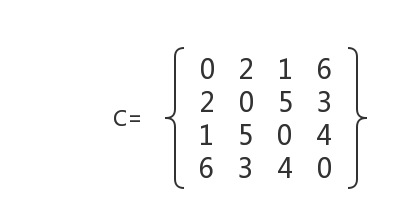
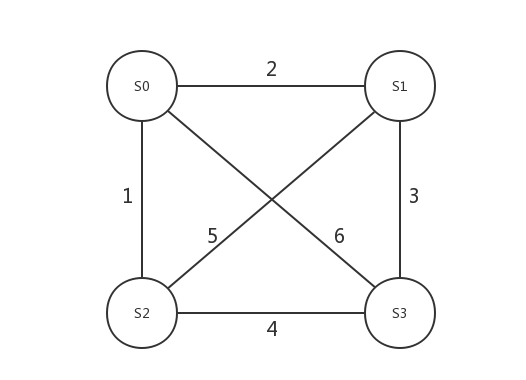
动态规划解TSP问题

**一、问题内容**

TSP问题（旅行商问题）是指假设有一个旅行商人要拜访n个城市，他必须选择所要走的路径，路径的限制是每个城市只能拜访一次，而且最后要回到原来出发的城市。每两个城市之间都有相连的路径。路径的选择目标是要求得的路径路程为所有路径之中的最小值。

**二、最优子结构证明**

我们现在假设有4个城市



假设存在一条最短回路L: S0->S1->S2->S3->S0

则S1->S2->S3->S0必然是S1通过其他各个点各一次到达S0的最短路径

证明：反证法，如果有比S1->S2->S3->S0更短的路径，则S0->S1->S2->S3->S0不是最短回路

此问题符合最优子结构

**三、解决思路**

由此启发可得最短路径求法

假如想知道S0->S1->S2->S3->S0的最短路径，即求从S0出发过{ S1,S2,S3} 3个点回到0的最短路径

即求S0->S1 + S1出发过{S2,S3}两点回到S0的最短路径和

S0->S2 + S2出发过{S1,S3}两点回到S0的最短路径和

S0->S3 + S3出发过{S1,S2}两点回到S0的最短路径，这三个选项中选一个最短的路径，

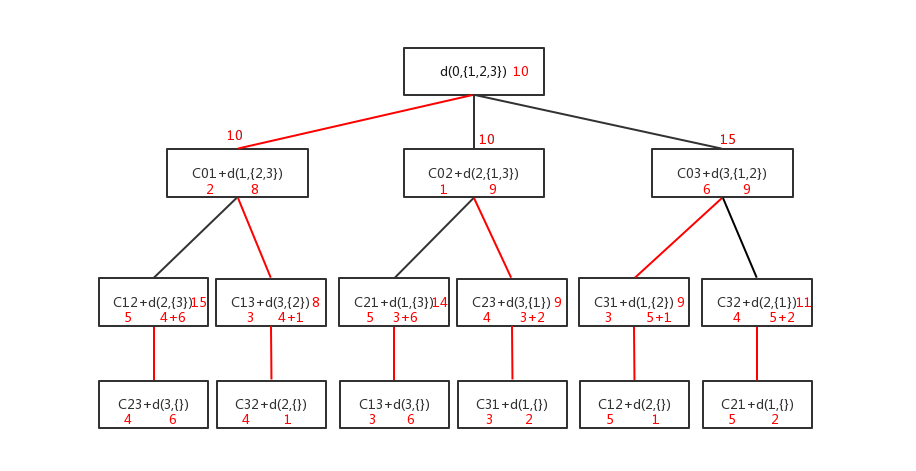
那么要求S0->S1 + S1出发过{S2,S3}两点回到S0的距离，只需再求出S1->S2 + S2出发过{S3}回到S0的最短路径和

