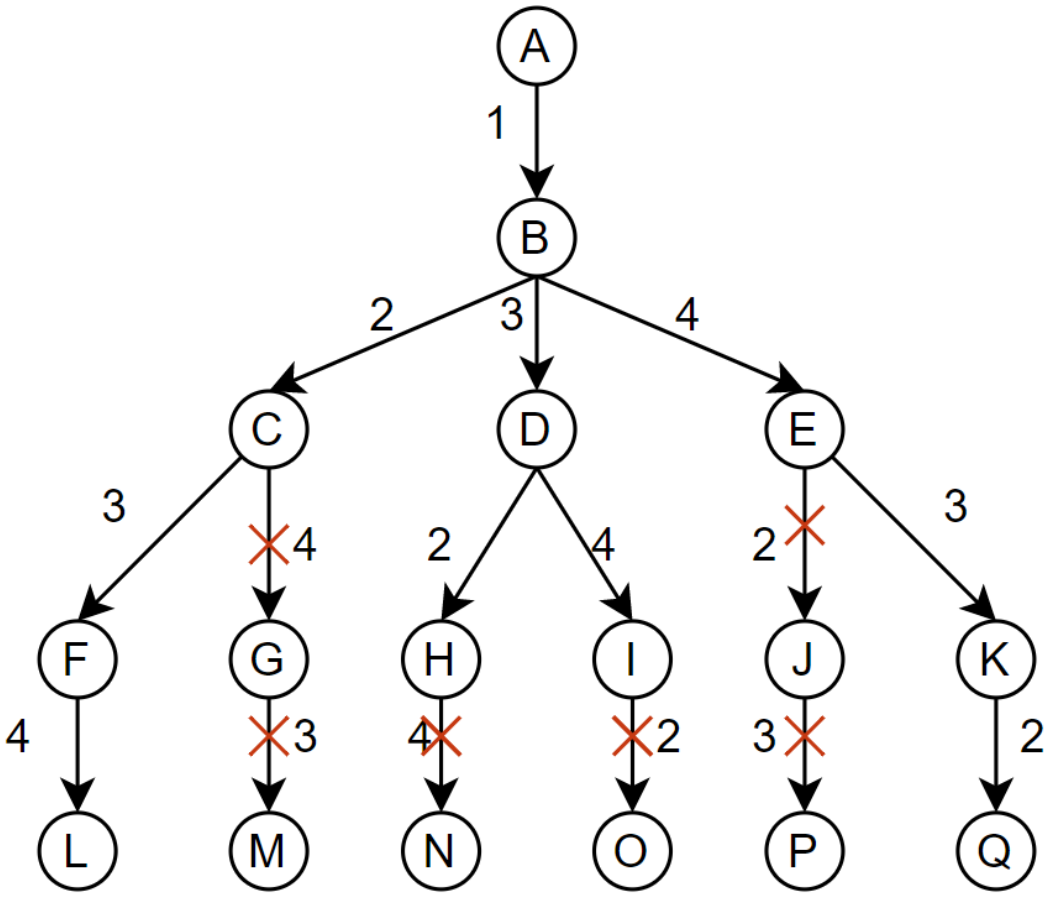
当前已有的部分路径为 $<1,2,?,?>$,路径末端结点为2;

根据路径组成规则，下一步可考虑将顶点3、4加入到部分路中。

但是，顶点2与4间无边，$w(2,4)=\infty$，因此在解空间树中，可以不必考虑顶点4所在分支

同理，对于$<1,3,2,?>$，$<1,3,4,?>$，$<1,4,?,?>$，也有类似的剪枝过程

在解空间树中表示剪枝如下：



剪枝策略2

由于我们生成解空间树的方式为深度优先搜索，因此在搜索过程中是可以得一些完整的可行解的。

用bestw表示当前得到的所有可行解中，在前面的搜索中，从其它已经搜索过的路径中，找到的最佳完整回路的权和（总长度）

如果我们在搜索某条路径时，它的部分权和已经超过了bestw，那最终搜索出这条路径的完整回路权和必然大于bestw，因此他一定不比bestw对应的路径优，我们不必往下搜索。

