第二十章 广度优先搜索

1. 广度优先搜索

广度优先搜索（BFS Breadth-First Search）
它的基本思路是：指从某一起始节点开始, 依据某种策略，逐层向外扩展新的节点（或状态），逐个检验新扩展出的节点（或状态），一直到找到答案或遍历完所有的节点（或状态） 为止。
因为它的思想是从一个节点开始，辐射状地同时遍历周围的区域，因此得名广度优先搜索。

- 应用场景：
  广度优先搜索在很多领域都有广泛的应用， 包括但不限于
  图的遍历：用于遍历图中的所有节点， 查找特定节点或找到从起始节点到目标节点的最短路径
  迷宫问题：用于找到从起点到终点的最短路径，或者找到是否有路径可以从起点到达终点
  社交网络分析：用于查找两个用户之间的最短路径或共同的联系人
  网络路由：用于寻找最优路由和解决网络拓扑问题

2.示例

BFS （广度优先搜索）从一个初始节点开始，向其周围未访问过的节点扩散。这种扩散类似于在图中创建一圈新节点，然后再对这些新节点进行相同的操作。这个过程会持续进行，直到满足以下结束条件：已遍历所有节点、 到达目标节点、或满足其他特定条件。更简洁来说，BFS像波纹一样从起点开始向四周扩散， 每次只考虑当前节点的邻居，直到我到目标或遍历完所有可能路径
一般BFS使用队列这种数据结构来保存待访问节点，保证了节点访问的顺序性一先进入队列的节点总是先被访问， 即先按层级（距离原点的距离）搜索，再按同一层级的入队顺序搜索

根据上述分析，可以创建一个结构体，结构题变量x，y分别表示节点的行号和列号，dis表示（1，1）点到达这个点的最小步数，再定一个队列q，存储扩展的节点

```
struct Node  //创建结构体定义行号列号及最小步数
{
    int x, y;  // x表示行号 y表示列号
    int dis;  // 保存到达当前点的移动步数
}
queue<Node> q;  //定义队列q
```

首先将起始点（1，1）入队，移动步数为0
vis[i][j] 用来标记（i，j）是否已经被访问过，避免重复访问

```
q.push((Node){1,1,0});   // 将（1，1）点添加到队列q中，并记录移动步数0
vis[1][1] = true;   //将（1，1）点状态改为true（已访问过）
```

开启循环， 如果队列不为空，取出队头节点，如果这个点就是终点，返回该节点记录的步数，即为最小步数。

```
while(!q.empty())//判断队列q是否为空，不为空进入循环
{
    Node t = q.front(); // 取出队头元素
    q.pop();    //出队
    if(t.x == n && t.y ==m)    //如果队头就是终点
        return t.dis;  //返回该节点记录的步数，即为最小步数
}
```

否则， 取出队头元素，以这个点为基准，向上、下、左、右4个方向扩展，如果扩展的点在迷范围内，且之前没有被访问过，并且这个点是，将其入队，并标记为已访问， 避免重复访问

```
while(!q.empty()){
    ......
    int tx, ty;
    for(int i = 0; i < 4; i++){                          // 循环四次，表示上下左右扩展的点
        tx = t.x + dx[i];                                // 计算相邻节点的行下标
        ty = t.y + dy[i];                                // 计算相邻节点的列下标
        if(tx >= 1 && tx <= n && ty >= 1 && ty <= m){    // 如果访问的节点在迷宫范围内
            if(!vis[tx][ty] && a[tx][ty] == 0){          // 并且当前节点未被访问，当前节点值为 0
                vis[tx][ty] = true;                      // 将该节点状态改为已访问
                q.push((Node){tx, ty, t.dis + 1});       // 将新节点入队，将最小移动步数加 1
            }
        }
    }
}
```

如果循环条件不成立（队列为空），则说明的所有能扩展的点已经扩展过了，无法到达终点

```
int bfs()
{
    while(!q.empty())  //判断队列q是否为空，不为空进入循环
    {
        .....
    }
    return -1;   //如果队列为空返回-1
}
```

- 参考代码

```cpp
#include<iostream>
#include<queue>
using namespace std;
const int N = 1005;
int dx[4] = {0,1,0,-1};  //四个方向的行偏移量
int dy[4] = {1,0,-1,0};  // 四个方向的列偏移量
struct Node
{
    int x,y;     // x表示行号，y表示行号
    int dis;     // 移动步数
}
int n, m, a[N][N];   //a[N][N]存储迷宫
bool vis[N][N];   // vis[i][j] (i,j) 的访问状态，初始化为未访问
queue<Node> q;   //队列q 存储节点
int bfs()
{
    q.push((Node){1, 1, 0});  // 将 (1, 1) 点添加到队列q中， 并记录移动步数 0
    vis[1][1] = true;  // 将 (1, 1) 点设置为已访问
    while(!q.empty())    //如果q不为空
    {
        Node t = q.front();    //取出队头
        q.pop();               //出队
        if(t.x == n && t.y ==m) return t.dis;   //如果到达终点，则返回该接待你记录的步数
        int tx, ty;
        for(int i = 0; i < 4;++i)  //搜索上下左右四个点
        {
            tx = t.x + dx[i];   //计算相邻节点的行下标
            ty = t.y + dy[i];   //计算相邻节点的列下标
            if(tx >= 1&& tx<=4 && ty >=1 && ty<= m)  //如果访问的节点在迷宫的范围内
            {
                if(!vis[tx][ty]&& a[tx][ty] == 0)   //如果当前节点未被访问且当前节点值为
0
                {
                    vis[tx][ty] = true;  // 将该节点状态标记为已访问
                    q.push((Node){tx, ty, t.dis+1}); // 将新节点添加到队列，然后将移动步
数+1
                }
            }
        }
    }
    return -1;  //如果队列为空返回-1
}
int main()
{
    cin >> n >> m;   //输入迷宫矩阵的行数和列数
    for(int i = 0;i<=n;++i)  // 输入迷宫矩阵， 0表示可走，1表示不可走
    {
        for(int j = 1; j <= m;j++)
            cin >> a[i][j];
    }
    cout << bfs();
    return 0;
}
```

示例总结：
首先将初始节点加入队列， 然后开始循环， 直至队列为空；
每次循环从队列取出队头节点并对其处理，然后再将队头节点的相邻节点加入队列，之后出队

在处理节点过程中，还需要标记已被访问过的位置，避免重复访问；
直至队空、 或者到达目标节点、 或者满足结束条件时才停止上述过程