```
#author :weihaoysgs@gmail.com
#date : 2022-2-20
```

Search Based Path Finding

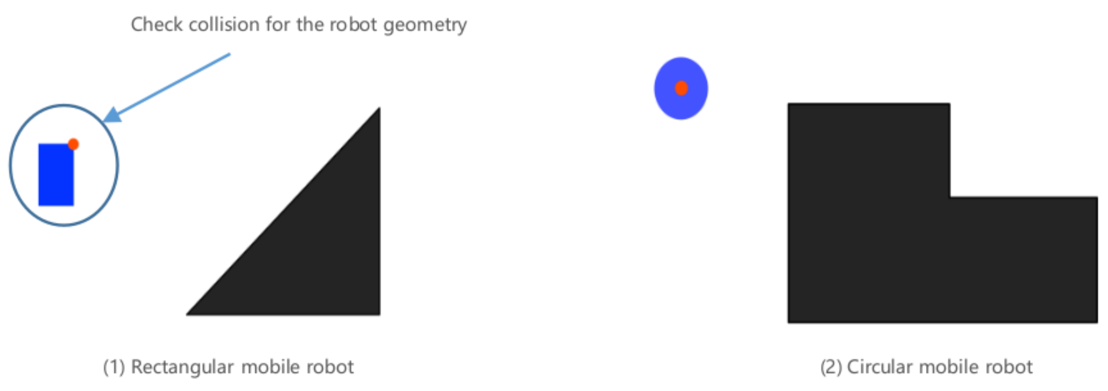
在进入正式的介绍之前我们要先有一些基于图搜索的路径规划的相关知识。下面我将进行详细的介绍

Configuration Space (构形空间)

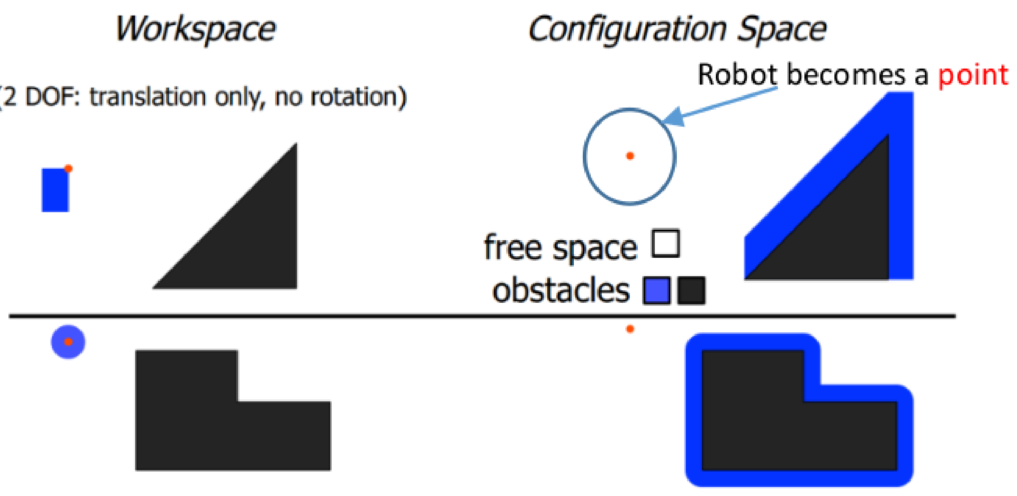
- Robot configuration: a specification of the positions of all points of the robot,One robot configuration defines a specific state of the robot.
- The minimum number n of real-valued coordinates needed to represent the robot configuration. 简单来说就是通过一个 n 维向量来代表机器人的状态或者说是机器人自身的自由度。
- Robot configuration space: a n-dim space containing all possible robot configurations, denoted as C-space. 一个可能包含机器人所有状态的 n 维空间，我们称之为机器人的构型空间，简称 C-sapce。
- Each robot pose is a point in the C-space. 每一个机器人在 C-space 中只是一个点。

Configuration Space Obstacle(C-space 中的障碍物)

- Planning in workspace(在工作空间中进行规划)
    - 首先我们先定义工作空间，注意这个工作空间和 ros 中的工作空间不一样。简单来说就是机器人当时所处的真实世界，或者更形象一点的说是机器人通过自身传感器信息所构造的世界，比如说 SLAM 技术生成的地图等。
    - Robot has different shape and size(真实世界中每个机器人是拥有不同的形状和大小的)
    - Collision detection required knowing the geometry data. 机器人运动规划过程中需要知道自己的几何信息，形状大小之类的。要做出更多的计算，这是很耗时的，并且难度较大。

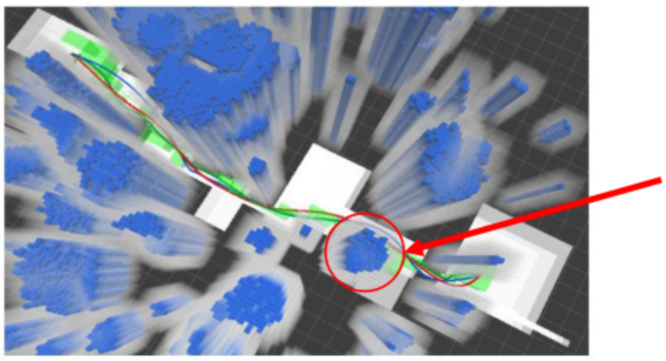

Check collision for the robot geometry
(1) Rectangular mobile robot
(2) Circular mobile robot

- Planning in configuration space (在 C-space 中进行规划)
    - Robot is represented by a point in C-space, e.g. position (a point in R^3), pose (a point in $SO(3)$),etc.而在 C-space 中，机器人的位置只是R^3 空间中的一个点，位姿则是 $SO(3)$ 中的一个点。
    - Obstacles need to be represented in configuration space (one-time work prior to motion planning),called configuration space obstacle, or C-obstacle
    - $C-space = (C-obstacle) \cup (C-free)$
    - The path planning is finding a path between start point q_{start} and goal point q_{goal} within C-free



