第二次作业

每个题目算法代码保存在对应文件夹内，算法输出保存在对应文件夹中的 test 文件夹内。

2-1 Hanoi

证明：

设 hanoi(n,A,B,C)是将塔座 A 上的 n 个圆盘，以塔座 C 为辅助塔座，移到目的塔座 B

A:0(n)1(n-1)01...01

B:空

C:空

Hanoi 塔算法 hanoi(n,A,B,C)由以下个步骤组成：

（1）hanoi(n-1,A,B,C)

（2）move(A,B)

（3）hanoi(n-1,C,A,B)

1. 我们假设步骤(1)不违反规则，将 A 上 1~n-1 圆盘经 B 移动到 C，且在移动过程中，塔座 C 上最低圆盘的编号与 n-1 具有相同颜色，塔座 B 上最低圆盘的编号与 n-1 具有不同颜色，从而 B 塔座上最低圆盘的编号与 n 具有相同颜色，塔座 C 上最低圆盘的编号与 n 相同颜色。

A:0(n)

B:空

C:1(n-1)0101...01

2. 步骤(2)也不违反规则，且塔座 B 上最低圆盘的编号与 n 相同颜色。

A:空

B:0(n)

C:1(n-1)0101...01

3. 步骤(3)不违反规则，且在移动过程中，塔座 B 上倒数第二个圆盘的编号与 n-1 具有相同颜色，塔座 A 上最低圆盘与 n-1 具有不同的颜色，从而塔座 B 上倒数第二个圆盘的编号与 n 具有不同颜色，塔座 A 上最低圆盘与 n 具有相同颜色。因此在移动过程中，塔座 B 上圆盘不违反规则，而且塔座 B 上最低圆盘的编号与 n 具有相同颜色，塔座 C 上最低圆盘的编号与 n 具有不同颜色。

A:空

B:0(n)1(n-1)01...01

C:空

由数学归纳法可知，hanoi(n,A,B,C)不违反规则。

算法思想：

采用递归方法，

（1）当没有圆盘可移动时，跳出；

（2）分两步，第一步通过辅助位 c，将 a 中圆盘转到 b；再通过辅助位 a，将 b 中圆盘转到 c。

实验核心源代码加注释：

```
    char a = 'A', b = 'B', c = 'C';
    // 将 n 个盘从 A 全部送到 C
    // a 原位置， b 辅助，c 目标位置
    void hanoi(char a, char b, char c, int n, fstream &f) {
        // 递归出口
        if (n < 1) return;
        hanoi(a, c, b, n - 1, f);
        f << n << " " << a << " " << c << endl;
        hanoi(b, a, c, n - 1, f);
    }
    // 读取输入
    void input_hanoi(string path, string filename) {
        fstream file;
        string data;
        path += filename;
        file.open(path);
        if (!file.is_open()) {
            cout << "open file failure" << endl;
            return;
        }
        while (!file.eof())
            file >> data;
        file.close();
        n = atoi(data.c_str());
    }
    // 主函数
    int main() {
        string str;
        string path = "E:\\tools\\Graduated\\algorithm\\Postraduate-Class-
Algorithm\\assignment 2-1-Hanoi\\test\\";
        do
        {
            cout << "请输入文件名：" << endl;
            cin >> str;
            if (str == "END") break;
            // 读取输入
            input_hanoi(path, str);
            fstream file;
            file.open(path + "output.txt", ios::app | ios::out);
            file << "output from: " << str << endl;
            // 调用 hanoi 递归函数，c 作为辅助位，将 a 的圆盘移到 c
            hanoi(a, c, b, n, file);
            file.close();
            cout << "输出结束" << endl;
```

```
    } while (true);
    cout << "程序结束" << endl;
    return 0;
}
```