S1->S3 + S3出发过{S2}回到S0的最短路径

这两个选项中选一个最短路径。

或者我们可以从后往前逆推，首先确定过1个特定点回到出发点的最短路径，再确定过2个特定点回到出发点的最短路径，以此类推，确定过所有点回到出发点的最短路径

把我们的思路化成图例来方便理解



d(i, V)表示从顶点i出发经过V(是一个点的集合)中各个顶点一次且仅一次，最后回到出发点S0的最短路径长度。

      ①当V为空集，那么d(i, V)，表示从i不经过任何点就回到出发点S0了。此时d(i, V)=Cis(就是 城市i 到 城市s 的距离)。

      ②如果V不为空，那么就是对子问题的最优求解。

所求的最短距离等于i到k的距离加上从k出发经过点集V - {k} 中各个顶点一次且仅一次，最后回到出发点s的最短路径长度。

          d(i, V)=min{Cik +  d(k, V - {k})}

图中所示红线表示选择的决策

如图所示，我们目前的算法跟穷举是一样的，那么动态规划的优势如何体现呢？

我们现在所举的例子规模很小只有4个点，如果规模扩大，假如有6个点需要求解，那么在我们求解某个点过一个点集的最短路径的时候，就会出现完全重复的情况，我们可以把这种重复的情况记录下来，在遇到重复的地方直接使用已有的结论而不是像穷举法一样反复计算，从而体现出动态规划法算法的优势，实现算法的优化，那么我们下一个要解决的问题就是使用什么方法记录重复的情况，又用什么办法调用已有的数据

**四、实现**

在具体的编程实现过程中，我们可以把求解d(i, V)转化为填一个二维表，其中行数为n，列数为2^(n-1)，行表示出发的顶点i，列表示从i出发经过集合中的点回到起始点的最短路径。这样我们就可以把每一次所做的计算记录下来，并且能够能够很方便的从表中找到已有的结论

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| d[i][j] | { } 000  0 | {1}  001  1 | {2}  010  2 | {1,2}  011  3 | {3}  100  4 | {1,3}  101  5 | {2,3}  110  6 | {1,2,3}  111  7 |
| 0 |  |  |  |  |  |  |  | 10 |
| 1 | 2 |  | 6 |  | 9 |  | 8 |  |
| 2 | 1 | 7 |  |  | 10 | 9 |  |  |
| 3 | 6 | 5 | 5 | 9 |  |  |  |  |

表画出来之后另一个要解决的问题就是如何用10进制的数字来表示点集从而实现快速的直接寻址查找。

假设我们有一个点集来表示问题中所有需要经过的点，这个点集中的所有点都拥有存在和不存在两种状态也就是1和0，从而变成各个需要求解的子集，从而我们想到可以用用二进制转换十进制的方法表示点集。

假如一共有S4,S3,S2,S1四个点我们要表示一个点1和点3的集合我们用二进制数0101来表示，体现在数组角标上的十进制数就是5，也就是说二维数组每一个列角标的十进制数唯一对应一个二进制数并唯一对应一种点的集合，一共有2^(n-1)个点的集合。

最终我们把我们的算法转换成对这个表中的每一个d[i][j]求解

d[i][j] = min{Cik +  d[i][j – 2^(k-1)]}，k分别等于j表示的那个集合的每一个点

按照从上至下的顺序一列一列把表填满就可以得到最终解。

关于路径而非路径数值的记录问题，我们可以把表中的每一个元素声明成一个结构体，数据成员分别是路径和权值，在计算数值的时候顺便也把路径的决策记录下来，用string或者一维数组记录都是可以的

**五、动态规划递推公式**

d(i,V) = Cis 当i != s 且 V 为空集时成立

d(i, V)=min{Cik +  d(k, V - {k})} 当V不为空集时成立

**六、时间复杂度分析**

计算此算法时间复杂度就是看填表过程中的计算次数

我们要填的表是一个n 行乘以 2^(n-2) 列的表，在填表过程中需要比大小的次数也是线性的，不超过n，所以总的时间复杂度应该是

O(n^2 \* 2^n)

**七、心得体会**

TSP问题是知名的数学难题，NP完全问题，随着问题规模N的增大，求解时间急剧增大，且优化难度是很大的。我自己曾经写过一个穷举的算法来做实验，在求解10个点时，我的笔记本电脑就很难算出解了（出去吃了个饭回来还是算不完，我就放弃了）。利用之前老师在课上讲过的动态规划的思想，对穷举法进行优化之后，运行效率虽有提高，但是我们可以看到，经过计算，时间复杂度仍然很高。目前据我所知此问题最现实的解法仍然是用贪心法求近似解，贪心法的TSP问题我也做过实验，用了大量的实例，经过试验，贪心法得到的近似解和动态规划得到的精确解在大多数情况下还是很接近的。