

## 实验目标

- 1、熟悉回溯法、分支限界法应用场景及实现的基本方法步骤；
- 2、学会回溯法、分支限界法的实现方法和分析方法。

## 实验题目

问题 1：旅行售货员问题：

当结点数为 4，权重矩阵为  $\begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} & \begin{matrix} 0 & 9 & 4 \\ 2 & 9 & 0 & 6 \\ 3 & 4 & 6 & 0 \end{matrix} \end{matrix}$ ，求最优路径及开销，分别利用回溯法和分支限界法解决并比较。

问题 2：0-1 背包问题：

对于  $n=5$ ， $C=10$ ， $v_i=\{6,3,5,4,6\}$ ， $w_i=\{2,2,6,5,4\}$ ，计算  $x_i$  及最优价值  $V$ 。

分别利用动态规划、回溯法和分支限界法解决此问题，比较并分析这三种算法实现！

## 第一个实验（旅行售货员问题）

### 算法问题：

旅行售货员问题：

当结点数为 4，权重矩阵为  $\begin{matrix} & \begin{matrix} 0 & 1 & 2 & 3 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \\ 3 \end{matrix} & \begin{matrix} 0 & 9 & 4 \\ 2 & 9 & 0 & 6 \\ 3 & 4 & 6 & 0 \end{matrix} \end{matrix}$ ，求最优路径及开销，分别利用回溯法和分支限界法解决并比较。

## 算法原理

初步分析：

旅行售货员问题需要找到一条从一点出发经过所有节点回到起点的最优路径，在本题中，可以建立一个搜索树，搜索树的节点值为访问的节点，从上一层的节点(i)到下一层(j)的为从  $i \rightarrow j$  的路径，节点值为路径的长度。在这题中，可以根据节点的访问顺序建立一个搜索树。

回溯法类似于深度优先的策略，及从一个节点出发，先访问其子节点，然后再访问其兄弟节点。再回溯法中，约束条件是两个节点之前是否有路径，限界条件是当前的节点的  $cost$

是否超过当前的最大值。

再回溯法中，决定算法的关键之处在于根据限界条件排除不必要搜索的分支，为了提高回溯法的效率，常常在往下递归时选择最优的路径（需要优先队列的支持）。

分支限界法类似于宽度优先的策略，宽度优先的访问顺序需要借助队列的容器，即 FIFO，为了提高分支限界法的效率，可以把队列改成优先队列，这样可以提高搜索的效率（可以更快的收敛）。

回溯法的伪代码：

```
node:
    path  # 路径
    cost  # 花费

best = 0  # 最优值
data  # 权重矩阵

def search_d(node x):
    nodes = _extend(x)  # 扩展此节点
    if nodes.empty():  # 当前此节点
        cost = x.cost + data[x.path.last, 0]  # 计算节点的值
        if cost < best:
            best = cost  # 如果当前的花费更优，则用此节点来替换
        return x
    else:
        return best([search_d(node) for node in nodes if node.cost < best])  # 获取子节点中最优的值

input(data)
search(node(x=[0],0))
```

分支限界法的伪代码：

```
node:
    path  # 路径
    cost  # 花费

data  # 权重矩阵

def search_d(node x):
    nodes = _extend(x)  # 扩展此节点，其中 nodes 是一个优先队列
    while not nodes.empty():
        node = nodes.top()
        nodes.pop()  # 获取优先队列中的第一个节点
        t_nodes = _extend(node)
        if t_nodes.empty():  # 已经遍历到最后一层节点
            t_nodes.cost += data[t_nodes.path.last][0]
```

```

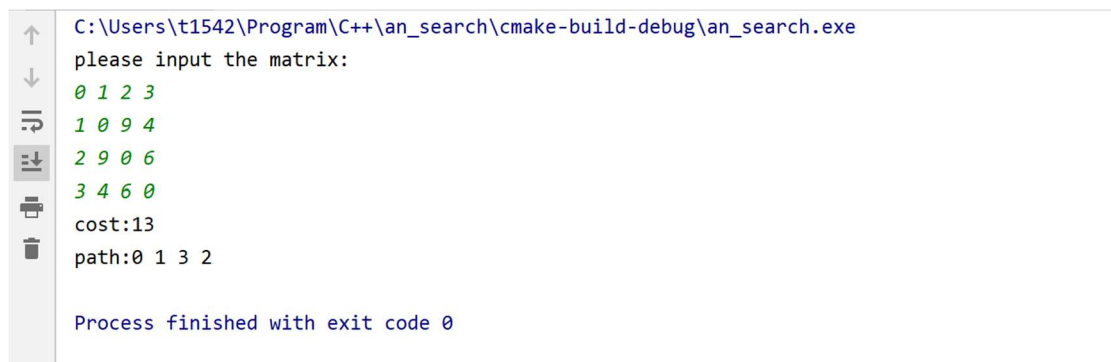
        return node
    else: # 没有遍历到最后一层节点，则将其扩展节点加入到优先队列
        中。

        for t_node in t_nodes:
            nodes.push(t_node)

input(data)
search(node(x=[0]))

```

结果:



