数据结构 Project：智能仓储机器人路径规划实验报告

徐晨欣 24300130064

2025 年 12 月 28 日

1 项目背景

- **项目名称：** 智能仓储机器人 (Smart Warehouse AGV)
- **背景描述：** 在现代化的电商物流中心，成千上万的 AGV (自动导引车) 在货架间穿梭。它们需要从充电站 (S) 出发，前往指定的货架 (T) 取货，最后将货物运送到打包台 (E)。本项目的任务是设计并实现一个程序，为机器人规划一条满足作业顺序且不发生碰撞的最短路径。

2 实验环境

Windows 操作系统，VSCode C++ 开发环境

3 实现思路

1. 整体设计思想
   使用面向对象的思想，将智能仓储机器人的路径规划问题封装为一个完成的类，通过模块划分，实现“地图读取-路径搜索-结果判断-输出与可视化”。
   (a) 基础数据结构定义
   描述地图中的节点与搜索状态
   (b) 核心类　SmartWarehouseAGV
   封装地图、搜索算法与求解流程

(c) 路径搜索算法模块（BFS）
实现最短路径计算

(d) 主函数模块

2. 基础数据结构定义

(a) 地图节点结构 Node

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

用于表示地图中的一个坐标
在路径回溯时判断是否回到起点

(b) BFS 搜索状态：BFSsearch

```
struct BFSsearch{
    int x, y;
    int step;
};
```

表示 BFS 队列中的一个搜索状态
step 用于记录从起点到当前位置的步数

3. 核心类　SmartWarehouseAGV 的设计
SmartWarehouseAGV 类负责管理路径规划的全部逻辑，其中的变量包括：

- map：地图数据，用于存储仓库布局
- start：起点；target：货物点；end：终点
- foundST, foundTE：分段搜索状态标识
- M、N：地图尺寸；K：电量限制
- visited：访问标记数组，用于避免重复搜索
- parent：前驱节点数组，用于路径回溯
- pathST, pathTE：分段路径存储结构

```
private:
    vector<string> map;//存储地图
    Node start, end, target;
    bool foundST, foundTE;
    int M, N, K;
    vector<vector<bool>> visited; //路径访问
    vector<vector<Node>> parent; //路径回溯
    vector<Node> pathST; //S -> T
    vector<Node> pathTE; //T -> E
```

4. 构造函数定义
   保证创建 SmartWarehouseAGV 对象后，处于初始化状态，避免未初始化变量导致的逻辑错误

```
public:
    SmartWarehouseAGV() : foundST(false), foundTE(false) {}
```

5. 地图读取模块 readMap()
   readMap() 函数负责从输入文本中读取地图规模，电量上限以及具体地图内容，并确定起点 S，货物点 T，终点 E 的坐标。

6. BFS 最短路径搜索函数

```
vector<Node> BFS(Node startPoint, Node endPoint)
```

   用于在地图中搜索从起点到终点的最短路径，包括：

   - 初始化 visited 和 parent 数组
   - 将起点加入队列，广度优先搜索
   - 按照上下左右四个方向搜索节点
   - 排除越界、障碍物以及以搜索过的节点
   - 当到达终点时，通过 parent 数组回溯得到完整路径

7. 分段路径规划
   在实现路径中，并不是直接从起点到达终点，而是从起点 S 先到达货物点 T，再从 T 到达终点 E，当两个阶段都为可行路径时，任务才能完成。

(a) 第一次 BFS（S->T）

- 以 S 为起点，T 为终点
- 搜索从 S 到 T 的最短路径
- 若搜索失败，则说明无法到达货物点，判断为“无解”

```
//S -> T
vector<Node> pathST = BFS(start, target);
if(pathST.empty()){
    cout << "无解" << endl;
    return;
}
```

(b) 第二次 BFS（T->E）

- 以 T 为起点，E 为终点
- 搜索从 T 到 E 的最短路径
- 若搜索失败，则说明无法从货物点到达终点，判断为“无解”

```
//T -> E
vector<Node> pathTE = BFS(target, end);
if(pathTE.empty()){
    cout << "无解" << endl;
    return;
}
```

