# 算法训练营2021版-期中考试题及参考答案

## 单项选择题

#### 第一题，数组相关

1. 将10阶对称矩阵（指10行10列的矩阵，且ai=aj）压缩存储到一维数组S中，则数组S的长度最少为（C）

A. 100

B. 40

C. 55

D. 80

难度：3星（总5星）

【参考解析】主对角线都存：10个，剩下的90个只存一半45个，共55个。

#### 第二题，链表相关

1. 下面哪项不是链表优于数组的特点?（D）

A. 方便删除

B. 方便插入

C. 长度可变

D. 存储空间小

难度：2星（总5星）

【参考解析】链表方便删除和插入，只需知道结点和要插入的信息即可；长度可变，一般链表是动态分配内存空间；链表的结点信息至少包含数据域和指针域，相同数据下：数组的大小是链表大小的子集，所以选D。

#### 第三题，链表、复杂度相关

1. 对于长度为n的数组，建立其对应的单链表的时间复杂度为（C）。

A. O(1)

B. O(log2n)

C. O(n)

D. O(n^2)

难度：2星（总5星）

【参考解析】本题主要考查的知识点是单链表的建立。无论采用什么方式建立单链表，都需要扫描这n个元素，边扫描边创建单链表中的结点并链接起来，其时间复杂度为O(n)。

#### 第四题，前缀和相关

1. 已知二维前缀和数组S，其中S[i,j]表示前i行前j列的元素之和，则以(x,y)为左上角、(i,j)为右下角的二维子矩阵的元素之和为（C）

A. S[i,j]-S[x,y]

B. S[i,j]-S[i,y]-S[x,j]+S[x,y]

C. S[i,j]-S[i,y-1]-S[x-1,j]+S[x-1,y-1]

D. S[i,j]-S[i-1,j-1]+S[x,y]-S[x-1,y-1]

难度：4星（总5星）

【参考解析】通过二维前缀和计算二维子矩阵和的基本公式（模板代码）。

#### 第五题，队列相关

1. 已知循环队列存储在一维数组A[0..n-1]中，且队列非空时 front 和 rear 分别指向队头和队尾元素。若初始时队列为空，且要求第 1 个进入队列的元素存储在 A[0]处，则初始时 front和 rear 的值分别是（B ）。

A. 0， 0

B. 0， n-1

C. n-1， 0

D. n-1， n-1

难度：4星（总5星）

【参考解析】插入时，队头指针不变，队尾指针后移一位。该题约定队列非空时 front 和 rear 分别指向队头和队尾元素，即插入第一个元素在下标为0的位置后，队头队尾指针皆指向A[0]，此时可以反推出插入前，队头指针仍旧指向下标0，而队尾指针应指向前一位，也就是下标n-1的位置。注意，这道题的约定与大多数题约定队列为空时front=rear=0是不一样的约定，都是根据题意解题。

#### 第六题，数组相关

1. 在一个长度为n的数组中删除第i个元素，要移动**\_\_**\_个元素。如果要在第i个元素前插入一个元素，要后移**\_\_**\_\_\_个元素。(A)

A. n-i，n-i+1

B. n-i+1，n-i

C. n-i，n-i

D. n-i+1，n-i+1

难度：2星（总5星）

【参考解析】：删除第i个元素，要移动后面n-i个元素。在第i个元素之前插入，要移动包括i在内的n-i+1个元素。（注：一般约定俗成的第几个元素，没有特别指明，我们是从1开始计数的。如果从0开始计数，长度为n的数组，那么它最后一个元素对应的索引是n-1，对应的删除和插入则分别为n-i-1和n-i。故答案为A）

#### 第七题，二叉树相关

1. 二叉树的先序遍历和中序遍历如下：先序遍历：EFHIGJK；中序遍历：HFIEJKG。则二叉树根结点为（A）

A. E

B. F

C. G

D. H

难度：1星（总5星）

【参考解析】先序遍历是先遍历二叉树的根结点，由先序遍历为：EFHIGJK可知根结点为：E。

#### 第八题，二叉树相关

1. 在一棵二叉树上第四层的结点数最多是（A）

A. 8

B. 9

C. 10

D. 11

难度：1星（总5星）

【参考解析】二叉树第1层最多1个结点，第二层最多2个结点，第三层最多4个结点，第n层最多2(n-1)个结点，因此第4层最多2(4-1)=8个结点。

#### 第九题，DFS相关

1. 下列关于树的深度优先搜索算法描述错误的是？(B)

A. 按照某种条件往前试探搜索，如果前进中遭到失败，则退回头另选通路继续搜索，直到找到条件的目标为止。

B. 先记录该节点所有的子节点，然后选取它未访问过的子节点重复上述过程，直到找到条件的目标为止。

C. 假设树的顶点数为V，则算法的空间复杂度为O(V)

D. 深度优先算法非常适合使用递归来实现

难度：3星（总5星）

【参考解析】B选项描述的是广度优先遍历（BFS）。

#### 第十题，贪心相关

1. 有关贪心法叙述正确的是（A）

A. 采用局部最优策略

B. 采用全局最优策略

C. 在贪心法中采用逐步构造最优解的方法

D. 把问题分解为简单的问题求解

难度：2星（总5星）

【参考解析】贪心算法，是一种在每一步选择中都采取在当前状态下最好或最优（即最有利）的选择，从而希望导致结果是最好或最优的算法。贪心算法在有最优子结构的问题中尤为有效。最优子结构的意思是局部最优解能决定全局最优解。简单地说，问题能够分解成子问题来解决，子问题的最优解能递推到最终问题的最优解。

#### 第十一、十二题，二分查找相关

1. 二分查找的时间复杂度（C）

A. O(N\*log(N))

B. O(N)

C. O(log(N))

D. O(N^2)

难度：3星（总5星）

【参考解析】二分法每次比较会去掉一半的数据，也就是说比较次数为n,数据为m个则2^n>=m,m=log(N),时间复杂度为O(log(N))

1. 对有序数组{2、11、15、19、30、32、61、72、88、90、96}进行普通的二分查找（移动left,right时双侧均不包含，一旦找到立刻停止），则成功找到15需比较（C）次

A. 3

B. 4

C. 2

D. 5

难度：3星（总5星）

【参考解析】

0 1 2 3 4 5 6 7 8 9 10

2、11、15、19、30、32、61、72、88、90、96

第一次 index=（0+10）/2=5 对应32 ； 比15大 所以下次范围是 0到4

第二次 index=（0+4）/2=2 对应15 找到

#### 第十三题，哈希相关

1. 以下哪个不属于单向哈希表的特征（假设哈希函数设计良好）（B）

A. 它把任意长度的信息转换成固定的长度输出

B. 它把固定的信息转换成任意长度信息输出

C. 根据特定的哈希值，它可以找到对应的原信息值

D. 不同的信息很难产生一样的哈希值

难度：3星（总5星）

【参考解析】哈希表（Hash Table）是一种根据关键字直接访问内存存储位置的数据结构。通过哈希表，数据元素的存放位置和数据元素的关键字之间建立起某种对应关系。

A，hash函数可以把字符串等任意长度的输入映射成固定长度的整数，也就是哈希值

B，与A说法相反，错误

C，哈希表建立了哈希值与原值信息存储之间的联系，可以通过哈希值查找到原值信息

D，不同的信息产生相同的哈希值叫哈希冲突。设计哈希函数应尽量避免哈希冲突。因此一般很难冲突。

#### 第十四题，分治相关

1. 下面哪个不是使用分治法的特征（ C ）

A. 该问题可以分解为若干个规模较小的相同问题

B. 子问题的解可以合并为该问题的解

C. 子问题必须是一样的

D. 子问题之间不包含公共的子问题

难度：3星（总5星）

【参考解析】分治法是建基于多项分支递归的一种很重要的算法范式。字面上的解释是“分而治之”，就是把一个复杂的问题分成两个或更多的相同或相似的子问题（同类子问题），直到最后子问题可以简单的直接求解，原问题的解即子问题的解的合并。

#### 第十五题，排序相关

1. 若希望保证对一个数组进行高效、稳定的排序（排序前后相同元素的相对顺序不变），则应该使用（D）排序算法。

A. 桶

B. 堆

C. 快速

D. 归并

难度：4星（总5星）

【参考解析】参考第9课PPT中的排序阵营九宫格。

## 编程题

1. [四数之和](https://leetcode-cn.com/problems/4sum/)（力扣第 18 题）

难度：中等

【参考答案】

无论是几数之和，重点是用排序+双指针解决二数之和，然后就可以通过枚举第三个数解决三数之和，再枚举第四个数解决四数之和，以此类推…… （迭代法或递归法）

建议研习下面的国际版力扣Python高票答案。

* C++ 解法：

class Solution{  
 public:   
 vector<vector<int>> fourSum(vector<int>& nums, int target) {  
 sort(nums.begin(),nums.end());  
 vector<vector<int> > res;  
 if(nums.size()<4)  
 return res;  
 int a,b,c,d,\_size=nums.size();  
 for(a=0;a<=\_size-4;a++){  
 if(a>0&&nums[a]==nums[a-1]) continue; //确保nums[a] 改变了  
 for(b=a+1;b<=\_size-3;b++){  
 if(b>a+1&&nums[b]==nums[b-1])continue; //确保nums[b] 改变了  
 c=b+1,d=\_size-1;  
 while(c<d){  
 if(nums[a]+nums[b]+nums[c]+nums[d]<target)  
 c++;  
 else if(nums[a]+nums[b]+nums[c]+nums[d]>target)  
 d--;  
 else{  
 res.push\_back({nums[a],nums[b],nums[c],nums[d]});  
 while(c<d&&nums[c+1]==nums[c]) //确保nums[c] 改变了  
 c++;  
 while(c<d&&nums[d-1]==nums[d]) //确保nums[d] 改变了  
 d--;  
 c++;  
 d--;  
 }  
 }  
 }  
 }  
 return res;  
 }  
};  
  
作者：misakasagiri-2  
链接：https://leetcode-cn.com/problems/4sum/solution/shuang-zhi-zhen-jie-fa-can-zhao-san-shu-zhi-he-ge-/  
来源：力扣（LeetCode）

* Python 解法（国际版力扣高票答案）：

The core is to implement a fast 2-pointer to solve 2-sum, and recursion to reduce the N-sum to 2-sum. Some optimization was be made knowing the list is sorted.

