# A\*算法研究与实践（C++实现）

潘韬睿 22180304

## 1 A\*算法基本原理解析

### 1.1 A\*算法介绍

#### 1.1.1 算法原理

A\* 算法是一种常用的启发式路径搜索算法，它结合了广度优先搜索的完整性和深度优先搜索的效率，通过在搜索过程中引入启发函数来提高效率。A\* 使用估价函数 f(n)=g(n)+h(n)，其中：

* g(n)表示从起点到节点n的实际代价，被称作代价函数；
* h(n)表示从节点n到终点的启发式估计代价，被成为启发函数。

A\* 算法的目标是找到从起点到终点的最短路径，它在每次扩展节点时选择具有最小 f(n)值的节点来探索，以保证路径是最优的。

#### 1.1.2 算法实现

A\* 算法使用优先队列来保持待探索的节点。其主要步骤如下：

1. **初始化**：将起点添加到开放列表（优先队列），并设置初始估价 f(n)。
2. **节点扩展**：从开放列表中选取估价最小的节点进行扩展，并将其邻居加入开放列表openlist。如果某个邻居节点的 g(n)值小于当前值，则更新其 g(n)值及父节点。
3. **目标判断**：当当前节点为终点时，停止搜索并回溯路径。
4. **重复搜索**：不断从开放列表中选取代价最小的节点，直到找到终点或无路径可达。

该算法的流程图如下：

退出，未找到路线

是

是

退出，输出路线

否

把附有f(S0)的初始节点S0放入OPEN表

移出OPEN表中第一个节点N放入CLOSED表中, 并冠以顺序编号n

OPEN表为空

若目标节点Sg=N, 则搜索成功, 结束

将节点N的邻近节点放入OPEN表

### 1.2 启发函数和代价函数的作用

#### 1.2.1 启发函数h(n)

启发函数h(n)是从当前节点n到目标节点的预估代价。常见的启发函数包括：

* **曼哈顿距离**：用于网格中只允许水平和垂直移动的情况。
* **欧氏距离**：用于网格中允许对角线移动且实际代价为路程的情况。
* **切比雪夫距离**：用于网格中允许八个方向移动且实际代价为步数的情况。

启发函数决定了算法的效率和性能。如果启发函数低估了实际代价（即是“乐观的”），则 A\* 算法的搜索路径接近最优，扩展节点较少。如果启发函数过于保守或不精确，搜索会变得接近于无启发式的广度优先搜索，扩展更多的节点。

可以说，启发函数的设计就是A\*算法的灵魂，只有启发函数的设计接近实际的需求，A\*算法才可以给出最优的路线。1.2.3会对该算法的最优性进行深入讨论。

#### 1.2.2 代价函数 g(n)

代价函数g(n)是从起点到节点n的实际代价，即已经走过的路径长度。它对路径的准确性有决定性影响，较小的g(n)值意味着当前路径的代价较低，因此节点会优先被扩展的该概率更大（还需要看启发函数的设计）。

在 A\* 算法中，启发函数和代价函数共同作用，既保证了路径的准确性，又提高了搜索效率。

#### 1.2.3 A\*算法的最优性条件

在课上第一次了解A\*算法的时候，听说这是一种启发式的搜索策略，理所当然地认为该算法就像遗传算法和模拟退火算法无法保证一定找到全局最优解。但是随着学习实验的深入，我总结了A\*算法在最优性方面的相关特性。

在实际运用A\*的时候，通常需要将二维地图进行网格化，并且规定路线需要经过网格格点（或者网格中心），在这种限制下必然会出现精度的损失，导致规划出来的路线必然不是实际问题的最优解，网格越精细，精度越高。

而到A\*算法负责的路径规划阶段，A\*算法是能够找到全局最优解，但有一个前提条件：启发式函数必须是可接受的（admissible）且一致的（consistent）。

