11S103001 郝亚峰 2 班 第十次算法作业

1、 完成旅行商问题。

解：旅行商问题的求解方案如下：

a) 对代价矩阵进行预处理，处理的目的是为了使每行每列都至少有一个 0，若果该行没有 0 元素，则该行的元素都减去该行中最小的那一个元素，列也同样处理。把所有可能的解作为根节点，把代价矩阵中减去的所有值加起来构成旅行商问题最优解代价的下界，同时该下界作为根节点的代价。

b) 分支的策略采用爬山法，选择所有子节点中代价最小的进行进一步分支，产生分支的方法如下：

   a) 选择边(i, j)，产生“包含边(i, j)”和“不包含边(i, j)”的左右两个子节点。选择边(i, j)要满足条件 cost(i, j)=0 and (i,j)=arg max{min{cost(i, k)}+min{cost(h, j)}}，即使得左子节点的代价下界不变，使右子节点的代价下界增加最大，增加为根节点的代价加上 min{cost(i, k)}+min{cost(h, j)}。

   b) 修改代价矩阵，对于左子节点，因为其中已经包含了(i, j)，所以把第 i 行和第 j 列的元素都删掉，把(j,i)的代价改为∞；然后对新的代价矩阵进行 1)中那样预处理，把所有减掉的值加到左子节点的代价上。对于右子节点，将(i, j)在代价矩阵中值置为∞，然后进行与 1)中相同的处理，把所有的减掉的值加到右子节点的代价上(对于右子节点代价是否要加上这部分值，我们后面会讨论，对于这个例子我们先按加上算，但是个人觉得是不合理的)。

c) 不断的进行步骤 2)，但是要避免产生局部回路，直到找到一个可能的解，记录该解的代价，把树中所有代价大于该值的节点全剪掉，对于剩下的节点再进行分支，找到一个可能的解，若其代价小于之前的代价，则记录该代价，并用这个代价进行之后的剪枝，直到遍历所有的分支，找到最优解。

简单分析一下上述方案的合理性：首先，将解集划分为包含某一条边和不包含某一条边的两个子集，不会有情况漏掉；其次每次都更新矩阵，进行 1)中的处理，这样保证每次都能找到满足划分条件的边；最后，采用 2)中的条件选择边，是的左子节点下界增加的小，而右子节点的增加的大，这样利用爬山策略，可以很快的找到可能解的代价，而且这个代价还是比较小的，对于右子节点代价增加的比较大，可以在很早的阶段进行剪枝，提高效率。

下面我们来完成课件中旅行商问题。

图 1 是最原始的代价矩阵，经过步骤 1)的处理(如图 2 所示)，变为图 3 所示的数据。将图 2 中的减掉的数据(red color)加起来即为根节点的代价(可能解的下界)，为 96。

$$
\begin{array}{c|ccccccc|}
 & j=1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
i=1 & \infty & 3 & 93 & 13 & 33 & 9 & 57 \\
2 & 4 & \infty & 77 & 42 & 21 & 16 & 34 \\
3 & 45 & 17 & \infty & 36 & 16 & 28 & 25 \\
4 & 39 & 90 & 80 & \infty & 56 & 7 & 91 \\
5 & 28 & 46 & 88 & 33 & \infty & 25 & 57 \\
6 & 3 & 88 & 18 & 46 & 92 & \infty & 7 \\
7 & 44 & 26 & 33 & 27 & 84 & 39 & \infty
\end{array}
$$