Python

def fourSum(self, nums, target):
  
 def findNsum(nums, target, N, result, results):
  
 if len(nums) < N or N < 2 or target < nums[0]\*N or target > nums[-1]\*N: # early termination
  
 return
  
 if N == 2: # two pointers solve sorted 2-sum problem
  
 l,r = 0,len(nums)-1
  
 while l < r:
  
 s = nums[l] + nums[r]
  
 if s == target:
  
 results.append(result + [nums[l], nums[r]])
  
 l += 1
  
 while l < r and nums[l] == nums[l-1]:
  
 l += 1
  
 elif s < target:
  
 l += 1
  
 else:
  
 r -= 1
  
 else: # recursively reduce N
  
 for i in range(len(nums)-N+1):
  
 if i == 0 or (i > 0 and nums[i-1] != nums[i]):
  
 findNsum(nums[i+1:], target-nums[i], N-1, result+[nums[i]], results)
  
 results = []
  
 findNsum(sorted(nums), target, 4, [], results)
  
 return results

1. [迷宫](https://leetcode-cn.com/problems/the-maze/)（力扣第490题）

难度：中等

【参考答案】

* 官方题解（Java 解法）：

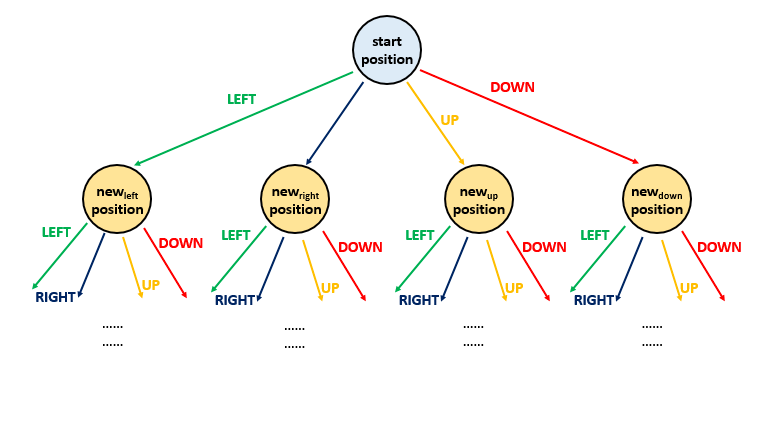
作者：LeetCode

链接：<https://leetcode-cn.com/problems/the-maze/solution/mi-gong-by-leetcode/>

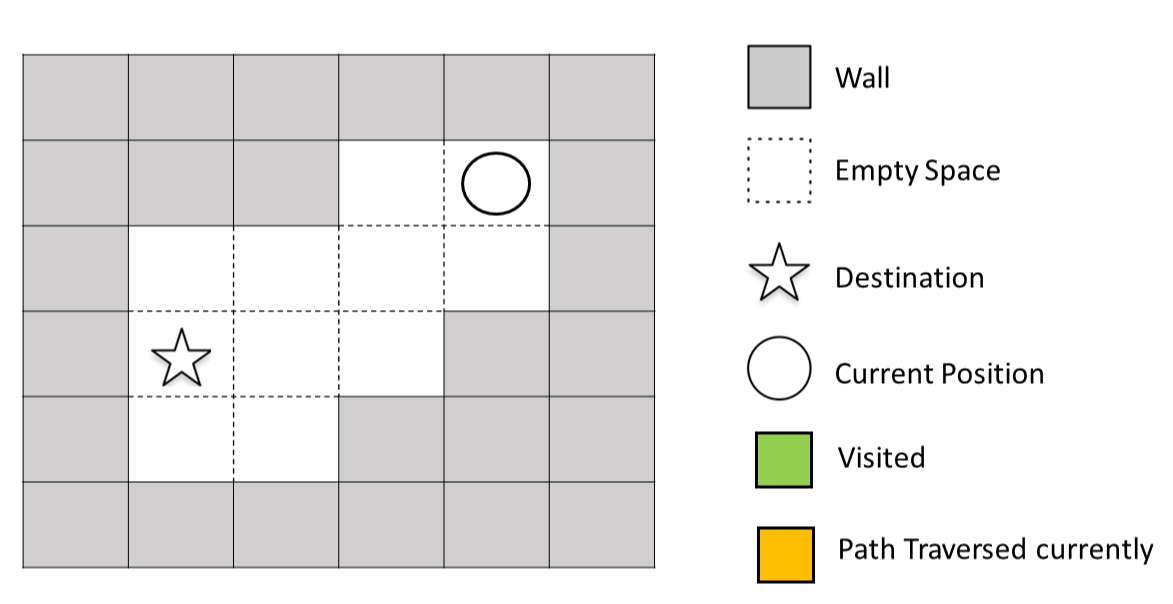
来源：力扣（LeetCode）

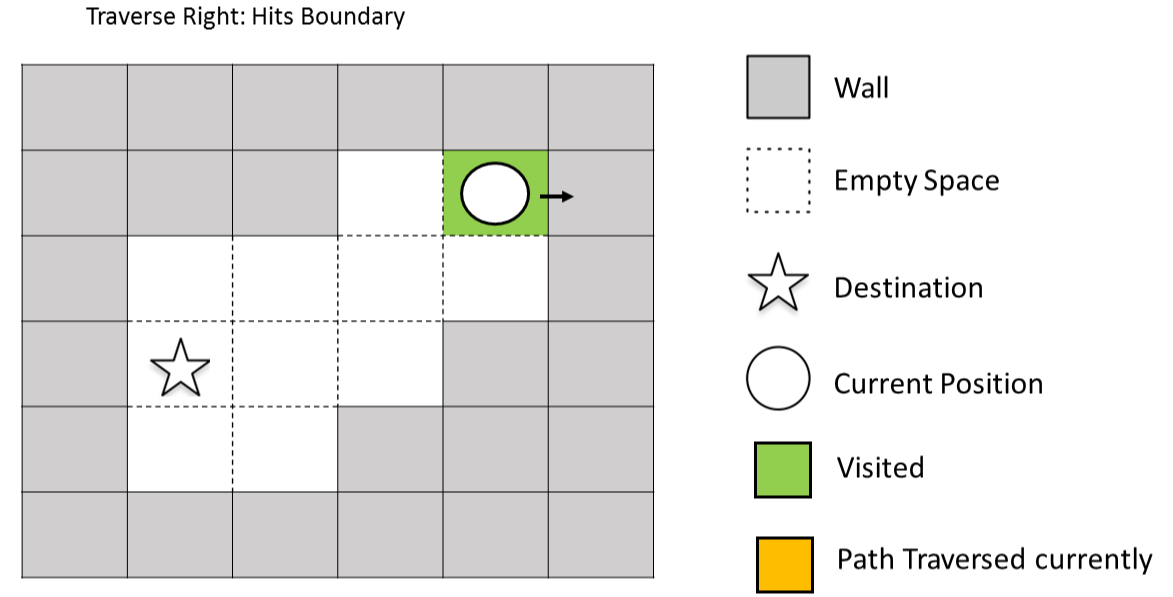
* 方法一：深度优先搜索

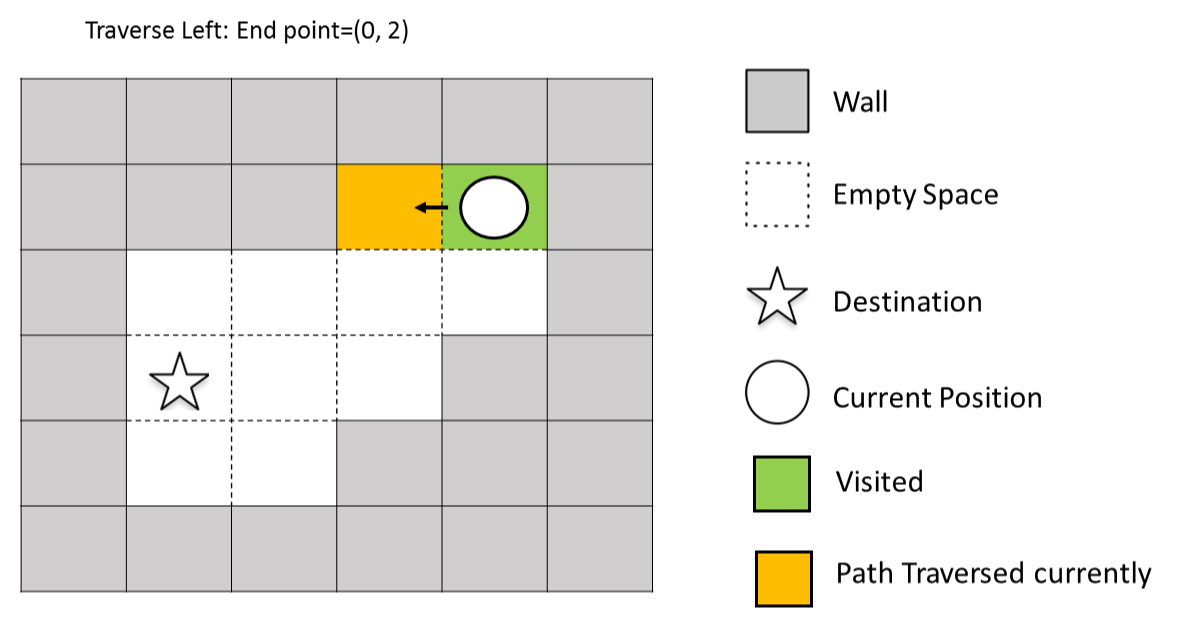
我们可以用搜索树的形式来展开搜索空间。如下图所示，根节点代表起始位置，每个节点有 4 个孩子，表示 4 种不同的路线：左、右、上、下。经过某条路线到达一个新的节点，就表示在迷宫中选择某个方向滚动直到停止。