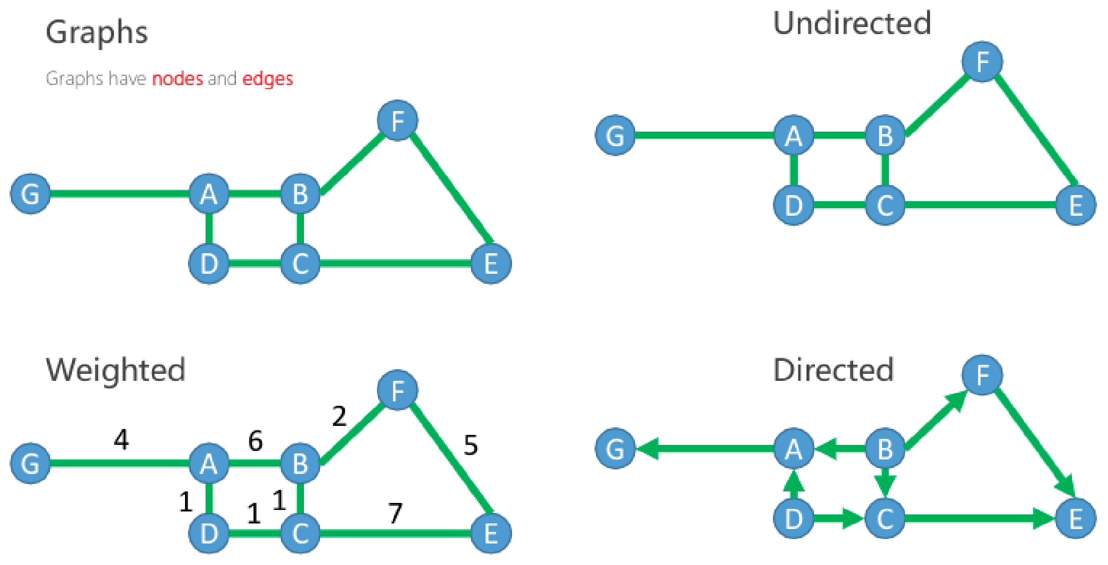
Workspace and Configuration Space Obstacle

- In workpace
  - Robot has shape and size (i.e:hard for motion planning)
- In configuration Space: C-space
  - Robot is a point (i.e:easy for motion planning)
  - Obstacle are represented in C-space prior to motion planning
- Representing an obstacle in C-space can be extremely complicated. So approximated (but more conservative(谨慎的，保守的，适当的)) represention are used in practice.

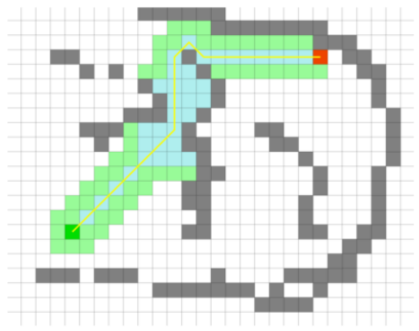

If we model the robot conservatively as a ball with radius δ_r,
then the C-space can be constructed by inflating obstacle at all directions by δ_r.

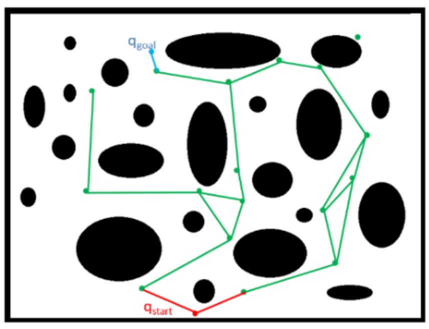
Graph and Search Method

关于图其实是图论中的一些知识点。图是由边和节点构成的，而根据边和节点的不同属性，又可以将图进行不同种类的分类。分别由无向图，无向赋权图，有向图，有向赋权图等。下面是一些例子。



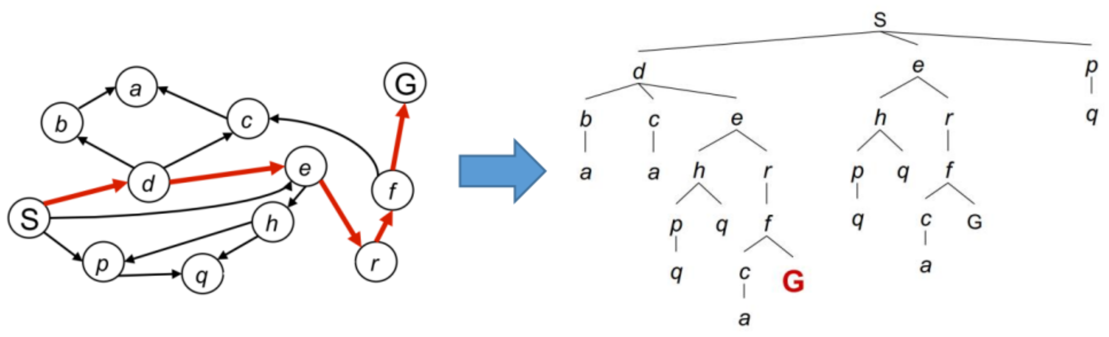
- State space graph: a mathematical representation of a search algorithm （状态空间图，是搜索算法在数学上的一种代表）
    - For everyu search problem, there is a corresponding state space graph. （对于每一个图索索问题，都有一个对应的状态空间图）
    - Connectivity between nodes in the graph is represented by (directed or undirected) edges. （由无向边或者有向边链接途中的节点）


Grid-based graph: use grid as vertices and grid connections as edges



The graph generated by probabilistic roadmap (PRM)

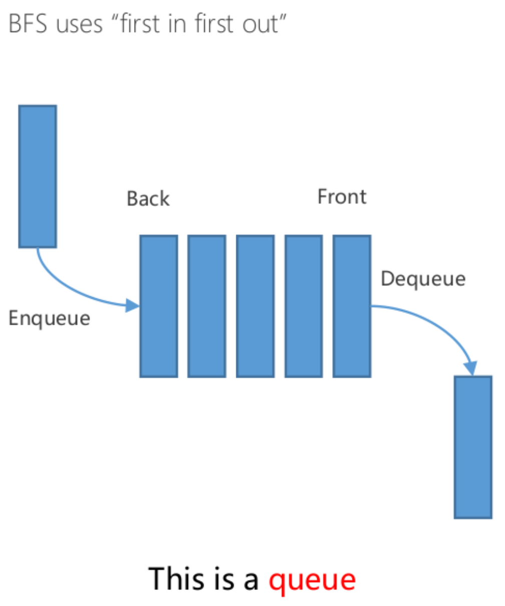
- The search always start from start state X_x
    - The Searching graph produces a search tree （根据搜索图可以生成一个搜索树）
    - Back-tracing a node in the search tree gives us a path from the start state to that node (反向跟踪搜索树中的一个节点，可以给出从起始状态到该节点的路径)
    - For many problems we can never actually build the whole tree, too large or inefficient – we only want to reach the goal node asap.（对于许多问题，我们永远无法真正构建整个树，因为太大或效率太低——我们只想尽快到达目标节点）

