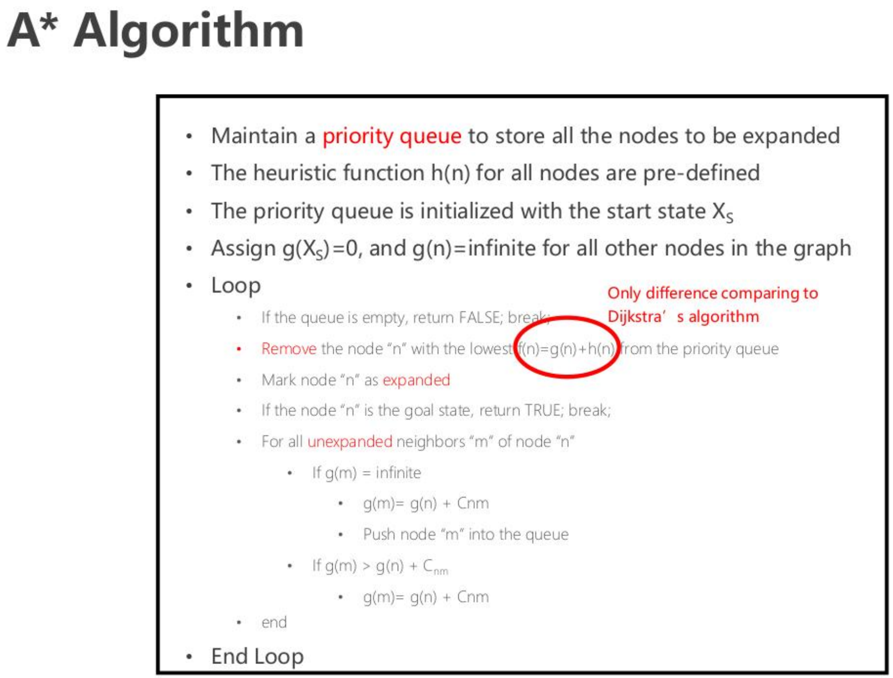
深蓝学院
shenlanxueyuan.com

第八章作业思路提示

主讲人 朱旺旺

A*

1. **预测模型 更新**
   1. A*算法的整体框架可以参考右图

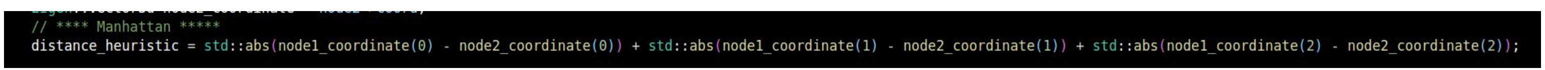


A* Algorithm

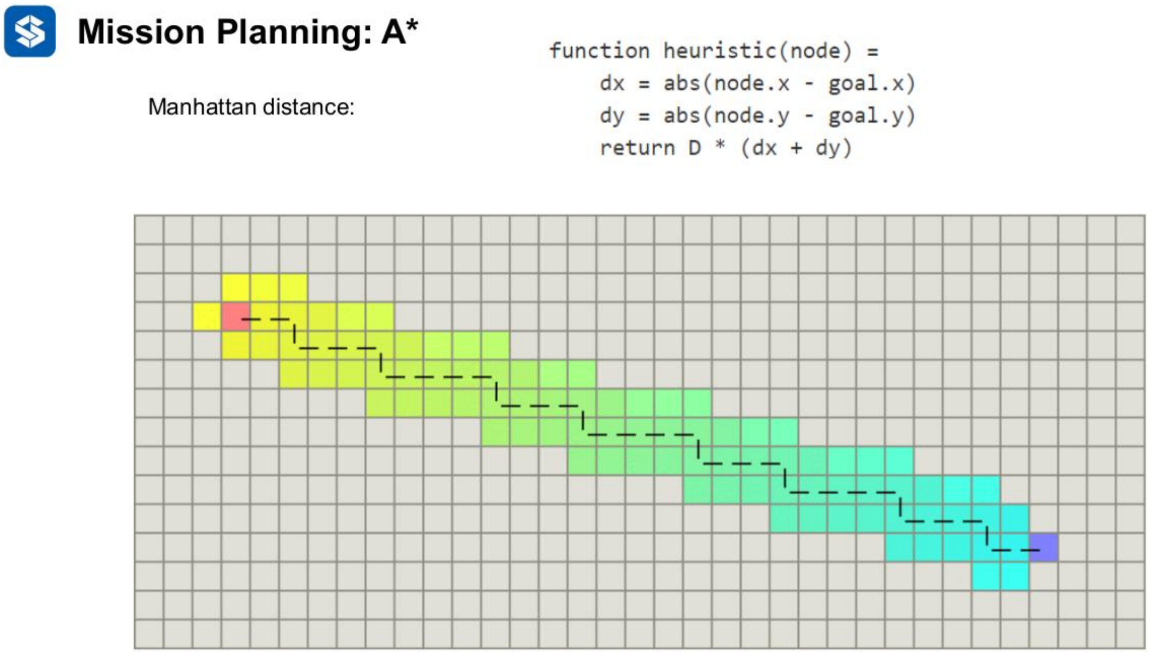
```
• Maintain a priority queue to store all the nodes to be expanded
• The heuristic function h(n) for all nodes are pre-defined
• The priority queue is initialized with the start state $X_S$
• Assign $g(X_S)=0$, and $g(n)$=infinite for all other nodes in the graph
• Loop
    • If the queue is empty, 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 $g(m) > g(n) + C_{nm}$
            • $g(m)= g(n) + Cnm$
    • end
• End Loop
```

