# 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 班

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 |

大数据与软件学院制

# 《算法设计与分析》实验报告

**开课实验室：DS1501** **2025 年 5 月 23 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 | 蛮力法实验和回溯法实验 | 指导教师 | 付春雷 | | |
| 教师评语 | 教师签名：<br>年 月 日 | | | | | | |

## 一、实验目的

- 掌握蛮力法和回溯法的设计思想，包括蛮力法的基本应用、回溯法的算法框架等。
- 熟练掌握“钱币兑换问题求解算法”的实现，熟练掌握“填字游戏问题求解算法”的实现，提供输入案例检测算法的正确性。
- 主要任务：实现教材配套实验指导书“第 2 章 2.4.2 小节 求解钱币兑换问题”和“第 2 章 2.5.2 小节 求解填字游戏问题”。

## 二、使用仪器、材料

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

## 三、实验步骤

1.钱币兑换问题：某个国家仅有 1 分、2 分和 5 分硬币,将钱 n(n>5)兑换成硬币有很多种兑法。编写个实验程序计算出 10 分钱有多少种兑法,并列出每种兑换方式。

用蛮力法求解，可以直接嵌套三层循环

```
#include<iostream>
using namespace std;
int main()
{
    int n = 10;
    cin >> n;
    int count = 0;
    cout << "当 n= " << n << endl;
    cout << "5 分， 2 分， 1 分" << endl;
    for (int i = 0; i <=n / 5; i++) {
        for (int j = 0; j <=n/2; j++)
        {
            for (int k = 0; k <= n; k++) {
                if (i * 5 + j * 2 + k == n) {
                    count++;
                    cout << i << "          " << j << "          " << k << "          " << endl;
                }
            }
        }
```

```
    }
    cout << "totall: " << count <<"种"<< endl;
    return 0;
}
```

2.求解棋盘问题：
在 3X3 个方格的方阵中要填入数字 1~10 的某 9 个数字,每个方格填一个整数,使所有相邻两个方格内的两个整数之和为素数。编写一个实验程序,求出所有满足这个要求的数字填法。

用回溯法求解：

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

// 判断是否为素数
bool is_prime(int n) {
    if (n <= 1) return false;
    if (n == 2) return true;
    if (n % 2 == 0) return false;
    for (int i = 3; i * i <= n; i += 2) {
        if (n % i == 0) return false;
    }
    return true;
}

// 预计算两数之和是否为素数
vector<vector<bool>> generate_prime_sums() {
    vector<vector<bool>> ps(11, vector<bool>(11, false));
    for (int a = 1; a <= 10; ++a) {
        for (int b = 1; b <= 10; ++b) {
            int sum = a + b;
            if (is_prime(sum)) {
                ps[a][b] = true;
            }
        }
    }
    return ps;
}

const vector<vector<bool>> prime_sums = generate_prime_sums();

// 回溯函数
void backtrack(int row, int col, vector<vector<int>>& grid, vector<bool>& used, vector<vector<vector<int>>>& solutions) {
    if (row == 3) { // 找到解，保存结果
        solutions.push_back(grid);
        return;
    }

    // 计算下一个位置
    int next_row = row;
    int next_col = col + 1;
    if (next_col == 3) {
        next_row++;
        next_col = 0;
    }

    // 尝试所有可能的数字
```

```
        for (int num = 1; num <= 10; ++num) {
            if (!used[num]) {
                bool valid = true;

                // 检查左方邻居
                if (col > 0 && !prime_sums[num][grid[row][col - 1]]) {
                    valid = false;
                }

                // 检查上方邻居
                if (row > 0 && !prime_sums[num][grid[row - 1][col]]) {
                    valid = false;
                }

                if (valid) {
                    used[num] = true;              // 标记为已使用
                    grid[row][col] = num;      // 填入当前数字
                    backtrack(next_row, next_col, grid, used, solutions);
                    grid[row][col] = 0;           // 回溯
                    used[num] = false;
                }
            }
        }
    }

    int main() {
        vector<vector<int>> grid(3, vector<int>(3, 0)); // 3x3 网格
        vector<bool> used(11, false);                                  // 已用数字标记
        vector<vector<vector<int>>> solutions;                  // 存储所有解

        backtrack(0, 0, grid, used, solutions);              // 从左上角开始填充

        // 输出结果
        cout << "共有 " << solutions.size() << " 种解法：" << endl;
        for (const auto& grid : solutions) {
            for (const auto& row : grid) {
                for (int num : row) {
                    cout << num << "\t";
                }
                cout << endl;
            }
            cout << "--------------------" << endl;
        }
        return 0;
    }
```

四、实验过程原始记录(数据、图表、计算等)

1. 运行结果：

```
10
当n= 10
5分，  2分，  1分
0       0       10
0       1       8
0       2       6
0       3       4
0       4       2
0       5       0
1       0       5
1       1       3
1       2       1
2       0       0
totall: 10种
```

2. 运行结果：共有 128 种解法，这里截取部分结果

```
共有 128 种解法:
1       2       5
4       3       8
7       10      9
------------------
1       2       5
4       9       8
7       10      3
------------------
1       2       5
10      3       8
7       4       9
------------------
1       2       5
10      9       8
7       4       3
------------------
1       4       7
2       3       10
5       8       9
------------------
1       4       7
2       9       10
5       8       3
------------------
1       10      7
2       3       4
5       8       9
------------------
```

```
------------------
2       1       6
3       4       7
8       9       10
------------------
2       1       6
3       10      7
8       9       4
------------------
2       1       6
9       4       7
8       3       10
------------------
2       1       6
9       10      7
8       3       4
------------------
2       3       8
1       4       9
6       7       10
------------------
```

**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。