2-2 Permutation with Repetition

算法思想：

使用 DFS 的递归方法。

（1） 当递归深度超过元素数量 n 时，跳出。

（2） 当递归深度小于 n 时，循环选择一个数量大于 0 的元素进入下一层，元素数量减一；之后再恢复这个元素的数量。

实验核心源代码加注释：

```
// 变量定义
const int maxn = 510;          // 题目给定长度不超过 500
int n, res;
int mp[30]{ 0 };               // 只有 26 个小写字母,只需要 26 个索引点模拟 map
char str_input[maxn], cstr[maxn];
// 重置 map
void init() {
    res = 0;
    for (int i = 0; i < 26; i++)
        mp[i] = 0;
}
// 模拟 map，记录每个字母出现次数
void mapping() {
    for (int i = 0; i < n; i++)
        mp[(int)(str_input[i] - 'a')]++;
}
// 读取输入数据
int input(string path, string filename) {
    fstream file;
    file.open(path + filename);
    if (!file.is_open()) {
        cout << "open file failure" << endl;
        return NULL;
    }
    string num;
    file >> num;
    int n = atoi(num.c_str());
    for (int i = 0; i < n; i++)
        file >> str_input[i];
    file.close();
    return n;
}
// DFS
void dfs(int depth, fstream &f) {
    if (depth >= n)
    {
        res++;
        for (int i = 0; i < n; i++)
            f << cstr[i];
        f << endl;
        return;
    }

    for (int i = 0; i < 26; i++)
```

```
    {
        if (mp[i] > 0)
        {
            cstr[depth] = i + 'a';
            mp[i]--;
            dfs(depth + 1, f);
            mp[i]++;
        }
    }
}
// 主函数
int main() {
    string str;
    string  path  =  "E:\\tools\\Graduated\\algorithm\\Postraduate-Class-
Algorithm\\assignment 2-2-Permutation with Repetition\\test\\";
    do
    {
        cout << "请输入文件名：" << endl;
        cin >> str;
        if (str == "END") break;
        n = input(path, str);
        // 初始化 map 序列
        init();
        // 处理字母序列
        mapping();

        fstream file;
        // permXX.in
        file.open(path + "output.txt", ios::app | ios::out);
        file << "output from: " << str << endl;

        dfs(0, file);
        file << res << endl;
        file.close();
        cout << "输出结束" << endl;
    } while (true);
    return 0;
}
```

2-3 Set Partition 2

算法思想：

利用递归思想：

（1） 不会存在不够分的情况，也不会存在一下都不分的情况，所以返回 0；

（2） n 个元素放入 1 个集合，以及 n 个元素放入 n 个集合都是 1 中情况，返回 1；

（3） 存在两种分法：

a. 将 n-1 个元素放入 m-1 个集合中，剩余 1 个自己 1 个集合，算 1 种情况；

b. 将 n-1 个元素放入 m 个集合中，剩余 1 个可以放入这 m 个集合中，算 m 种情况；

返回 a + m*b。

实验核心源代码加注释：

```
// 全局变量
int n, m;              // n 表示元素个数，m 表示划分的非空子集个数
double res = 0;        // 记录组合个数
// 读取输入数据
void input(string path, string filename) {
    fstream file;
    file.open(path + filename);
    if (!file.is_open()) {
        cout << "open file failure" << endl;
        return;
    }
    string N, M;
    file >> N >> M;
    n = atoi(N.c_str());
    m = atoi(M.c_str());
    file.close();
}
// 集合划分函数
double partition(int n, int m) {
    // 不会存在不够分的情况，也不会存在一下都不分的情况
    if (n < m || m == 0) return 0;
    // 把 n 个元素放进 1 个集合，和把 n 个元素放进 n 个集合都是一种情况
    if (m == 1 || m == n) return 1;

    return partition(n - 1, m - 1) + m * partition(n-1, m);
}
// 主函数
int main() {
    string str;
    string  path  =  "E:\\tools\\Graduated\\algorithm\\Postraduate-Class-
Algorithm\\assignment 2-3-Set Partition 2\\test\\";
    do
    {
        cout << "请输入文件名：" << endl;
        cin >> str;
        if (str == "END") break;

        input(path, str);
        fstream file;
```

```
        file.open(path + "output.txt", ios::app | ios::out);
        file << "output from: " << str << endl;
        // 重置 res 记录组合个数
        res = 0;
        res = partition(n, m);
        file << res << endl;
        file.close();
        cout << "输出结束" << endl;
    } while (true);
    return 0;
}
```

