搜索

TEACHING COURSEWARE POWPOINT

授课时间：2025.08.09

初赛

链表不具有的特点是（）。

- A. 可随机访问任一元素
- B. 不必事先估计存储空间
- C. 插入删除不需要移动元素
- D. 所需空间与线性表长度成正比

1字节等于多少位

面向对象的语言有哪些（三个）

面向过程的语言是什么（一个）

初赛

在 C++ 中，下面哪个关键字用于声明一个变量， 其值不能被修改?

- A. `unsigned`
- B. `const`
- C. `static`
- D. `mutable`

以下哪个不是操作系统?()

- A. Linux
- B. Windows
- C. Android
- D. HTML

初赛

操作系统有：Windows、macOS、Linux、Android、iOS、。
可能不是操作系统的有：应用软件（如 Word、浏览器）、编程语言（如 C、Python）、硬件（如 CPU、内存）等

目录

01

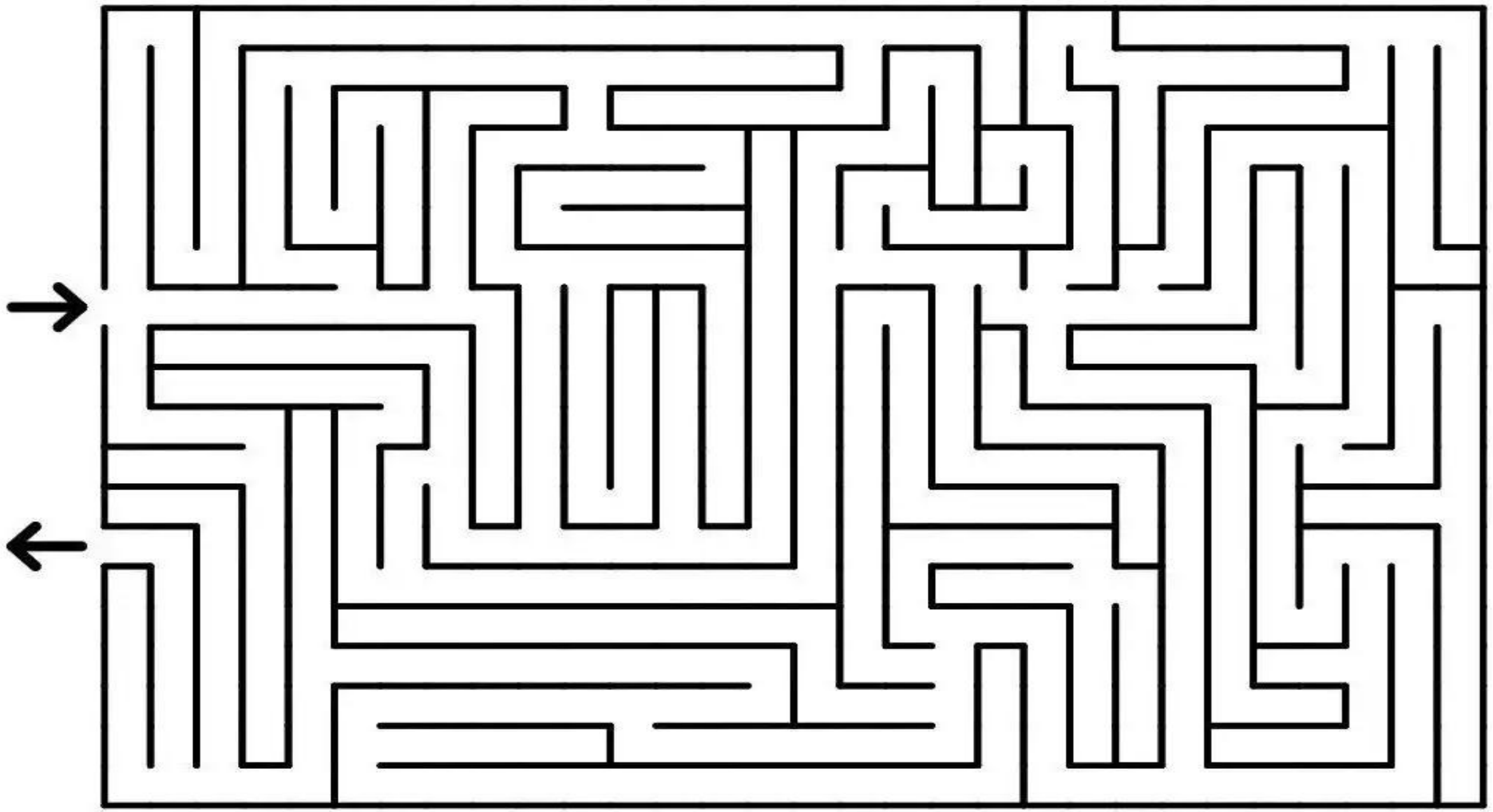
概念引入

TEACHING
COURSEWARE

TEACH

概念引入

迷宫



概念引入

搜索就是 “在一堆东西 / 一个空间里，按照一定规则找目标”，而计算机的 “搜索算法” 就是给计算机制定的 “找东西的规则”。

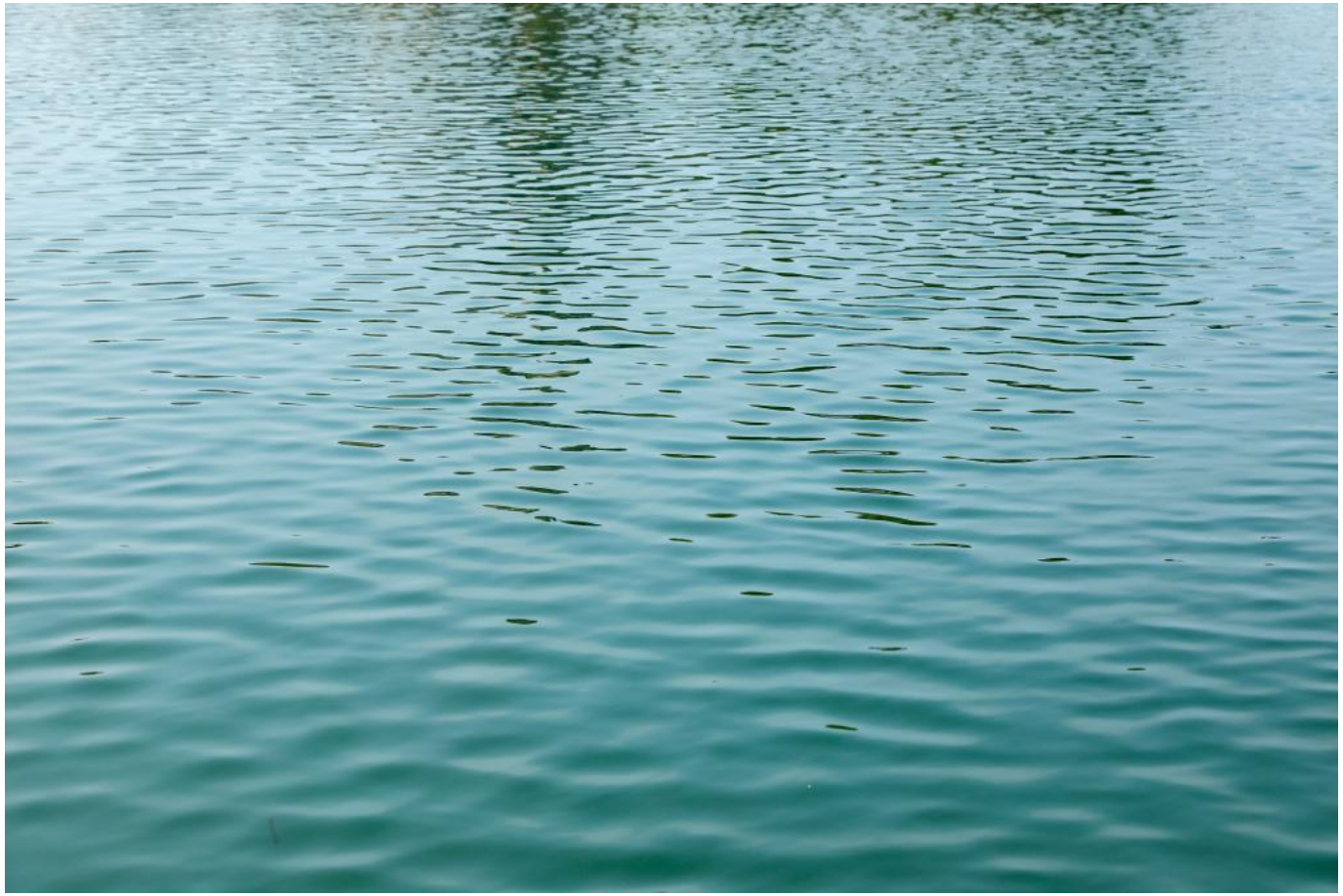
02

广度优先搜索

TEACHING COURSEWARE

TEACH

广度优先搜索



广度优先搜索

S . .
. 墙 .
. . E

广度优先搜索

核心：“先宽后深”，优先访问距离起点近的节点。

优势：能找到最短路径

关键：用 “队列”（Queue）存储待处理的节点（先进先出，保证逐层处理）。

需要记录 “已访问” 的节点（避免重复搜索，比如绕圈走）。

BFS：全称 Breadth-First Search，“广度” 指的是“宽度”，即先扩展所有近距离的节点，再扩展远距离的。

03