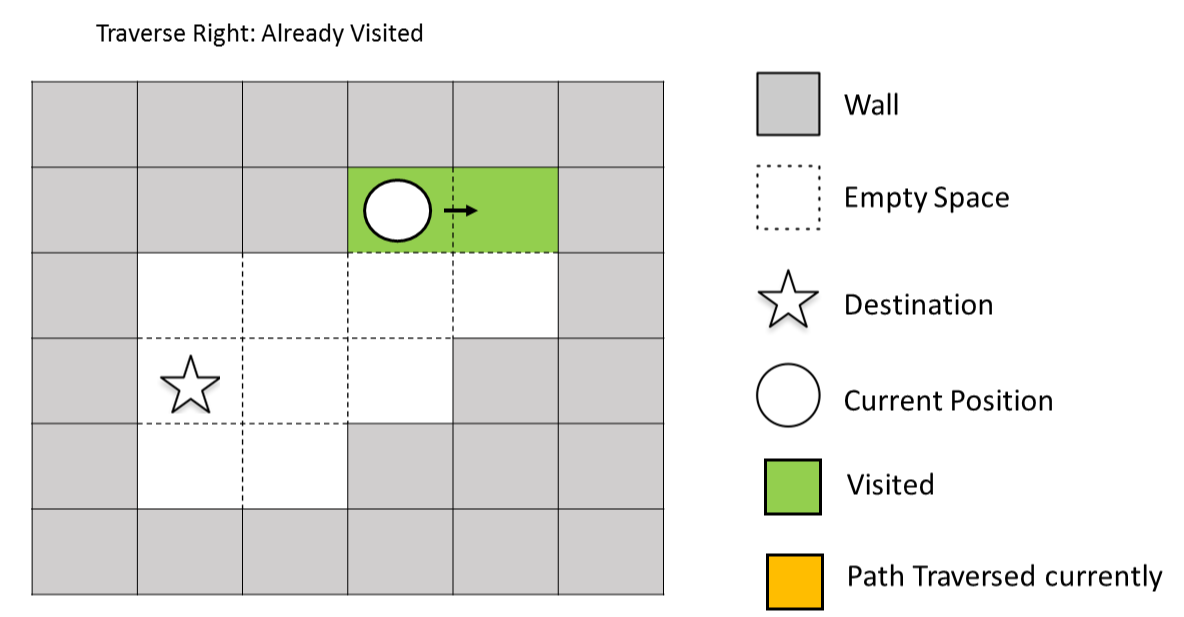


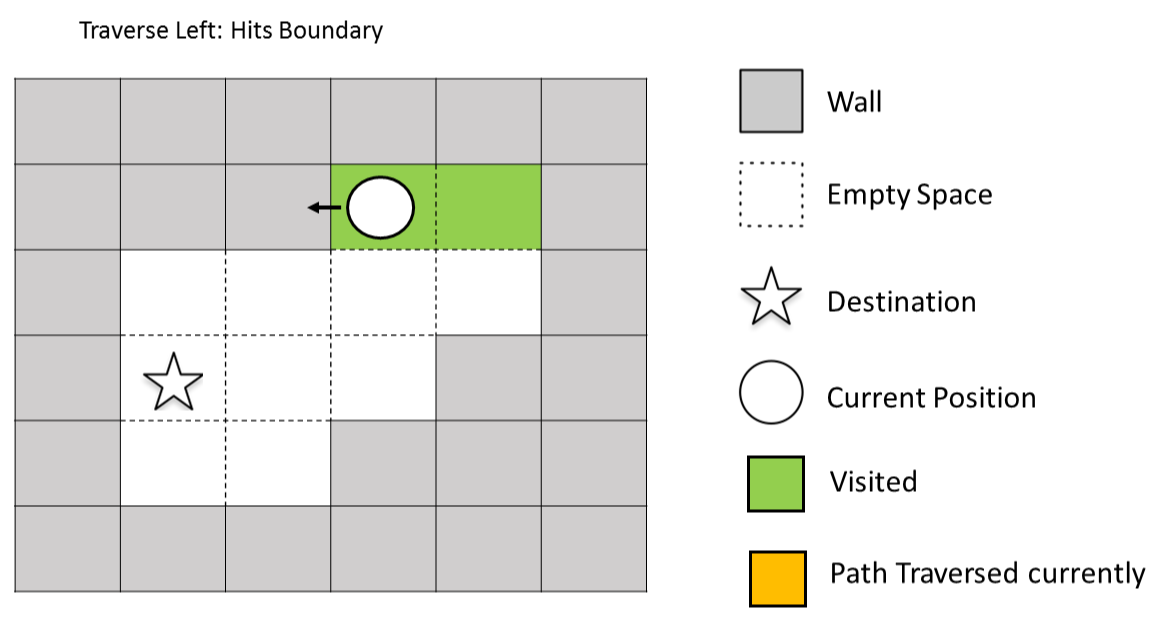
我们可以使用深度优先搜索对整颗搜索树进行遍历。我们从起始位置开始，每次选择一条路线进行搜索，并用一个二维布尔数组 visited 表示是否曾经到达过位置 (i, j) ，若在某一次搜索到位置 (i, j) 时，visited[i, j] = true，那么我们可以进行回溯，不必继续搜索下去。

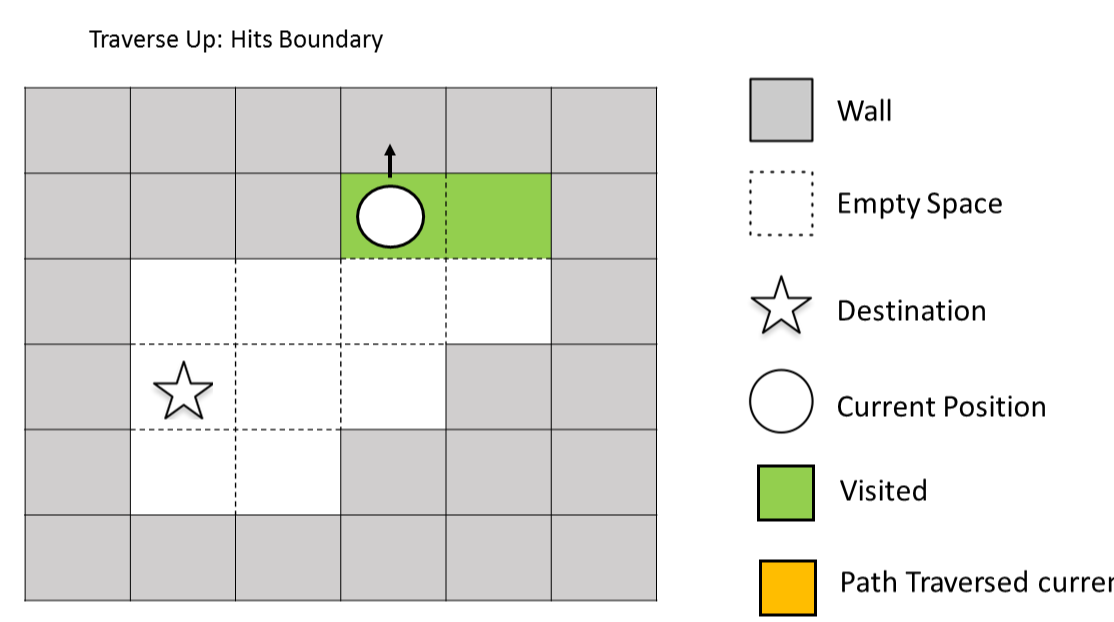
1. 

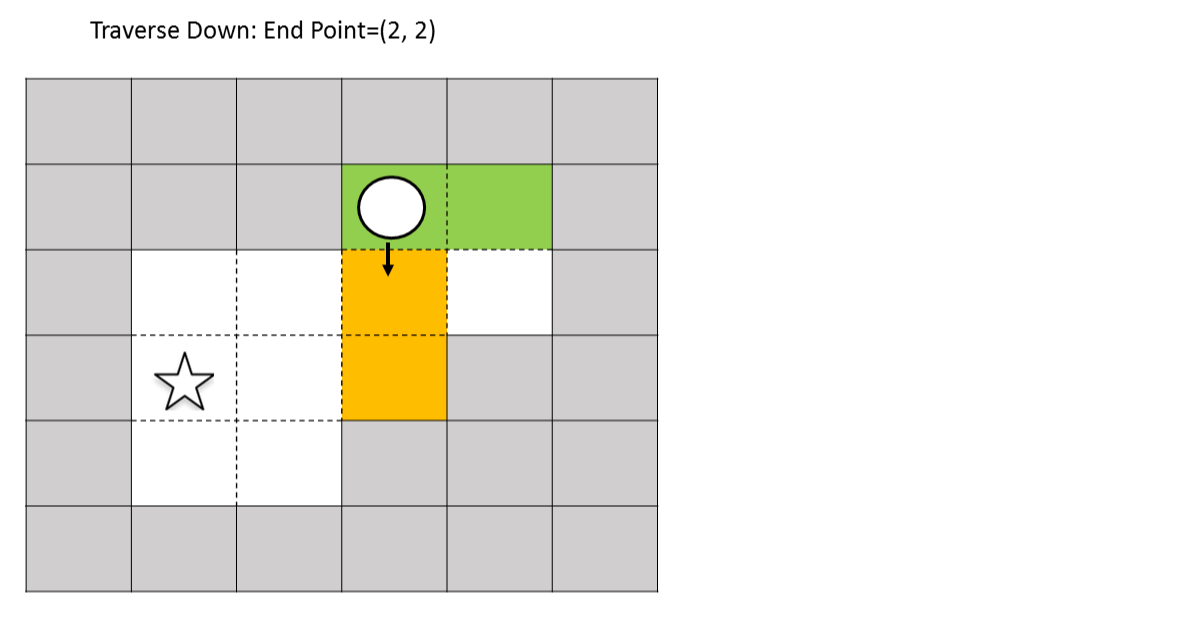
2.

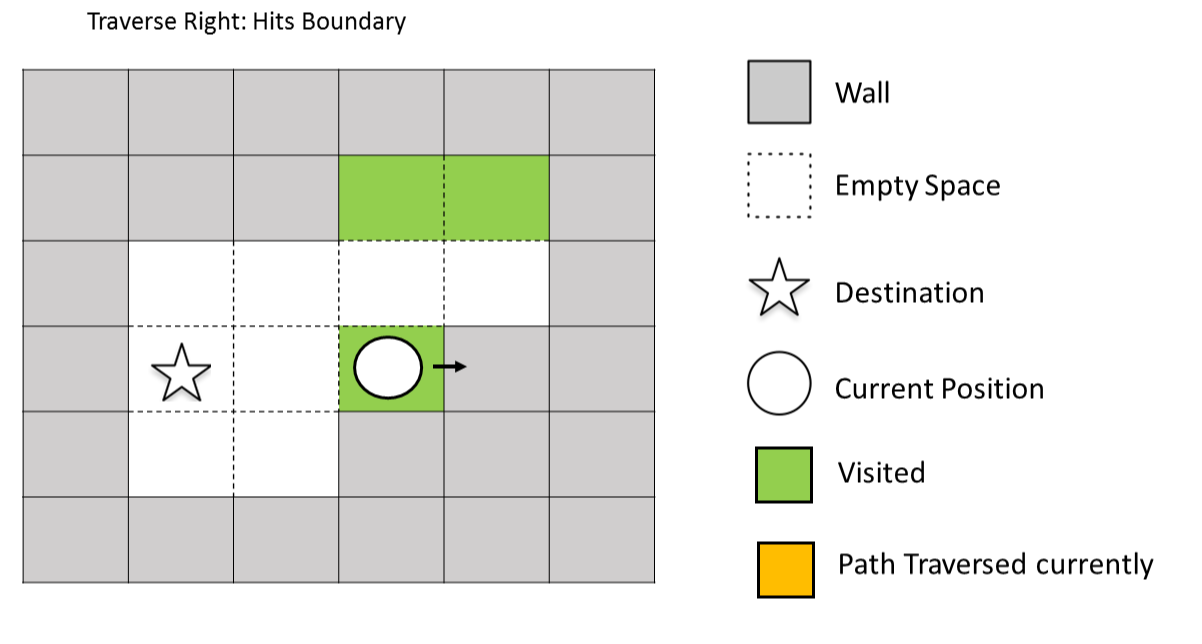
3.4.

