深蓝学院
shenlanxueyuan.com

Autonomous Valet Parking（AVP）Theory and Practice
自主代客泊车理论与实践

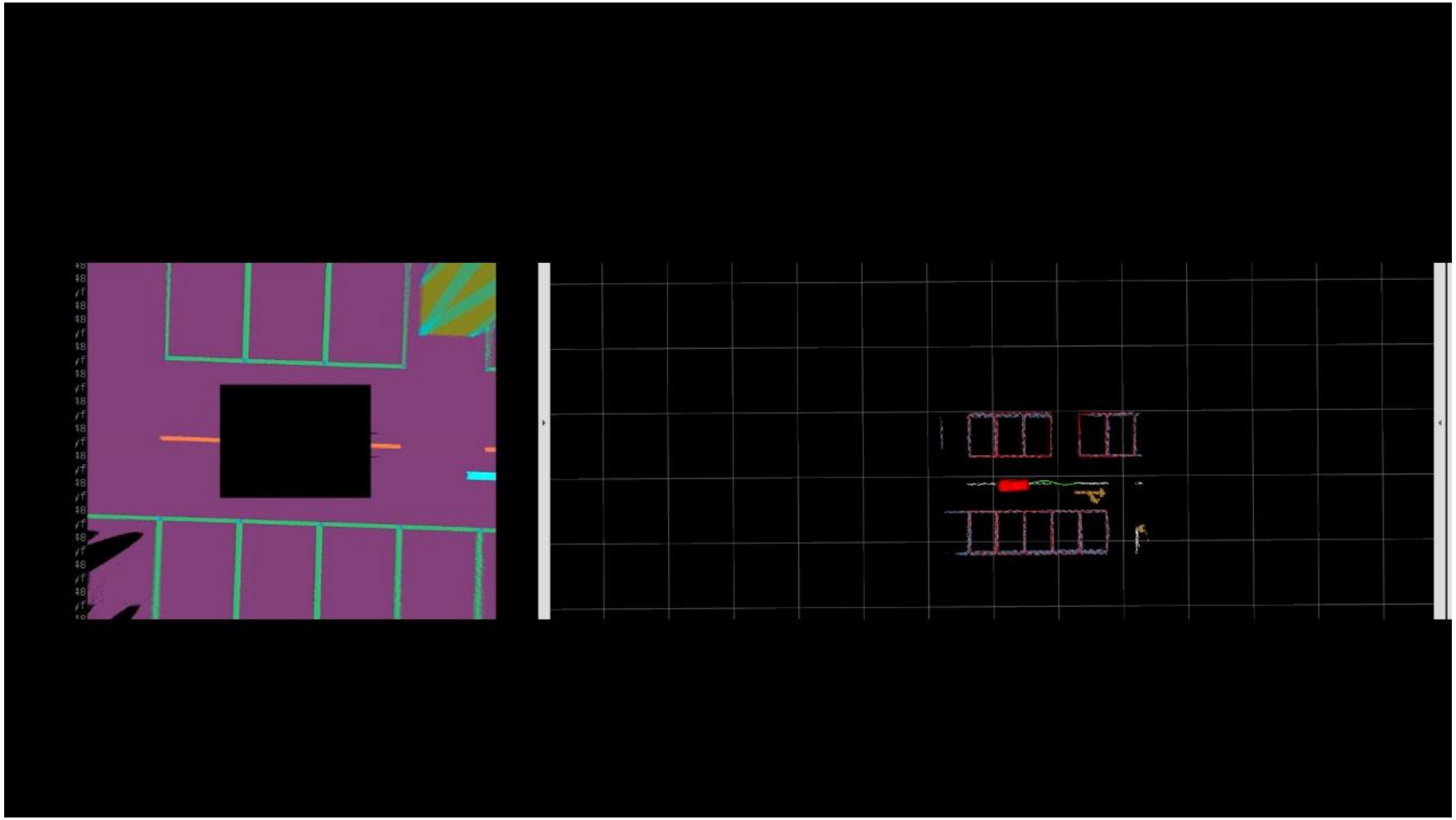
Lecture 7：Path Planning

Lecturer Tong QIN

Associate Professor
Shanghai Jiao Tong University
Global Institute of Future Technology
qintong@sjtu.edu.cn