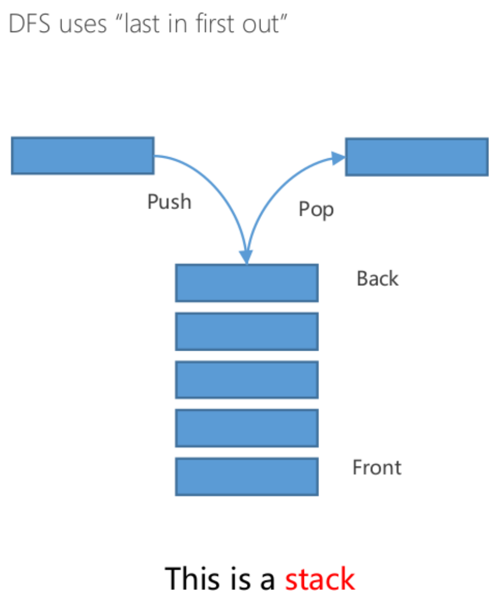


- Maintain a container to store all the nodes to be visited. （维护一个容器去保存所有将要被访问的节点）
- The container is initialized with the start state X_s。（容器初始化为空）
- Loop（循环）
    - Remove a node from the container according to some pre-defined score function（通过一个预先定义好的得分计算函数将容器中的某一个节点移除）
        - Visit a node（访问一个节点）
    - Expansion: Obtain all neighbors of the node（进行节点的扩展，得到移除节点的所有邻接节点，即邻居节点）
        - Discover all its neighbors（发现它所有的邻居）
    - Push them (neighbors) into the container（将发现的邻居节点送入容器中去）
- End Loop

以上的循环可以通过队列实现，也可以通过栈来实现，均符合上述的循环操作，但是在基于图的路径规划中，通常使用的是基于 BFS 的遍历，而 BFS 是通过队列实现的，为什么不用 DFS 呢，因为在栅格地图中 DFS 是无法完成每一个点的遍历的，可能会出现无法到达目标点的情况出现。

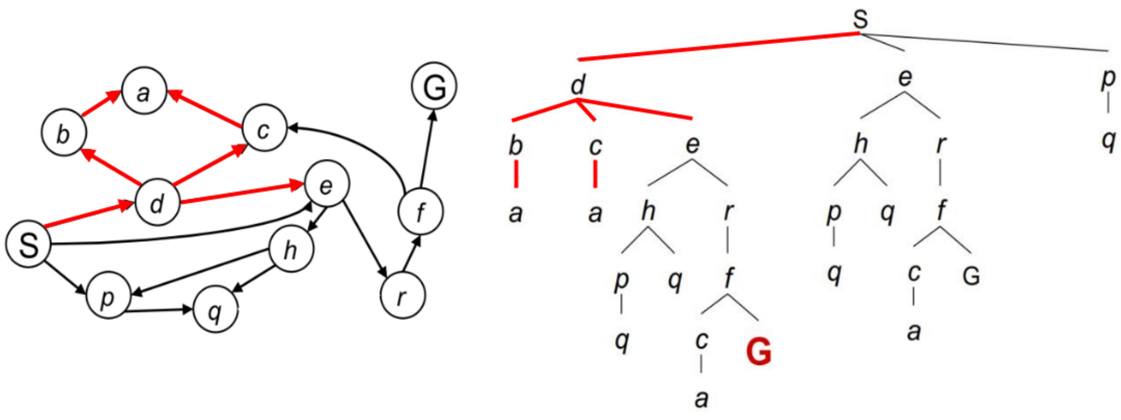
- Question One : When to end the loop ?
    - Possible option : when the container is empty
- Question Two : What if the graph is cyclic ?
    - When a node is removed from the container (expanded/visited), it should be add back to the container again.
- Question Three : In what way to remove the right node such that the goal state can be reached as soon as possible, which results in less expansion of the graph node. （以何种方式删除正确的节点，以便能够尽快达到目标状态，从而减少图节点的扩展），这个问题就是今天要讨论的问题。
- Breadth First Search (BFS) vs. Depth First Search (DFS)。（深度优先遍历 VS 广度优先遍历）



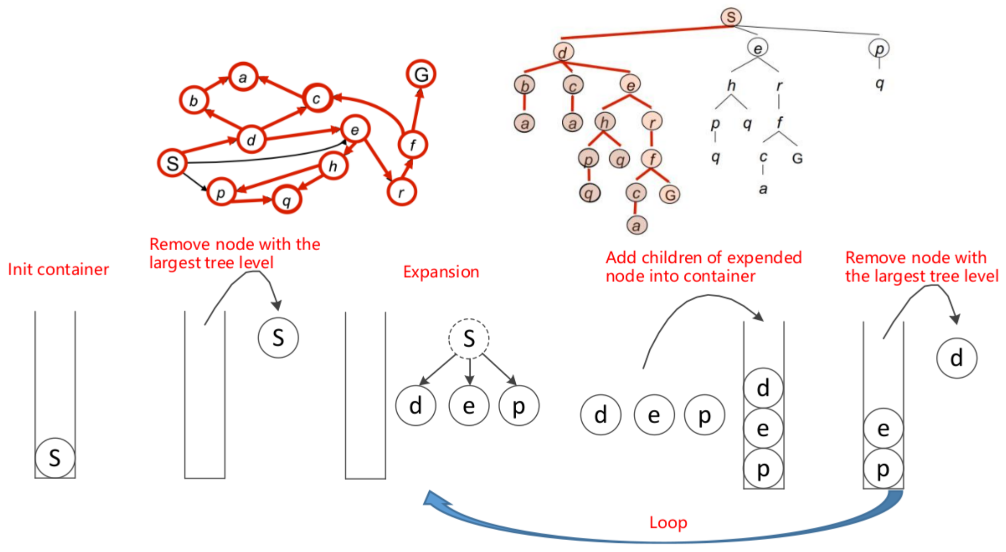


Depth First Search(DFS)

- Strategy : remove / expand the deepest node in the container.（策略：删除/扩展容器中最深的节点，注意是使用栈来进行存储的，在压栈之前先根据 Score 函数对即将压栈的节点进行排序，得分最高或者说代价最小的后压栈）



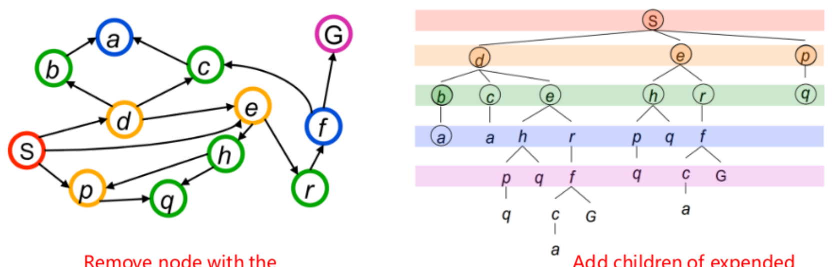
- Implentation : maintain a last in first out (LIFO) container (i.e. stack)