令到第i层结点为止，构造的部分解路径为
$< 1, x[2], x[3], \ldots, x[i-1], x[i], ?, ?, ? >$

我们用 cw 表示路径的权值总和

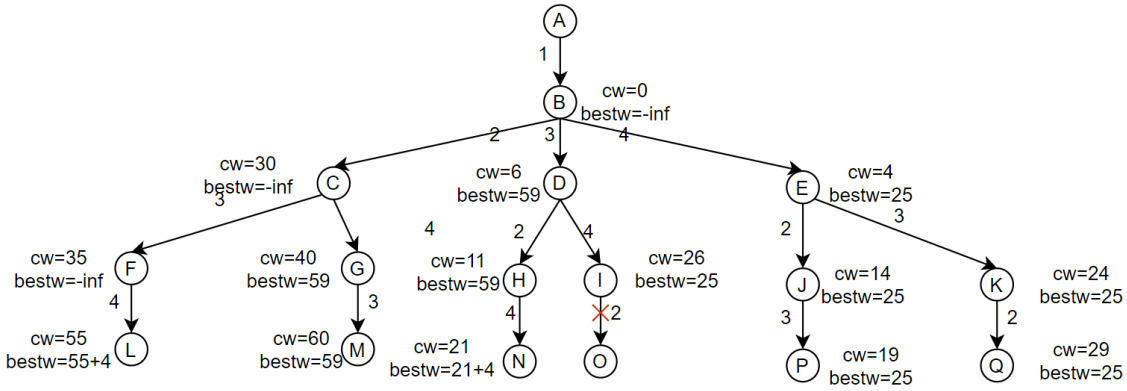
$$cw[i] = \sum_{j=2}^{i} w(x[j-1], x[j]) \quad 且 x[1] = 1$$

则当

$$cw(i-1) + w(x[i-1], x[i]) >= bestw$$

时，停止搜索x[i]分支及其下面的层。

画出剪枝后的解空间树：



得到该问题最优解: <1,3,2,4,1>, <1, 4,2, 3,1>，对应的bestw=25.

贪心策略

在上述过程中我们看到，bestw在搜索到第一个可行解时为59，而真正的bestw=25在很后面的位置才得到，这就导致前期我们拿着一个较大的bestw搜索了很多无用分支，剪枝效果并不理想。为了避免这个问题，我们可以在一开始用贪心的策略，从节点1开始，每次在尚未走过的节点中，选择一个路径开销最小的形成路径，得到一个贪心策略下的bestw。它虽然不是真正的bestw=25，但能在搜索前期剪掉更多的分支。

在本例中，贪心求解路径的过程如下：

路径矩阵为

$$\begin{bmatrix} \infty & 30 & 6 & 4 \\ 30 & \infty & 5 & 10 \\ 6 & 5 & \infty & 20 \\ 4 & 10 & 20 & \infty \end{bmatrix}$$

1. $w(1) = [\infty \quad 30 \quad 6 \quad 4]$

   选择4，解路径为$< 1, 4 >$，cw=4

2. $w(4) = [(4) \quad 10 \quad 20 \quad \infty]$

   选择10，解路径为$< 1, 4, 2 >$，cw=4+10=14

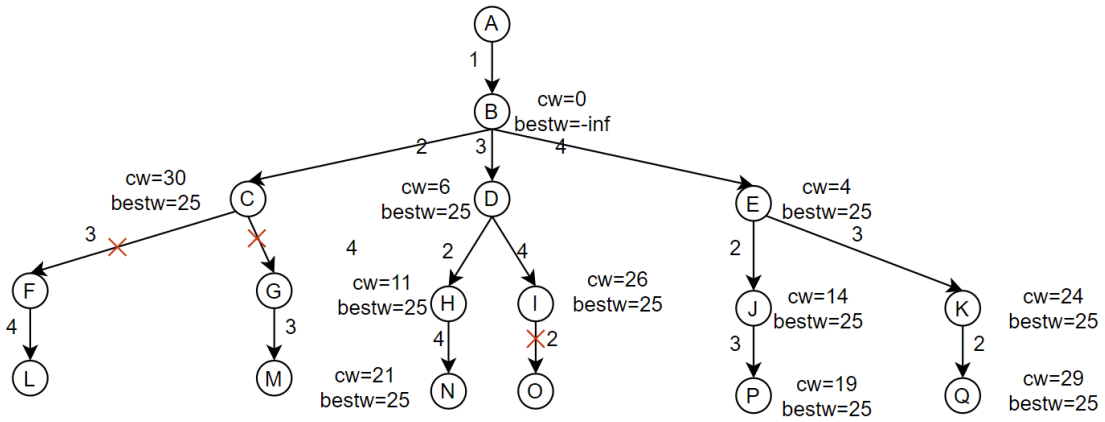
3. $w(2) = [(30) \quad \infty \quad 5 \quad (10)]$