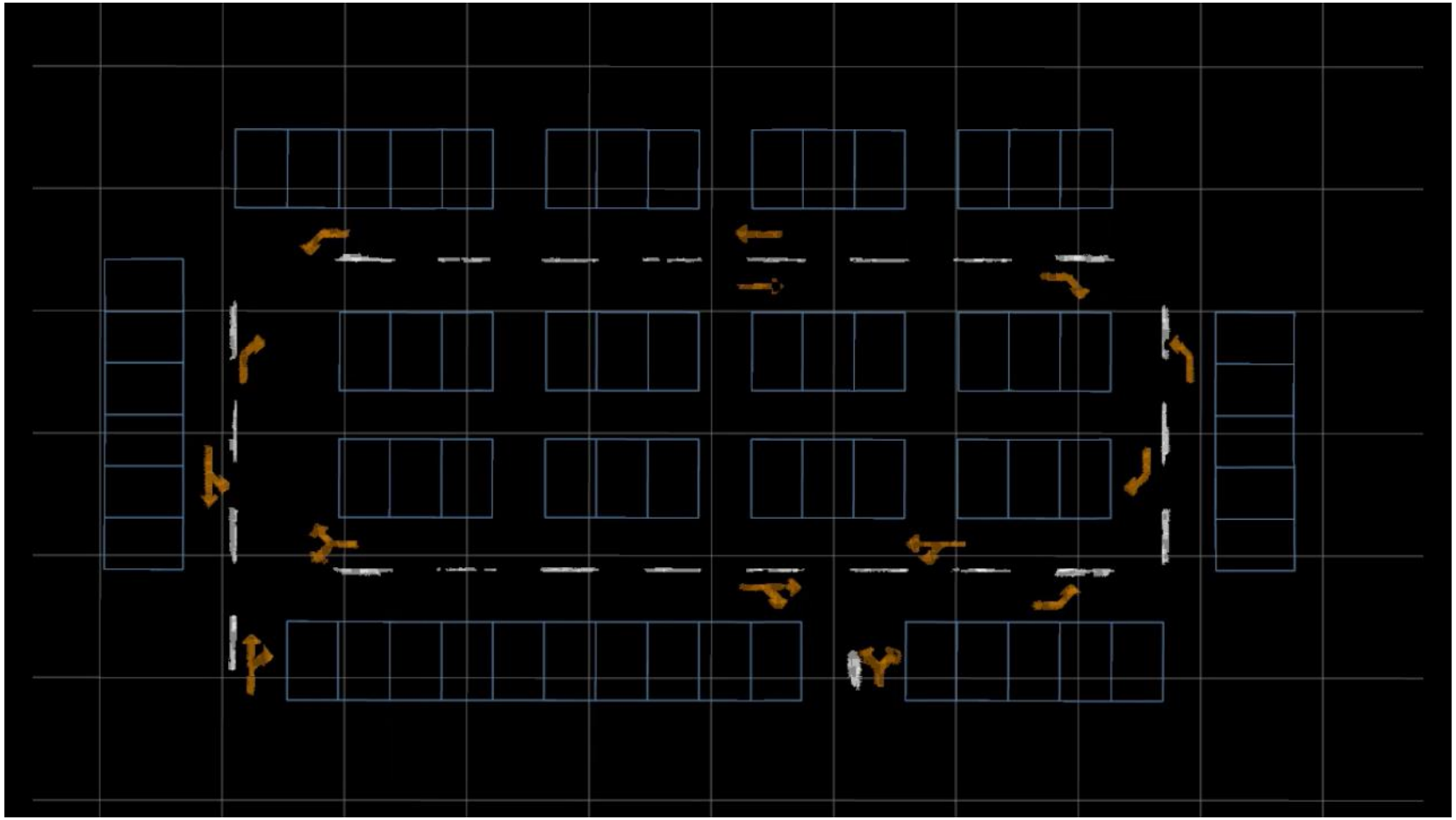
Recall

- Lecture 5 Parking Map Construction



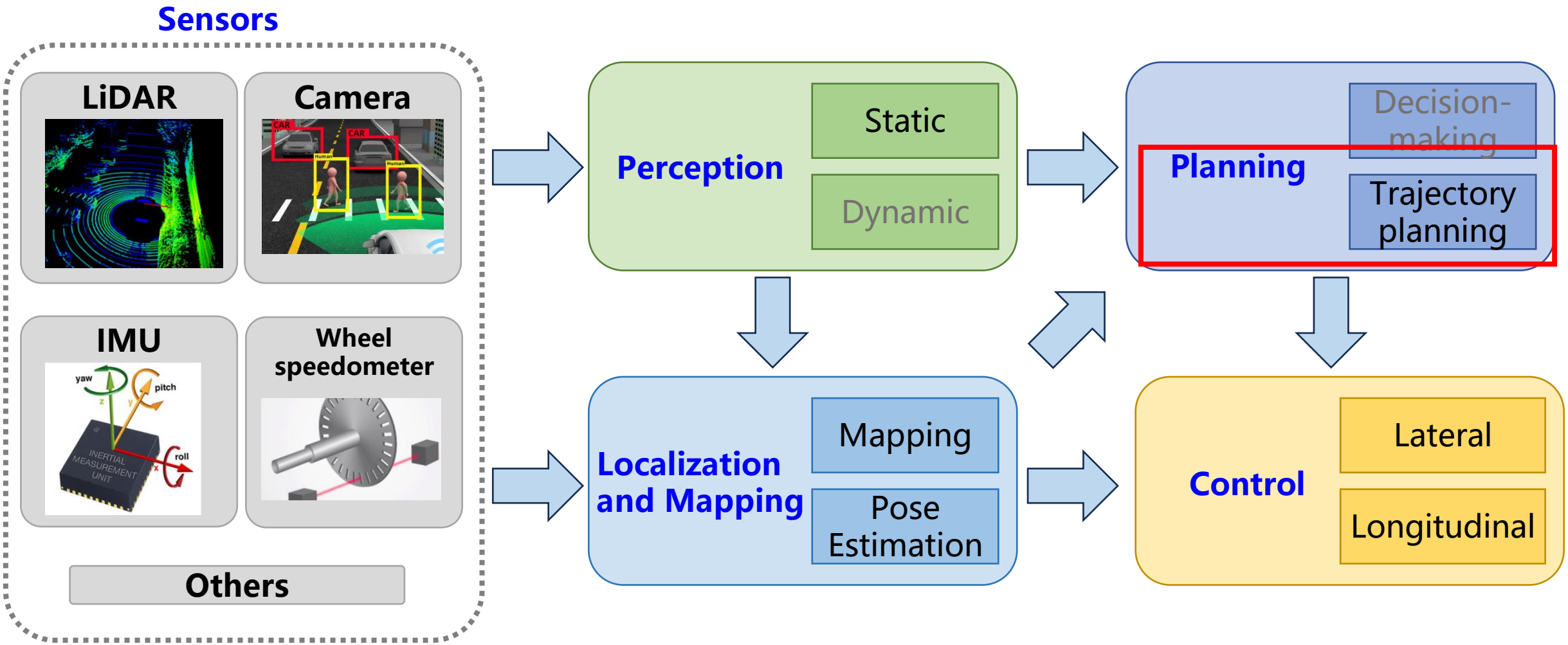
Recall

- Lecture 6 Semantic Localization

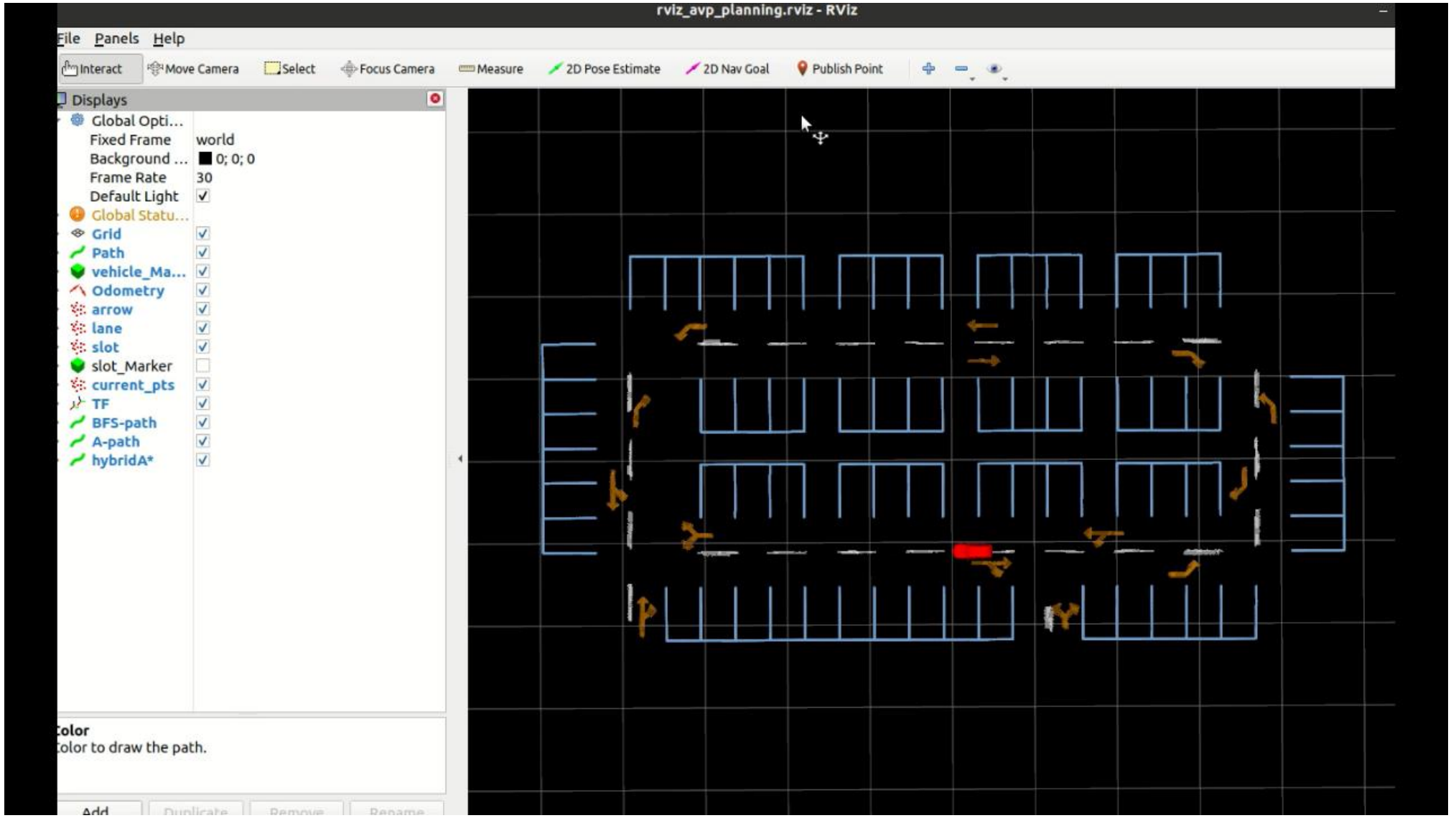


Content

AVP Architecture



Path Planning

