搜索算法

DFS 标准代码模板

代码实现：

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

// ================= 全局变量定义 =================
int n, m;               // 网格的行数n，列数m（题目通常从1开始计数）
int map[50][50];        // 存储网格状态（0表示可通行，1表示障碍）
bool vis[50][50];       // 访问标记数组，防止重复访问同一位置
int ans = 0;            // 结果计数器，存储从起点到终点的总路径数

// 方向数组：右(0,1)、下(1,0)、左(0,-1)、上(-1,0)
int dx[4] = {0, 1, 0, -1};
int dy[4] = {1, 0, -1, 0};

// ================= DFS深度优先搜索 =================
/**
 * 递归搜索所有从(x,y)到(n,m)的路径
 *
 * @param x 当前位置行坐标（从1开始）
 * @param y 当前位置列坐标（从1开始）
 */
void dfs(int x, int y) {

    // 1. 终点判断：到达右下角(n,m)时，找到一条完整路径
    if (x == n && y == m) {
        ans++;          // 路径计数加1
        return;         // 结束当前路径的搜索
    }

    // 2. 遍历四个方向进行探索
    for (int i = 0; i < 4; i++) {
        int nx = x + dx[i];    // 计算下一个位置的行坐标
        int ny = y + dy[i];    // 计算下一个位置的列坐标

        // 3. 可行性检查：
        //    a) 不越界：nx∈[1,n]，ny∈[1,m]
        //    b) 可通行：map[nx][ny] == 0
        //    c) 未访问：!vis[nx][ny]
        if (nx >= 1 && nx <= n && ny >= 1 && ny <= m &&
            map[nx][ny] == 0 && !vis[nx][ny]) {

            vis[nx][ny] = true;    // 标记为已访问（防止循环）
            dfs(nx, ny);           // 递归深入搜索下一位置
            vis[nx][ny] = false;   // 回溯：恢复未访问状态（允许其他路径探索）
        }
    }
}

// ================= 主程序入口 =================
int main() {
    // 输入网格规格
    cin >> n >> m;

    // 输入网格数据（假设题目中0表示空地，1表示障碍）
```

```
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> map[i][j];
        }
    }

    // 初始化起点(1,1)为已访问状态
    vis[1][1] = true;

    // 开始DFS搜索
    dfs(1, 1);

    // 输出从起点到终点的总路径数
    cout << ans << endl;

    return 0;
}
```

BFS 标准代码模板

模板属于：二维网格 BFS

- BFS 的考题主要分为两类：

1. 走迷宫/地图类：比如“老鼠走迷宫”、“逃离火灾”、“最少移动步数”。
2. 状态变化类：比如“倒水问题”

- 代码实现：

```
#include <iostream>
#include <queue>  // 使用STL队列实现BFS
using namespace std;

// ================= 数据结构定义 =================
/**
 * 节点结构体：表示网格中的一个位置
 * @param x     行坐标（从1开始）
 * @param y     列坐标（从1开始）
 * @param step 从起点到当前节点的步数
 */
struct Node {
    int x, y;
    int step;
};

// ================= 全局变量定义 =================
int n, m;               // 网格的行数n，列数m
int map[50][50];        // 网格地图（0可通行，1障碍物）
bool vis[50][50];       // 访问标记数组，记录是否已访问

// 方向数组：右(0,1)、下(1,0)、左(0,-1)、上(-1,0)
int dx[4] = {0, 1, 0, -1};
int dy[4] = {1, 0, -1, 0};

// ================= BFS广度优先搜索 =================
/**
 * 使用BFS搜索从起点到终点的最短路径
 *
 * @param start_x 起点行坐标
 * @param start_y 起点列坐标
 * @param end_x   终点行坐标
 * @param end_y   终点列坐标
 * @return 最短路径步数（无法到达时返回-1）
```

```
 */
int bfs(int start_x, int start_y, int end_x, int end_y) {
    // 1. 初始化队列
    queue<Node> q;

    // 2. 创建起点节点并入队
    Node start = {start_x, start_y, 0};
    q.push(start);
    vis[start_x][start_y] = true;  // 标记起点已访问

    // 3. 队列非空时循环搜索
    while (!q.empty()) {
        // 3.1 取出队首节点（当前探索的节点）
        Node now = q.front();
        q.pop();

        // 3.2 终点判断：到达终点时返回步数
        if (now.x == end_x && now.y == end_y) {
            return now.step;
        }

        // 3.3 遍历四个方向
        for (int i = 0; i < 4; i++) {
            int nx = now.x + dx[i];  // 计算下一位置行坐标
            int ny = now.y + dy[i];  // 计算下一位置列坐标