图 1 原始代价矩阵

$$
\begin{array}{c} \\ i=1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ \\ \end{array}
\begin{array}{ccccccc} j=1 & 2 & 3 & 4 & 5 & 6 & 7 \\ \end{array}
$$

$$
\begin{array}{c|ccccccc|c}
 & j=1 & 2 & 3 & 4 & 5 & 6 & 7 & \\
\hline
i=1 & \infty & 3 & 93 & 13 & 33 & 9 & 57 & -3 \\
2 & 4 & \infty & 77 & 42 & 21 & 16 & 34 & -4 \\
3 & 45 & 17 & \infty & 36 & 16 & 28 & 25 & -16 \\
4 & 39 & 90 & 80 & \infty & 56 & 7 & 91 & -7 \\
5 & 28 & 46 & 88 & 33 & \infty & 25 & 57 & -25 \\
6 & 3 & 88 & 18 & 46 & 92 & \infty & 7 & -3 \\
7 & 44 & 26 & 33 & 27 & 84 & 39 & \infty & -26 \\
\hline
 & & & -7 & -1 & & & -4 &
\end{array}
$$

图 2 对原始代价矩阵预处理

$$
\begin{array}{c|ccccccc|}
 & j=1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
i=1 & \infty & 0 & 83 & 9 & 30 & 6 & 50 \\
2 & 0 & \infty & 66 & 37 & 17 & 12 & 26 \\
3 & 29 & 1 & \infty & 19 & 0 & 12 & 5 \\
4 & 32 & 83 & 66 & \infty & 49 & 0 & 80 \\
5 & 3 & 21 & 56 & 7 & \infty & 0 & 28 \\
6 & 0 & 85 & 8 & 42 & 89 & \infty & 0 \\
7 & 18 & 0 & 0 & 0 & 58 & 13 & \infty \\
\hline
\end{array}
$$

图 3 经过变换后的代价矩阵

下面我们按照 2)中的条件来选择边(i,j)来进行对代表所有解的集合的根节点进行扩展子节点。在图 3 中，找出所有的 cost(i, j)=0，然后选择边(i, j)使得不包含边(i, j)时的右子节点的代价增加的最大，cost(1, 2)=0，"不包含边(1, 2)"的代价增加为 cost(1,6)+cost(7, 2)=6，同理所有满足 cost(i, j)=0 的边，我们计算不包含其的代价增量，见表 1。

表 1 图 3 中不包含 cost(i, j)=0 的边的代价下界的增量

Cost(i, j)=0	不含(i,j)时代价增量	Cost(i, j)=0	不含(i,j)时代价增量
(1,2)	cost(1,6)+cost(7,2)=6	(6,1)	cost(6,7)+cost(2,1)=0
(2,1)	cost(2,6)+cost(6,1)=12	(6,7)	cost(6,1)+cost(3,7)=5
(3,5)	cost(3,2)+cost(2,5)=18	(7,2)	cost(7,3)+cost(1,2)=0
(4,6)	cost(4,1)+cost(5,6)=32	(7,3)	cost(7,2)+cost(6,3)=8
(5,6)	cost(5,1)+cost(4,6)=3	(7,4)	cost(7,2)+cost(5,4)=7

根据上表，我们第一次选的边为(4,6)。生成如图 4 所示的二叉树，左子节点为"包含(4,6)"，其代价下界暂时不变，仍为 96，右子节点为"不包含(4,6)"，由于其代价增加了 32，故为 128。

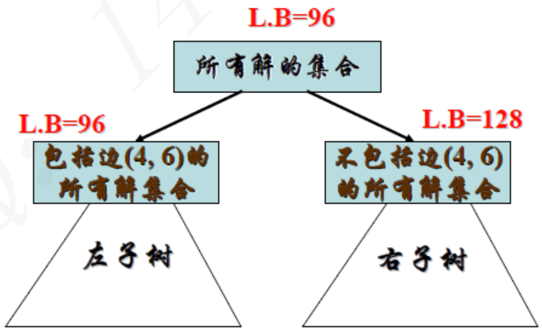

图 4 选择边(4,6)划分解集

然后，我们还得对左右子节点的矩阵，进行相应的处理，对于"包含(4,6)"将矩阵中的第 4 行，第 6 列元素都删掉，把原来矩阵中(6,4)改为∞，然后按照步骤 1)中进行预处理，发现第五列无 0 元素，故第 5 行元素都减去 3，如图 5 所示。