Only difference comparing to Dijkstra's algorithm

A*



```
Eigen::Vector3d node2_coordinate = node2->coord;
// **** Manhattan *****
distance_heuristic = std::abs(node1_coordinate(0) - node2_coordinate(0)) + std::abs(node1_coordinate(1) - node2_coordinate(1)) + std::abs(node1_coordinate(2) - node2_coordinate(2));
```

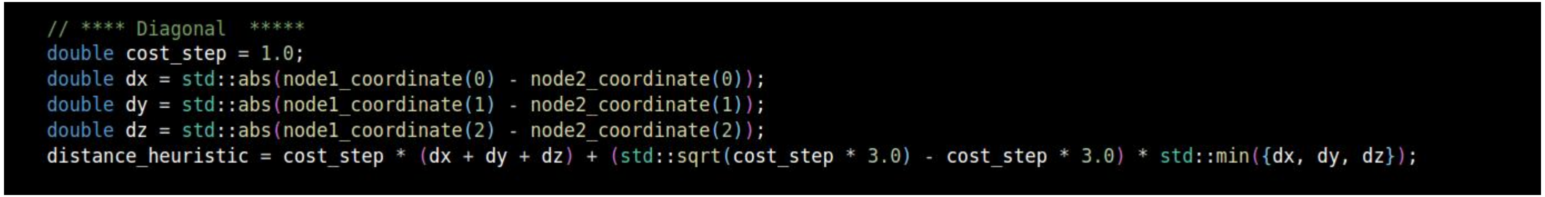


Mission Planning: A*

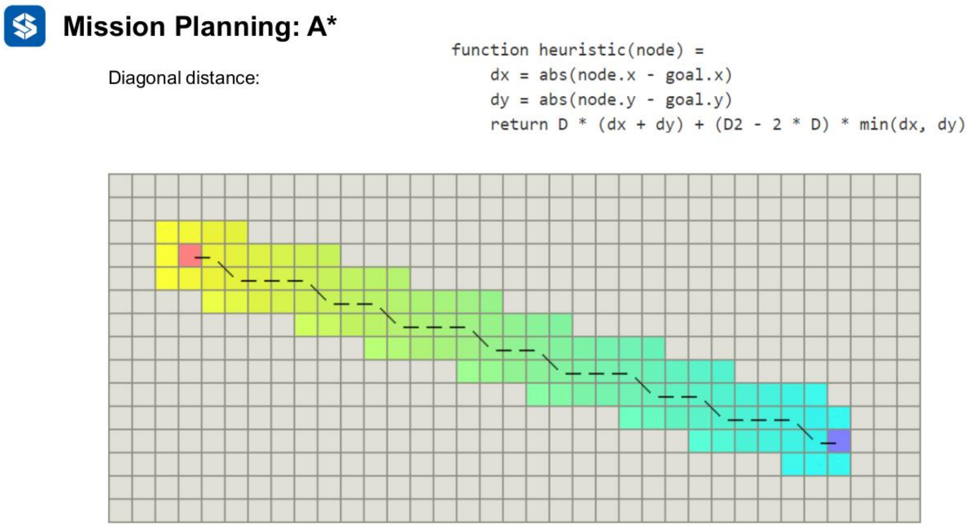
Manhattan distance:

```
function heuristic(node) =
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * (dx + dy)
```