可接受性意味着启发式函数从任意节点到目标节点的估计值不能超过实际最优路径的代价。这确保了A\*不会被错误的低估误导，从而确保找到最优解。

一致性（或称单调性）是指启发式函数满足三角不等式，即从一个节点直接到达目标的估计代价不应大于通过另一个中间节点再到达目标的代价总和。即对于任何两个相邻节点 n和 m，启发函数应该满足：

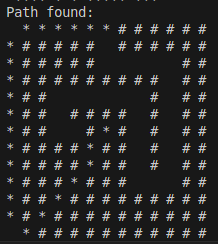
c(n,m)+h(m)

其中 c(n,m)是从n到m的实际代价（允许斜着走，可以是对角线距离）。如果启发式函数满足这两个条件，A\*算法可以保证找到从起点到目标的全局最优解。但如果启发式函数不满足这些条件，A\*算法可能会找到次优解或更长的路径。

## 2 A\*算法的C++实现与代码解析

### 2.1实验环境与背景介绍

完成在一个网格图上起点和终点之间的寻路操作，假设网格的边长为1,起点和终点在网格的中心点，可以在相邻节点的中心点之间移动，并且支持斜着走，可以从一个节点向8个方向移动，启发函数用欧式距离来实现。



### 2.2 节点类Node的实现

#### 2.2.1 节点坐标（x, y）

x 和 y 代表节点在网格中的坐标。A\* 算法在二维网格上搜索路径，因此每个节点都有对应的 (x, y) 坐标。

#### 2.2.2 gCost（从起点到该节点的实际代价）

gCost 是从起点到当前节点的实际代价。在 A\* 中，gCost 代表了从起点出发经过的路径长度（包括可能的障碍物绕行）。

#### 2.2.3 hCost（从该节点到终点的启发代价）

hCost 是启发函数估算的从当前节点到目标节点的代价。启发函数通常根据节点间的距离计算，例如欧氏距离、曼哈顿距离或切比雪夫距离。

#### 2.2.4 fCost（总代价）

fCost 是节点的总代价，即 gCost 与 hCost 之和，用于评估节点优先级。A\* 算法总是优先选择 fCost 最小的节点进行扩展，以保证找到最优路径。

#### 2.2.5 父节点（parent）

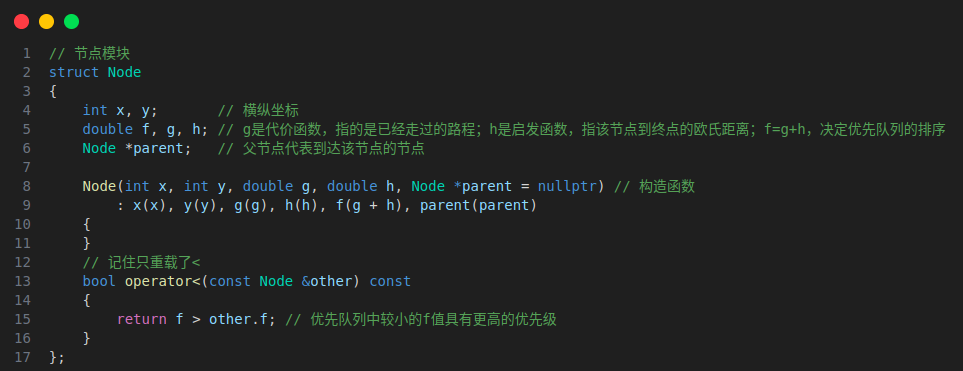
parent 是指向当前节点的父节点的指针，记录了路径的来源。在 A\* 完成搜索后，通过追溯父节点可以重建从起点到终点的路径。

#### 2.2.6 构造函数

构造函数初始化了节点的坐标和父节点，并默认将 gCost, hCost, 和 fCost 初始化为 0。

#### 2.2.7 calculateFCost()函数

calculateFCost() 用于计算节点的 fCost，即总代价 f(n)=g(n)+h(n)f(n) = g(n) + h(n)f(n)=g(n)+h(n)，这是 A\* 算法中决定扩展顺序的核心值。每次更新 gCost 和 hCost 后，都需要调用此函数来更新 fCost。



