

# 重庆大学

# 学生实验报告

实验课程名称 算法设计与分析

开课实验室 DS1501

学 院 大数据与软件学院 年级 2023 专业班 软工 01

学 生 姓 名        学 号       

开 课 时 间 2024 至 2025 学年第 2 学期

总 成 绩	
教师签名	付春雷

# 《算法设计与分析》实验报告

开课实验室：DS1501

2025年5月26日

学院	大数据与软件学院	年级、专业、班		姓名		成绩	
课程名称	算法设计与分析	实验项目名称	回溯法实验和分枝限界法实验	指导教师	付春雷		
教师评语	教师签名： 年 月 日						

## 一、实验目的

- 掌握回溯法算法框架，掌握分枝限界法的特点和算法框架。
- 熟练掌握“0/1 背包问题的回溯求解算法”的实现，熟练掌握“4 皇后问题求解算法”的实现。
- **主要任务：**实现教材“第 5 章 求解 0/1 背包问题”和实验指导书“第 2 章 2.6.1 小节 求解 4 皇后问题”。

## 二、使用仪器、材料

PC 微机；  
Windows 操作系统，VS CODE+MINGW64 编译环境（不限）；

## 三、实验步骤

1. 有 n 个重量分别为 w1、w2、…、wn, 的物品(物品编号为 1~n), 它们的价值分别为 v1、v2、v3…、vn., 给定一个容量为 W 的背包。设计从这些物品中选取一部分物品放入该背包的方案, 每个物品要么选中要么不选中, 要求选中的物品不仅能够放到背包中, 而且具有最大的价值。我们给出有 1, 2, 3, 4 个物品, 重量分别为 5, 3, 2, 1, 价值分别为 4, 4, 3, 1, 当 W=6 时求出所有解和最佳解

```
#include <iostream>
#include <vector>
using namespace std;

struct Item {
    int value;
    int weight;
    int number;
};

vector<vector<int>> all_solutions; // 所有可行解
int max_value = 0; // 最大价值
vector<int> best_combination; // 最佳解组合

// 回溯函数
void backtrack(const Item items[], int index, int current_weight, int current_value,
              vector<int>& selected, int W) {
    if (index == 4) { // 处理完所有物品
        if (current_weight <= W) { // 合法解
            all_solutions.push_back(selected);
            if (current_value > max_value) {
                max_value = current_value;
                best_combination = selected;
            }
        }
    } else {
        selected.push_back(index);
        backtrack(items, index + 1, current_weight + items[index].weight, current_value + items[index].value, selected, W);
        selected.pop_back();
        backtrack(items, index + 1, current_weight, current_value, selected, W);
    }
}
```

```

        all_solutions.push_back(selected); // 将每一次找到的求解方法作为元素 push 到所有
解决方法中
    if (current_value > max_value) { // 更新最佳解
        max_value = current_value;
        best_combination = selected;
    }
}
return;
}

// 选择当前物品（左子树）
if (current_weight + items[index].weight <= W) {
    selected.push_back(items[index].number);
    backtrack(items, index + 1, current_weight + items[index].weight,
              current_value + items[index].value, selected, W);
    selected.pop_back(); // 回溯
}

// 不选择当前物品（右子树）
backtrack(items, index + 1, current_weight, current_value, selected, W);
}

int main() {
    Item items[4] = {
        {4, 5, 1}, // 编号 1, 价值 4, 重量 5
        {4, 3, 2}, // 编号 2, 价值 4, 重量 3
        {3, 2, 3}, // 编号 3, 价值 3, 重量 2
        {1, 1, 4} // 编号 4, 价值 1, 重量 1
    };

    vector<int> selected;
    backtrack(items, 0, 0, 0, selected, 6);

    // 输出所有解
    cout << "所有解: " << endl;
    for (auto& solution : all_solutions) {
        int total_weight = 0;
        int total_value = 0;
        cout << "选中物品: ";
        for (int num : solution) {
            for (int i = 0; i < 4; ++i) {
                if (items[i].number == num) {
                    total_weight += items[i].weight;
                    total_value += items[i].value;
                }
            }
            cout << num << " ";
        }
        cout << ", 总重量: " << total_weight << ", 总价值: " << total_value << endl;
    }

    // 输出最佳解
    cout << "\n 最佳解: " << endl;
    int total_weight = 0;
    int total_value = 0;
    cout << "选中物品: ";
    for (int num : best_combination) {
        for (int i = 0; i < 4; ++i) {

```

```

        if (items[i].number == num) {
            total_weight += items[i].weight;
            total_value += items[i].value;
        }
    }
    cout << num << " ";
}
cout << ", 总重量: " << total_weight << ", 总价值: " << total_value << endl;

return 0;
}

```

2.在  $n \times n$  的方格棋盘上放置  $n$  个皇后,要求每个皇后不同行、不同列、不同左右对角线。现在  $n=4$ ,用 c++ 的分枝限界法求解

```

#include <iostream>
#include <vector>
#include <queue>
using namespace std;

// 检查当前列位置是否安全
bool isSafe(int new_col, const vector<int>& cols) {
    int new_row = cols.size();
    for (int i = 0; i < new_row; ++i) {
        if (cols[i] == new_col || abs(i - new_row) == abs(cols[i] - new_col)) {
            return false;
        }
    }
    return true;
}

// 使用分支限界法求解 N 皇后问题
vector<int> solveNQueens(int n) {
    queue<vector<int>> q;
    q.push(vector<int>()); // 初始化队列

    while (!q.empty()) {
        vector<int> cols = q.front();
        q.pop();

        int current_row = cols.size();
        if (current_row == n) { // 找到解
            return cols;
        }

        // 生成所有可能的子节点
        for (int new_col = 0; new_col < n; ++new_col) {
            if (isSafe(new_col, cols)) {
                vector<int> new_cols(cols);
                new_cols.push_back(new_col);
                q.push(new_cols);
            }
        }
    }

    return vector<int>(); // 无解
}

int main() {

```

```
int n = 4;
vector<int> solution = solveNQueens(n);

if (solution.empty()) {
    cout << "无解" << endl;
}
else {
    cout << "找到解 (columns per row): ";
    for (int col : solution) {
        cout << col << " ";
    }
    cout << "\nBoard:\n";
    for (int row = 0; row < n; ++row) {
        for (int c = 0; c < n; ++c) {
            cout << (c == solution[row] ? "Q " : ". ");
        }
        cout << endl;
    }
}
return 0;
}
```

四、实验过程原始记录(数据、图表、计算等)

1.

所有解：

```
选中物品：1 4 , 总重量：6, 总价值：5
选中物品：1 , 总重量：5, 总价值：4
选中物品：2 3 4 , 总重量：6, 总价值：8
选中物品：2 3 , 总重量：5, 总价值：7
选中物品：2 4 , 总重量：4, 总价值：5
选中物品：2 , 总重量：3, 总价值：4
选中物品：3 4 , 总重量：3, 总价值：4
选中物品：3 , 总重量：2, 总价值：3
选中物品：4 , 总重量：1, 总价值：1
选中物品：, 总重量：0, 总价值：0
```

最佳解：

```
选中物品：2 3 4 , 总重量：6, 总价值：8
```

2.

找到解：

Board:

```
. Q . .
. . . Q
Q . . .
. . Q .
```

D:\代码文件\c++\exer1\x64\D

要在调试停止时自动关闭控制台

按任意键关闭此窗口. . .

## **五、实验结果及分析**

结果都已对应显示在原始数据记录中，结果都与预期的分析符合。