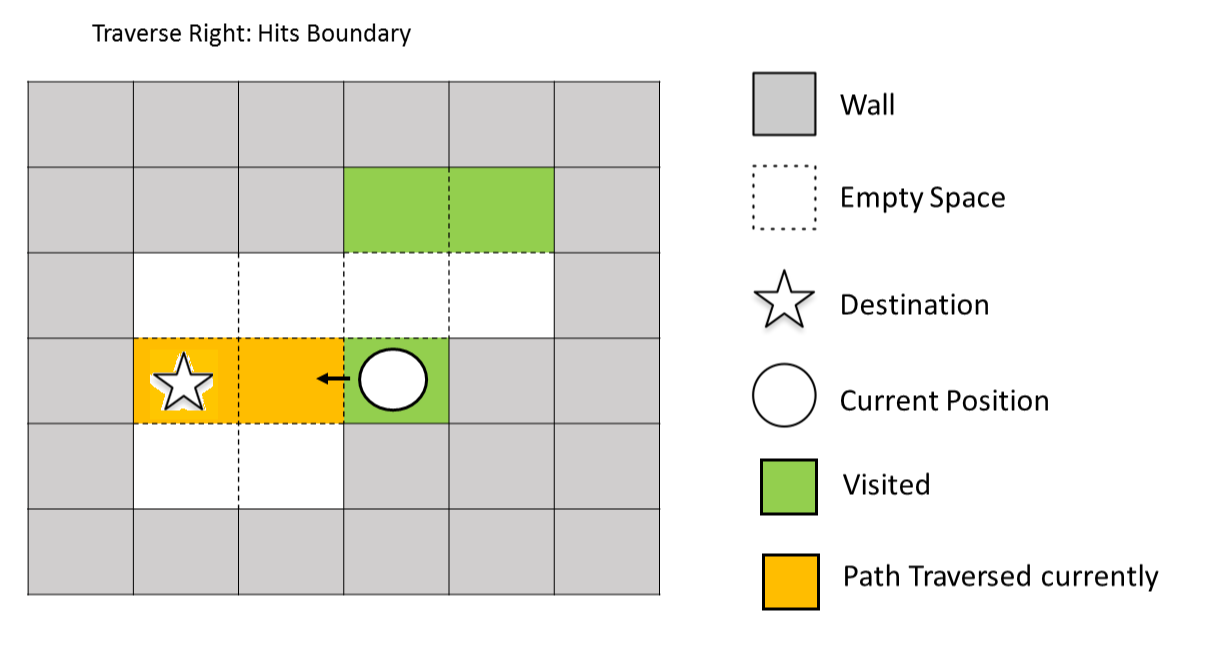
5.

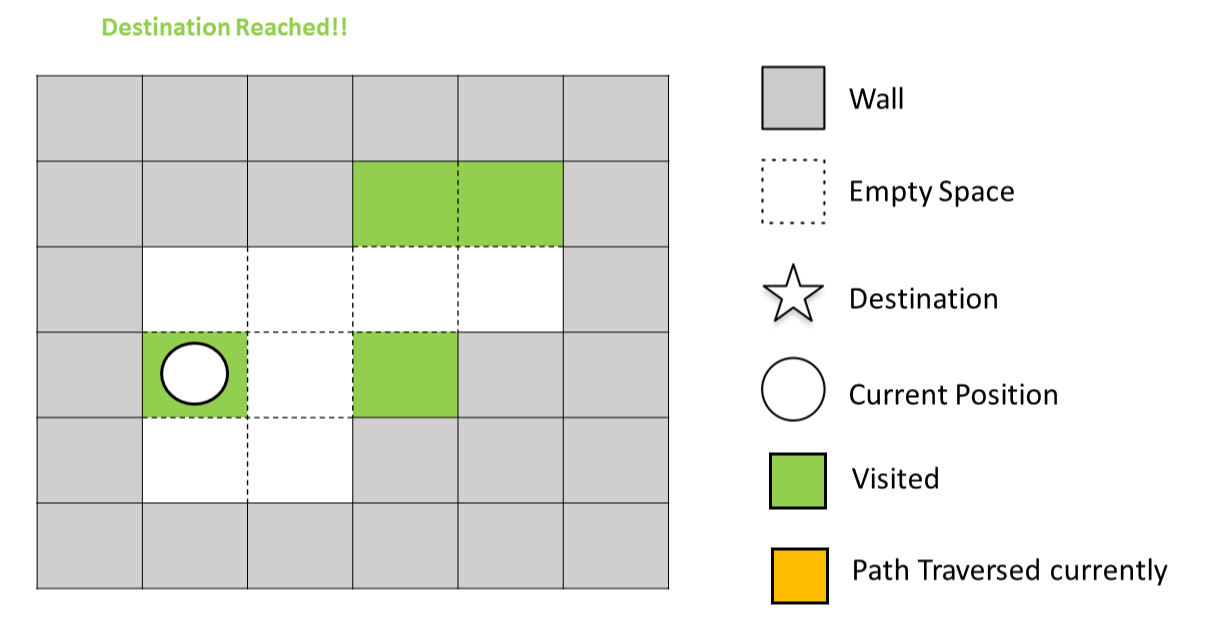
6.

7.

8.

9.

10.