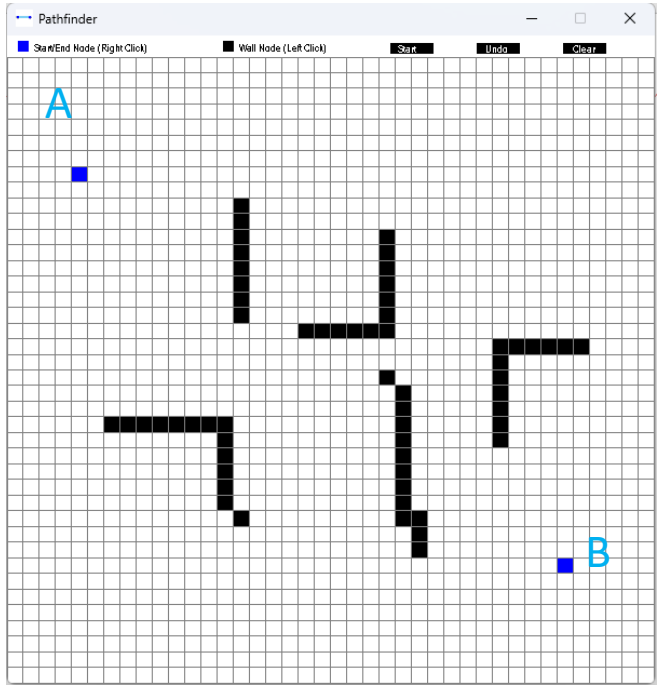


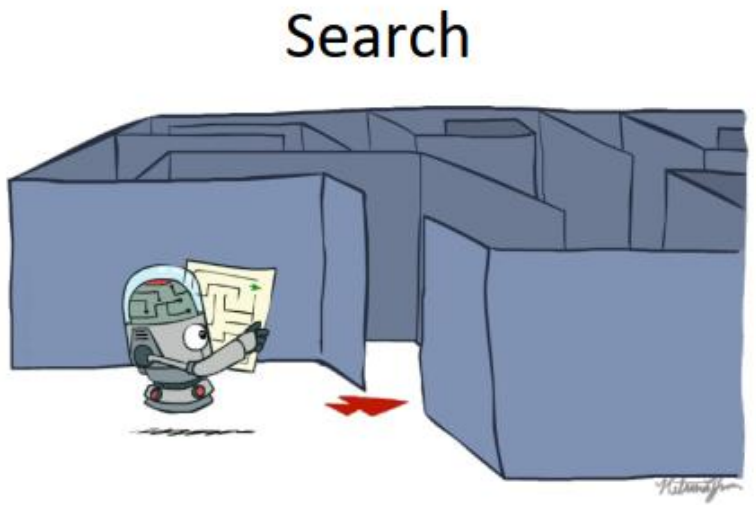
BFS
A*
Hybrid A*

Content

Path Planning and Search

Maze (迷宫)





How to find path from A to B?

How to find the shortest path from A to B?

Path Planning and Search