            // 3.4 可行性检查：
            //    a) 不越界：nx∈[1,n]，ny∈[1,m]
            //    b) 可通行：map[nx][ny] == 0
            //    c) 未访问：!vis[nx][ny]
            if (nx >= 1 && nx <= n && ny >= 1 && ny <= m
                && map[nx][ny] == 0 && !vis[nx][ny]) {

                vis[nx][ny] = true;  // 标记为已访问（防止重复入队）

                // 创建下一节点并入队（步数+1）
                Node next = {nx, ny, now.step + 1};
                q.push(next);
            }
        }
    }

    // 4. 队列空仍未找到终点，说明不可达
    return -1;
}

// ================= 主程序入口 =================
int main() {
    // 1. 输入网格规格
    cin >> n >> m;

    // 2. 输入网格数据（这里假设map已在全局初始化）
    // 实际使用时需要先读入map数据，例如：
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> map[i][j];
        }
    }

    // 3. 调用BFS计算从(1,1)到(n,m)的最短步数
    cout << bfs(1, 1, n, m) << endl;

    return 0;
}
```

- 代码疑点：

1. map 和 vis 里面的数字怎么定?

原则：看题目给的“数据范围”，然后稍微开大一点。

- 题目会说：$1 \leq N, M \leq 100$。

定义 `map[105][105]`。

- 题目会说：$1 \leq N, M \leq 1000$。

定义 `map[1010][1010]`。

> 数组大小 = 题目最大约束 + 5

2. 方向数组 dx dy 怎么定?

这是基于矩阵坐标系（行号是 x，列号是 y）：

- x (行)：向下是增加 (+)，向上是减少 (-)。
- y (列)：向右是增加 (+)，向左是减少 (-)。

> 示例：

- 题目：5. 走迷宫
- 题目描述

给定一个 $N \times M$ 的网格迷宫 G。G 的每个格子要么是道路，要么是障碍物（道路用 1 表示，障碍物用 0 表示）。

已知迷宫的入口位置为 (x_1, y_1)，出口位置为 (x_2, y_2)。问从入口走到出口，最少要走多少个格子。

- 输入描述

输入第 1 行包含两个正整数 N, M，分别表示迷宫的大小。

接下来输入一个 $N \times M$ 的矩阵。若 $G_{i,j} = 1$ 表示其为道路，否则表示其为障碍物。

最后一行输入四个整数 x_1, y_1, x_2, y_2，表示入口的位置和出口的位置。

- 数据范围：

$1 \leq N, M \leq 10^2$，$0 \leq G_{i,j} \leq 1$，$1 \leq x_1, x_2 \leq N$，$1 \leq y_1, y_2 \leq M$。

- 输出描述

输出仅一行，包含一个整数表示答案。

若无法从入口到出口，则输出 -1。

- 输入输出样例
  - 示例 1
  - 输入

```
5 5
1 0 1 1 0
1 1 0 1 1
0 1 0 1 1
1 1 1 1 1
1 0 0 0 1
1 1 5 5
```

- 输出

```
8
```

- 运行限制
  - 最大运行时间：1s
  - 最大运行内存：128M
- 解答：

```cpp
#include <iostream>
#include <queue>
using namespace std;

struct Node {
    int x, y;
    int step;
};

int n, m;
int g[110][110];      // 地图
bool vis[110][110];   // 标记数组

// 方向数组：右、下、左、上
int dx[4] = {0, 1, 0, -1};
int dy[4] = {1, 0, -1, 0};

int bfs(int x1, int y1, int x2, int y2) {
    queue<Node> q;

    // 起点入队，步数为0
    q.push({x1, y1, 0});
    vis[x1][y1] = true;

    while (!q.empty()) {
        Node cur = q.front();
        q.pop();

        // 到达终点，返回步数
        if (cur.x == x2 && cur.y == y2) {
            return cur.step;
        }

        // 遍历四个方向
        for (int i = 0; i < 4; ++i) {
            int nx = cur.x + dx[i];
            int ny = cur.y + dy[i];

            // 核心判断：
            // 1. 不越界
            // 2. 是路（题目说 1 是路！注意这里！）
            // 3. 没走过
            if (nx >= 1 && nx <= n && ny >= 1 && ny <= m && g[nx][ny] == 1 && !vis[nx][ny]) {
                vis[nx][ny] = true;
                q.push({nx, ny, cur.step + 1});
            }
        }
    }
    return -1; // 走不到
}

int main() {
    // 1. 输入 N 和 M
```

```
    cin >> n >> m;

    // 2. 输入地图矩阵
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= m; j++) {
            cin >> g[i][j];
        }
    }

    // 3. 输入起点和终点
    int x1, y1, x2, y2;
    cin >> x1 >> y1 >> x2 >> y2;

    // 4. 计算并输出
    cout << bfs(x1, y1, x2, y2) << endl;

    return 0;
}
```