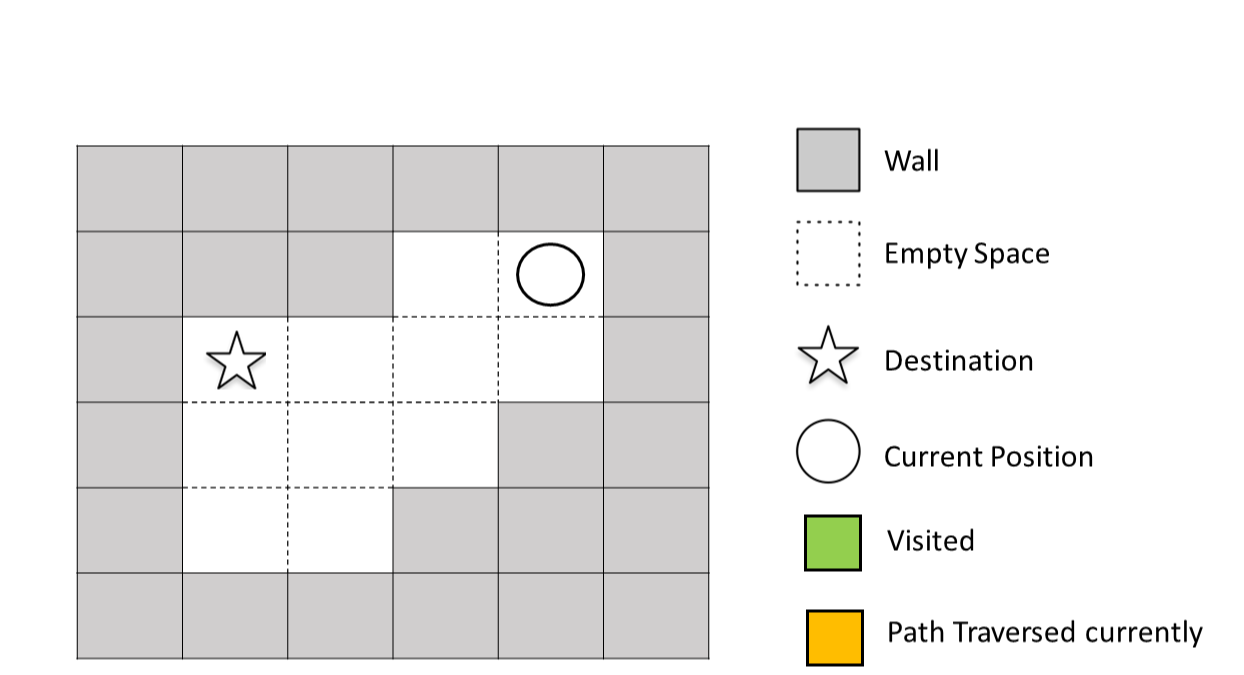
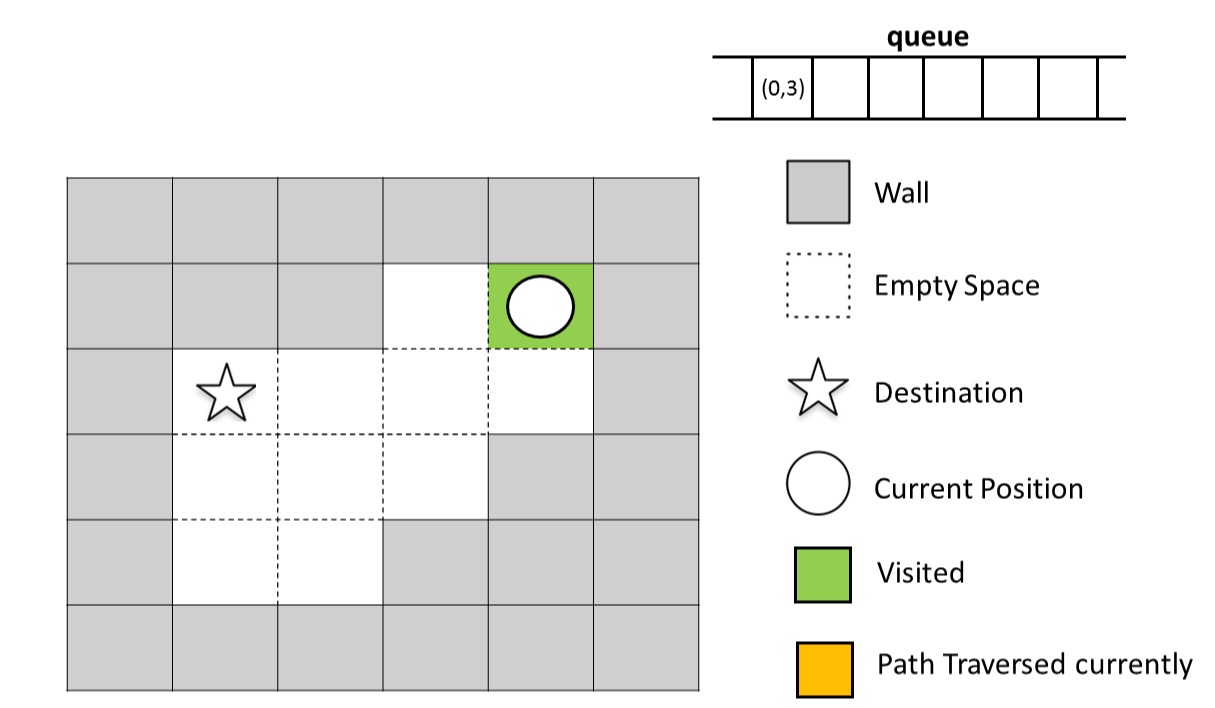
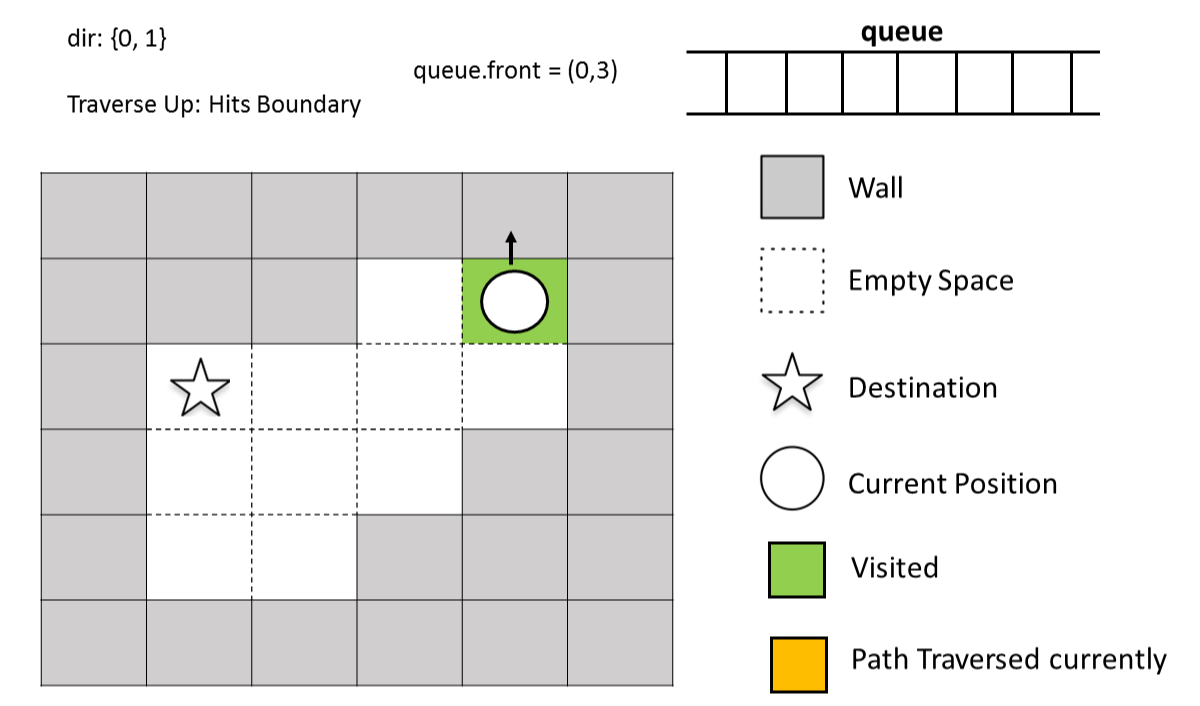
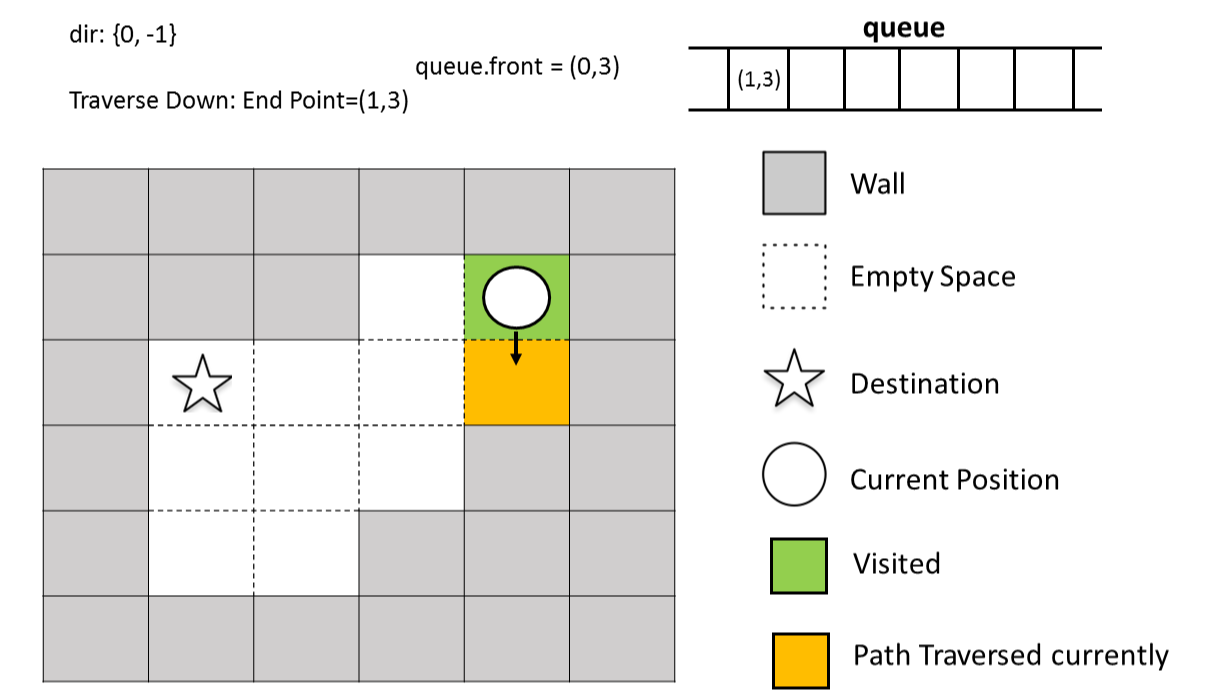
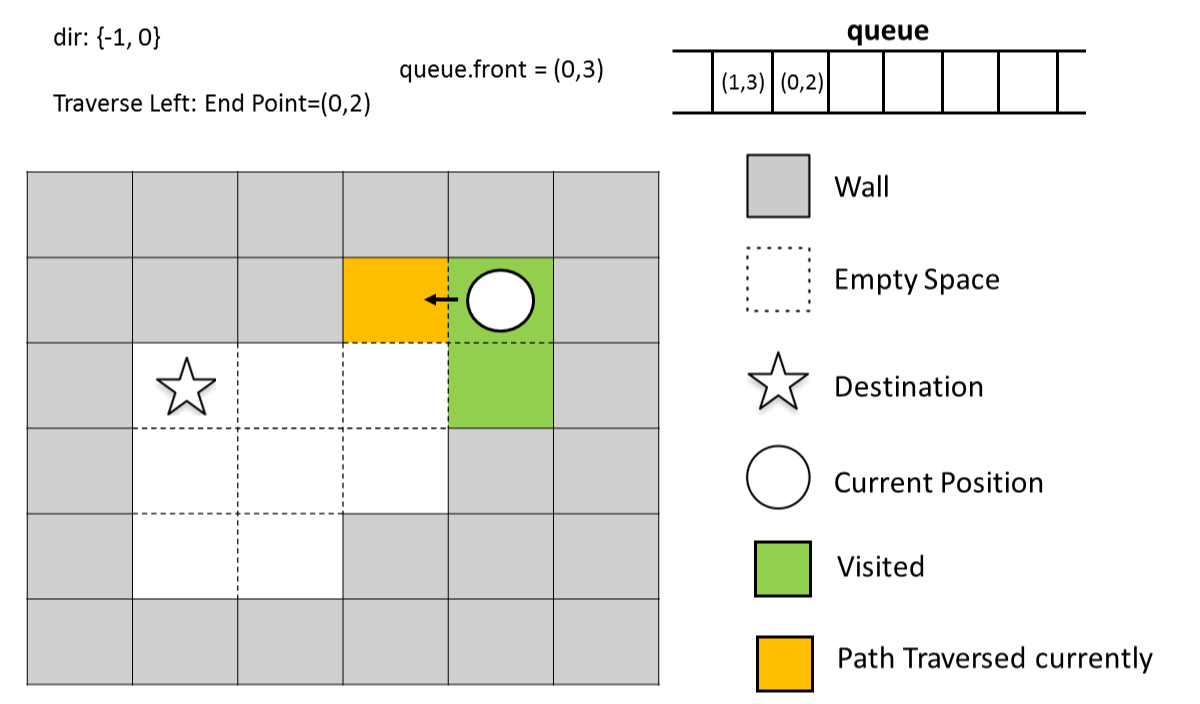