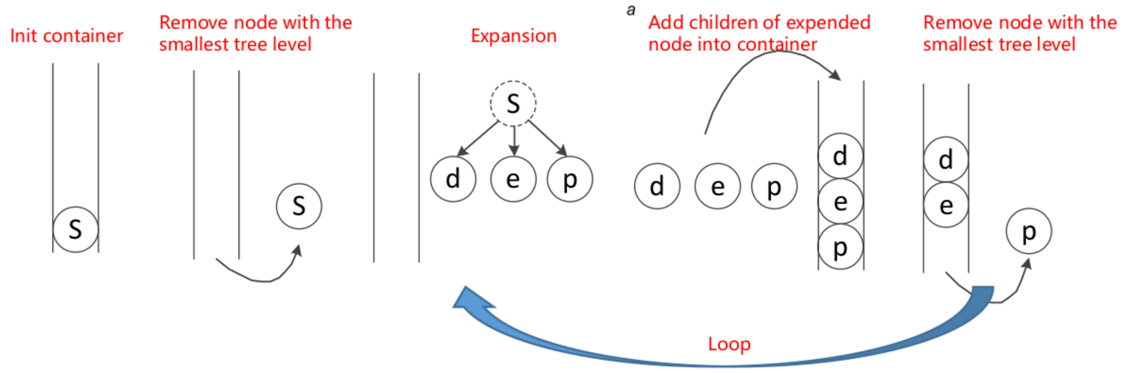


Init container
Remove node with the largest tree level
Expansion
Add children of expended node into container
Remove node with the largest tree level
Loop
S
d
e
p

Breadth First Search (BFS)

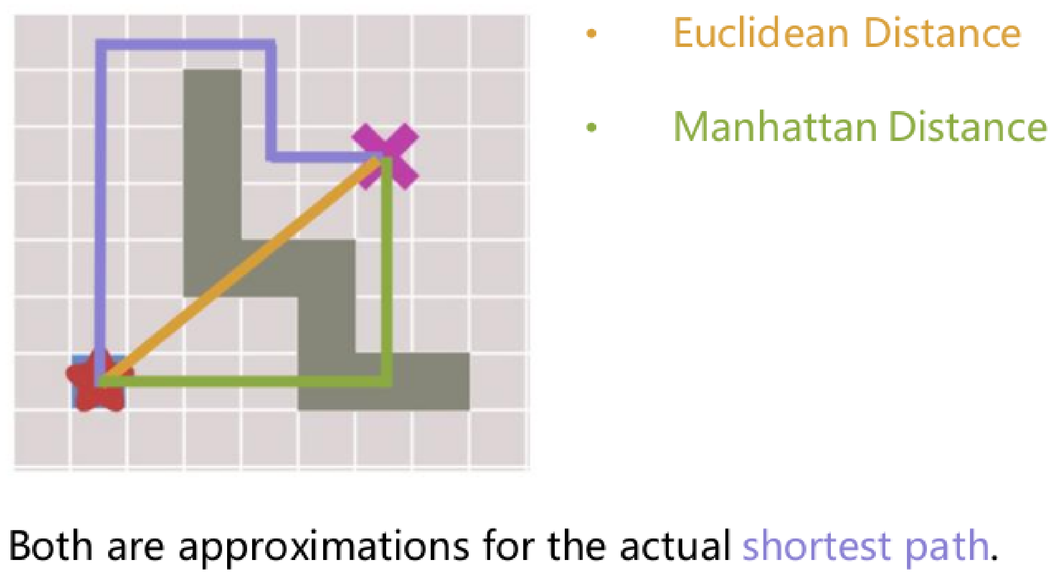
- Strategy: remove / expand the shallowest node in the container
- Implementation: maintain a first in first out (FIFO) container (i.e. queue)




Init container
Remove node with the smallest tree level
Expansion
Add children of expended node into container
Remove node with the smallest tree level
Loop

Heuristic search（启发式搜索）

Greedy Best First Search（贪心算法搜索）

- BFS and DFS pick the next node off the frontiers based on which was“first in” or “last in”.
- Greedy Best First picks the “best” node according to some rule, called a **heuristic**
- Definition: A heuristic is a **guess** of how close you are to the target.
- A heuristic guides you in the right direction.
- A heuristic should be easy to compute.
- 

  - Euclidean Distance
  - Manhattan Distance

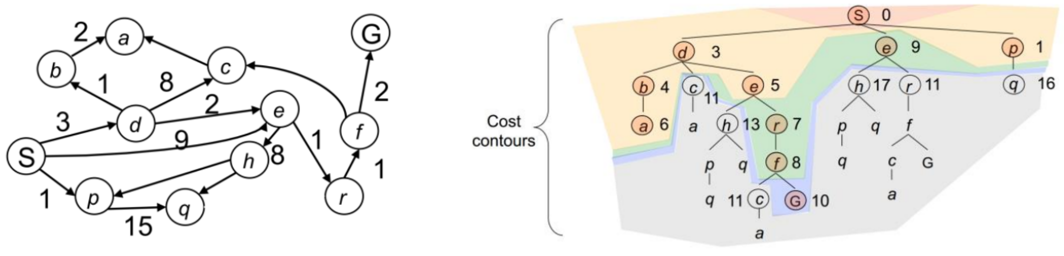
  Both are approximations for the actual shortest path.

Costs on Actions

- A practical search problem has a **cost “C”** from a node to its neighbor
  - Length, time, energy, etc.
- When all weight are 1, BFS finds the optimal solution
- For general cases, how to find the least-cost path as soon as possible?

