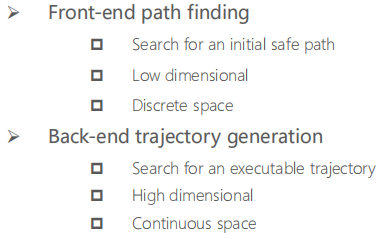
1. **主要结果**
2. 研究内容：移动机器人的运动规划。这门课的主讲老师，就是文章（TGK-planner）的通讯作者。课程内容与文章内容有很高的相关性。



1. 学习路线

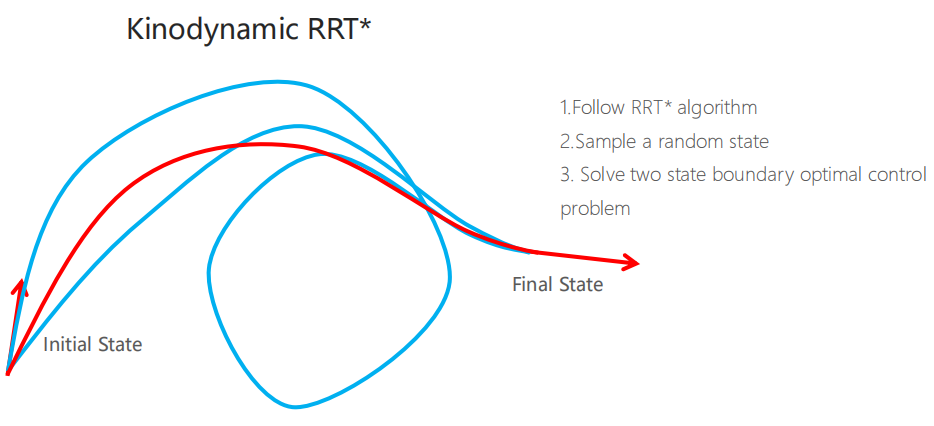


前端路径发现（搜索）：在一个相对低维的通常是离散的状态空间中，让机器人搜索一个初始的（解的质量不是特别好的）path,没有高维的信息。

后端的轨迹优化（生成）：在高维的连续空间中执行的（寻找一个可执行的轨迹），要用到很多的优化技巧。

**前端：Path finding**

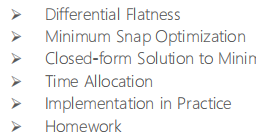
1. 基于搜索的Path finding
2. 图的基本概念：机器人为了在环境中进行导航，会用到很多种类型的地图。
3. Dijkstra and A\*算法
4. Jump Point Search(跳点算法，实际上是A\*的改良版本)
5. 基于采样的Path finding
6. Probabilistic Road Map概率路线图:经典的随机采样的路径规划方法
7. Rapidly-exploring Random Tree随机扩展快速采样的随机树算法（RRT）
8. Optimal Sampling-based Methods 具有渐进最优性的改良版本,称为RRT\*
9. Kinodynamic Path finding(满足动力学要求的轨迹规划，开始考虑动力学模型)
10. State-State Boundary Value Optimal Control Problem满足两点边界值的最优控制问题
11. State Lattice Search状态图的搜索（其实是高维的Dijkstra或A\*）
12. Kinodynamic RRT\* 引入动力学的RRT\*



1. Hybrid A\* 混合A\*算法

**后端：Trajectory Generation**

1. Minimum Snap 轨迹生成,再交给移动机器人执行



**2）Soft and Hard Constrained Trajectory Optimization**

A) 软约束轨迹优化

B) 硬约束轨迹优化

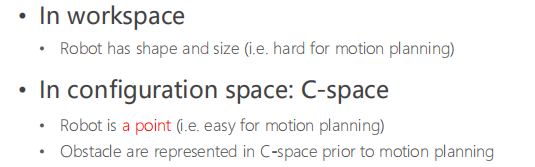
1. **本周具体学习内容(学习笔记，供复习备查）**

1）机器人规划的配置空间：

Configuration Space:对机器人上所有点的位置的描述。在配置空间中做规划：将工作空间中的planning变成配置空间中的planning。

在配置空间中要注意对障碍物的处理：C-obstacle；对障碍物和机器人的体积做膨胀。 处理过后，运动规划就是在C-free空间中找到一条从开始点到目标点的路径。





1. 图搜索算法的总体框架：
2. 总是要维护一个容器：这个容器装载着全部将来要去访问的节点。
3. 这个容器一开始是空的，最先放入的是起始点。
4. 最后进入一个循环。各种搜索算法其实都可以归结在这个循环框架里