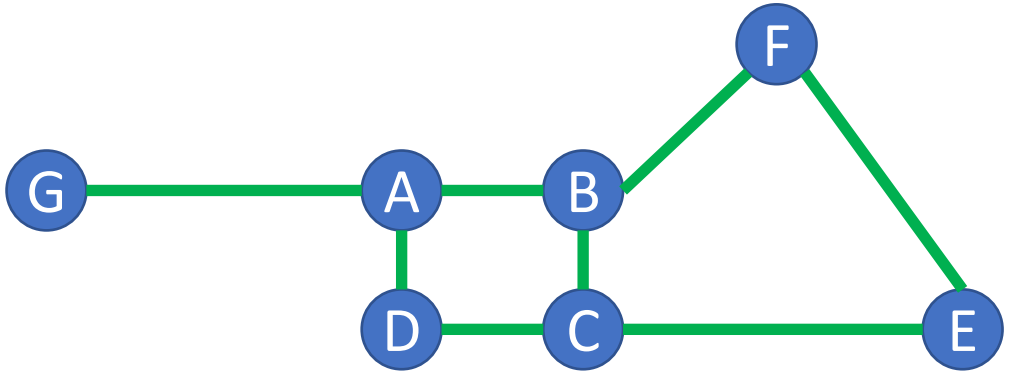
- **Graph and Search Method**
- **Uninformed (Blind) Search Methods**
  - BFS Breadth-First Search
  - DFS Depth-First Search
  - Dijkstra
- **Informed (Heuristic) Search Methods**
  - A algorithm
  - A* algorithm