   选择5，解路径为$< 1, 4, 2, 3 >$，cw=14+5=19

4. $w(3) = [6 \quad 5 \quad \infty \quad 20]$

   必须回到1号节点，解路径为$< 1, 4, 2, 3, 1 >$，bestw=19+6=25

用初始bestw=25进行剪枝：



可以看到剪枝效果显著

预估路径

我们在剪枝策略2中，只比较了当前走过的路径的权值之和与bestw的大小。事实上当前路径最终生成的完成回路的权值必然是大于当前这个部分路径的权值的，如果能预估之后路径权值大小，讲其加到当前路径上，则比较不等式变为：

$$cw(i-1)+w(x[i-1],x[i])+rest >= bestw$$

其中，rest为我们**预估接下来还要走的路径权值。**

显然，rest有很多个值，我们需要求出的是它的下界。

考虑当前解路径$<1,x[2],x[3],\ldots,x[i-1],x[i],?,?,?>$，?中一定是在$x[2],x[3],\ldots,x[i-1],x[i]$没有出现过的点，也就是说，这些点一定要经过一次。那么，rest的下界就是对于这些点所连接的所有路径，经过他们的时候都走了最小的那一条（当然，这种情况不一定能实现）。但我们能够保证不能找到一条比rest权值更小的线路，因此，rest就是我们需要的预估下界。

特别的，最后一定要回到1号结点，因此rest中也要加上1号结点连接的路径中最小的一条。

则，

$$rest=\sum_{i\notin X} Min\{w(i)\}+Min\{w(1)\}$$

由此，我们得到了剪枝策略2的优化版本：

当

$$cw(i-1)+w(x[i-1],x[i])+rest >= bestw$$

时，停止搜索x[i]分支及其下面的层。

此方法对于层数较高的树剪枝效果明显，对于本例不明显，故不再画出解空间树。

算法实现

初始化

```
void init()
{
    cin >> N;
    FOR(i, 1, N)
    {
```

```
        Wmin[i] = INF;
        FOR(j, 1, N)
        {
            cin >> W[i][j];
            if (i == j)
                W[i][j] = INF;
            else
                Wmin[i] = gmin(W[i][j], Wmin[i]);
        }
    }
    FOR(i, 1, N) X[i] = i;
    bestw = greedy();
    cout << "Initial Greedy:";
    FOR(i, 1, N)
        cout << bestX[i] << " ";
    cout << "bestw = " << bestw << endl;
}
```

贪心策略

```
int greedy()
{
    int val = 0, now = 1;
    int V[MAXN + 10] = { 0 };    //V代表已经经过的节点
    V[1] = 1;          //初始只经过1
    bestX[1] = 1;
    FOR(i, 2, N)
    {
        int to, Min = INF;
        FOR(j, 1, N)     //遍历还未访问过的节点，寻找最小权值路径
            if (W[now][j] < Min && !V[j])
            {
                Min = W[now][j];
                to = j;
            }
        bestX[i] = to;
        V[to] = 1;        //标记转移节点
        now = to;         //转移到该节点
        val += Min;       //加上权值
```

```
    }
    return val + W[now][1]; //最后还要再经过一次结点1
}
```

回溯过程

```
void BacktrackTSP(int i, int cw)
{
    if (i == N)      //已经搜索到叶节点，即到达最后1个城市
    {
        if (W[X[N - 1]][X[N]] != INF && W[X[N]][1] != INF)
 //最后1个城市与与第1个城市相连
        {
            cw = cw + W[X[N - 1]][X[N]] + W[X[N]][1];
            if (cw < bestw) //当前回路更优，更新最优搜索结果
            {
                cout << "Update:";
                bestw = cw;

                FOR(i, 1, N)//更新路径
                {
                    bestX[i] = X[i];
                    cout << bestX[i] << " ";
                }
                cout << "bestw =" << bestw << endl;
            }
        }
        return;
    }
    FOR(j, i, N)
    {
        swap(X[i], X[j]);
        int rest = Rest(i); //求出剩余节点的预估下界
        if (cw + W[X[i - 1]][X[i]] + rest >= bestw)
        {
            swap(X[i], X[j]);   //撤回此序列
            continue;
        }
```

```
            BacktrackTSP(i + 1, cw + W[X[i - 1]][X[i]]);
            swap(X[i], X[j]);    //回溯
        }
    }
```

计算预估路径

```
int Rest(int i)
{
    int val = 0;
    FOR(j, i + 1, N)     //遍历还未经过的结点
        val += Wmin[X[j]];        //加上连接它的最小权值路径
    return val + Wmin[1];         //最后还要再经过一次结点1
}
```

