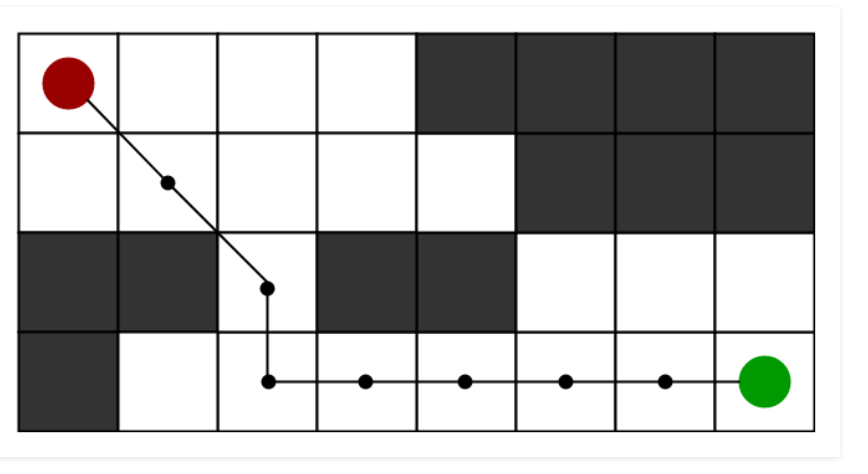
已知如下图地图，黑色表示障碍物无法通行，要求实现避障算法寻找从红色起点出发到达绿色终点的最优路径。