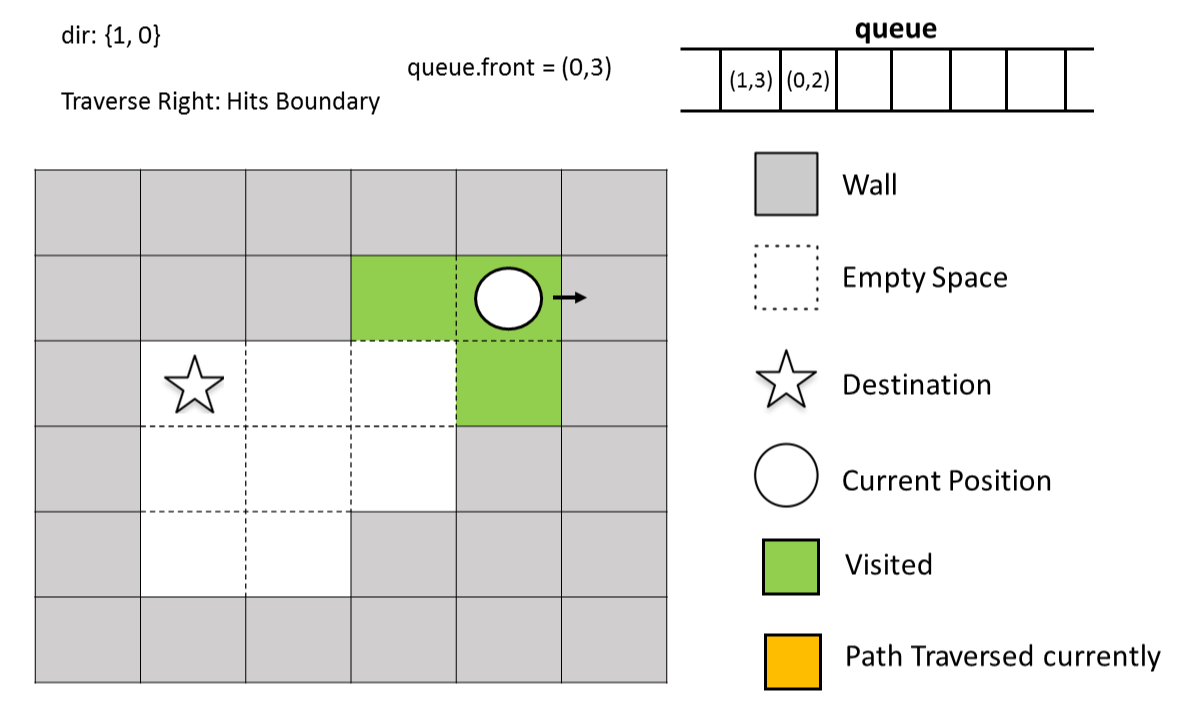
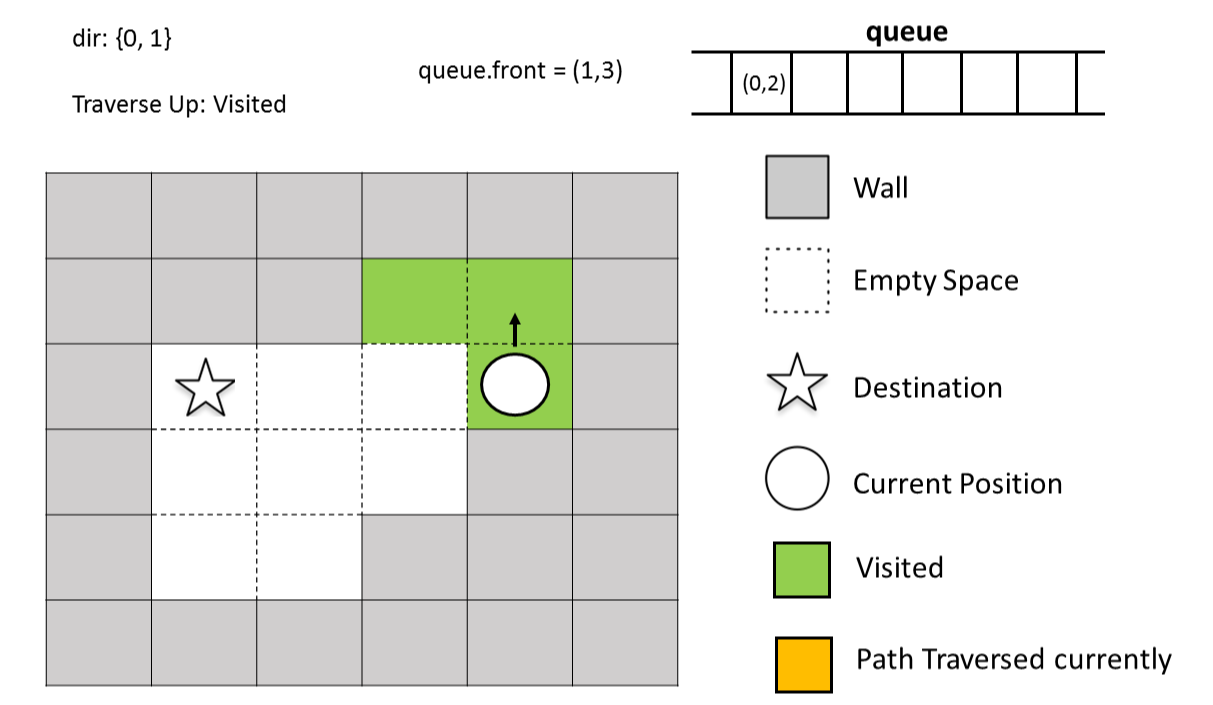
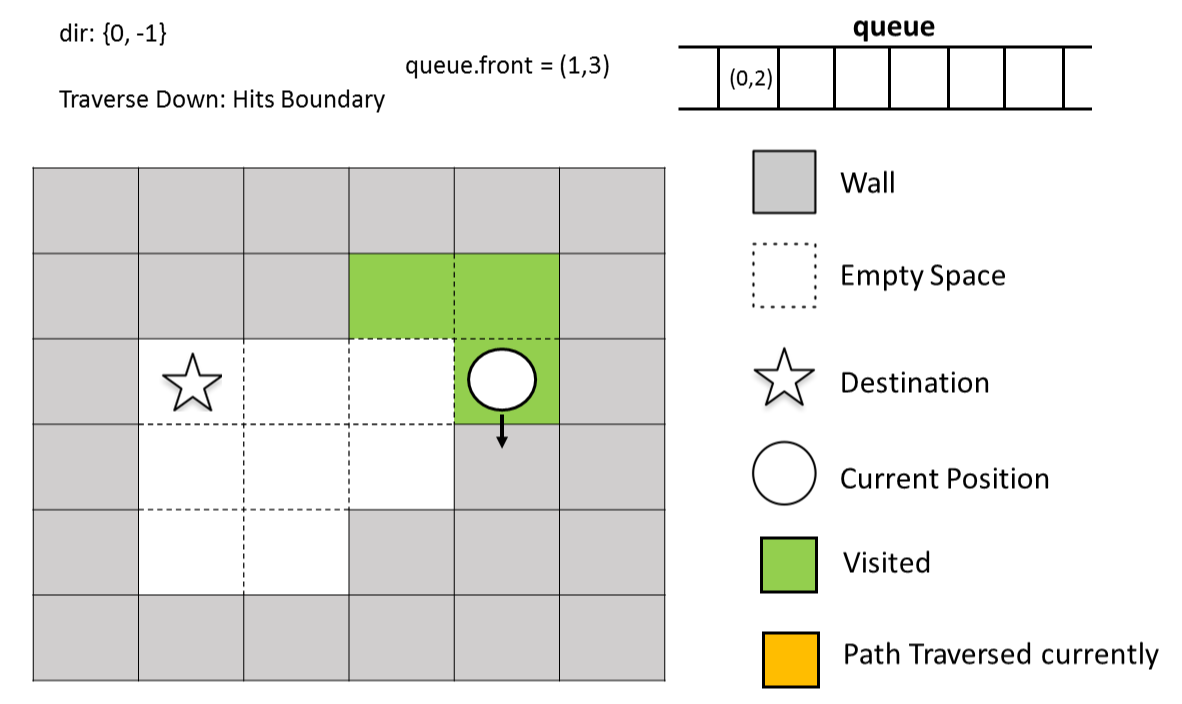
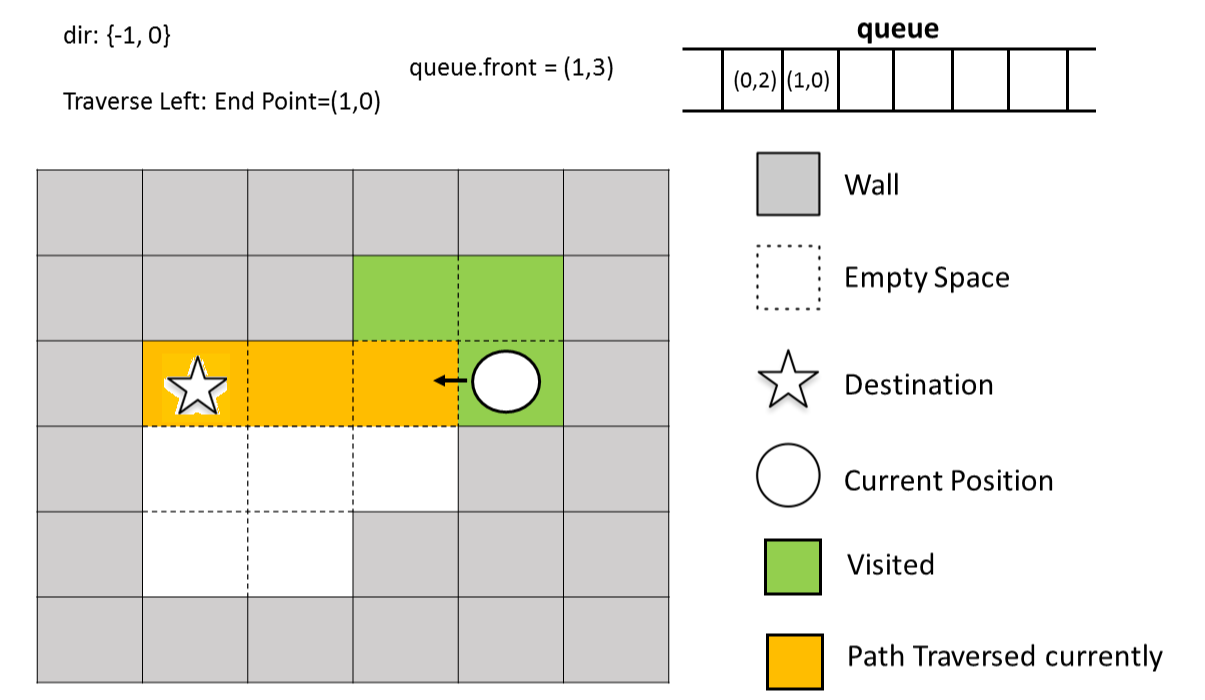
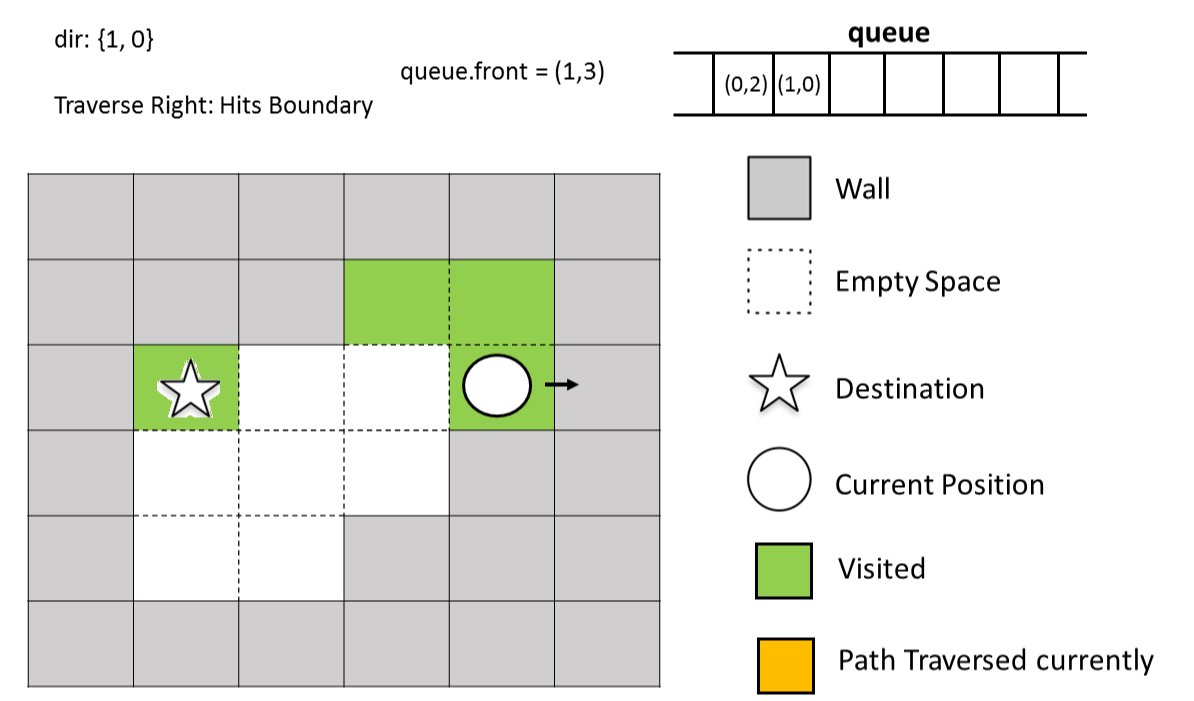