例题

TEACHING
COURSEWARE

TEACH

例题

题目描述

呵呵，有一天我做了一个梦，梦见了一种很奇怪的电梯。大楼的每一层楼都可以停电梯，而且第 i 层楼（$1 \leq i \leq N$）上有一个数字 K_i（$0 \leq K_i \leq N$）。电梯只有四个按钮：开，关，上，下。上下的层数等于当前楼层上的那个数字。当然，如果不能满足要求，相应的按钮就会失灵。例如：$3, 3, 1, 2, 5$ 代表了 K_i（$K_1 = 3$，$K_2 = 3$，……），从 1 楼开始。在 1 楼，按“上”可以到 4 楼，按“下”是不起作用的，因为没有 -2 楼。那么，从 A 楼到 B 楼至少要按几次按钮呢?

例题

输入格式

共二行。

第一行为三个用空格隔开的正整数，表示 N, A, B（$1 \le N \le 200$，$1 \le A, B \le N$）。

第二行为 N 个用空格隔开的非负整数，表示 K_i。

输出格式

一行，即最少按键次数，若无法到达，则输出 `-1`。

输入输出样例

输入 #1 复制

```
5 1 5
3 3 1 2 5
```

输出 #1 复制

```
3
```

说明/提示

对于 100% 的数据，$1 \le N \le 200$，$1 \le A, B \le N$，$0 \le K_i \le N$。

本题共 16 个测试点，前 15 个每个测试点 6 分，最后一个测试点 10 分。

例题

题目设定为：在一栋 n 层的大楼中，从 a 层出发，每层楼 i 都有固定跳跃距离 k [i]，每次可选择向上或向下跳 k [i] 层（不能超出大楼范围），需找到到达 b 层的最少步数，无法到达则返回 - 1。
用队列存储待访问的楼层节点（含当前楼层 x 和步数 y），用 vis [] 数组标记已访问楼层避免重复访问。
BFS 过程为：先将起始楼层 a 入队（步数 0）并标记访问；循环处理队列，取出队首节点后，若为目标楼层 b 则返回步数；否则计算向上、向下跳跃后的新楼层，对有效（在 1~n 范围且未访问）的新楼层入队并标记；若队列为空仍未找到目标，返回 - 1。、

例题


```
#include<bits/stdc++.h>
using namespace std;
int n,a,b,k[201];
bool vis[201];//标记数组，记录是否访问过
struct node{
    int x,y; //x表示当前楼层，y表示步数
};
queue<node> q;
int bfs(){
    q.push({a,0});//起始位置入队
    vis[a] = true;
    while(!q.empty()){
        int x = q.front().x;
        int y = q.front().y;
        q.pop();
        if(x == b) return y; //到达目标楼层，返回步数
```

例题

```
        //向上移动
        int xn = x + k[x];
        int yn = y + 1;
        if(xn <= n && xn > 0 && !vis[xn]){
            q.push({xn, yn});
            vis[xn] = true;
        }
        //向下移动
        xn = x - k[x];
        if(xn <= n && xn > 0 && !vis[xn]){
            q.push({xn, yn});
            vis[xn] = true;
        }
    }
    return -1; //无法到达
}
```

例题


```
int main(){
    cin >> n >> a >> b;
    for(int i = 1; i <= n; i++){
        cin >> k[i];
    }
    cout << bfs() << endl;
    return 0;
}
```