A*



```
// **** Diagonal  *****
double cost_step = 1.0;
double dx = std::abs(node1_coordinate(0) - node2_coordinate(0));
double dy = std::abs(node1_coordinate(1) - node2_coordinate(1));
double dz = std::abs(node1_coordinate(2) - node2_coordinate(2));
distance_heuristic = cost_step * (dx + dy + dz) + (std::sqrt(cost_step * 3.0) - cost_step * 3.0) * std::min({dx, dy, dz});
```

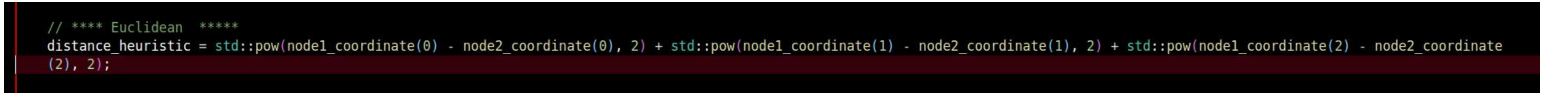


Mission Planning: A*

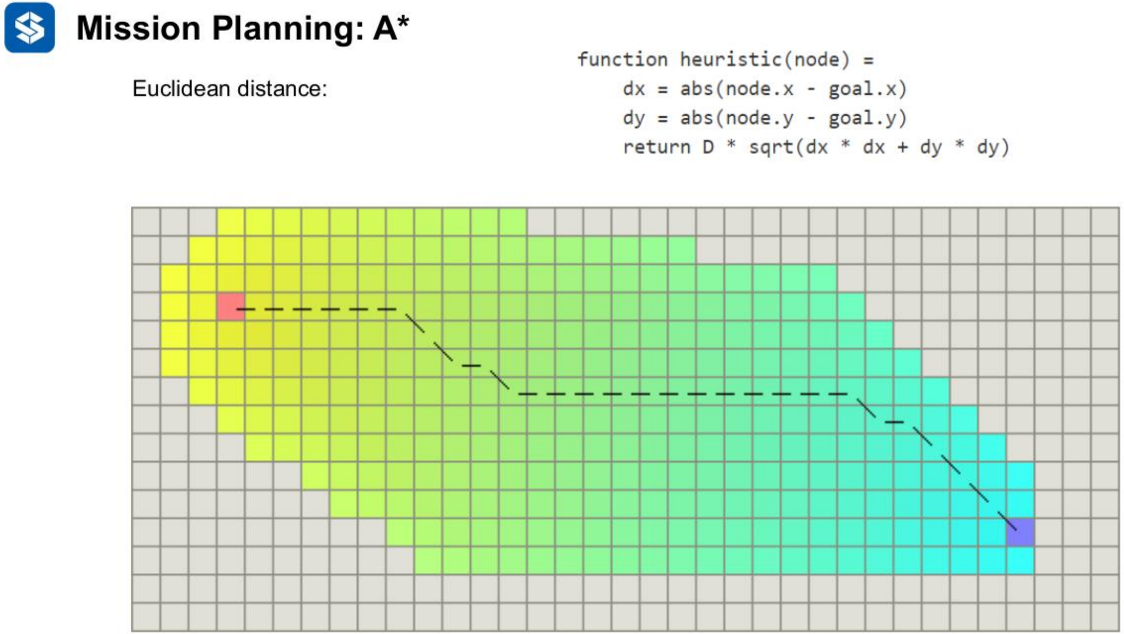
Diagonal distance:

```
function heuristic(node) =
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * (dx + dy) + (D2 - 2 * D) * min(dx, dy)
```

A*



```
// **** Euclidean  *****
distance_heuristic = std::pow(node1_coordinate(0) - node2_coordinate(0), 2) + std::pow(node1_coordinate(1) - node2_coordinate(1), 2) + std::pow(node1_coordinate(2) - node2_coordinate
(2), 2);
```



Mission Planning: A*

Euclidean distance:

```
function heuristic(node) =
    dx = abs(node.x - goal.x)
    dy = abs(node.y - goal.y)
    return D * sqrt(dx * dx + dy * dy)
```

在线问答

Q&A

感谢各位聆听！

Thanks for Listening