北京郵電大學

实验报告：旅行售货员 TSP 问题的回溯法求解探索

学院：计算机学院（国家示范性软件学院）

专业：计算机科学与技术

班级：2022211305

学号：2022211683

姓名：张晨阳

2025 年 1 月 1 号

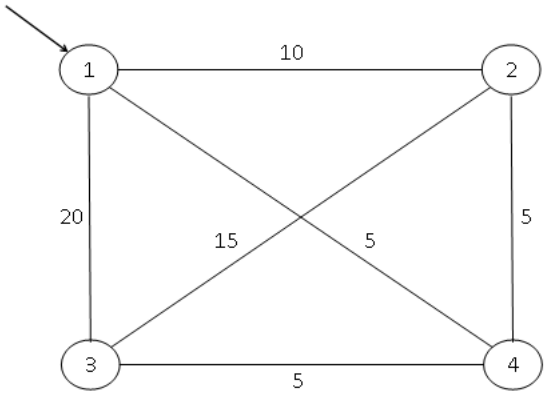
目录

1.实验内容

如图 1 所示，节点代表城市，节点之间的边代表城市之间的路径。每个城市都有一条进入路径和离开路径，不同的路径将耗费不同的旅费。旅行售货员选择城市 1 作为出发城市，途经其他每个城市，要求每个城市必须经过一次，并且只能经过一次，求一条具有最小耗费的路径，该路径从城市 1 出发，经过其余 5 个城市后，最后返回城市 1。

采用 C/C++/Java/Python 语言，采用回溯法，使用递归或非递归的方法，求解旅行售货员问题。要求完成以下内容：

1. 设计采用的界限函数（剪枝函数）；
2. 给出回溯过程中对结点采用的剪枝策略。
3. 编写基于回溯法的算法代码，求出一条最短回路及其长度；
4. 画出回溯搜索过程中生成的解空间树，说明发生剪枝的结点，以及树中各个叶结点、非叶结点对应的路径长度。



2.算法设计与实现

2.1. 算法设计

核心思想

1. **回溯法**：通过递归依次选择每个可能的城市，并尝试构造一条完整路径，逐层回溯寻找最优解。
2. **剪枝策略**：在搜索过程中，当发现某路径的部分结果已经不可能优于当前最优解时，立即停止该路径的搜索，以节省计算资源。

剪枝条件

1. **条件一**：当前路径长度 cv 加上前往下一城市的路径长度 w[x[i−1]][u] 超过当前最优路径长度 v_best，则不继续搜索。
2. **条件二**：当前城市和下一城市之间无路径（即 w[x[i−1]][u]=INF），则不继续搜索。

递归终止条件

当路径长度达到城市数量 n 时，检查是否可以回到起点。如果可以，并且总路径长度小于当前最优解，则更新最优解。

数据结构

- **w**：权重矩阵，存储城市间路径的长度。
- **x**：当前路径，记录已经访问的城市序列。
- **x_best**：最优路径，记录当前最短路径的城市序列。
- **cv**：当前路径长度。
- **v_best**：当前最优路径长度。

2.2. 算法实现

全局变量的初始化

```cpp
const int INF = numeric_limits<int>::max();  // 表示正无穷

// 权重矩阵
int w[5][5] = {
    {INF, 50, 25, INF, 20},
    {50, INF, 30, 20, 25},
    {25, 30, INF, 15, 10},
    {INF, 20, 15, INF, 135},
    {20, 25, 10, 135, INF}};

int n = 5;                          // 顶点数
vector<int> x(n), x_best(n);     // 当前路径和最优路径
int cv = 0;                         // 当前路径长度
int v_best = INF;                   // 最优路径长度
vector<bool> visited(n, false);  // 记录访问状态
```

- 权重矩阵 w 定义了城市之间的路径长度，INF 表示无路径。
- x 记录当前正在尝试的路径，x_best 存储最优路径。
- cv 和 v_best 分别表示当前路径长度和当前最优路径长度。
- visited 是一个布尔数组，用于标记某城市是否已访问。