启发的

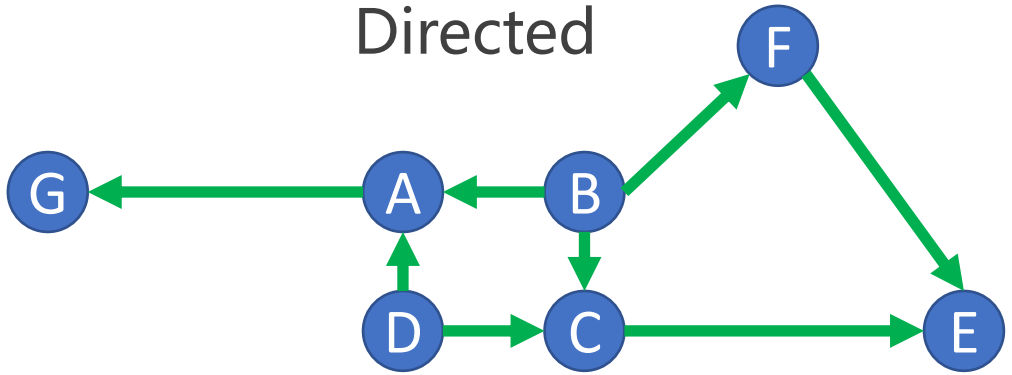
Path Planning and Search

Graphs

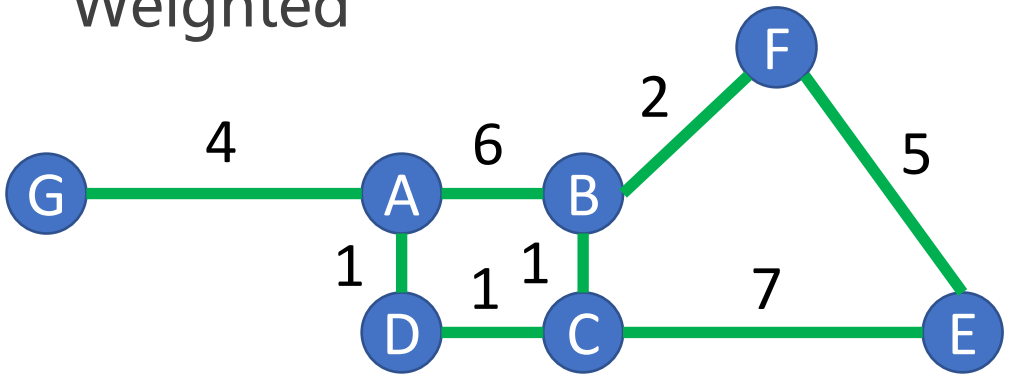
Graphs have nodes and edges



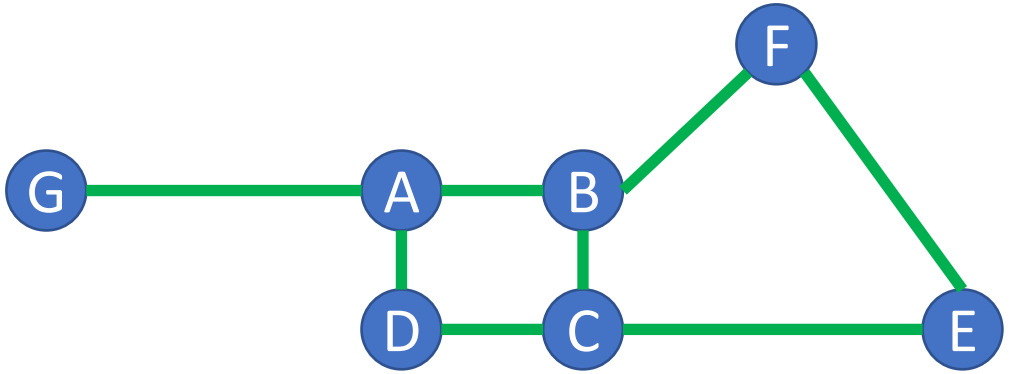
Directed



Weighted

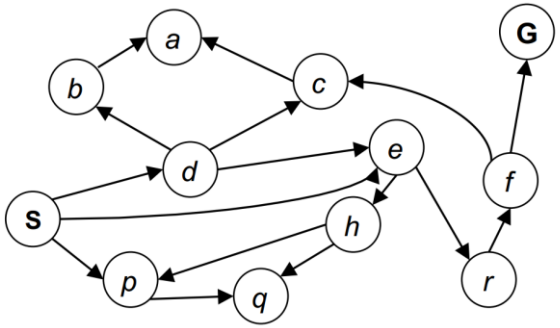


Undirected

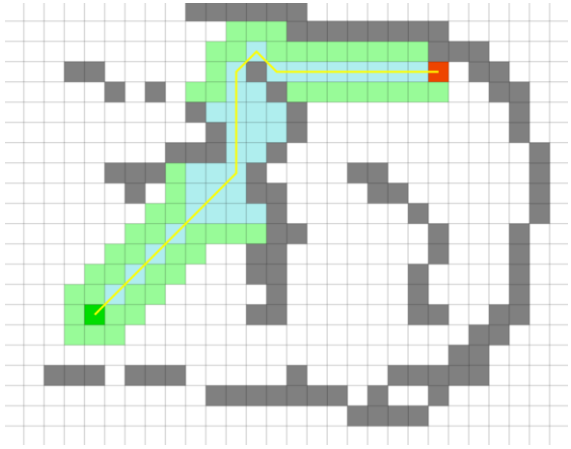


Path Planning and Search

- State space graph: a mathematical representation of a search algorithm
  - For every search problem, there's a corresponding state space graph
  - Connectivity between nodes in the graph is represented by (directed or undirected) edges



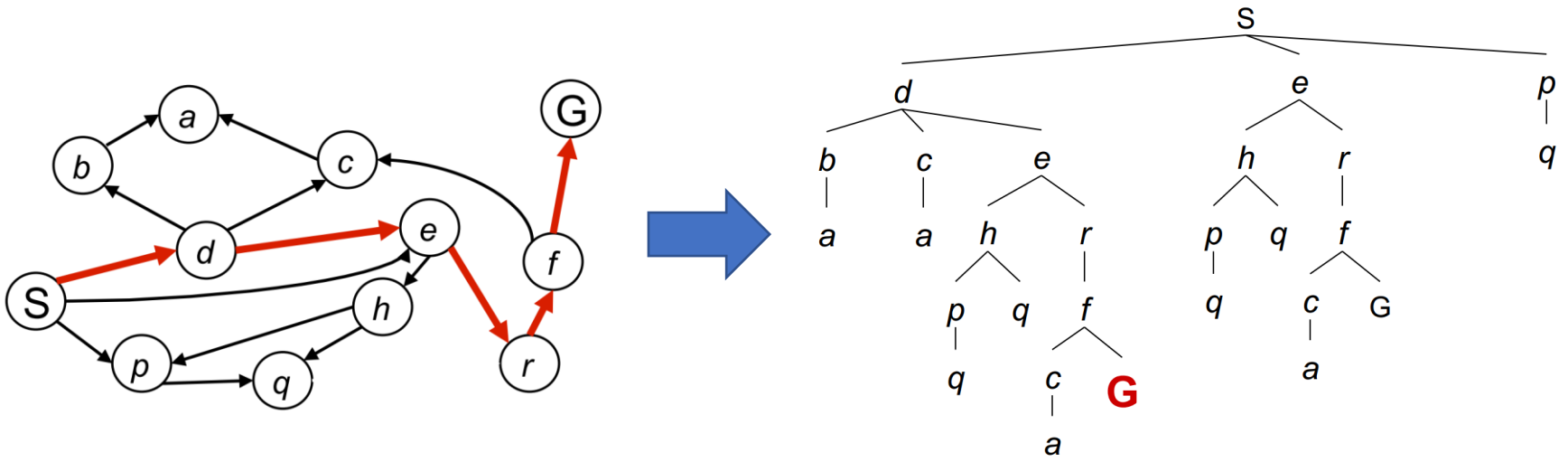
Ridiculously tiny search graph for a tiny search problem



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