$$
\begin{array}{c|cccccc|}
 & j=1 & 2 & 3 & 4 & 5 & 7 \\
\hline
i=1 & \infty & 0 & 83 & 9 & 30 & 50 \\
2 & 0 & \infty & 66 & 37 & 17 & 26 \\
3 & 29 & 1 & \infty & 19 & 0 & 5 \\
5 & 0 & 18 & 53 & 4 & \infty & 25 \\
6 & 0 & 85 & 8 & \infty & 89 & 0 \\
7 & 18 & 0 & 0 & 0 & 58 & \infty
\end{array}
$$

图 5 修改后的包含(4,6)的代价矩阵

所以，右子节点的代价加上 3，变为 96。对于左子节点，将(4,6)变为∞，也同样按照步骤 1)中进行同样处理，发现第 4 行元素中无 0 元素，则第四行都减去该行最小的元素 32，如图 6 所示，所以右子节点代价增加 32 变为 160。

$$
\begin{array}{c|ccccccc|}
 & j=1 & 2 & 3 & 4 & 5 & 6 & 7 \\
\hline
i=1 & \infty & 0 & 83 & 9 & 30 & 6 & 50 \\
2 & 0 & \infty & 66 & 37 & 17 & 12 & 26 \\
3 & 29 & 1 & \infty & 19 & 0 & 12 & 5 \\
4 & 0 & 51 & 34 & \infty & 17 & \infty & 48 \\
5 & 3 & 21 & 56 & 7 & \infty & 0 & 28 \\
6 & 0 & 85 & 8 & 42 & 89 & \infty & 0 \\
7 & 18 & 0 & 0 & 0 & 58 & 13 & \infty
\end{array}
$$

图 6 修改后的不包含(4,6)的代价矩阵

所以此时的二叉树如图 7 所示。

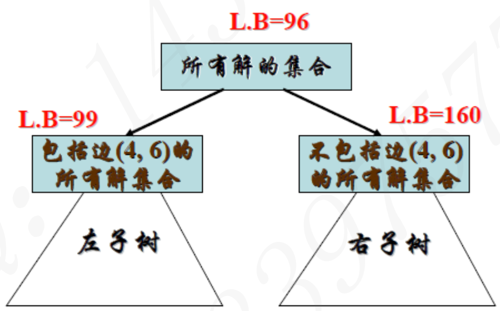

图 7 将左右子节点加入后最终的二叉树

按照爬山策略进行分支，所以选择代价为 99 的右子节点进行扩展，同样，我们要在图 5 所示的代价矩阵中选择一条边来进行划分右子节点所代表的解集，需要计算不包含边(i,j)时，代价的增量，如表 2 所示。

表 2 图 5 中不包含 cost(i, j)=0 的边的代价下界的增量

Cost(i, j)=0	不含(i,j)时代价增量	Cost(i, j)=0	不含(i,j)时代价增量
(1,2)	cost(1,4)+cost(7,2)=9	(6,7)	cost(6,1)+cost(3,7)=5
(2,1)	cost(2,5)+cost(6,1)=17	(7,2)	cost(7,3)+cost(1,2)=0
(3,5)	cost(3,2)+cost(2,5)=18	(7,3)	cost(7,2)+cost(6,3)=8
(5,1)	cost(5,4)+cost(6,1)=4	(7,4)	cost(7,2)+cost(5,4)=4
(6,1)	cost(6,7)+cost(2,1)=0		

所以根据上表，我们选择(3,5)来划分，产生子节点“包含(3,5)”和“不包含(3,5)”，右儿子的代价暂时为 99，左儿子的代价暂时为 99+18=117。分别对左右儿子对矩阵进行相应的处理之后，左儿子的代价矩阵如图 8 所示，右儿子的代价矩阵如图 9 所示，第 3 行的

