《算法设计与分析》习题 8 作业

学　号：1004191211　　姓　名：郎文鹏

日　期：2021/11/11　　得　分：

问题四：修改8.3.1节n皇后问题的算法，要求输出n皇后问题的所有解。

【问题】

在$8*8$的棋盘上摆放八个皇后，使其不能互相攻击，即任意两个皇后不能处于同一行、同一列、同一斜线上。可以把八皇后问题扩展到n皇后问题，即在$n*n$的棋盘上摆放n个皇后，使任意两个皇后都不能处于同一行、同一列或同一斜线上。

【原理】

显然棋盘的每一行都必须摆放一个皇后，所以我们在从上到下依次每一行的不同位置尝试摆放一个皇后，然后判断是否与题目相违背，否则尝试下一个位置。由于需要找出所有的解所以我们没找到一个答案记录下来，需要继续在解空间树上继续搜索。

【调试】

我们假设给第一行的皇后寻找合法的放置列数，因为需要找出所有的可能所以确定一个方案后应该继续尝试下一个列数放置而不能停止。所有将原题目的源码关键字$return$提前退出删去，同时为了能够尝试所有的列数所以需要加一个外部的遍历所有列数的循环。

【源码】

```
#include <bits/stdc++.h>
#pragma GCC optimize(2)
#pragma G++ optimize(2)
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
const int maxn = 25;

int x[maxn];

bool Place(int k) {
```

```
    for (int i = 0; i < k; ++i)
        if (x[i] == x[k] || abs(i - k) == abs(x[i] - x[k])) return 1;
    return 0;
}

void Queen(int n) {
    int k = 0;
    while (k >= 0) {
        ++x[k];  //给每个皇后寻找合适的列
        while (x[k] < n && Place(k) == 1) ++x[k];
        if (x[k] < n) {
            if (k == n - 1) {
                for (int i = 0; i < n; ++i)
                    cout << x[i] + 1 << ' ';
                cout << endl;
            } else
                x[++k] = -1;
        } else
            --k;
    }
}

void solve() {
    int n;
    cin >> n;
    Queen(n);
}

signed main() {
    ios::sync_with_stdio(false), cin.tie(0);
    freopen("IO\\in.txt", "r", stdin);
    freopen("IO\\out.txt", "w", stdout);
    solve();
    return 0;
}
```

【答案】

经过计算 8 皇后问题有 92 个解，为了方便验证演示这里以 4 皇后问题的解截图。

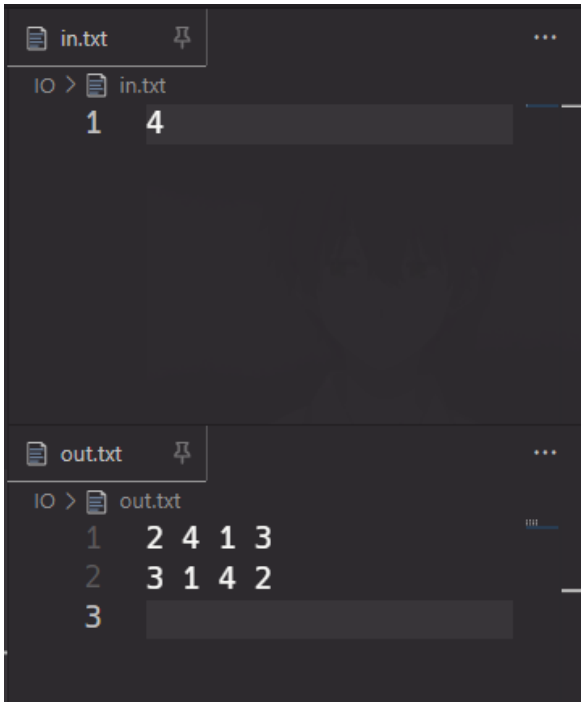

图 1 问题五答案截图

问题五：给定一个正整数集合$X=\{x_1,x_2,\ldots,x_n\}$和一个正整数y，设计一个回溯算法求集合X的一个子集Y，使得Y中元素之和等于y。

【原理】

结合背包问题的思考方式我们考虑对于集合X中的每个数都有选与不选两种可能，所以我们依次从第一个数到最后一个数向下搜索两个分支，直至找到一个满足条件的子集输出。

【调试】

设计输入样例为 X 集合为$1-10$共11个数，要求和等于21。为了找遍所有可能并输出，我们首先对于每个数都进行选与不选两种情况向下进行搜索，其次维护一个容器用来记录每个状态时子集的选择情况。

【源码】

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;

int n, y;
vector<int> x;
```

```
vector<vector<int> > xx;

void search() {
    int now = 0;
    while (now < n) {
        int sz = xx.size();
        for (int i = 0; i < sz; ++i) {
            vector<int> temp(xx[i]);
            temp.push_back(x[now]);
            if (accumulate(temp.begin(), temp.end(), 0) <= y)   //小优化
                xx.push_back(temp);
        }
        ++now;
    }
    for (auto &e : xx) {
        if (accumulate(e.begin(), e.end(), 0) == y) {
            for (auto &o : e)
                cout << o << ' ';
            cout << endl;
        }
    }
}

void solve() {
    cin >> n >> y;
    x.resize(n);
    for (int i = 0; i < n; ++i) cin >> x[i];
    xx.push_back(vector<int>{});
    search();
}