Graph Search Overview

- The search always start from start state X_S
  - Searching the graph produces a search tree
  - Back-tracking a node in the search tree gives us a path from the start state to that node
  - For many problems we can never actually build the whole tree, too large or inefficient – we only want to reach the goal node ASAP.
  - Search strategy to make the search efficient

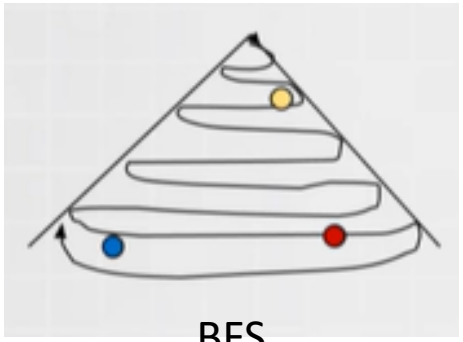


Content

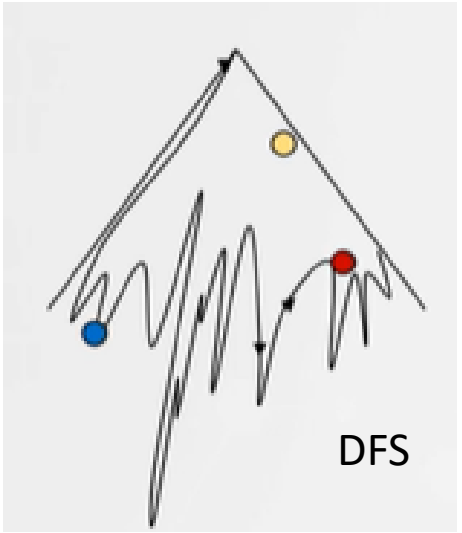
Blind Search: BFS & DFS

Search Strategy

- Blind search (uninformed search) is generally only suitable for solving relatively simple problems.
  - Breadth-First Search
  - Depth-First Search