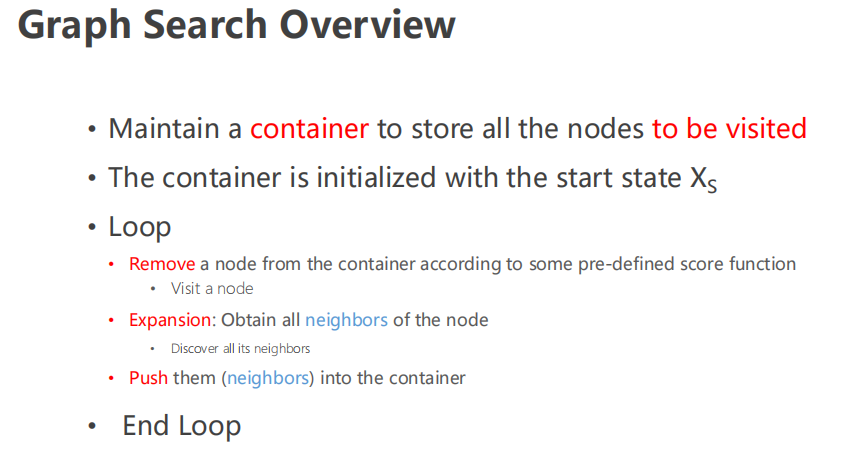
循环分为三步：

1）弹出一个节点（根据一项指标、一种目的）Remove

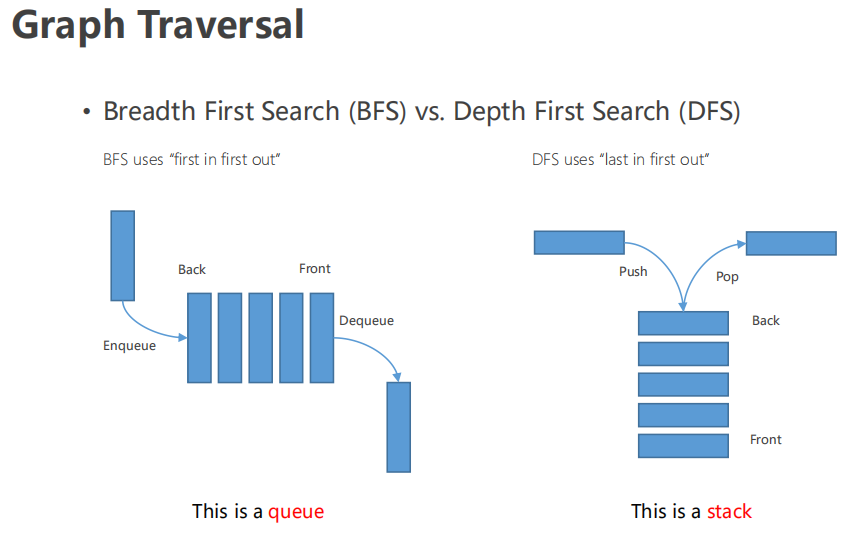
2）扩展这个节点所有的邻居Expansion

3）对于所有扩展出来邻居节点，再重新装入容器中。Push

循环持续到：容器中没有任何节点可以被弹出为止。



图的遍历：



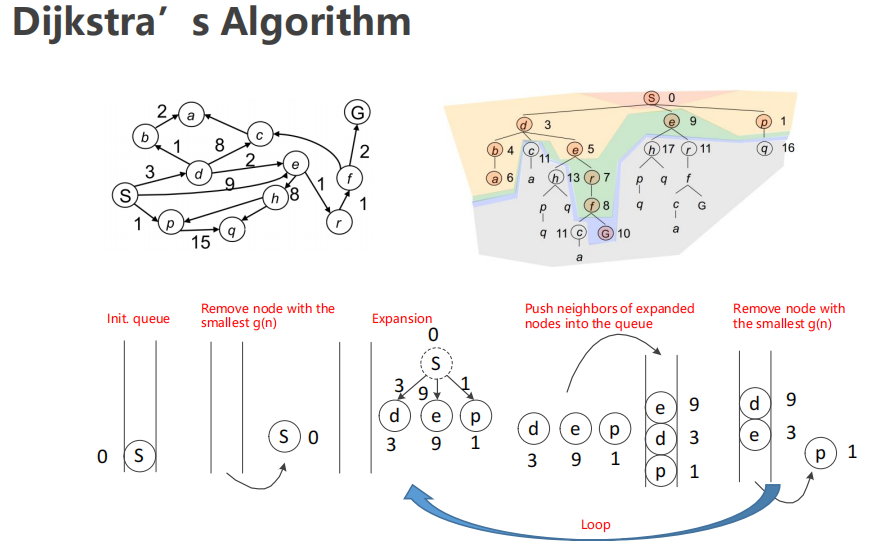
DFS弹出容器中最深的节点，在每次扩展时把每个可能的分支走到底。