算法分析

如果不考虑更新bestw所需的计算时间，则算法backtrack需要 $O((n-1)!)$ 计算时间。由于算法backtrack在最坏的情况下可能需要更新当前最优解 $O((n-1)!)$ 次，每次更新bestw需 $O(n)$ 计算时间，从而整个算法的计算时间复杂性为O(n!)。

算法测试

输入

```
4
0 5 7 2
5 0 1 2
7 1 0 10
2 2 10 0
```

输出

```
Initial Greedy:1 4 2 3 bestw = 12
1 4 2 3 1
12
```

输入

```
10
0 1 2 10 1 2 2 10 3 9
1 0 4 9 6 1 3 8 4 2
2 4 0 10 7 2 8 4 2 5
10 9 10 0 9 6 6 9 1 5
1 6 7 9 0 10 5 4 4 2
2 1 2 6 10 0 6 2 10 6
2 3 8 6 5 6 0 7 10 8
10 8 4 9 4 2 7 0 4 5
3 4 2 1 4 10 10 4 0 2
9 2 5 5 2 6 8 5 2 0
```

输出

```
Initial Greedy:1 2 6 3 9 4 10 5 8 7 bestw = 27
Update:1 2 3 6 8 5 10 9 4 7 bestw =26
Update:1 2 6 3 8 5 10 9 4 7 bestw =25
Update:1 2 10 5 8 6 3 9 4 7 bestw =24
Update:1 5 10 4 9 3 8 6 2 7 bestw =23
1 5 10 4 9 3 8 6 2 7 1
23
```

输入

```
20
0 1 2 6 3 8 7 3 7 5 4 7 7 10 3 3 7 10 6 5
1 0 9 2 10 9 7 10 6 6 4 6 3 9 3 9 6 3 1 2
2 9 0 10 3 5 1 3 3 9 5 4 6 9 10 4 5 9 3 2
```

```
6 2 10 0 2 2 9 8 4 3 2 10 7 2 9 8 8 4 1 6
3 10 3 2 0 7 1 3 8 5 10 10 7 1 6 2 9 3 5 5
8 9 5 2 7 0 1 9 5 4 10 9 6 1 10 2 6 1 5 2
7 7 1 9 1 1 0 10 8 10 9 9 4 4 8 5 4 5 2 2
3 10 3 8 3 9 10 0 8 8 10 6 5 8 8 7 4 1 8 7
7 6 3 4 8 5 8 8 0 2 1 2 1 6 1 5 2 4 10 5
5 6 9 3 5 4 10 8 2 0 4 8 5 5 4 7 3 5 1 1
4 4 5 2 10 10 9 10 1 4 0 8 9 1 4 2 7 10 8 9
7 6 4 10 10 9 9 6 2 8 8 0 1 10 1 6 9 7 4 7
7 3 6 7 7 6 4 5 1 5 9 1 0 6 6 4 1 5 2 1
10 9 9 2 1 1 4 8 6 5 1 10 6 0 2 1 10 10 1 4
3 3 10 9 6 10 8 8 1 4 4 1 6 2 0 1 5 6 8 8
3 9 4 8 2 2 5 7 5 7 2 6 4 1 1 0 1 10 10 1
7 6 5 8 9 6 4 4 2 3 7 9 1 10 5 1 0 8 6 5
10 3 9 4 3 1 5 1 4 5 10 7 5 10 6 10 8 0 4 8
6 1 3 1 5 5 2 8 10 1 8 4 2 1 8 10 6 4 0 9
5 2 2 6 5 2 2 7 5 1 9 7 1 4 8 1 5 8 9 0
```

输出

```
Initial Greedy:1 2 19 4 5 7 3 20 10 9 11 14 6 18 8 17 13
12 15 16 bestw = 28
Update:1 2 4 5 3 7 19 10 20 16 17 13 12 15 9 11 14 6 18 8
bestw =27
Update:1 2 4 5 7 3 19 10 20 16 17 13 12 15 9 11 14 6 18 8
bestw =26
Update:1 2 4 11 9 15 12 13 17 16 20 10 19 14 6 18 8 5 7 3
bestw =25
Update:1 2 4 19 10 20 16 17 13 12 15 9 11 14 5 7 6 18 8 3
bestw =24
1 2 4 19 10 20 16 17 13 12 15 9 11 14 5 7 6 18 8 3 1
24
```