BFS



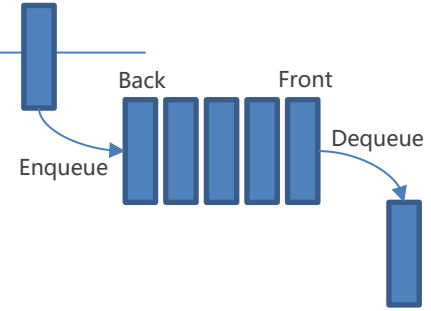
DFS

Blind Search: BFS

Breadth-First Search

Prioritize searching nodes (states) in the state space that are close to the initial state.

It is complete but consumes space.

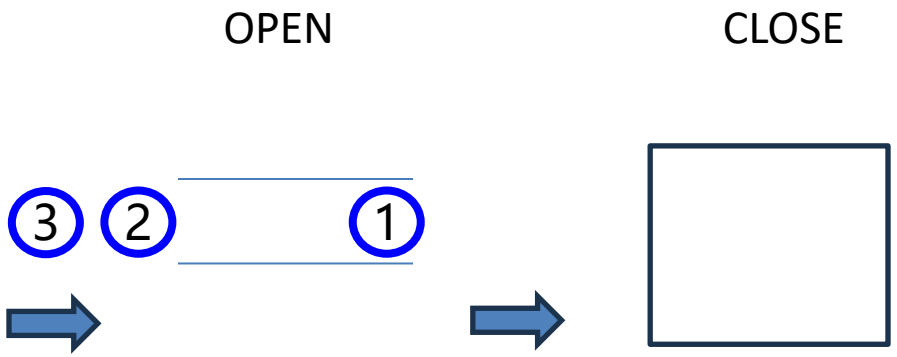


Data structures used in the search:

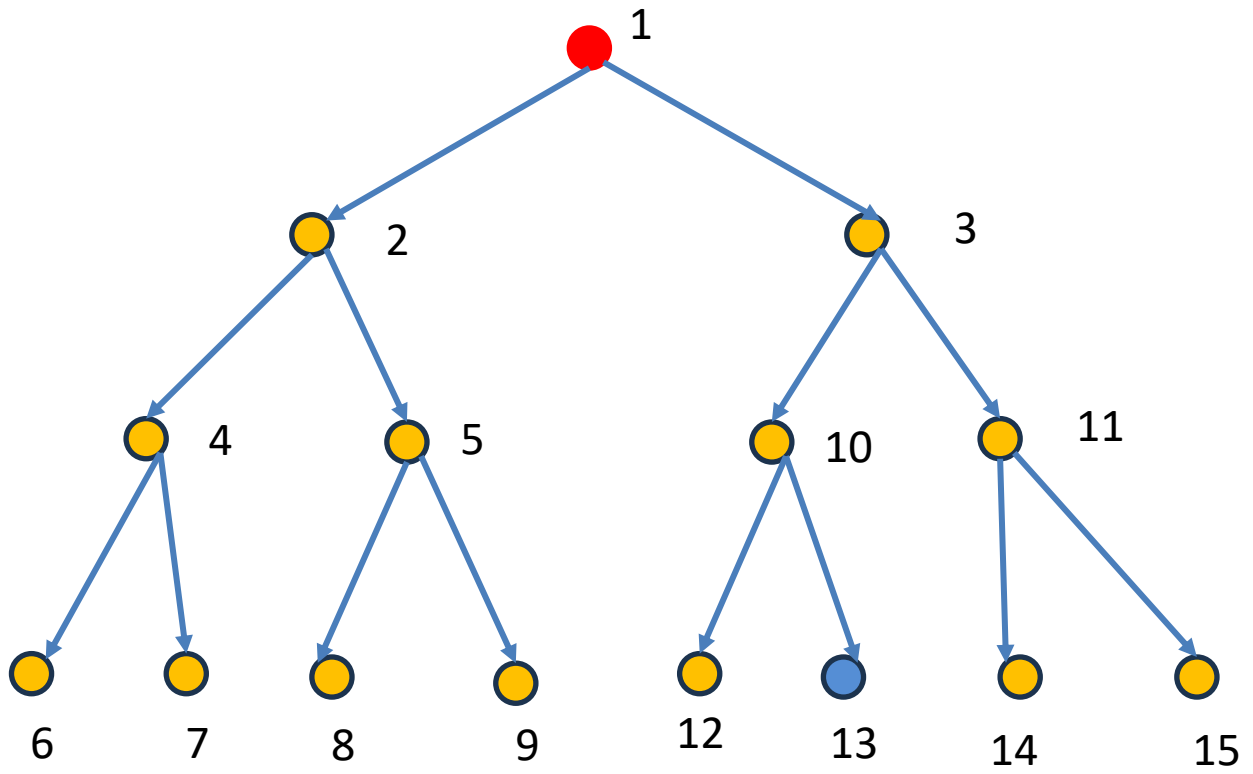
Open Table A first-in, first-out queue that stores nodes to be expanded.