BFS弹出容器中最浅的节点，所以BFS是分层遍历的。

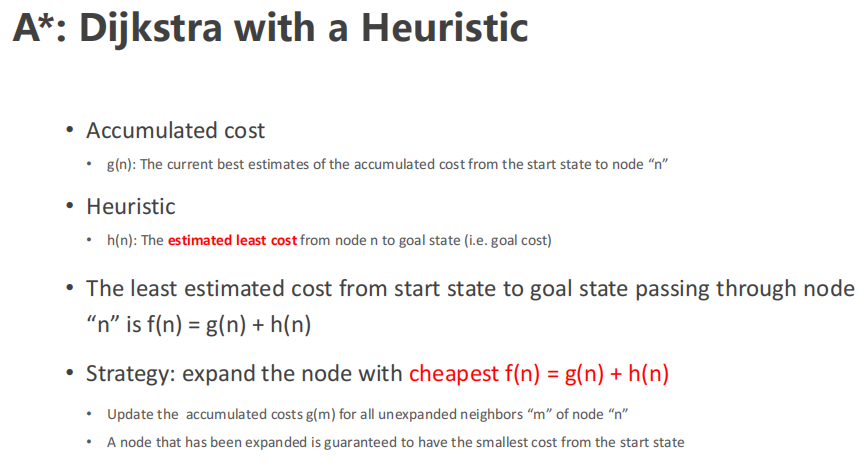
贪心算法：每次都选择局部最优。

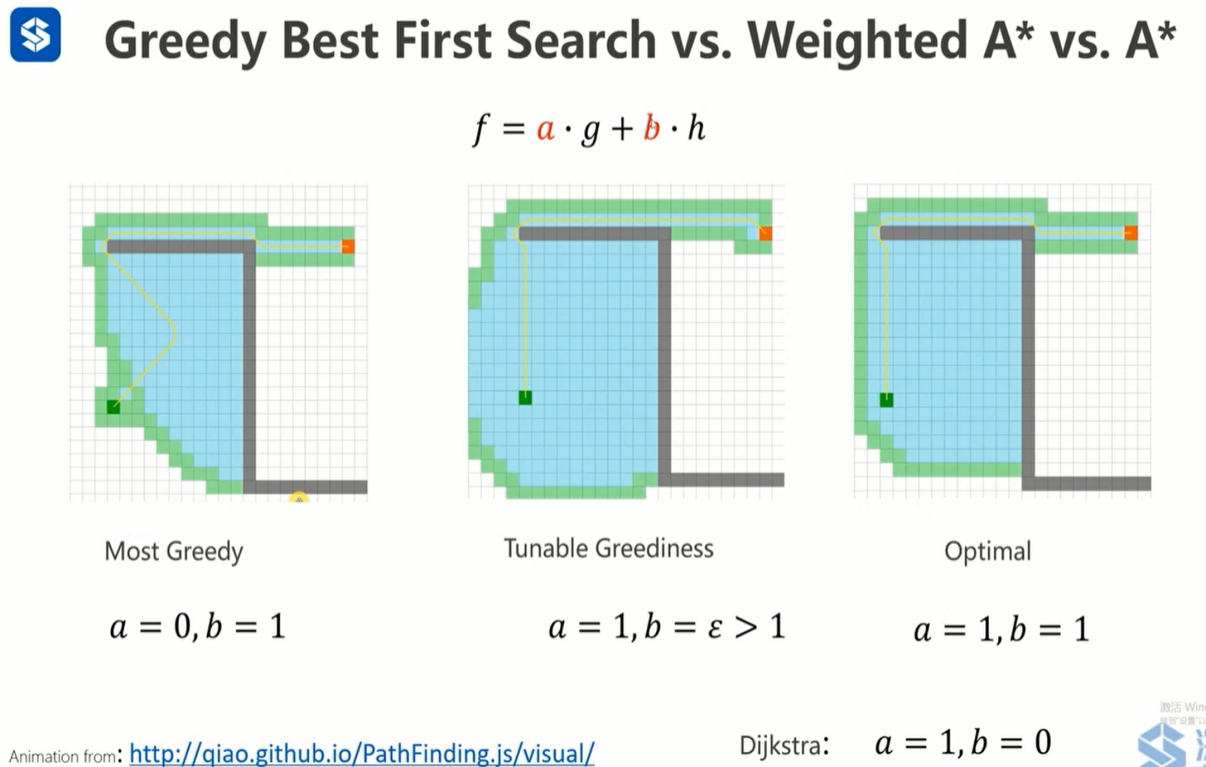
用BFS搜索的前提是图中每条边的权重是相同的，比如都是1.

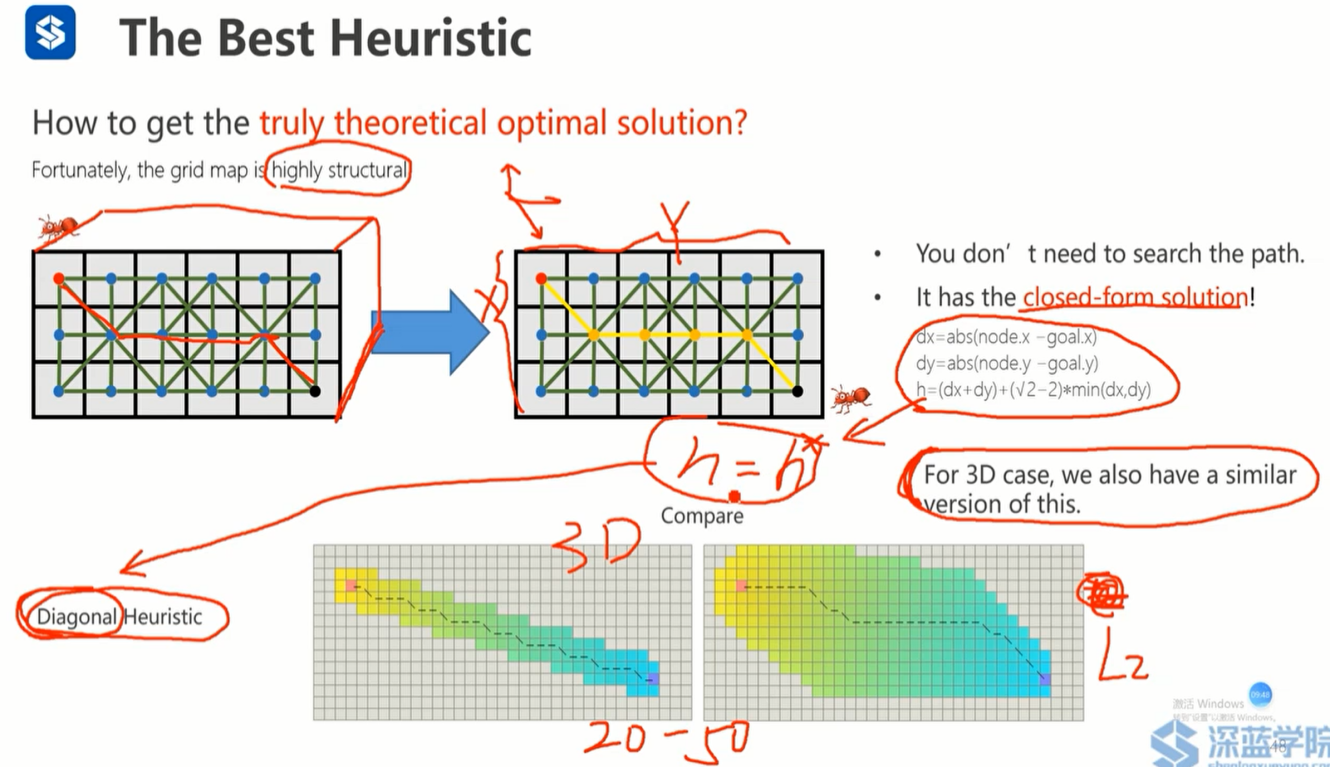
Dijkstra算法：弹出的节点n需具有最小的cost: g(n):从起点到n节点的代价总和最小。如果所有边的权重都是1的话，Dijkstra算法等同于BFS。