Dijkstra and $A*$

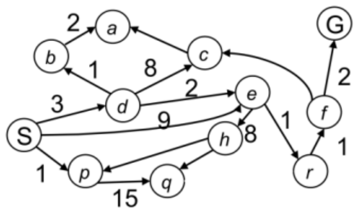
Dijkstra’s Algorithm

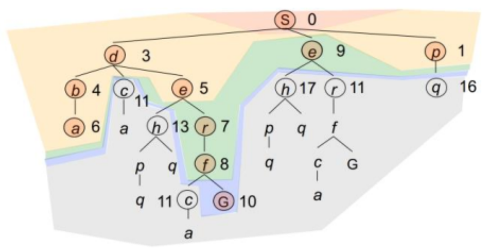
- Strategy: expand/visit the node with cheapest accumulated cost g(n)
    - g(n): The current best estimates of the accumulated cost from the start state to node “n”
    - Update the accumulated costs g(m) for all unexpanded neighbors “m” of node “n”
    - A node that has been expanded/visited is guaranteed to have the smallest cost from the start state
    - 

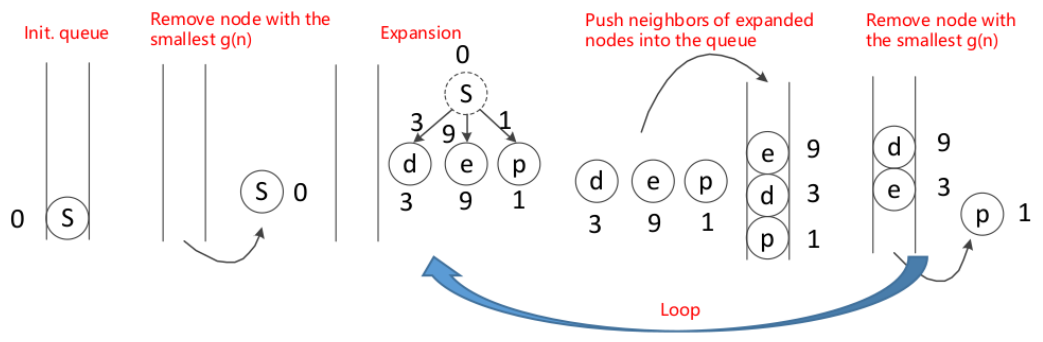
算法整体流程

- Maintain a priority queue to store all the nodes to be expanded
    - The priority queue is initialized with the start state XS
    - Assign g(XS)=0, and g(n)=infinite for all other nodes in the graph
    - Loop
        - If the queue is empty, return FALSE; break;
        - **Remove** the node “n” with the lowest g(n) from the priority queue
        - Mark node “n” as **expanded**
        - If the node “n” is the goal state, return TRUE; break;
        - For all **unexpanded** neighbors “m” of node “n”
            - If g(m) = infinite
                - g(m)= g(n) + Cnm
                - Push node “m” into the queue
            - If g(m) > g(n) + Cnm
                - g(m)= g(n) + Cnm
        - end
    - End Loop

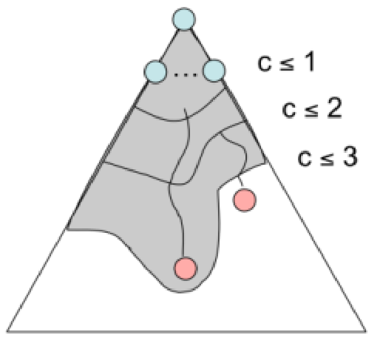
如图所示

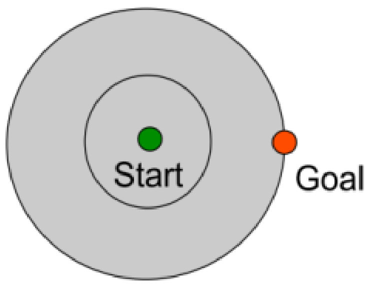





Init. queue
Remove node with the smallest g(n)
Expansion
Push neighbors of expanded nodes into the queue
Remove node with the smallest g(n)
Loop

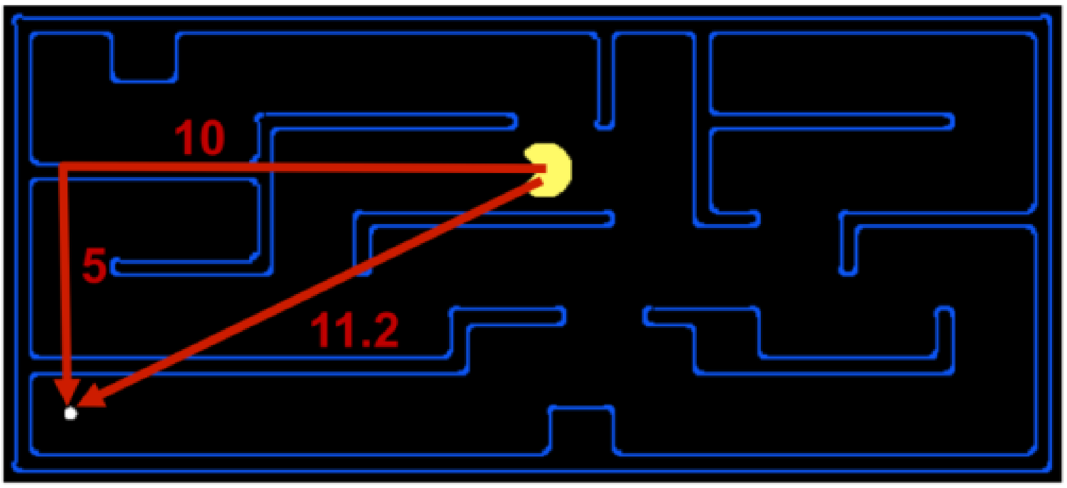
Pros and Cons of Dijkstra’s Algorithm（Dijkstra 算法的优缺点）

- The good:
  - Complete and optimal
- The bad:
  - Can only see the cost accumulated so far (i.e. the uniform cost), thus exploring next state in every “direction”
  - No information about goal location
  - 



Search Heuristics（启发式搜索）

- Recall the heuristic introduced in **Greedy Best First Search**
- Overcome the shortcomings of uniform cost search by **inferring the least cost to goal (i.e. goal cost)**
- Designed for particular search problem（为特定的搜索问题设定启发函数）
- Examples: Manhattan distance VS. Euclidean distance