附录 完整代码

```
#include <cstdio>
#include <iostream>
#include <algorithm>
#include <cstring>
#include <vector>
#include <ctime>
#include <cmath>
#include <stack>
#define Reg register
#define gmax(_a, _b) ((_a) > (_b) ? (_a) : (_b))
#define gmin(_a, _b) ((_a) < (_b) ? (_a) : (_b))
#define cmax(_a, _b) (_a < (_b) ? _a = (_b) : 0)
#define cmin(_a, _b) (_a > (_b) ? _a = (_b) : 0)
#define FOR(_i,_a,_b) for (Reg int _i = (_a) ; _i <= (_b)
; _i++)
#define REP(_i,_a,_b) for (Reg int _i = (_a) ; _i >= (_b)
; _i--)
#define FOR_LETTER(_i) for (Reg int _i = 'A'; _i <= 'z';
_i != 'Z' ? (_i++) : (_i = 'a'))
#define ll long long
using namespace std;
const int INF = 9999999;
const int MAXN = 100;
int W[MAXN + 10][MAXN + 10];
int X[MAXN + 10], bestX[MAXN + 10];
int bestw, N;
int Wmin[MAXN + 10];

int Rest(int i)
{
    int val = 0;
    FOR(j, i + 1, N)     //遍历还未经过的结点
        val += Wmin[X[j]];       //加上连接它的最小权值路径
    return val + Wmin[1];        //最后还要再经过一次结点1
}

void BacktrackTSP(int i, int cw)
{
```

```
    if (i == N)      //已经搜索到叶节点，即到达最后1个城市
    {
        if (W[X[N - 1]][X[N]] != INF && W[X[N]][1] != INF)
 //最后1个城市与与第1个城市相连
        {
            cw = cw + W[X[N - 1]][X[N]] + W[X[N]][1];
            if (cw < bestw) //当前回路更优，更新最优搜索结果
            {
                cout << "Update:";
                bestw = cw;

                FOR(i, 1, N)//更新路径
                {
                    bestX[i] = X[i];
                    cout << bestX[i] << " ";
                }
                cout << "bestw =" << bestw << endl;
            }
        }
        return;
    }
    FOR(j, i, N)
    {
        swap(X[i], X[j]);
        int rest = Rest(i); //求出剩余节点的预估下界
        if (cw + W[X[i - 1]][X[i]] + rest >= bestw)
        {
            swap(X[i], X[j]);   //撤回此序列
            continue;
        }

        BacktrackTSP(i + 1, cw + W[X[i - 1]][X[i]]);
        swap(X[i], X[j]);   //回溯
    }
}

int greedy()
{
    int val = 0, now = 1;
    int V[MAXN + 10] = { 0 };   //V代表已经经过的节点
    V[1] = 1;        //初始只经过1
```

```
    bestX[1] = 1;
    FOR(i, 2, N)
    {
        int to, Min = INF;
        FOR(j, 1, N)     //遍历还未访问过的节点，寻找最小权值路径
            if (W[now][j] < Min && !V[j])
            {
                Min = W[now][j];
                to = j;
            }
        bestX[i] = to;
        V[to] = 1;       //标记转移节点
        now = to;        //转移到该节点
        // cout << Min << endl;
        val += Min;      //加上权值
    }
    return val + W[now][1]; //最后还要再经过一次结点1
}

void init()
{
    cin >> N;
    //随机生成数据
    srand(time(NULL));
    FOR (i, 1, N)
        FOR (j, i + 1, N)
        {
            W[i][j] = rand() % 10 + 1;
            W[j][i] = W[i][j];
        }

    FOR (i, 1, N)
    {
        FOR (j, 1, N)
            cout << W[i][j] << " ";
        cout << endl;
    }
    /*
    FOR(i, 1, N)
    {
        Wmin[i] = INF;
```

```
        FOR(j, 1, N)
        {
            cin >> W[i][j];
            if (i == j)
                W[i][j] = INF;
            else
                Wmin[i] = gmin(W[i][j], Wmin[i]);
        }
    }
    */
    FOR(i, 1, N) X[i] = i;
    bestw = greedy();
    cout << "Initial Greedy:";
    FOR(i, 1, N)
        cout << bestX[i] << " ";
    cout << "bestw = " << bestw << endl;
    // cout << bestw << endl;
}

int main()
{
    freopen("in.txt", "r", stdin);
    init();
    BacktrackTSP(2, 0);
    FOR(i, 1, N)
        cout << bestX[i] << " ";
    cout << "1" << endl;
    cout << bestw << endl;
}
```