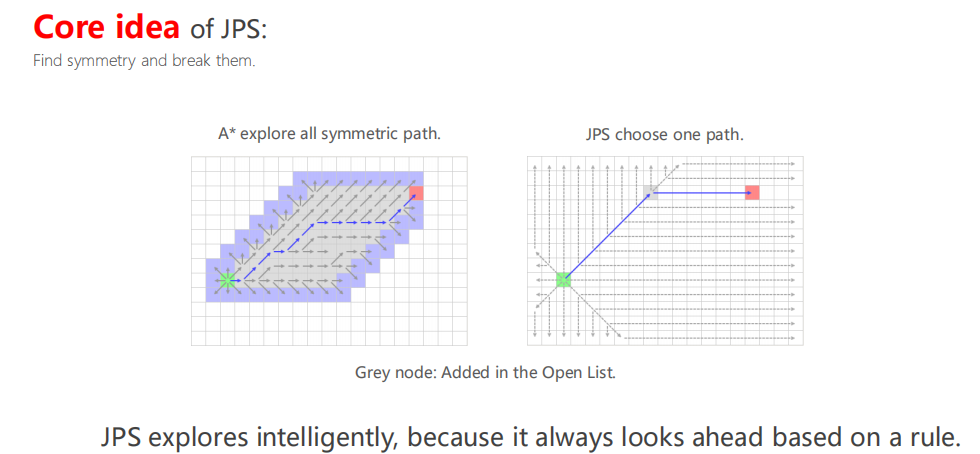
**A\*算法：Dijkstra算法Cost g(n) + 启发式h(n)**



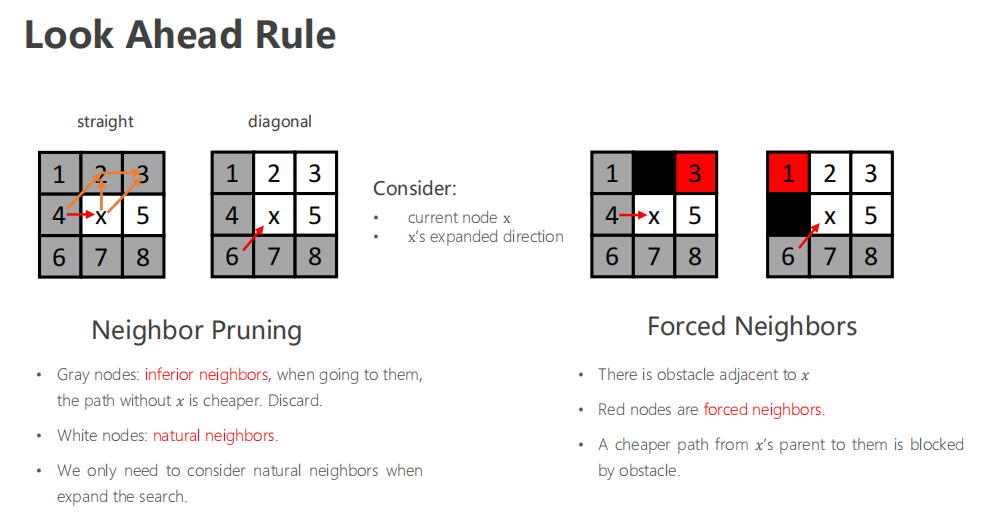




**Jump Point Search跳点搜索：打破对称性**



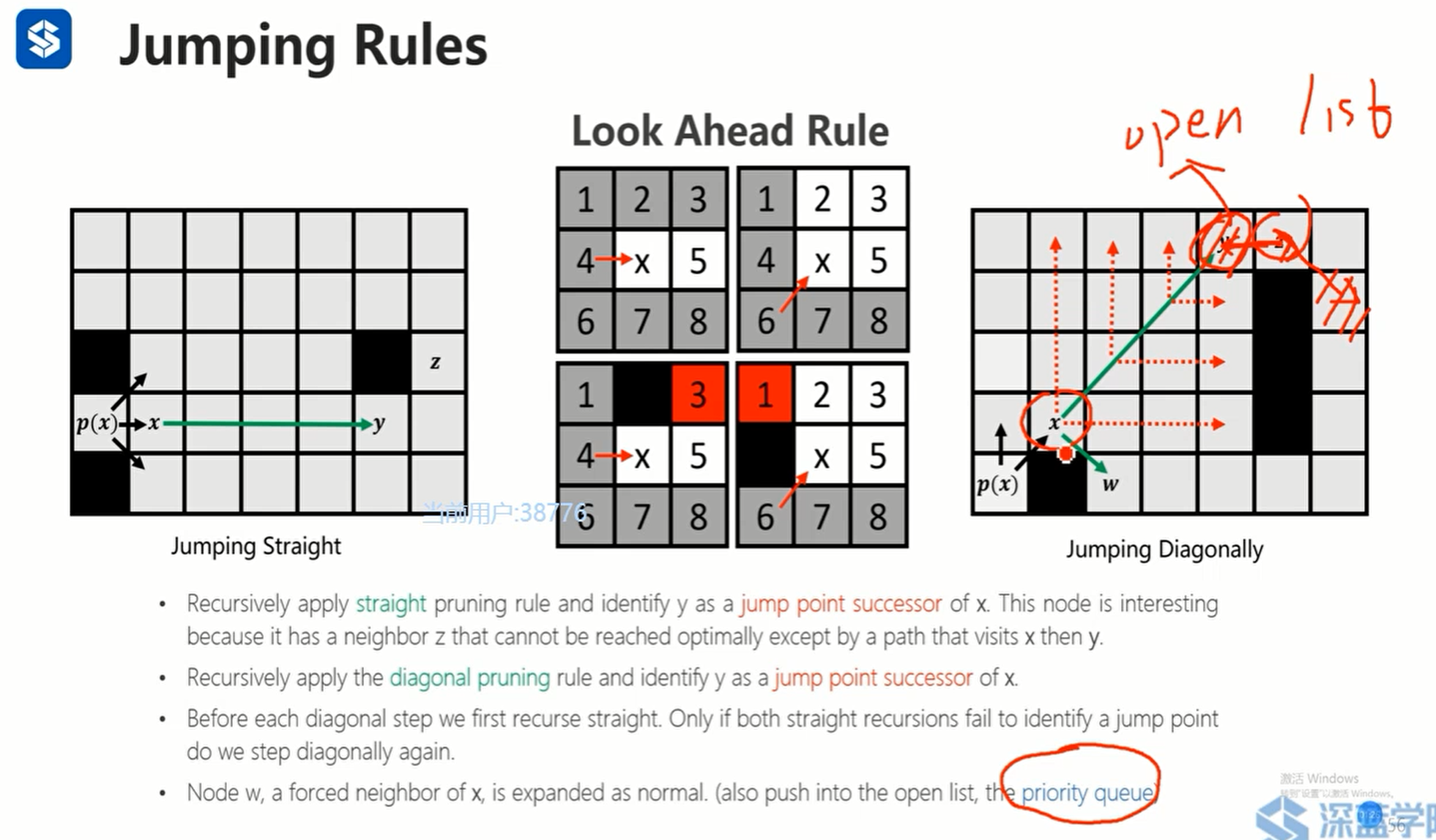
Look Ahead Rule: (Straight pruning rule)如果一个节点可以通过x的父亲到达（在cost小于等于的前提下），那就没有必要通过x到达。(跳点，打破对称性）



