目录

一、问题描述

分支限界法 PPT 中的例子

代价矩阵如下：

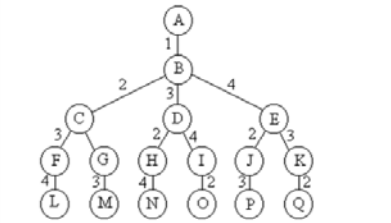
0	3	1	5	8
3	0	6	7	9
1	6	0	4	2
5	7	4	0	3
8	9	2	3	0

在这五个城市中旅行，要求每个城市走过一次，且最终回到出发的位置，求最少的路程（代价）。

二、算法思想

使用回溯法。所谓回溯法，是利用深度优先搜索思想，遍历解空间树，搜索满足约束条件的解。搜索时判断对应的部分解是否满足约束条件或者目标函数的边界。如果不包含最优解，跳过对其孩子的遍历，所谓“剪枝”。否则继续搜索。

该问题的解空间树为一个排列树。



遍历排列树需要O(n!)计算时间

```
void backtrack (int t)
{
 if (t>n) output(x);
  else
   for (int i=t;i<=n;i++) {
    swap(x[t], x[i]);
    if (legal(t)) backtrack(t+1);
    swap(x[t], x[i]);
   }
```

遍历过程中，如果没有遍历到叶子节点，路程就已经比最优情况大了，这个时候就要舍弃这种情况，即剪枝。到达叶子节点时，如果得到的路径长度小于先前得到的最佳路径长度，则更新

三、核心代码

```
void backtrack(int t)
{
    int i,j;
    if(t==CityNum-1)
    {//已经搜索到叶节点，已经选择了最后1个城市
```

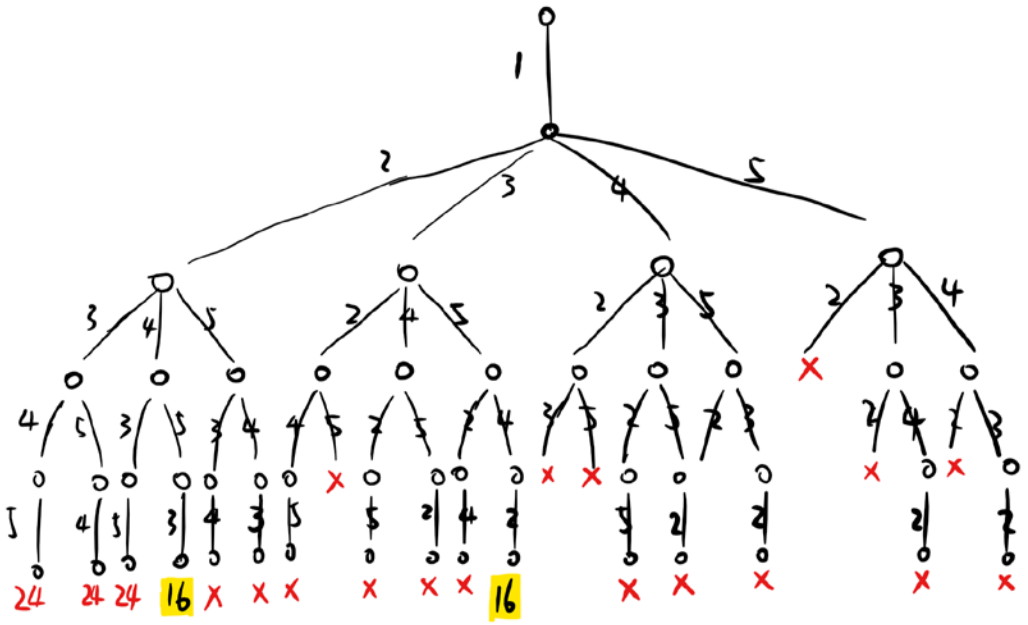
```
        if(jour[path[t-1]][path[t]]!=0&&jour[path[t]][0]!=0)
        {//最后1个城市与前一城市相连与第1个城市相连，组成1个满足条件的回路
            if(cost+jour[path[t]][0]<bestcost&&jour[path[t]][0]!=0)
            {
                bestcost=cost+jour[path[t]][0];
                for(j=0; j<=CityNum-1; j++)
                    bestpath[j]=path[j];
            }
        }
    }
    else
    {
        for(j=t-1;j<=CityNum-1;j++)
        {//考察x[i]的各个可能取值
            if(jour[path[t-1]][path[j]]!=0
                &&
                cost+jour[path[t-1]][path[j]]<bestcost)
            {
                swap(path[t],path[j]);//加入第i个城市
                cost+=jour[path[t-1]][path[t]];//更新扩展后的路径的代价
                backtrack(t+1);//递归搜索以x[i]为根的后续子树
                cost-=jour[path[t-1]][path[t]];//搜索失败，回溯，回到
                swap(path[t], path[j]);
            }
        }
    }
}
```

四、设计剪枝函数

```
if(jour[path[t-1]][path[j]]!=0&&
    cost+jour[path[t-1]][path[j]]<bestcost)
```

这个判断语句即可保证剪枝的实现。即上述的“遍历过程中，如果没有遍历到叶子节点，路程就已经比最优情况大了，这个时候就要舍弃这种情况，即剪枝”。

五、最终剪枝后的解空间树



六、收获

重温了一下深度优先遍历的方法，了解了剪枝的思想。对解空间树的两种形式，即排列树和子集树有了很深的认识。结合 PPT 上其他的问题，我会进一步实现一些关于子集树的题目。