值减掉 1，第 5 列的值减掉 17，一共减掉 18，所以右儿子代价为 117+18=135。

$$
\begin{array}{c|ccccc}
 & j=1 & 2 & 3 & 4 & 7 \\
\hline
i=1 & \infty & 0 & 83 & 9 & 50 \\
2 & 0 & \infty & 66 & 37 & 26 \\
5 & 0 & 18 & \infty & 4 & 25 \\
6 & 0 & 85 & 8 & \infty & 0 \\
7 & 18 & 0 & 0 & 0 & \infty
\end{array}
$$

图 8 修改后的图 5 包含(3,5)所对应的代价矩阵

$$
\begin{array}{c|cccccc}
 & j=1 & 2 & 3 & 4 & 5 & 7 \\
\hline
i=1 & \infty & 0 & 83 & 9 & 13 & 50 \\
2 & 0 & \infty & 66 & 37 & 0 & 26 \\
3 & 28 & 0 & \infty & 18 & \infty & 4 \\
5 & 0 & 18 & 53 & 4 & \infty & 25 \\
6 & 0 & 85 & 8 & \infty & 72 & 0 \\
7 & 18 & 0 & 0 & 0 & 41 & \infty
\end{array}
$$

图 9 修改后的图 5 不包含(3,5)所对应的代价矩阵

所以，此时生成的二叉树如图 10 所示。

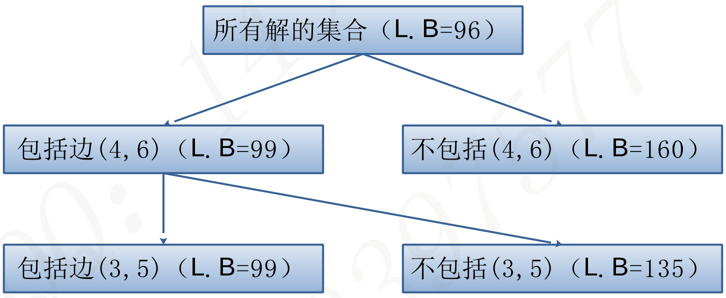

图 10 图 7 左儿子做扩展

重复以上步骤，选择“包括(3,5)”的左儿子进行扩展，在图 8 所示的矩阵上进行选择(i,j)。选择的表如表 3 所示。

表 3 图 8 中不包含 cost(i, j)=0 的边的代价下界的增量

Cost(i, j)=0	不含(i,j)时代价增量	Cost(i, j)=0	不含(i,j)时代价增量
(1,2)	cost(1,4)+cost(7,2)=9	(6,7)	cost(6,1)+cost(5,7)=25
(2,1)	cost(2,7)+cost(6,1)=26	(7,2)	cost(7,3)+cost(1,2)=0
(5,1)	cost(5,4)+cost(6,1)=4	(7,3)	cost(7,2)+cost(6,3)=8
(6,1)	cost(6,7)+cost(2,1)=0	(7,4)	cost(7,2)+cost(5,4)=4

所以，我们选择边(2,1)来扩展，得到的“包括边(2,1)”的左儿子的下界为：99+4=103，未修改完全的矩阵如图 11 所示，第 5 行要减去 4。右儿子的下界为 99+26=125，其对应的代价矩阵如图 12 所示。