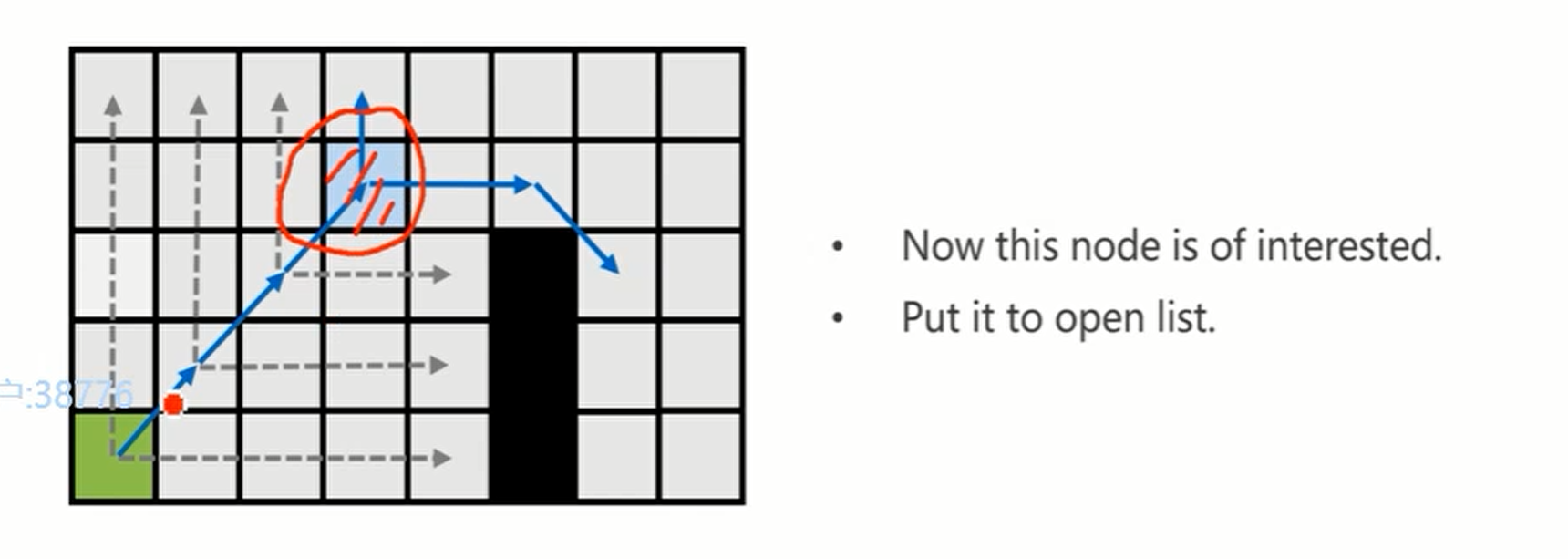
上方左图红色箭头代表父节点的来源。（分为直线来源和对角线来源）

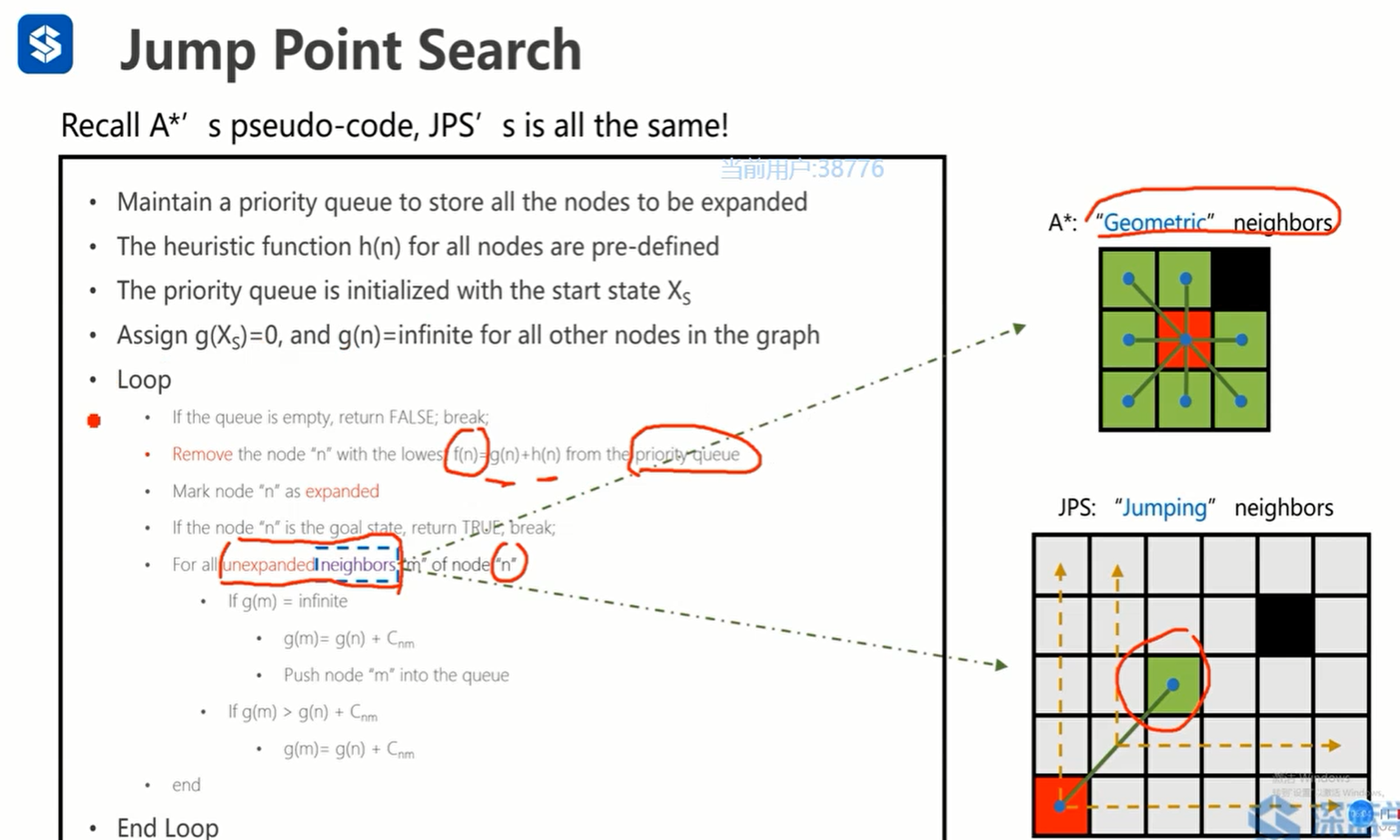
当出现障碍物邻居的时候，就会出现Forced Neighbors.

优先考虑水平和垂直方向的跳跃。