回溯函数实现

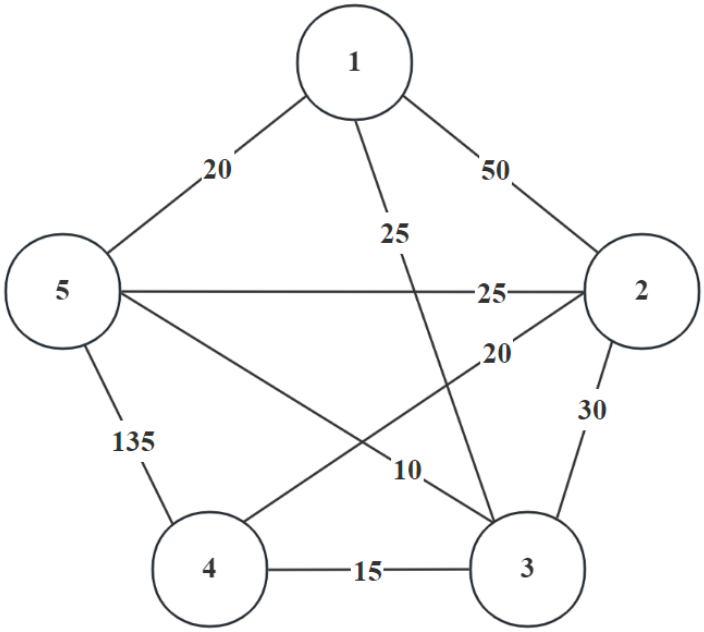
```cpp
void TspDFS(int i) {
    if (i == n) {                                // 递归结束条件
        if (w[x[i - 1]][0] < INF) {               // 剪枝 2
            if (cv + w[x[i - 1]][0] < v_best) {  // 更新最优解
                cv += w[x[i - 1]][0];             // 回到起点
                v_best = cv;                     // 更新最优路径长度
                x_best = x;                      // 保存最优路径
                cv -= w[x[i - 1]][0];             // 回溯
            }
        }
    }

    for (int u = 0; u < n; u++) {                 // 遍历所有城市
        if (!visited[u]) {                       // 城市未访问
            if (w[x[i - 1]][u] < INF) {           // 剪枝 2
                if (cv + w[x[i - 1]][u] < v_best) {  // 剪枝 1
                    cv += w[x[i - 1]][u];        // 更新当前路径长度
```

```
                    x[i] = u;                    // 选择城市 u
                    visited[u] = true;           // 标记城市 u 为已访问
                    TspDFS(i + 1);               // 递归搜索下一层
                    visited[u] = false;          // 回溯
                    x[i] = -1;                   // 清除选择
                    cv -= w[x[i - 1]][u];        // 回溯
                }
            }
        }
    }
}
```

- **递归结束**：当所有城市都访问过时（i == n），尝试回到起点。如果路径可行且长度小于当前最优路径，则更新最优解。
- **回溯核心**：遍历每个未访问的城市，判断是否满足剪枝条件，如果满足，则递归搜索下一层。