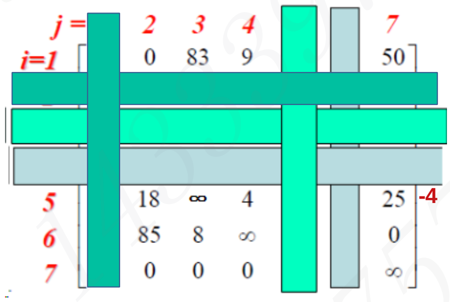

图 11 未修改完全的图 8 包含(2,1)所对应的代价矩阵

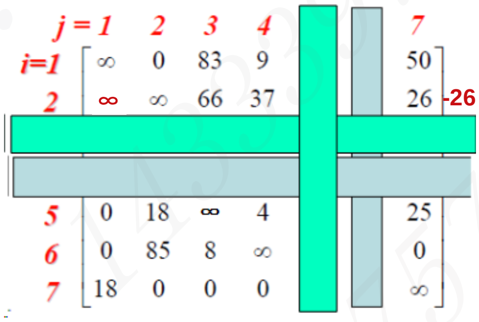

图 12 未修改完全的图 8 不包含(2,1)所对应的代价矩阵

所以，我们最终形成新的二叉树如图 13 所示。

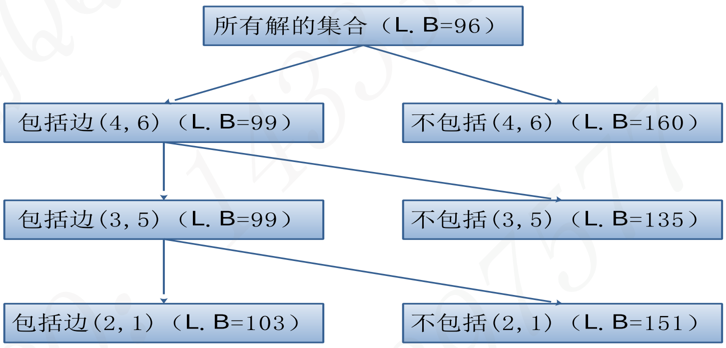

图 13 图 10 左儿子做扩展

之后也是同样处理，只不过我们不叙述详细过程了，画出得到第一个可能的解时的二叉树，如图 14 所示。

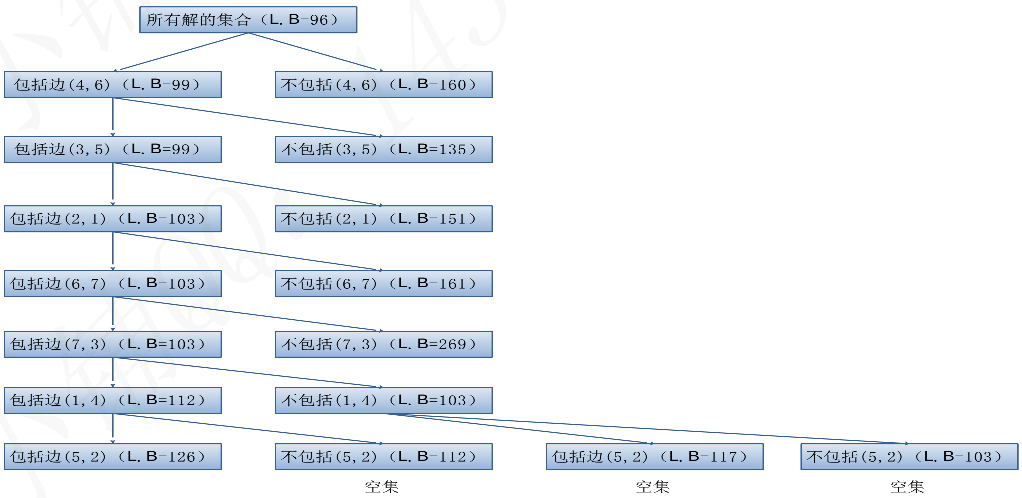

图 14 对于包括(7,3)的节点不能再继续扩展

在此处，我们还要讨论一开始步骤 2)中的问题，即对于不包括(i,j)的节点的代价，是否除了加上增加的代价之外，还要加上矩阵修改之后减掉的所有的值。其实应该是不用加的，因为这两个值表示的意义是相同的，就是不包括边(i,j)，那么 i 必须从指向某一点，并且，必须有另一个点指向 j，找到 i 行的最小值，j 列的最小值(当然都是除了(i,j)之外的)，两者相加即为增加的代价，在新的矩阵中，我们把不包含(i,j)的代价矩阵的第 i 行第 j 列的位置置为∞；这样只有第 i 行和第 j 列可能没有 0 元素，所以要减去第 i 行，第 j 列的最小值，这两个最小值即为求代价增加的那两个值，是相同的，所以不需要重复加。所以我们对上面生成的二叉树进行改正，如图 15 所示。