04

习题

TEACHING
COURSEWARE

TEACH

例题

题目描述

复制 Markdown 展开 进入 IDE 模式

小 H 在一个划分成了 $n \times m$ 个方格的长方形封锁线上。 每次他能向上下左右四个方向移动一格（当然小 H 不可以静止不动）， 但不能离开封锁线，否则就被打死了。 刚开始时他有满血 6 点，每移动一格他要消耗 1 点血量。一旦小 H 的血量降到 0， 他将死去。 他可以沿路通过拾取鼠标（什么鬼。。。）来补满血量。只要他走到有鼠标的格子，他不需要任何时间即可拾取。格子上的鼠标可以瞬间补满，所以每次经过这个格子都有鼠标。就算到了某个有鼠标的格子才死去， 他也不能通过拾取鼠标补满 HP。 即使在家门口死去， 他也不能算完成任务回到家中。

地图上有五种格子：

`0`： 障碍物。

`1`： 空地， 小 H 可以自由行走。

`2`： 小 H 出发点， 也是一片空地。

`3`： 小 H 的家。

`4`： 有鼠标在上面的空地。

小 H 能否安全回家？如果能， 最短需要多长时间呢？

例题

输入格式

第一行两个整数 n, m， 表示地图的大小为 $n \times m$。

下面 n 行， 每行 m 个数字来描述地图。

输出格式

一行， 若小 H 不能回家， 输出 `-1`，否则输出他回家所需最短时间。

输入输出样例

输入 #1	复制	输出 #1	复制
3 3 2 1 1 1 1 0 1 1 3		4	

说明/提示

对于所有数据，$1 \leq n, m \leq 9$。

2021.9.2 增添一组 hack 数据 by @囧仙

例题

解题时，用二维数组g存储地图信息，三维数组vis[x][y][hp]标记不同位置（x,y）和血量（hp）的访问状态，避免重复搜索。BFS 队列存储每个状态的坐标、血量和步数，从起点开始尝试四个方向移动：计算新位置和剩余血量，若新位置是家且血量有效，返回当前步数 + 1；若为鼠标格子，补满血量至 6；将未访问的有效状态加入队列继续搜索。若队列为空仍未到家，返回 - 1。该方法利用 BFS 特性保证最短路径，结合血量管理满足题目限制，高效求解

例题


```cpp
#include <bits/stdc++.h>
using namespace std;
// 方向数组：上下左右四个方向
int dx[] = {-1, 1, 0, 0};
int dy[] = {0, 0, -1, 1};
// 定义状态结构体：位置(x,y)、剩余血量(hp)、已走步数(steps)
struct Node {
    int x, y, hp, steps;
};
int main() {
    int n, m;
    cin >> n >> m;
    int map[10][10];  // 存储地图信息
    int start_x, start_y;  // 起点坐标
    // 读取地图并找到起点
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> map[i][j];
            if (map[i][j] == 2) {  // 记录起点位置
                start_x = i;
                start_y = j;
            }
        }
    }
    // 三维访问标记：visited[x][y][hp]表示在(x,y)位置且血量为hp时是否访问过
    bool visited[10][10][7] = {false};
    // BFS队列初始化
    queue<Node> q;
    q.push({start_x, start_y, 6, 0});  // 起点状态：初始血量6，步数0
    visited[start_x][start_y][6] = true;
```

例题

```
// 执行BFS
while (!q.empty()) {
    Node curr = q.front();
    q.pop();
    // 尝试四个方向移动
    for (int i = 0; i < 4; i++) {
        int nx = curr.x + dx[i];  // 新x坐标
        int ny = curr.y + dy[i];  // 新y坐标
        // 检查是否超出地图范围
        if (nx < 0 || nx >= n || ny < 0 || ny >= m) {
            continue;
        }
        // 检查是否为障碍物
        if (map[nx][ny] == 0) {
            continue;
        }
        // 计算移动后的血量（消耗1点）
        int new_hp = curr.hp - 1;
        if (new_hp <= 0) {  // 血量耗尽，无法继续
            continue;
        }
        // 到达家且血量有效，返回步数
        if (map[nx][ny] == 3) {
            cout << curr.steps + 1 << endl;
            return 0;
        }
        // 踩到鼠标格子，补满血量
        if (map[nx][ny] == 4) {
            new_hp = 6;
        }
        // 未访问过的状态，加入队列
        if (!visited[nx][ny][new_hp]) {
            visited[nx][ny][new_hp] = true;
            q.push({nx, ny, new_hp, curr.steps + 1});
        }
    }
}
```

例题


```
    }
    // 无法到达家
    cout << -1 << endl;
    return 0;
}
```