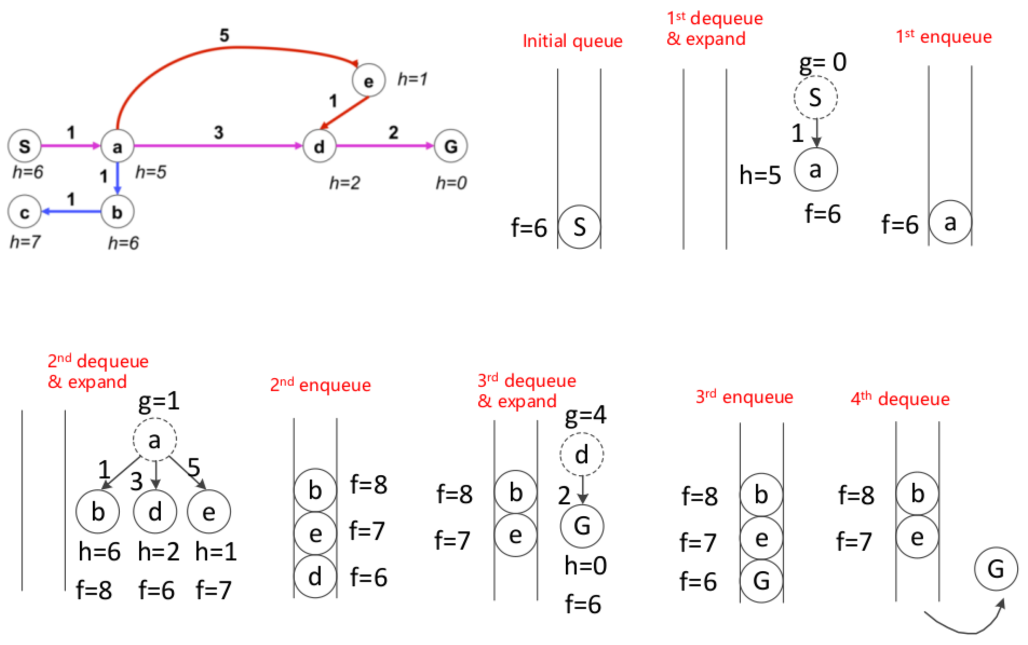
- 

A*: Dijkstra with a Heuristic

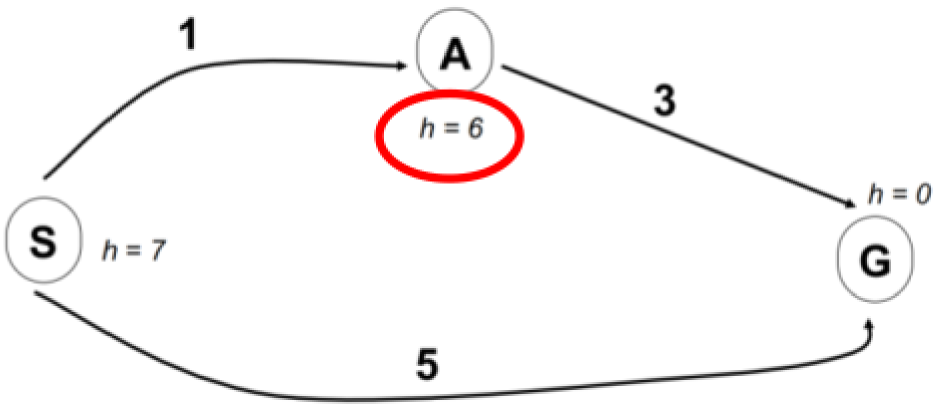
- Accumulated cost
    - g(n): The current best estimates of the accumulated cost from the start state to node “n”
- Heuristic
    - h(n): The **estimated least cost** from node n to goal state (i.e. goal cost)
- The least estimated cost from start state to goal state passing through node “n”is f(n) = g(n) + h(n)
- Strategy: expand the node with **cheapest f(n) = g(n) + h(n)**
    - Update the accumulated costs g(m) for all unexpanded neighbors “m”of node “n”
    - A node that has been expanded is guaranteed to have the smallest cost from the start state

算法整体流程

- Maintain a priority queue to store all the nodes to be expanded
    - The priority queue is initialized with the start state XS
    - Assign g(XS)=0, and g(n)=infinite for all other nodes in the graph
    - Loop
        - If the queue is empty, return FALSE; break;
        - **Remove** the node “n” with the lowest f(n)=g(n)+h(n) from the priority queue
        - Mark node “n” as **expanded**
        - If the node “n” is the goal state, return TRUE; break;
        - For all **unexpanded** neighbors “m” of node “n”
            - If g(m) = infinite
                - g(m)= g(n) + Cnm
                - Push node “m” into the queue
            - If g(m) > g(n) + Cnm
                - g(m)= g(n) + Cnm
        - end
    - End Loop
- A* Example

- 

A* Optimality

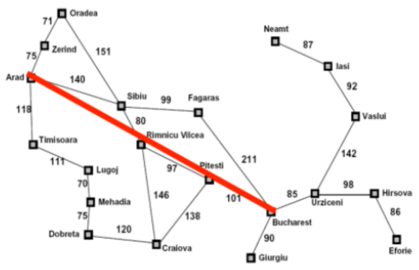
- 

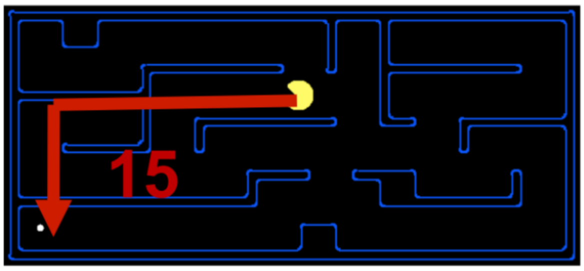
- What went wrong?
- For node A: actual least cost to goal (i.e. goal cost) < estimated least cost to goal (i.e. heuristic)
- We need the estimate to be **less than** actual least cost to goal(i.e. goal cost) **for all nodes**!

Admissible Heuristics

- A Heuristic h is **admissible** (optimistic) if:
    - h(n) <= h*(n) for all node "n",where h*(n) is the true least cost to goal from node "n"
- If the heuristic is admissible, the A* search is optimal
- Coming up with admissible heuristics is most of what'sinvolved in using A* in practice.

- Example:

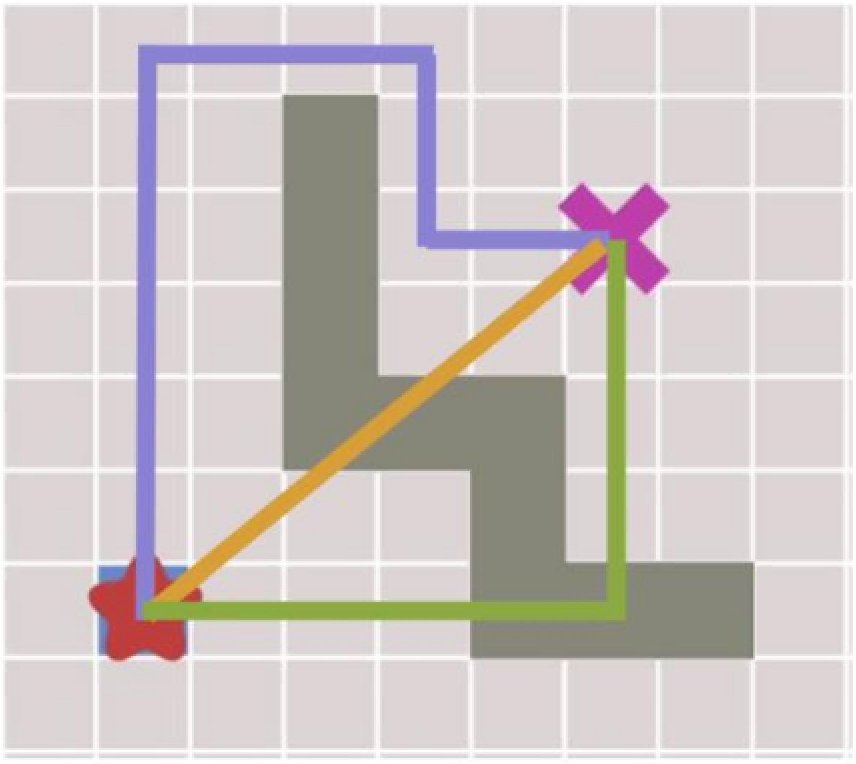




Heuristic Design

An admissible heuristic function has to be designed case by case.

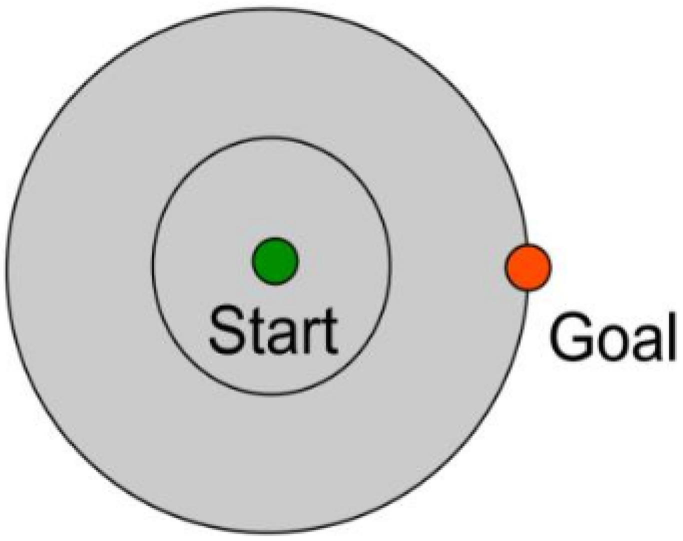
- Euclidean Distance
- Manhattan Distance



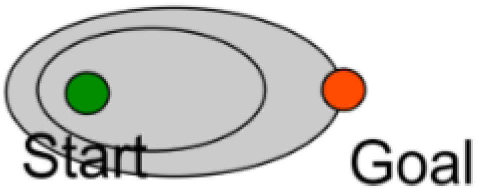
- Is Euclidean distance (L2 norm) admissible? **Always**
- Is Manhattan distance (L1 norm) admissible? **Depends**
- Is L∞ norm distance admissible? **Always**
- Is 0 distance admissible? **Always**

Dijkstra’s VS A*

- Dijkstra’s algorithm expanded in all directions

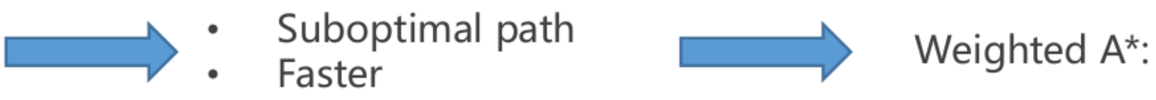


- A* expands mainly towards the goal, but does not hedge its bets to ensure optimality

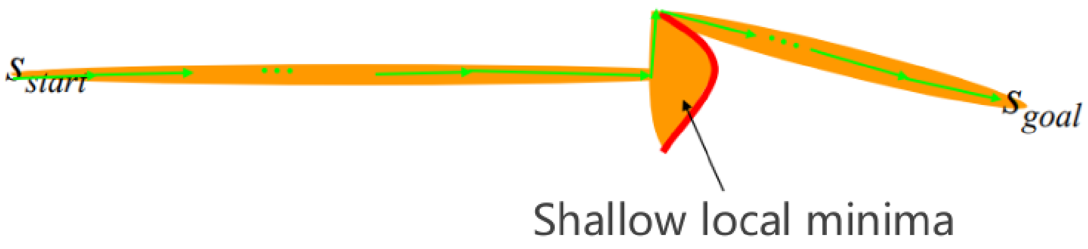


Sub-optimal Solution

What if we intend to use an over-estimate heuristic?



- Weighted A* : Expands states based on f=g+εh,ε>1=bias towards states that are closer to goal.



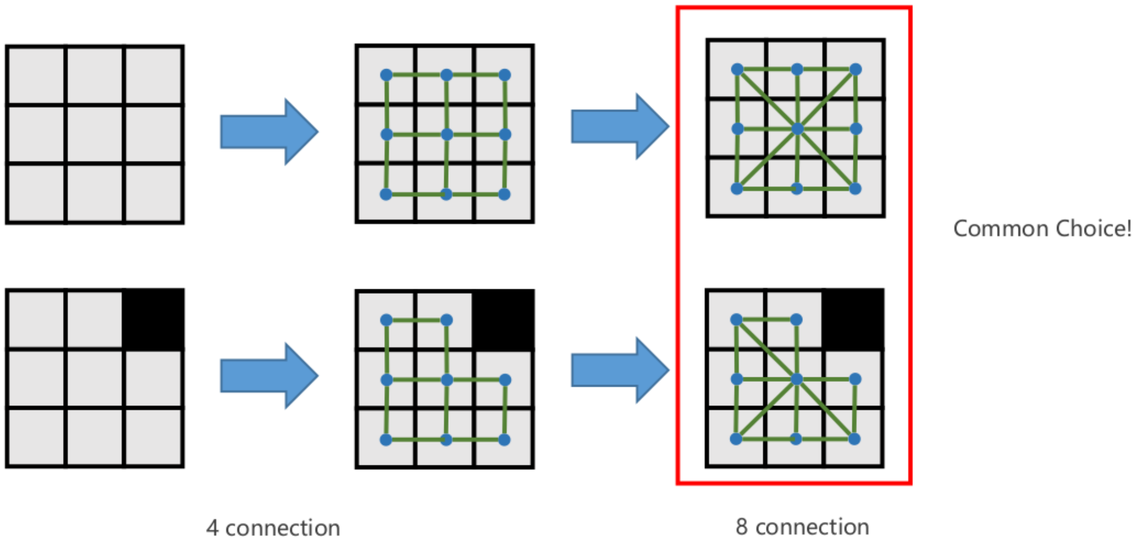
- Weighted A* Search:
  - Optimality vs. speed
  - ε-suboptimal:
    - cost(solution) <= εcost (optimal solution)
  - It can be orders of magnitude faster than A*

Engineering Considerations

Example：Grid-based Path Search

How to represent grids as graphs?