3.算法结果与分析

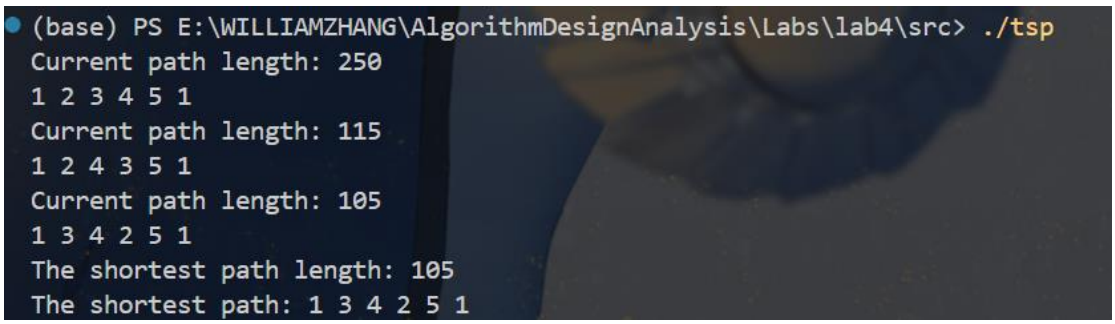
3.1. 地图展示



对于该地图，最短路径长度为 105

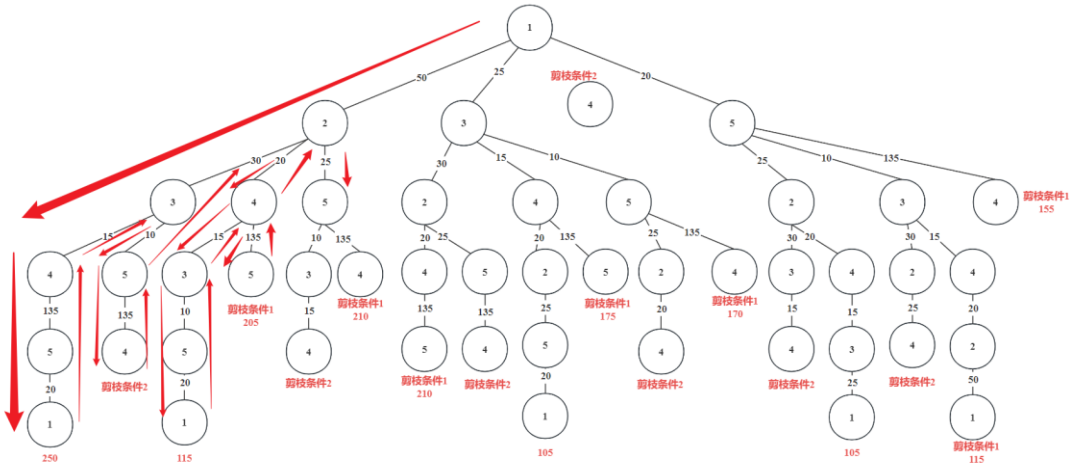
路线为$1 \to 3 \to 4 \to 2 \to 5 \to 1$

3.2. 算法执行过程显示



第一次找到 250 的路径，然后更新为 115 的新路线，最后更新为最短路线后，没有发现更短的路线。

3.3. 解空间树分析



回溯法的遍历顺序如图中的红色箭头，先向下遍历到能够达到的最深节点，然后返回一个节点，去向另外的子节点，之后再返回上一级节点，以此类推，直到遍历完整棵树。

对于剪枝条件，被剪去的位置已经在图中标出减去的原因和当时的路径值。

我们设定第一层为起始点 1.

由于当前路线 ＞ 当前最优路线而被减去的节点：

第三层最后一个节点；

第四层第 4，6，10，12 个节点；

第五层第 5 个节点；

由于没有可达路径（INF）而被减去的节点：

第二层第 3 个节点；

第五层第 2，4，6，8，9，11 个节点。

从第六层我们可以观察到最优路线的变化：

第一次：250，1 → 2 → 3 → 4 → 5 → 1；

第二次：115，1 → 2 → 4 → 3 → 5 → 1；

第三次：105，1 → 3 → 4 → 2 → 5 → 1；

第四次：105，1 → 5 → 2 → 4 → 3 → 1。

由于我们是**无向图**，所以在遍历过程中，一定存在两条最优路线。且两条路线的途径点顺序刚好相反。

4.算法效率分析

4.1. 时间复杂度分析

旅行售货员问题 (TSP) 是一个 NP-hard 问题。通过回溯法求解，该算法的时间复杂度主要由以下因素决定：

1. **节点搜索空间：**

- TSP 的搜索空间规模是$n!$，因为从起点出发，每次选择一个未访问的城市，直到所有城市都访问过。
- 每次递归中，需要遍历所有未访问的节点，因此原始的回溯算法时间复杂度为 $O(n!)$。

2. **剪枝优化：**

在代码中，通过以下两种剪枝策略减少搜索空间：

- **条件一**：当前路径长度 cv 加上前往下一城市的路径长度 w[x[i−1]][u] 超过当前最优路径长度 v_best，则不继续搜索。
- **条件二**：当前城市和下一城市之间无路径（即 w[x[i−1]][u=INF），则不继续搜索。

剪枝策略有效减少了不必要的分支搜索，但不能完全避免指数级增长。

因此，算法的时间复杂度在最优剪枝情况下约为 $O(C \cdot n!)$，其中 C 是剪枝后剩余的有效路径比例。

4.2. 性能优化

对于目前的回溯+剪枝优化，存在一些缺点：

- **无法处理大规模问题**：由于时间复杂度的指数增长，当城市数量较大（如 $n > 15$）时，计算量会变得不可承受。
- **受限于剪枝效率**：当路径权重较均匀或剪枝条件不充分时，剪枝的效果有限。

我们有如下改进方向：

1. 结合动态规划（DP+状态压缩）：

- 通过记录子问题的最优解来减少重复计算，将时间复杂度降至$O(n^2 \cdot 2^n)$。

2. 使用启发式算法（如遗传算法、模拟退火算法、蚁群算法等）：

- 在大规模城市情况下，可以使用启发式方法找到近似解，提升计算效率。

5.总结心得

在本次实验中，我通过回溯法成功解决了小规模的旅行售货员问题（TSP）。

通过递归函数实现，我对回溯法如何通过递归和状态回退系统地搜索解空间有了更深刻的认识。同时，我认识到剪枝优化在解决组合优化问题中的重要性。

本次实验进一步提升了我的递归调试能力。在递归算法中，调试往往较为复杂，而我通过逐步打印路径和路径长度验证了递归函数的正确性，使我更好地掌握了递归回溯的过程。同时，我也深刻体会到暴力解法的局限性。

此外，本次实验也让我更加意识到代码规范和算法设计的重要性。清晰的变量命名和注释不仅提高了代码的可读性，还为后续的调试和维护提供了便利。例如，通过直观命名路径长度 cv 和最优路径长度 v_best，使得逻辑更为清晰。

总的来说，本次实验让我不仅巩固了回溯法的理论知识，也提升了代码实现和调试能力。我对 TSP 问题的复杂性和优化潜力有了更深的理解，也意识到算法设计中效率优化与问题建模同等重要。这些收获和反思将成为我未来学习和研究的重要指导。