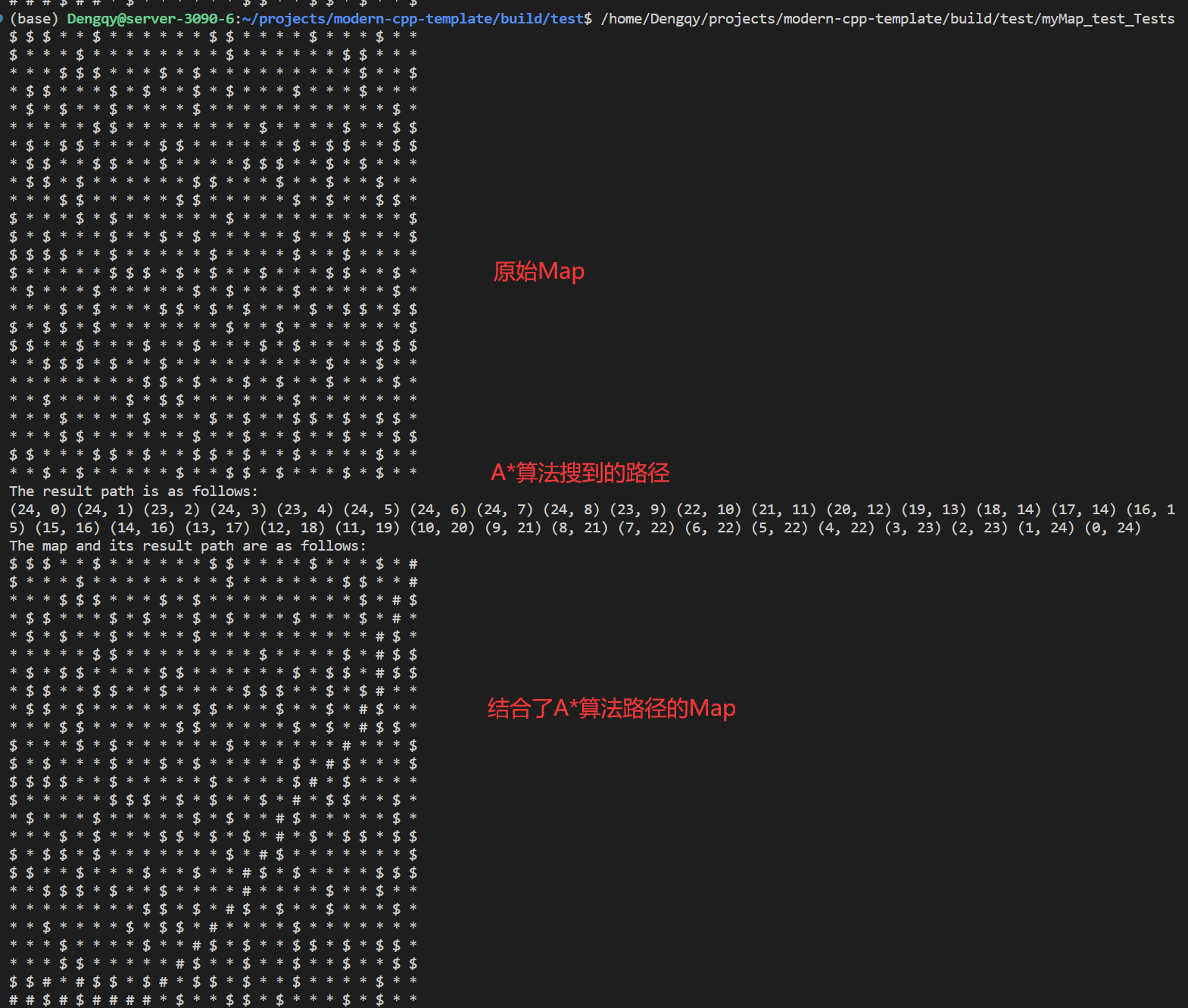
要求：

1. 对图中的地图进行建模，抽象成类，对数据进行封装；
2. 思考寻路算法的实现，对问题进行拆解，对算法实现也要求抽象接口类；
3. 使用给定的C++工程模板，按照模板编写CMakeLists.txt，以及Google Test单元测试，DoxyGen注释的使用。

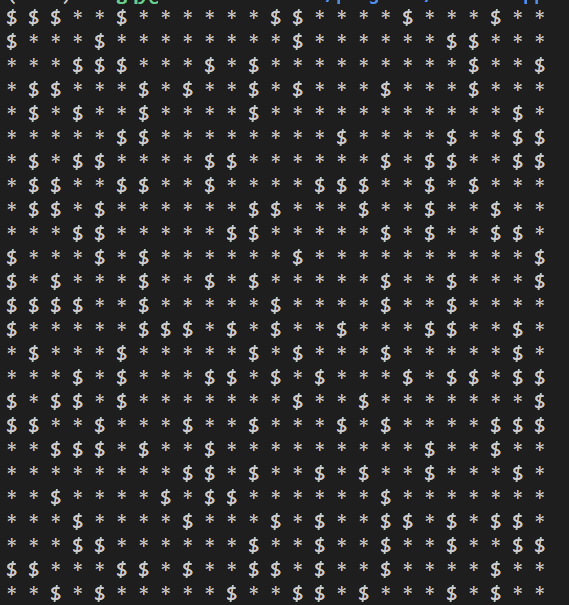
工程模板如下：

**<https://github.com/filipdutescu/modern-cpp-template.git>**

实验结果：



原始Map：

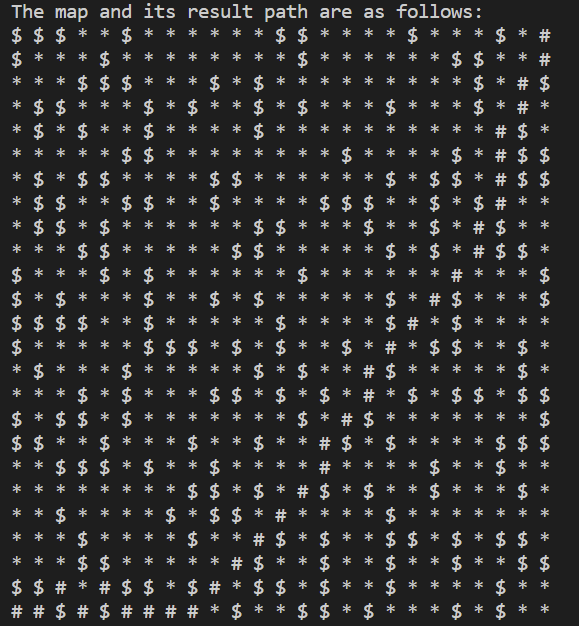


搜索到的路径为：

(24, 0) (24, 1) (23, 2) (24, 3) (23, 4) (24, 5) (24, 6) (24, 7) (24, 8) (23, 9) (22, 10) (21, 11) (20, 12) (19, 13) (18, 14) (17, 14) (16, 15) (15, 16) (14, 16) (13, 17) (12, 18) (11, 19) (10, 20) (9, 21) (8, 21) (7, 22) (6, 22) (5, 22) (4, 22) (3, 23) (2, 23) (1, 24) (0, 24)