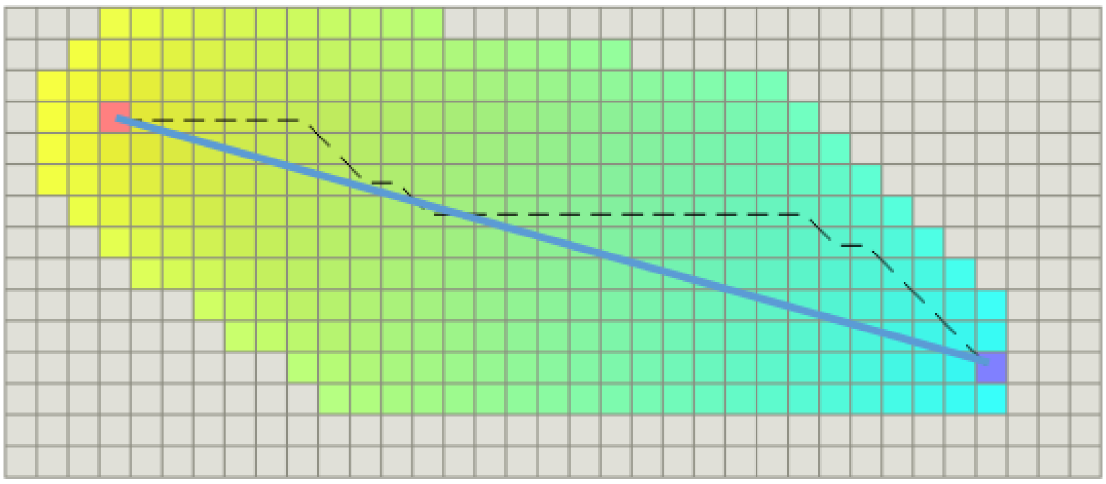
Each cell is a node. Edges connect adjacent cells.



The Best Heuristic

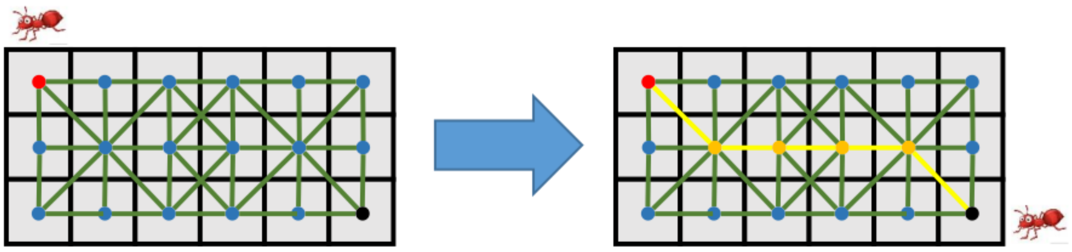
- [x] Is Euclidean distance (L2 norm) admissible?
- [ ] Is Manhattan distance (L1 norm) admissible?
- [x] Is L∞ norm distance admissible?
- [x] Is 0 distance admissible?

- They are useful, but none of them is the best choice, why?
  - Because none of them is tight.Tight means who close they measure the true shortest distance
- Euclidean Heuristic



- Why so many nodes expanded?
  - Because Euclidean distance is far from the truly theoretical optimal solution.
- **How to get the truly theoretical optimal solution?**

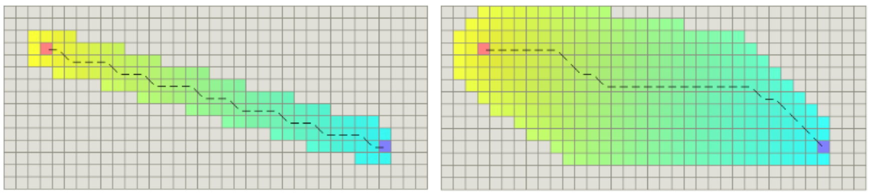
- Fortunately, the grid map is highly structural.



- You don't need to search the path.
- It has the **closed-form solution!**

$$dx = abs(node.x - goal.x)$$
$$dy = abs(node.y - goal.y)$$
$$h = (dx + dy) + (\sqrt{2} - 2) * min(dx, dy)$$

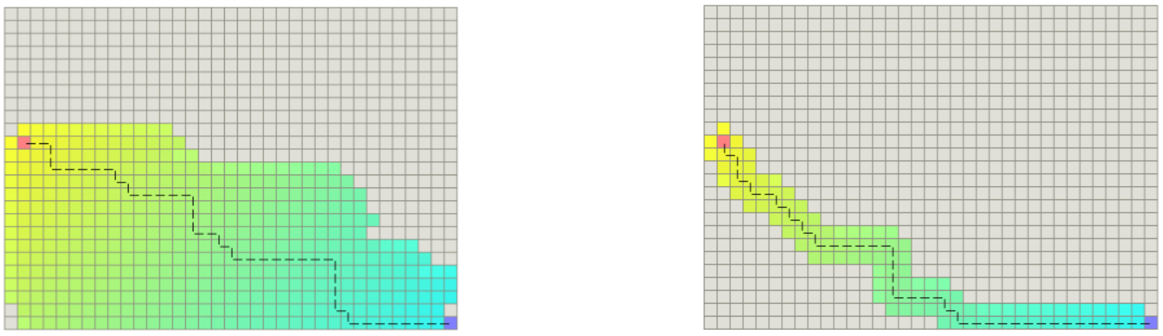
Diagonal Heuristic



Tie Breaker

- Many paths have the same f value.
- No differences among them making them explored by A* equally.
- Manipulate the f value breaks the tie.
- Make same f values differ.
- Interfere h slightly.

$$h = hx(1.0 + p)$$
$$p < \frac{minimun\ cost\ of\ one\ step}{expected\ maximun\ path\ cost}$$



Core idea of tie breaker

Find a preference among same cost paths.

- When nodes having same f, compare their h.
- Add deterministic random numbers to the heuristic or edge costs (A hash of the coordinates).
- Prefer paths that are along the straight line from the starting point to the goal.

$$dx_1 = abs(node_.x - goal_.x)$$
$$dy_1 = abs(node_.y - goal_.y)$$
$$dx_2 = abs(start.x - goal_.x)$$
$$dy_1 = abs(start.y - goal_.y)$$
$$cross = abs(dx_1 * dy_2 - dx_2 * dy_1)$$
$$h = h + crpss * 0.001$$

效果如下图：