### 2.3 AStar函数的实现

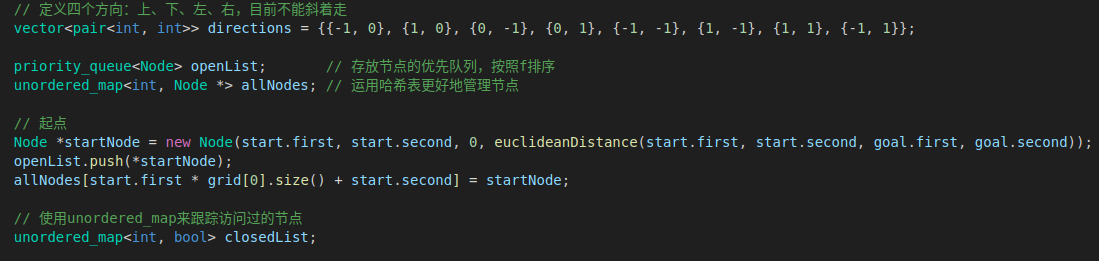
#### 2.3.1函数输入

地图储存在二维数组中，用vector实现，并且需要输入起点和终点的坐标。



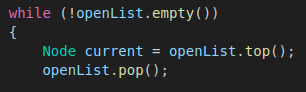
#### 2.3.2 函数初始化

可以向四周共八个方向移动，并且定义一个存储节点的优先队列，按照估价函数f来排序。

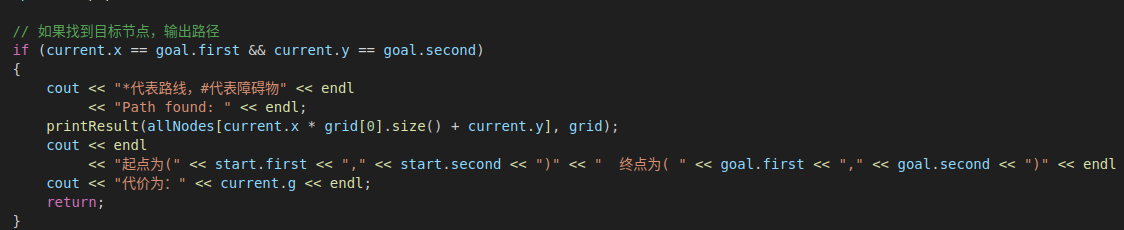


#### 2.3.3当OPEN表不为空时的关键循环

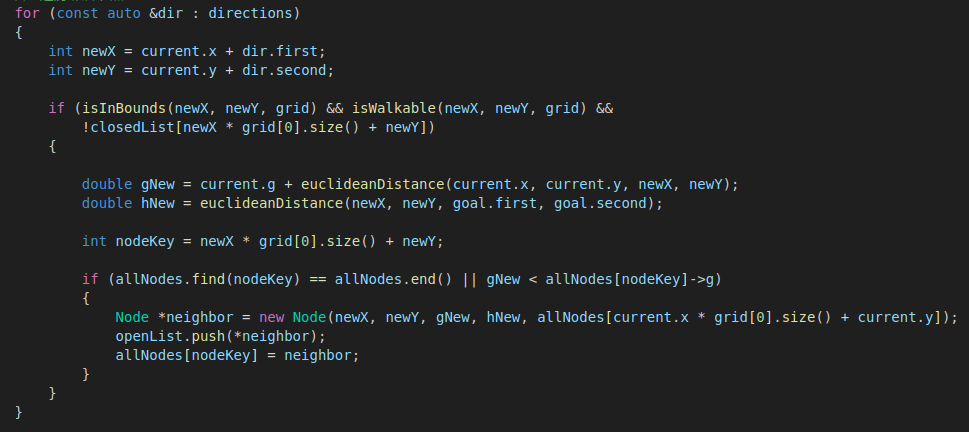
定义current为OPEN表中f值最小的节点。



找到的节点为目标节点则输出线路，退出循环。



如果不为目标节点，就将邻居节点遍历，符合条件的放入OPEN表。



## 3 实验结果分析

## 4 A\*算法和相近算法比较分析