优先考虑水平和垂直方向的跳跃。（地图边界等同于障碍物处理，在右三图中，水平和垂直方向的跳跃都会失败，因为都会碰到障碍物）于是进行对角线跳跃，到y节点。y节点是一个关键节点。

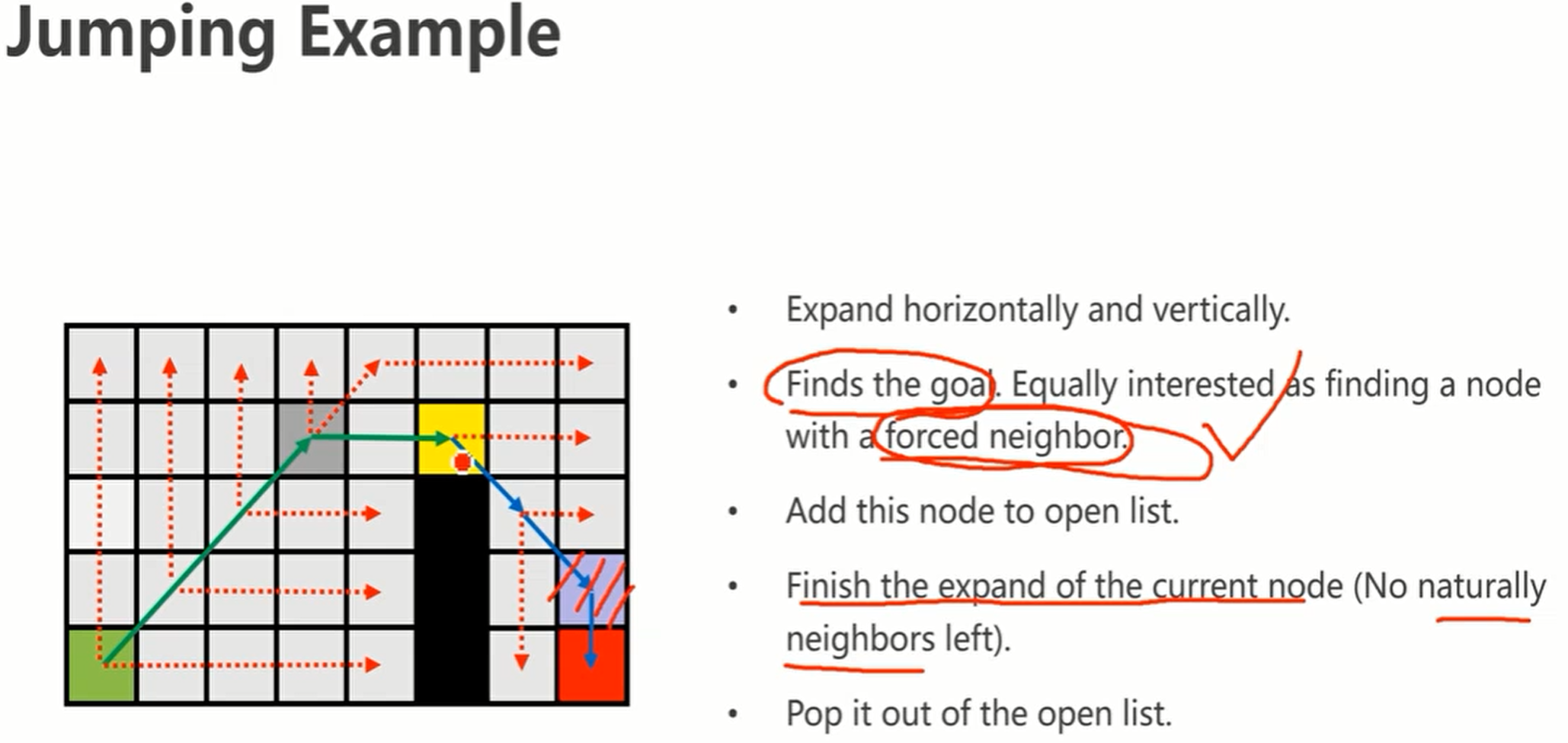
将Forced neighbors(expanded)加入优先级队列中(push),将父节点x弹出(Remove).