signed main() {
    ios::sync_with_stdio(false), cin.tie(0);
    freopen("IO\\in.txt", "r", stdin);
    freopen("IO\\out.txt", "w", stdout);
    solve();
    return 0;
}
```

【答案】

最终计算出共 33 种不相同的方案。

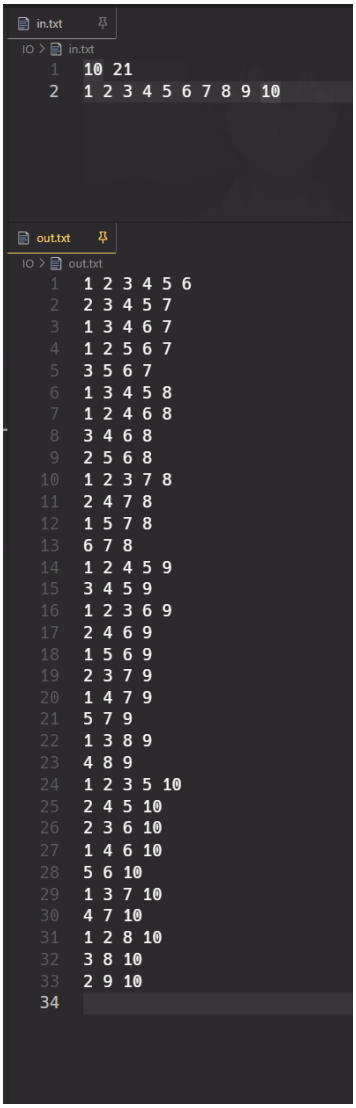

图 2 问题五答案截图

问题六：迷宫问题。迷宫问题的求解是实验心理学中的一个经典问题，心理学家把一只老鼠从一个无顶盖的大盒子的入口处赶进迷宫，迷宫中设置很多隔壁，对前进方向形成了多出障碍，心理学家在迷宫的唯一出口处放置了一块奶酪，吸引老鼠在迷宫中寻找通路以到达出口。设计回溯算法实现如图所示迷宫的求解。

【原理】搜索经典问题。通俗来说就是反悔思想，走错路了就回头原路返回到上一个分叉点然后尝试其他未知方向。关键点就在于回溯，我们标记走过的路，那么每次分叉点走向另一个方向时，之前走过的一些路就应该回溯清除标记。

【调试】

我们从(0,0)开始进行搜索，每次走过一个点就标记为-1，然后考虑走错了回头的

情况，这时候我们需要重新清除标记当做没来过。找到第一个到达(5,7)的时候终止搜索，并从起点根据一路打上−1的标记输出即可打印出整个路径。

【源码】

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
typedef pair<int, int> pii;
const int d[8][2] = {{-1, 0}, {1, 0}, {0, 1}, {0, -1}, {1, 1}, {-1, -1},
{1, -1}, {-1, 1}};  //顺时针行走

int mz[15][15];  //存储每一个可以走的点距离终点的距离
struct pos {
    int x, y;  //横纵坐标以及面对方向
    pos(int x, int y) : x(x), y(y) {}
};

bool check(pos temp) {
    if (temp.x < 0 || temp.x > 5) return false;
    if (temp.y < 0 || temp.y > 7) return false;  //判断是否越界
    if (mz[temp.x][temp.y] != 0) return false;   //判断能否走和是否走过
    return true;
}

void find(int x, int y) {
    stack<pos> sta;
    sta.push(pos(0, 0));
    while (sta.size()) {
    Loop:
        pos now = sta.top();
        mz[now.x][now.y] = -1;
        if (now.x == 5 && now.y == 7)
            return;

        for (int i = 0; i < 8; ++i) {
            pos nex = pos(now.x + d[i][0], now.y + d[i][1]);
            if (check(nex)) {
                sta.push(nex);
                goto Loop;
            }
        }

        pos top = sta.top();
```

```
        mz[top.x][top.y] = 0;
        sta.pop();
    }
}

void solve() {
    for (int i = 0; i <= 5; ++i)
        for (int j = 0; j <= 7; ++j)
            cin >> mz[i][j];
    find(0, 0);
    // for (int i = 0; i <= 5; ++i) {
    //     for (int j = 0; j <= 7; ++j)
    //         cout << mz[i][j] << ' ';
    //     cout << endl;
    // }
    cout << "(0, 0)";
    int x = 0, y = 0;
    mz[x][y] = 0;
    while (x != 5 || y != 7) {
        for (int i = 0; i < 8; ++i) {
            int tx = x + d[i][0];
            int ty = y + d[i][1];
            if (tx >= 0 && ty >= 0 && tx <= 5 && ty <= 7) {
                if (mz[tx][ty] == -1) {
                    mz[tx][ty] = 0;
                    x = tx, y = ty;
                    cout << " -> (" << tx << ", " << ty << ")";
                    break;
                }
            }
        }
    }
}

signed main() {
    ios::sync_with_stdio(false), cin.tie(0);
    freopen("IO\\in.txt", "r", stdin);
    freopen("IO\\out.txt", "w", stdout);
    solve();
    return 0;
}
```

【答案】

(0, 0) -> (1, 1) -> (2, 2) -> (2, 3) -> (1, 3) -> (2, 4) -> (3, 4) -> (4, 5) -> (4, 6) -> (4, 7)

-> (5, 7)

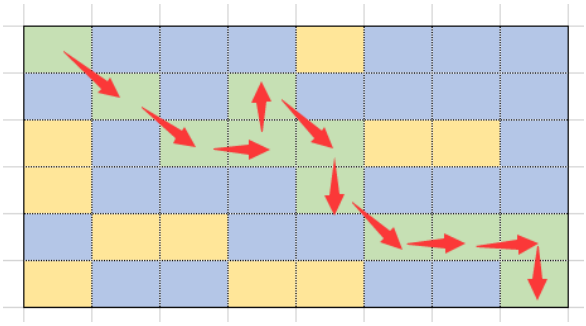

图 3 问题 6 答案图示

可以看出来回溯法成功找出了一条路径，但由于方向设定不合理导致饶远了。