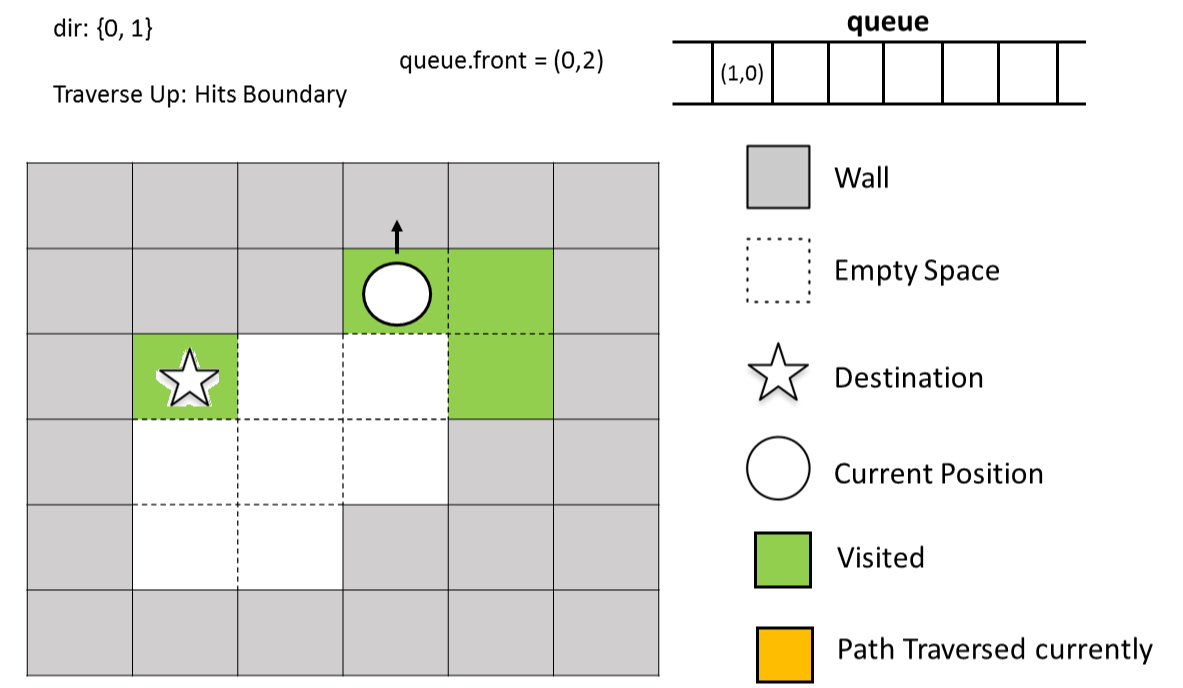
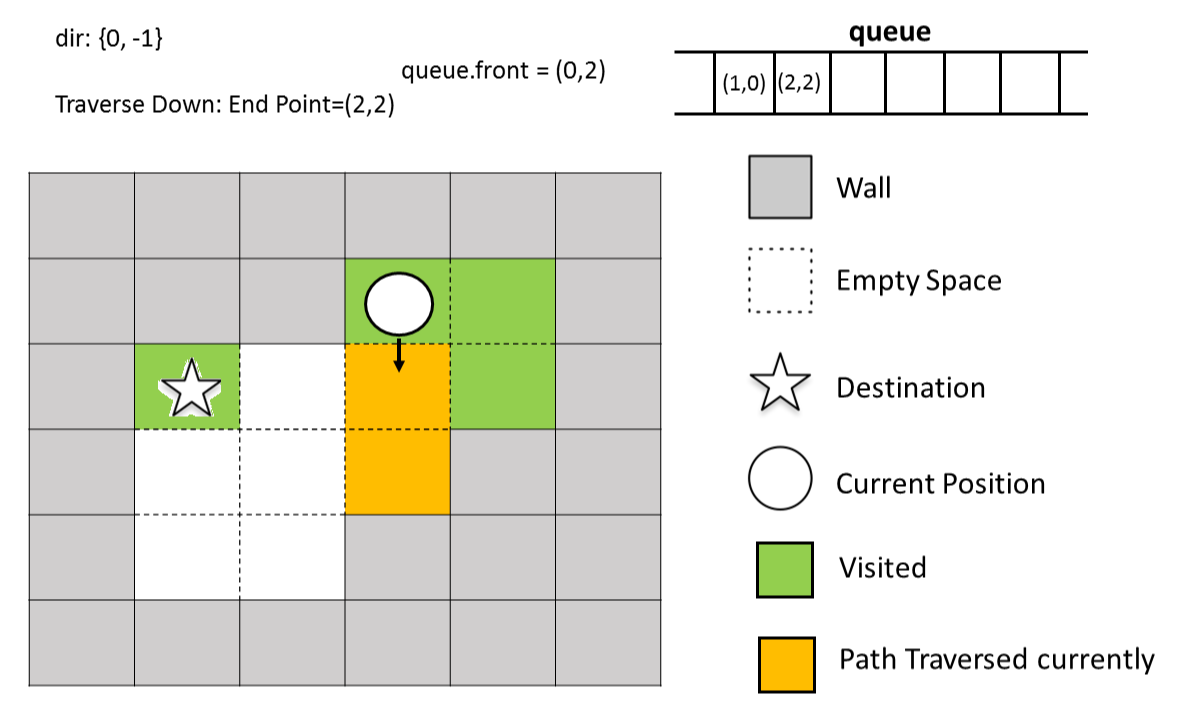
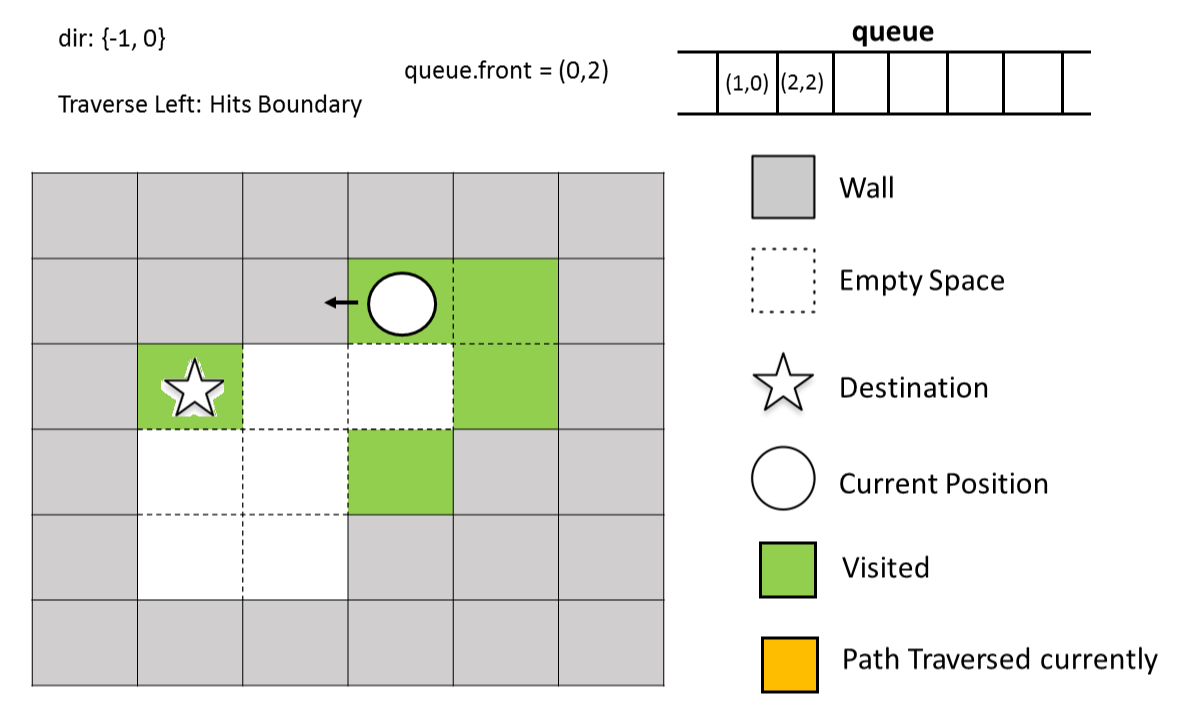
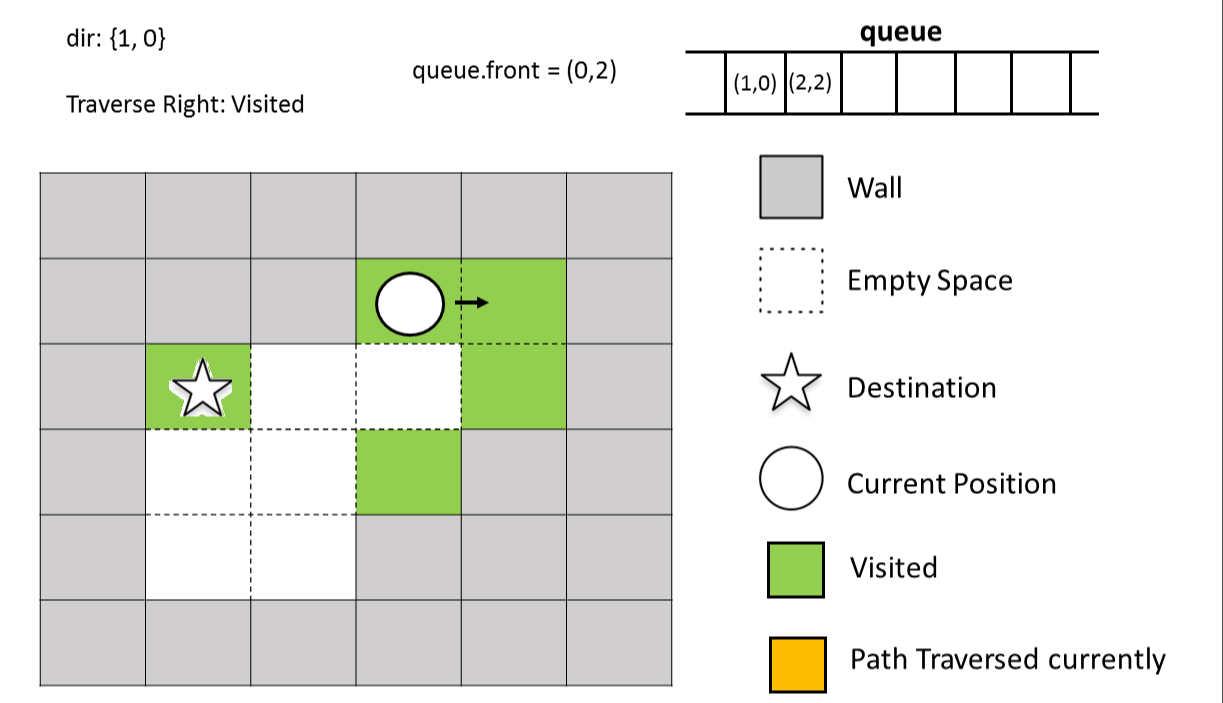
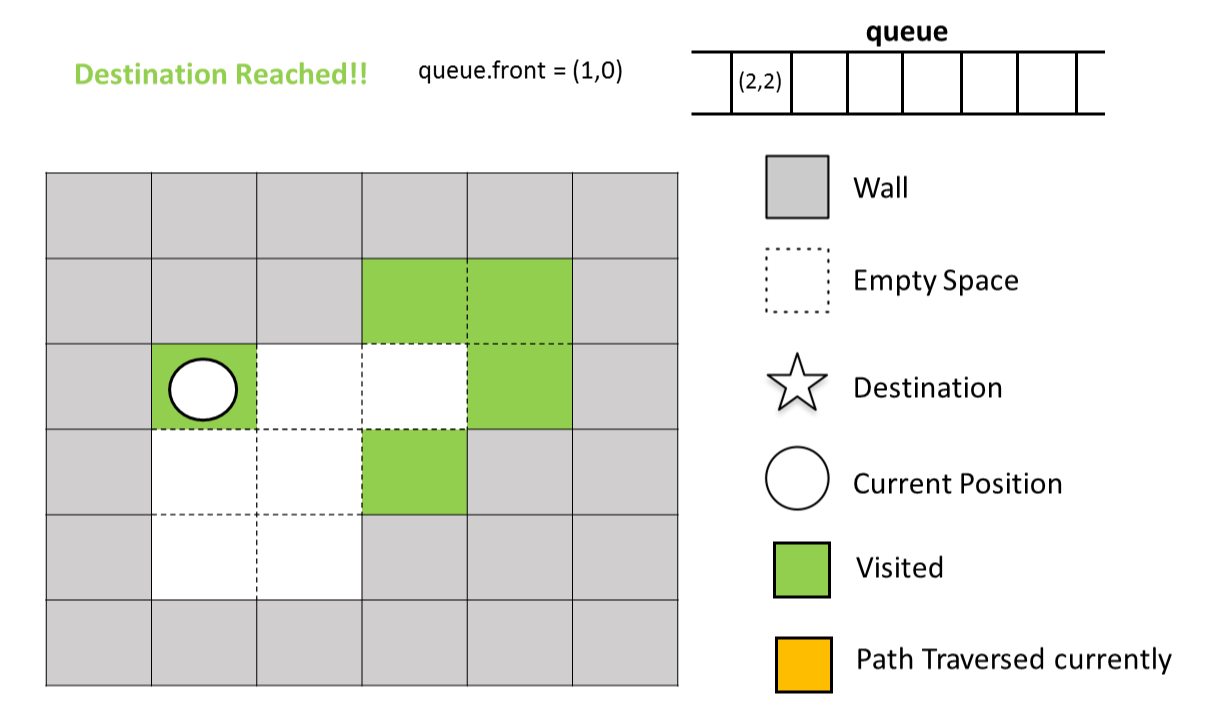