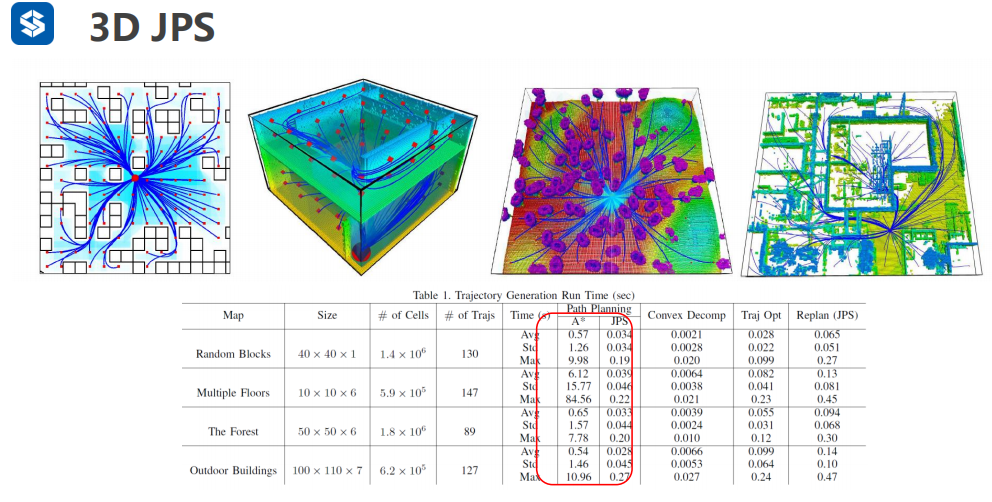
不会有折线跳跃，每一次的跳跃一定是直线的。



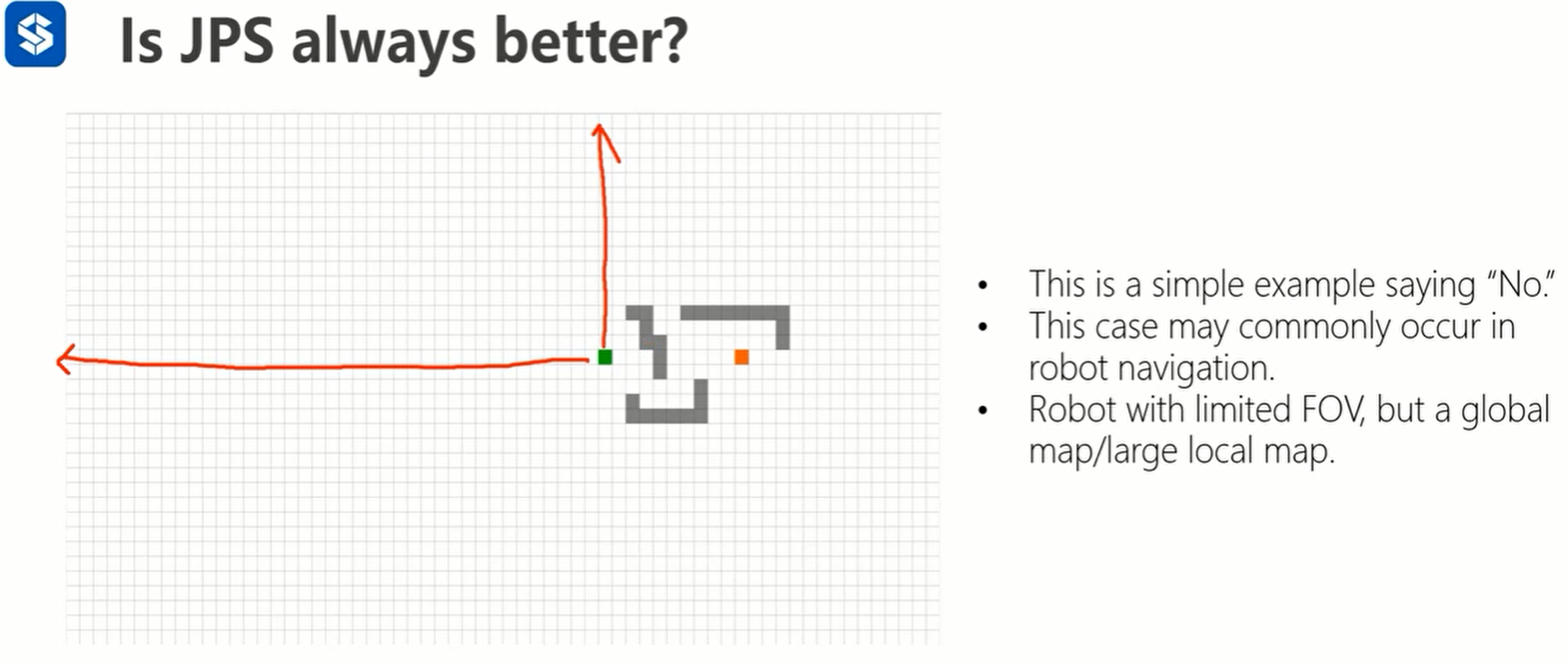
A\*的代码与JPS的代码几乎一致，唯一的区别在于怎么去找n节点未扩展的邻居。

A\*的邻居找的是几何上能达到的邻居（除去障碍物的邻居节点）。但JPS找的邻居是根据跳点规则和look Ahead规则找到的跳点邻居。 A\*在扩展几何上紧密连接的邻居，而JPS在地图上空阔区域进行大范围的跳跃。