图 15 对图 14 修正

所以，我们得到一个可能的解 4-6-7-3-5-2-1-4，其代价为 126，那么我们就可以用这个代价作为上界来剪枝，对于所有大于等于 126 的节点都可以考虑不用扩展了。而且对于这个问题，我们在不断的迭代步骤 2)的时候，可能有满足划分条件的边，但是加入这条边会产生回路，那么我们就不能取这样的边，所有的 cost(i, j)=0 的边都不满足，只能取 cost(i,j) ≠0 的边，但是从代价小的便开始取，按这条边进行分支，对于分支的各个子节点的代价矩阵还是按照步骤 2)的进行，例如图 15 中的节点“包括边(7,3)”。所以我们接着来扩展图 15 中的“不包括(3,5)”和“不包括(2,7)”节点，分别如图 16 和图 17 所示。

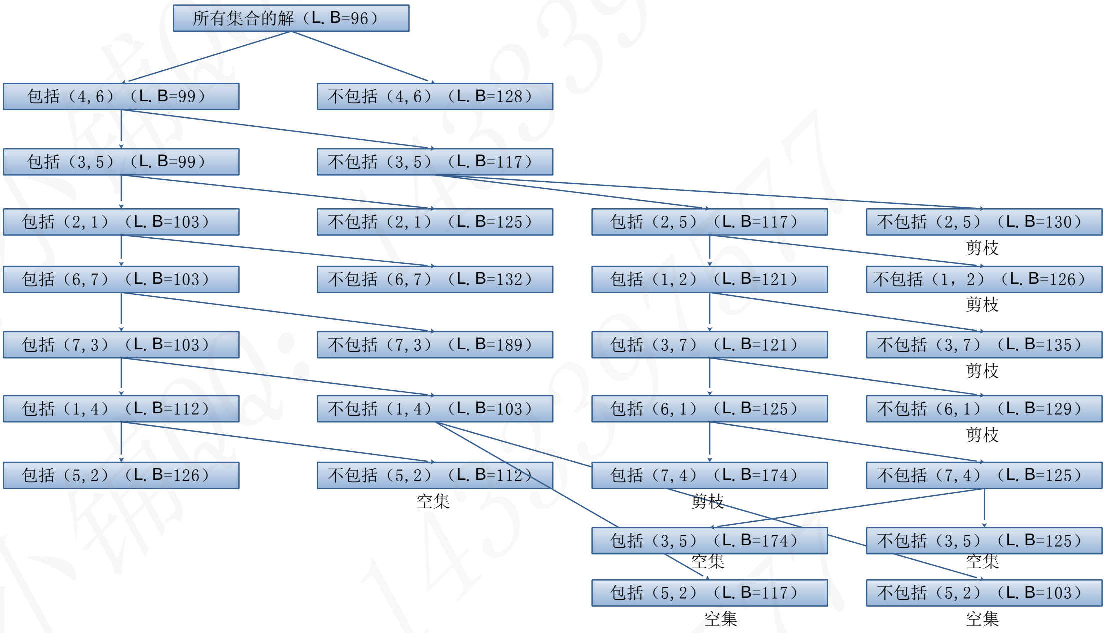

图 16 扩展图 15 中“不包括(3,5)”节点

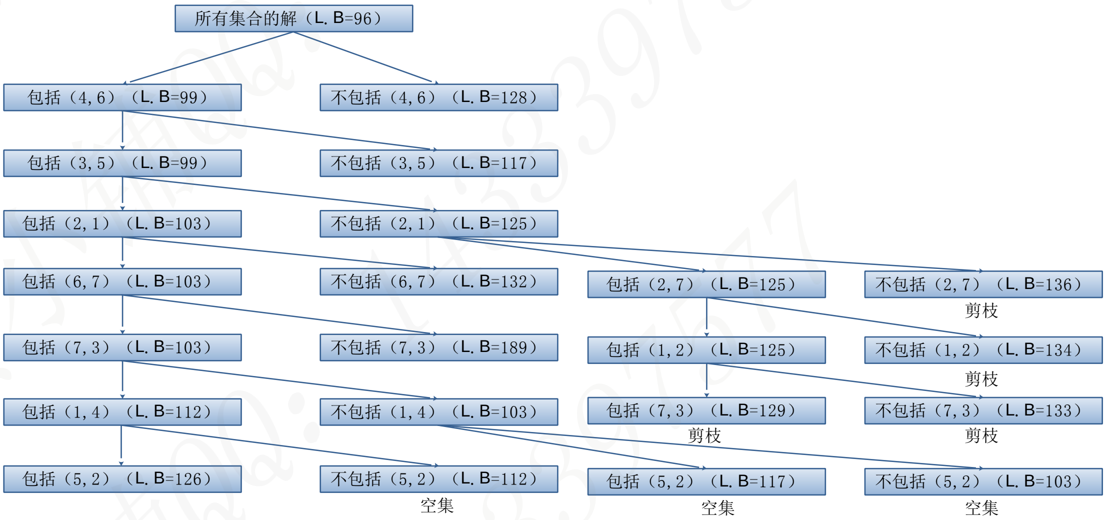

图 17 扩展图 15 中“不包括(2,1)”节点

综上，我们得到最优解为：4-6-7-3-5-2-1-4，其代价为 126。

说明:在处理左子节点时,我们在构造新的代价矩阵没有把(i,j)所对应的(j,i)变为无穷大，其实应该是变为无穷大，这样做是为了避免在选(j,i)形成局部回路，但是，整个循环中，我们对于形成局部循环这种情况一直在监控，所以没有这样修改的话得到的结果也是正确的。

2、 给出旅行商问题的详细算法。

解：由求解上面的问题，我们给出旅行商问题的详细算法。

1) 数据结构

```
struct TreeNode{
    int id1;                //id1、id2 分别为边的起点和终点
    int id2;
    bool flag;              //表示该节点是包括还是不包括该边
    int cost;               //节点代价
```

```
int matrix[][];                    //节点代价矩阵
struct TreeNode *Parent;           //指向父节点
struct TreeNode *Lchild;           //指向左儿子节点
struct TreeNode *Rchild;           //指向右儿子节点
}
```

我们使用该数据结构来建立树，进行分支界限算法，建树的目的是为了记录最优解，也是为了利于判断是否已经形成了可能的解，是否会形成局部回路，因为如果我们在栈上采用爬山策略操作，当前节点的祖先节点都被弹出栈了，对于一些界限的判断和最优解的查找比较费事，所以我们利用此结构来建树，当然建树会有额外的开销，但是，我们会对于不可能是解的分支进行及时的剪枝，这样树的空间就不会太大。