Java

public class Solution {
  
 public boolean hasPath(int[][] maze, int[] start, int[] destination) {
  
 boolean[][] visited = new boolean[maze.length][maze[0].length];
  
 return dfs(maze, start, destination, visited);
  
 }
  
 public boolean dfs(int[][] maze, int[] start, int[] destination, boolean[][] visited) {
  
 if (visited[start[0]][start[1]])
  
 return false;
  
 if (start[0] == destination[0] && start[1] == destination[1])
  
 return true;
  
 visited[start[0]][start[1]] = true;
  
 int r = start[1] + 1, l = start[1] - 1, u = start[0] - 1, d = start[0] + 1;
  
 while (r < maze[0].length && maze[start[0]][r] == 0) // right
  
 r++;
  
 if (dfs(maze, new int[] {start[0], r - 1}, destination, visited))
  
 return true;
  
 while (l >= 0 && maze[start[0]][l] == 0) //left
  
 l--;
  
 if (dfs(maze, new int[] {start[0], l + 1}, destination, visited))
  
 return true;
  
 while (u >= 0 && maze[u][start[1]] == 0) //up
  
 u--;
  
 if (dfs(maze, new int[] {u + 1, start[1]}, destination, visited))
  
 return true;
  
 while (d < maze.length && maze[d][start[1]] == 0) //down
  
 d++;
  
 if (dfs(maze, new int[] {d - 1, start[1]}, destination, visited))
  
 return true;
  
 return false;
  
 }
  
}
  
// 作者：LeetCode链接：https://leetcode-cn.com/problems/the-maze/solution/mi-gong-by-leetcode/来源：力扣（LeetCode）

* 方法二：深度优先搜索

我们同样可以使用广度优先搜索，实现细节与深度优先搜索类似，同样会使用一个二维布尔数组 visited，若 visited[i, j] = true，则表示曾经到达过位置 (i, j)。

1.2.3.4.5.6.7.8.9.10.11.12.13.14.15.

Java

public class Solution {
  
 public boolean hasPath(int[][] maze, int[] start, int[] destination) {
  
 boolean[][] visited = new boolean[maze.length][maze[0].length];
  
 int[][] dirs={{0, 1}, {0, -1}, {-1, 0}, {1, 0}};
  
 Queue < int[] > queue = new LinkedList < > ();
  
 queue.add(start);
  
 visited[start[0]][start[1]] = true;
  
 while (!queue.isEmpty()) {
  
 int[] s = queue.remove();
  
 if (s[0] == destination[0] && s[1] == destination[1])
  
 return true;
  
 for (int[] dir: dirs) {
  
 int x = s[0] + dir[0];
  
 int y = s[1] + dir[1];
  
 while (x >= 0 && y >= 0 && x < maze.length && y < maze[0].length && maze[x][y] == 0) {
  
 x += dir[0];
  
 y += dir[1];
  
 }
  
 if (!visited[x - dir[0]][y - dir[1]]) {
  
 queue.add(new int[] {x - dir[0], y - dir[1]});
  
 visited[x - dir[0]][y - dir[1]] = true;
  
 }
  
 }
  
 }
  
 return false;
  
 }
  
}
  
// 作者：LeetCode链接：https://leetcode-cn.com/problems/the-maze/solution/mi-gong-by-leetcode/来源：力扣（LeetCode）

* C++ 解法：

与常规迷宫略有一些不同，本题并不需要枚举出地图中每一个点，只需要枚举出靠墙的点即可。

C++

class Solution {
  
public:
  
 int dir[2][4] = {{0,0,1,-1},
  
 {1,-1,0,0}};
  
 int maxx,maxy,dex,dey;
  
 bool vis[105][105]={0};
  
 inline bool legal(int x,int y)//判断点是否越界
  
 {
  
 return (x>=0&&y>=0&&x<maxx&&y<maxy);
  
 }
  
 bool hasPath(vector<vector<int>>& maze, vector<int>& start, vector<int>& destination)
  
 {
  
 dex=destination[0],dey=destination[1];
  
 maxx = maze.size(),maxy = maze[0].size();
  
 return dfs(start[0],start[1],maze);
  
 }
  
 inline bool dfs(int x,int y,vector<vector<int>>& graph
  
 {
  
 vis[x][y] = 1;//已经访问过该节点
  
 if(x==dex&&y==dey)//可以到达目的地
  
 return true;
  
 for(int i=0;i<4;i++) {
  
 int tempx = x + dir[0][i];
  
 int tempy = y + dir[1][i];
  
 while(legal(tempx,tempy)&&graph[tempx][tempy]!=1) { //沿着某个方向走到靠墙的点
  
 tempx += dir[0][i];
  
 tempy += dir[1][i];
  
 }
  
 tempx -= dir[0][i];//回溯到最后合法的靠墙点
  
 tempy -= dir[1][i];
  
 if(vis[tempx][tempy])//如果访问过就不再访问
  
 continue;
  
 if(dfs(tempx, tempy,graph))//没访问就继续深搜
  
 return true;//已经可以到达目的地，返回true
  
 }
  
 return false;//所有方向都无法到达目的地
  
 }
  
};
  
   
// 作者：zhu-yi-hang-de-lao-fu-qin链接：https://leetcode-cn.com/problems/the-maze/solution/cdfs-by-zhu-yi-hang-de-lao-fu-qin-71gd/来源：力扣（LeetCode）