每一次进行 BFS 搜索时都是在原始地图上独立进行，每次搜索前需要重新初始化 visited 和 parent 数组，确保不会互相干扰。

(c) 路径长度与电量约束

在分别获得两段最短路径后，第一段步数为：pathST-1，第二段步数为：pathTE-1，总步数为两段步数之和；

将总步数与最大电量 K 比较，小于等于 K 则可以完成，大于 K 则判断为“电量不足”，无法完成。

8. 输出与可视化

在程序求解成功后，以坐标序列形式输出完整路径，在原地图上对路径进行标记，实现可视化输出。

4 测试结果

- 测试环境:

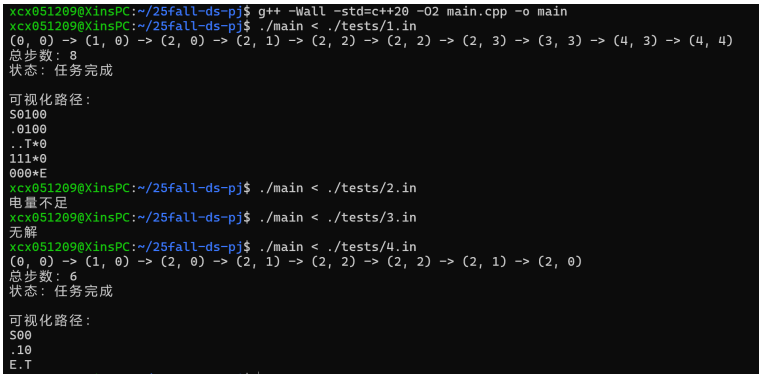
  Linux环境 6.6.87.1-microsoft-standard-WSL2 Ubuntu 24.04

- 编译命令:

  g++ -Wall -std=c++20 -O2 main.cpp -o main

- 测试指令 e.g.:

  ./main < ./tests/1.in

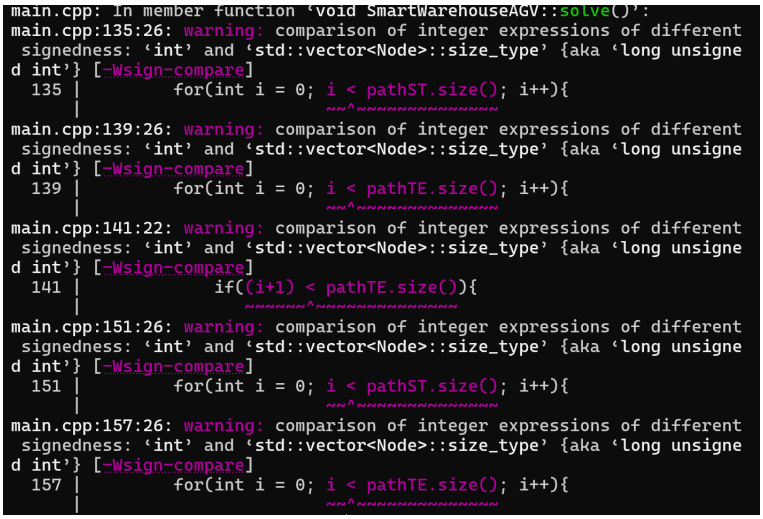


程序读取“1.in”“2.in”“3.in”“4.in”中的地图数据，输出的内容分别与提供的“1.out”“2.out”“3.out”“4.out”一致，程序能正常运行，测试通过。

5 问题与收获

1. 出现的问题

   - 在第一次编译 main.cpp 时，出现了以下报错内容：



     说明在定义变量类型时，最初将循环变量 i 定义为 int 型，与容器长度的变量类型不匹配，对于路径长度而言，是由 size() 获得的，返回类型为 size_t，将 i 定义为 size_t 类型后可以通过编译。

   - 路径保存问题
     在 BFS 搜索中，如果只记录步数而不保存路径来源，则无法输出完整的行驶路径，因此引入了 parent 数组，记录每个节点的前驱节点，并在到达终点后回溯完整路径。

2. 实验收获
   通过本次实验，我进一步理解了广度优先搜索在无权图最短路径中的使用，学会了如何通过路径回溯将搜索结果转化为可输出的具体路径；同时，我也体会到了面向对象的程序设计中，将问题划分成不同阶段，可以使程序逻辑更加直观。