结果Map：





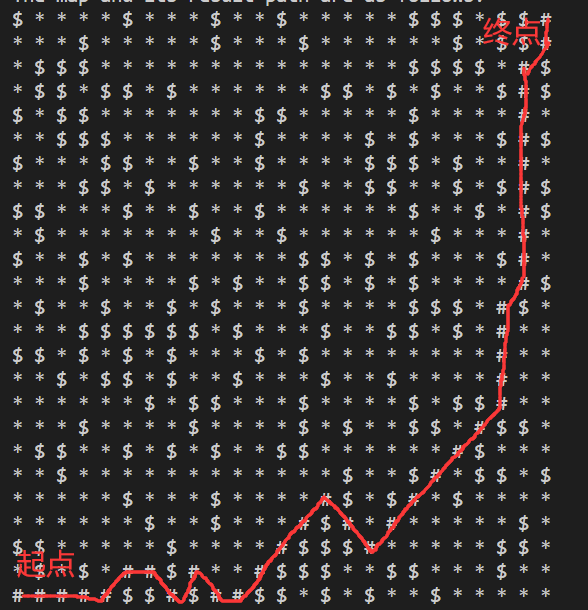
测试2：

原始图：



结果路径：(24, 0) (24, 1) (24, 2) (24, 3) (24, 4) (23, 5) (23, 6) (24, 7) (23, 8) (24, 9) (24, 10) (23, 11) (22, 12) (21, 13) (20, 14) (21, 15) (22, 16) (21, 17) (20, 18) (19, 19) (18, 20) (17, 21) (16, 22) (15, 22) (14, 22) (13, 22) (12, 22) (11, 23) (10, 23) (9, 23) (8, 23) (7, 23) (6, 23) (5, 23) (4, 23) (3, 23) (2, 23) (1, 24) (0, 24)

结果图：



1、Dijkstra算法与A\*算法

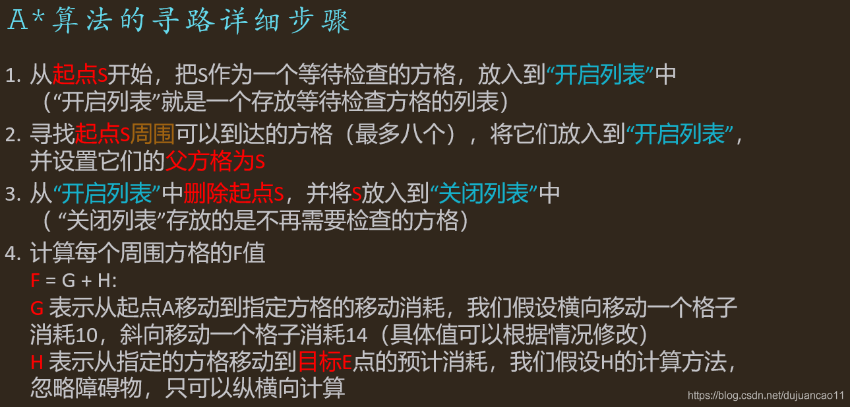
（1）Dijkstra算法（贪心策略 + 优先队列）：

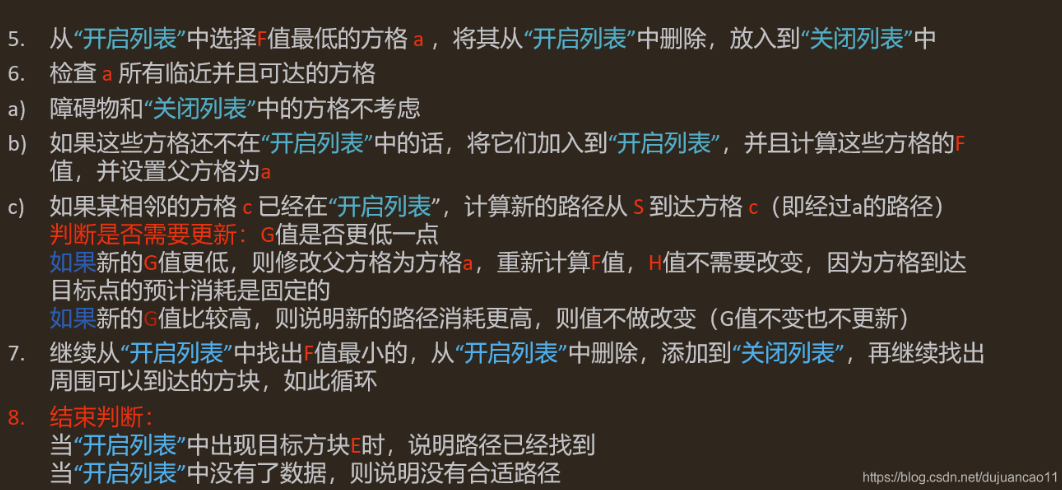
* 集合S：已确定的顶点集合，初始只含源点s。
* 集合T：尚未确定的顶点集合。
* 算法反复从集合T中选择当前到源点s最近的顶点u，将u加入集合S，然后对所有从u发出的边进行松弛操作。
* 算法步骤：维护一个优先队列，将集合T中的顶点到源点s的距离，作为这些点的优先级，距离越低，优先级越高。那么只要从优先队列中取出队首元素，即可获得当前离源点s最近的顶点。

附注：松弛操作：若d(u) + w(u, v) < d(v)，就更新d(v) = d(u) + w(u, v),其中w(u, v)表示边的权重，d(u)表示从顶点u到达源点s的最短距离（目前已知）

（2）A\*算法（启发式搜索）：

* F = G+ H，G表示从起点到某中间节点的移动代价，启发函数H（Heuristic）表示从某中间节点到终点的估算移动代价，当H始终为0，算法就退化为Dijkstra算法。
  + 曼哈顿距离
  + 欧式距离
* 算法步骤：





最后找到的路径顺着终点的父节点一路往前，直到起点，则为路径