2-4 推箱子问题

算法思想：

利用 BFS 广度优先搜索，在四个方向上遍历小地图，同时判断小人是否走到箱子坐标上。

（1） 若没有，则标记当前情况，同时继续遍历；

（2） 若小人和箱子的坐标相同，则模拟小人按当前方向推一下箱子；

（3） 当箱子坐标和目标位置相同时，返回步数。

实验样例：

```
// X 玩家， * 箱子， # 障碍， @ 目的地
```

```
4   4
. . . .
. . * @
. . . .
. X . .
6   6
. . . # . .
. . . . . .
# * # # . .
. . # # . #
. . X . . .
. @ # . . .
0
```

答案：

```
3
11
```

实验核心源代码加注释：

```
// 变量定义
// 上 下 左 右 移动方向
const vector<pair<int, int>> dir = { { 0,-1 },{ 0,1 },{ -1,0 },{ 1,0 } };
int n, m;
// path 存储地图
vector<string> path;
// visit 记录小人和箱子的移动路径
bool visit[10][10][10][10];
// node 结构记录当前步数，小人和箱子的坐标
struct node {
    int x, y, bx, by, step;
    node(int x, int y, int bx, int by, int step) :
        x(x), y(y), bx(bx), by(by), step(step) {};
};
// 判断小人是否可以移动到给定的坐标
bool valid(int x, int y) {
    if (x < 0 || x >= n || y < 0 || y >= m) return false;
    if (path[x][y] == '#') return false;
    return true;
}
// BFS 遍历地图 寻找最优路径
int bfs(int x_begin, int y_begin, int x_end, int y_end) {
    queue<node> q;
    q.push(node(x_begin, y_begin, x_end, y_end, 0));
    visit[x_begin][y_begin][x_end][y_end] = 1;
    while (!q.empty())
```

```
    {
        node d = q.front();
        q.pop();
        int bx = d.bx, by = d.by, step = d.step;
        // 四个方向尝试
        for (int i = 0; i < 4; i++)
        {
            int x = d.x + dir[i].first;
            int y = d.y + dir[i].second;
            // 假设箱子被从这个方向推一下后的位置
            int next_x = bx + dir[i].first;
            int next_y = by + dir[i].second;
            if (!valid(x, y)) continue;
            // 记录小人走的路径
            // 先走到箱子旁边
            if ((x != bx || y != by) && !visit[x][y][bx][by])
            {
                visit[x][y][bx][by] = 1;
                q.push(node(x, y, bx, by, step + 1));
            }
            // 当小人和箱子坐标相同时，说明要推一下箱子了，就按之前模拟的方向更新箱子坐标
            else if (x == bx && y == by && valid(next_x, next_y) && !visit[x][y][next_x][next_y]) {
                visit[x][y][next_x][next_y] = 1;
                if (path[next_x][next_y] == '@')
                    return step + 1;
                q.push(node(x, y, next_x, next_y, step + 1));
            }
        }
    }
    return -1;
}
// 主函数
int main() {
    do
    {
        cin >> n;
        // 题目要求，当 n=0 时退出程序
        if (n == 0) break;
        cin >> m;
        // 每次 BFS 前重置 visit
        memset(visit, 0, sizeof(visit));
        path = vector<string>(n, string(""));
```

```
        for (int i = 0; i < n; i++)
            cin >> path[i];
        // 找到箱子和小人的位置
        int x_begin, y_begin, x_end, y_end;
        for (int i = 0; i < n; i++)
        {
            for (int j = 0; j < m; j++)
            {
                if (path[i][j] == 'X')      // 找到小人
                    x_begin = i, y_begin = j;
                else if (path[i][j] == '*')
                    x_end = i, y_end = j;   // 找到箱子
            }
        }
        // 输出步数
        cout << bfs(x_begin, y_begin, x_end, y_end) << endl;
    } while (true);

    return 0;
}
```