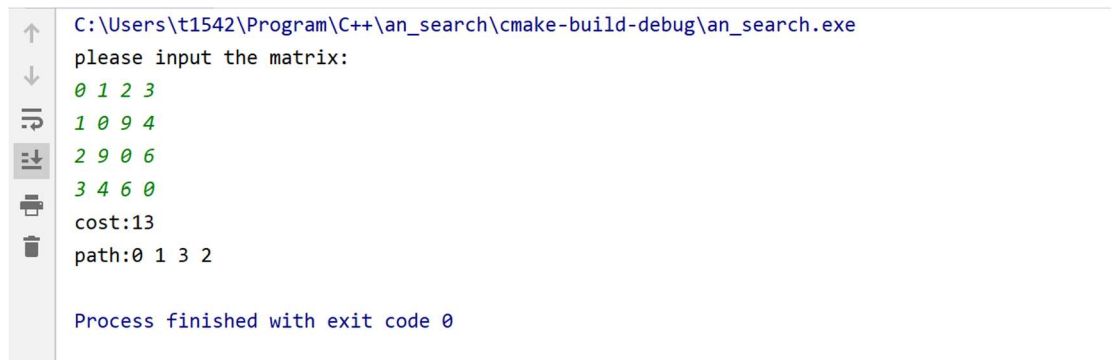
```

C:\Users\t1542\Program\C++\an_search\cmake-build-debug\an_search.exe
please input the matrix:
0 1 2 3
1 0 9 4
2 9 0 6
3 4 6 0
cost:13
path:0 1 3 2

Process finished with exit code 0

```

图 5-1 实验 1 使用回溯法的实验结果



```

C:\Users\t1542\Program\C++\an_search\cmake-build-debug\an_search.exe
please input the matrix:
0 1 2 3
1 0 9 4
2 9 0 6
3 4 6 0
cost:13
path:0 1 3 2

Process finished with exit code 0

```

图 5-2 实验 2 使用分支限界法的实验结果

分析:

回溯法和分支限界法是对一般的搜索算法的改进，一般只要问题能够使用选择树和排序树来进行表示，都能够使用回溯法和分支限界法来进行解决，搜索算法效率和访问节点的数目密切相关。在旅行商问题中，使用传统的搜索方法需要访问约 $n!$ 个节点，使用回溯法和分支限界法可以提高算法的搜索效率，但是由于其本质上仍属于遍历的搜索算法，因此算法的时间复杂度并没有降低。

## 第二个实验 (0-1 背包问题)

### 算法问题:

问题 2: 0-1 背包问题:

对于  $n=5$ ,  $C=10$ ,  $v_i=\{6,3,5,4,6\}$ ,  $w_i=\{2,2,6,5,4\}$ , 计算  $x_i$  及最优价值  $V$ 。

分别利用动态规划、回溯法和分支限界法解决此问题, 比较并分析这三种算法实现!

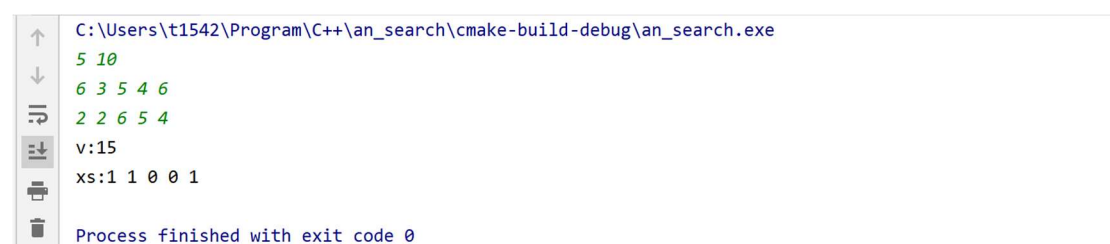
### 算法原理:

#### 初步分析

0-1 背包问题中, 可以将当前问题分解为两个子问题, 其中一个子问题是放入当前的物品, 另外一个子问题是不放入当前的物品。即  $E[i][j] = \max(E[i-1][j-w[i]] + v[i], E[i-1][j])$ 。

在回溯法和分支限界法中, 与上一题不同的是, 这里使用的搜索状态树是选择树, 即每个节点的值 0-1, 为了便于更好的搜索效率, 一般常常根据  $v[i]/w[i]$  的值从大到小进行排序, 在两个方法中, 约束条件是当前的重量和  $\leq$  背包的容量, 限界条件是当前的价值+剩余的价值  $\geq$  当前最优的价值。

### 结果:



```
C:\Users\t1542\Program\C++\an_search\cmake-build-debug\an_search.exe
5 10
6 3 5 4 6
2 2 6 5 4
v:15
xs:1 1 0 0 1
Process finished with exit code 0
```

图 5-3 0-1 背包的问题的实验结果

### 分析:

在本例中, 0-1 背包问题分别使用了动态规划、回溯法和分支限界法进行问题的求解。经过计算我们可以发现, 动态规划的时间复杂度为  $O(n^2)$ , 而回溯法和分支限界法的时间复杂度为  $O(2^n)$ 。前者的时间复杂度为多项式时间, 而后者为指数项时间。

动态规划能够大大降低时间复杂度在于其能够使用问题本身的性质, 相对来说, 动态规划使用的范围更窄。而回溯法和分支限界法由于需要展开各个节点, 一般来说其时间复杂度为达到指数和阶乘的时间复杂度, 因此在数据规模增长时, 执行算法所需要的时间也会快速增长。因此, 根据实际问题, 设计一个较好的算法是十分必要的过程。