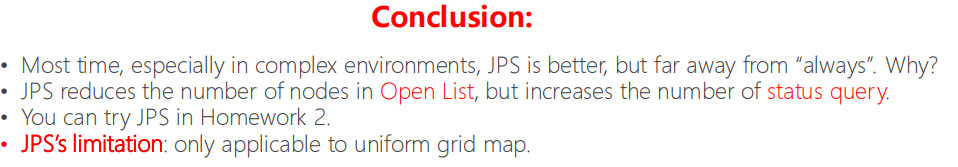




在大多数情况下，尤其是复杂环境中，JPS比A\*更快。因为它减少了优先级队列中扩展的节点数量，在队列中对节点进行的操作次数也会变少，从而节省计算量；但是它增加了环境中碰撞查询（collision query）的次数,积少成多也会是一笔计算开销。



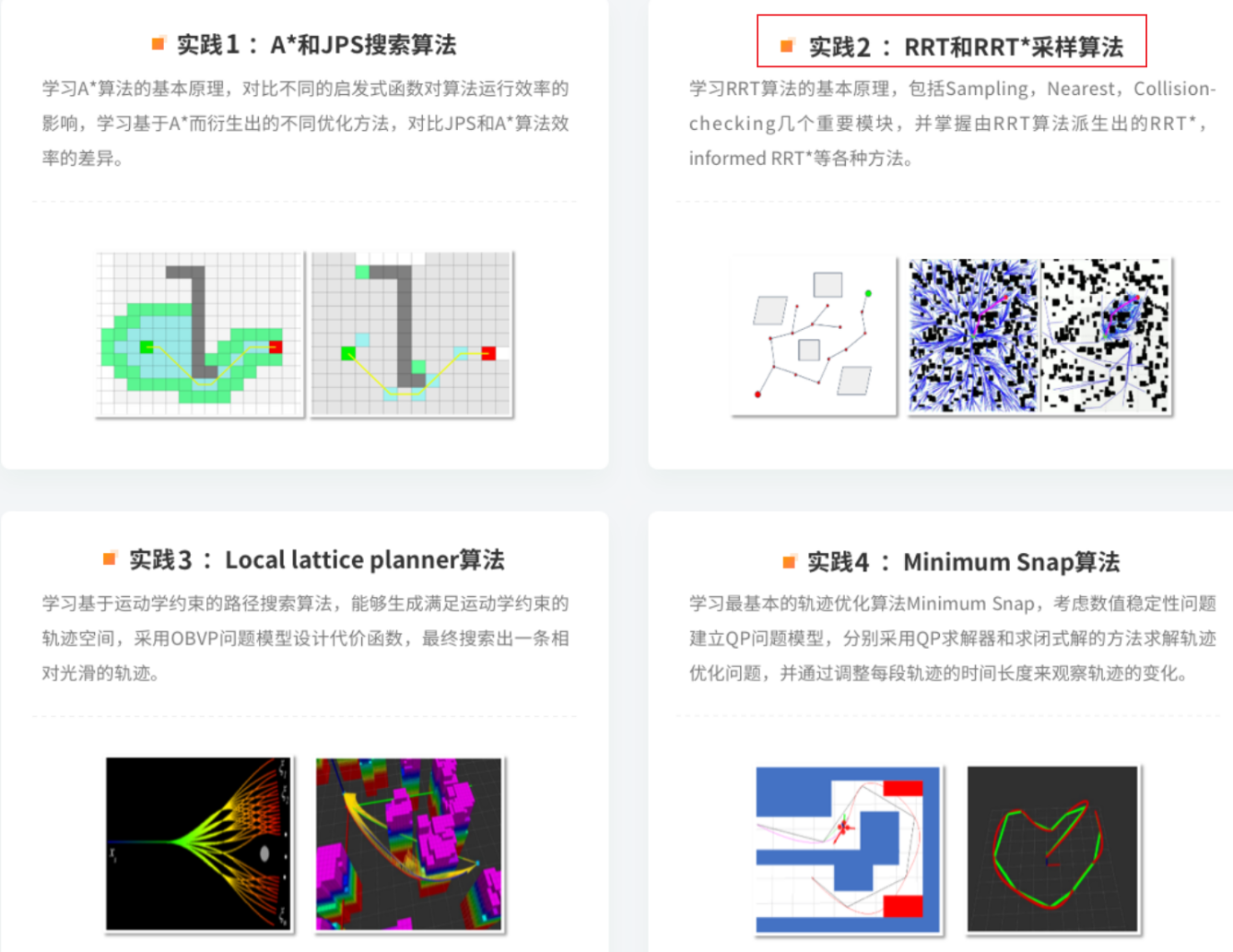
JPS的限制：只适用于规则的栅格地图。



基于搜索的路径规划方法建立在地图的基础之上，通过对图的搜索，规划出从点A到点B的最优路径。

基于采样的路径规划：不需要遍历从点A到点B的所有网格，而是通过在空间中随机洒点，通过一些线段把这些点连接起来的方式，构建一颗树（图），来表示空间。尤其适用于复杂空间，适用于三维或更高维的情况。

1. **结论**
2. 目前学完了前两章，了解了总体框架和学习路线，学习了基于搜索的和基于采样的路径规划方法，实践了A\*和JPS算法。因为课程的作业要求ROS下的c++实现，所以对于理解开源代码也有好处。



1. 下一步工作思路及计划：后续重点关注RRT\*算法和kino RRT\*算法，并进行编程实践。基本上学完前6章，应该能把动力学规划这一块弄懂了。



**Motion planning当前的研究难点：**

kinodynamic path searching。纯geometric path search的planning前端已经是几年前的事情了，要让无人机飞行动态性能好（速度快，运动smooth，节省能量），现在普遍需要的是on the flight的onboard kinodynamic path finding，之后再接上后端的轨迹优化。kinodynamic path searching解空间维度高，问题复杂度大，引申出来的问题包括：如何快速求解two point bounda value problem, 如何针对你的系统设计高效的heuristic，如何保证每次replan前后的motion consistency。最简单的测试指标，复杂环境里，你能让无人机飞多快？环境越复杂，飞的越快，就越NB。

*---转自 高飞*