2) 算法设计：

在叙述详细算法之前，我们先给出一些辅助函数：

A. int preprocess(M)：对矩阵 M 进行预处理，使得 M 满足每行每列至少都有一个 0，即每行每列都减去该行该列的最小值，把这些减去的值之和作为返回值。

B. delete(M,i,j)：删除矩阵 M 中的第 i 行和第 j 列的所有元素。

C. alter(M,i,j)：修改矩阵 M 第 i 行第 j 列的元素为无穷大，即 M[i][j]←∞。

D. int(i,j) maxIncrement(M,T)：返回矩阵 M 中满足以下条件的元素的下标(i,j)：

① (i,j)加入树 T 后，使所在的分支不能产生局部回路；

② 按照(i,j)分支，应该使得“包含(i,j)的”增加的最小，“不包含(i,j)”的增加的最大。

要实现上面这两个条件，我们可以先把矩阵 M 中所有 M(i,j)=0 的元素，找出来，然后，对于某一个(i,j)计算“不包含(i,j)”增加的代价：就是矩阵 M 中第 i 行中除了 M(i,j)最小的元素和第 j 列中除了 M(i,j)最小的元素之和。我们取“不包含(i,j)”增加的代价最大的那个(i,j)，当然不能使“包含(i,j)”所在的分支产生局部回路，若有局部回路，则取“不包含(i,j)”增加代价次大的，若所有的 M(i,j)=0 的边(i,j)都会产生局部回路，则我们取 M(i,j)≠0,且最小的那一个(i,j)。

E. bool isLocalCircuit(N,T)：通过树 T 查看加入节点 N 是否会在该分支上产生局部回路，若是则返回 true，否则返回 false。

F. bool isPossibleSolution(N,T)：判断节点 N 是不是可能解的最后一个元素，若是则返回 true，否则返回 false。

G. TreeNode buildNode(i,j,flag,cost,M)：建立一结构体节点，比如建立根节点为代码为：

```
struct TreeNode root=(strcut TreeNode*)malloc(sizeof(struct TreeNode));
root.id1=0;
root.id2=0;
root.flag=true;
root.cost= preprocess(M)
root.matrix=M;
root.Parent=NULL;
root.Lchild=NULL;
root.Rchild=NULL;
```

我们就可以写为：buildNode(0,0,true,preprocess(M),M)

下面给出旅行商问题的详细算法：

```
Travel-Salesman(M)
{
    x←∞;
    struct TreeNode root= buildNode(0,0,true,preprocess(M),M);
    PUSH(root,S);                    //将根节点 root 压入栈 S 中
    WHILE S 不为空  DO
    {
        IF((TOP(S).matrix≠∅ and  TOP(S).matrix(maxIncrement(TOP(S).matrix,T))=∞)
        or TOP(S).cost>=x)
        THEN
            在 TOP(S)所在的分支中，从 TOP(S)节点开始连续的删除在删除未进行
            前只含一个儿子的祖先，直到遇到包含两个儿子的祖先为止
            POP(S);
        ELSE IF isPossibleSolution(TOP(S),T)
            x←TOP(S).cost;
            POP(S);
        ELES
            matrix1←TOP(S).matrix;
            matrix2←TOP(S).matrix;
            (i,j)←maxIncrement(matrix1,T);
            cost←TOP(S).cost+matrix1(i,j);
            delete(matrix1,i,j);
            alter(matrix1,j,i);
            cost←TOP(S).cost+preprocess(matrix1);
            struct TreeNode temp1= buildNode(i,j,true,cost,matrix1);
            alter(matrix2,i,j);
            cost←TOP(S).cost+preprocess(matrix2);
            struct TreeNode temp2= buildNode(i,j,false,cost,matrix2);
            IF temp1.cost>temp2.cost
            THEN
                TOP(S).Rchild=temp1;
                TOP(S).Lchild=temp2;
            ELSE
                TOP(S).Rchild=temp2;
                TOP(S).Lchild=temp1;
            POP(S);
            IF temp1.cost>temp2.cost
            THEN
                PUSH(temp1,S);
                PUSH(temp2,S);
            ELSE
                PUSH(temp2,S);
                PUSH(temp1,S);
            }
```

}

以上算法循环中，分为三种情况：

- 对于代价大于 x 的，要进行剪枝；对于无法得到可能解的也要剪枝；
- 对于某一节点是可能解的最后一个元素，则要记录其代价，用来剪枝；
- 对于其他情况，我们进行扩展。