Close Table Stores nodes that have already been expanded.

Blind Search: BFS



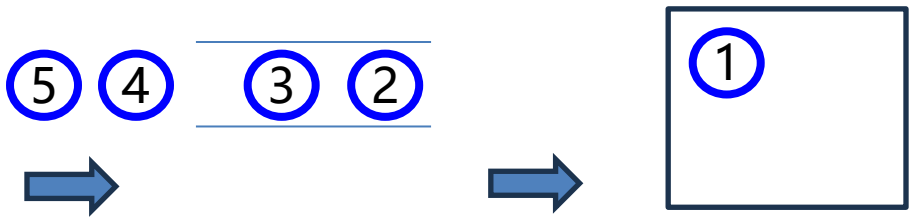
first-in, first-out

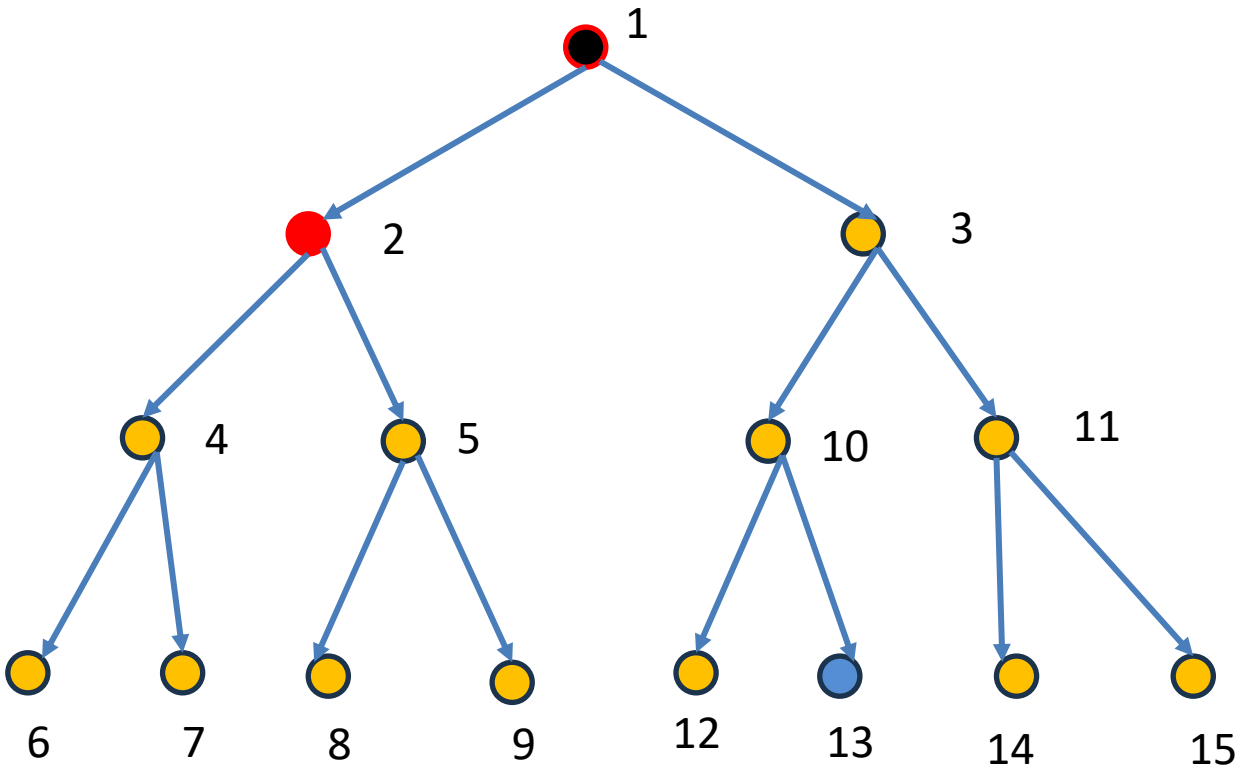


Blind Search: BFS

OPEN

CLOSE

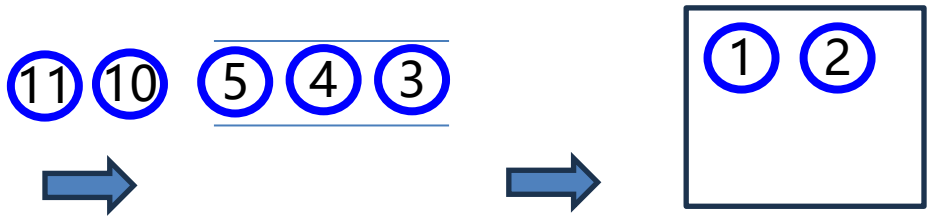


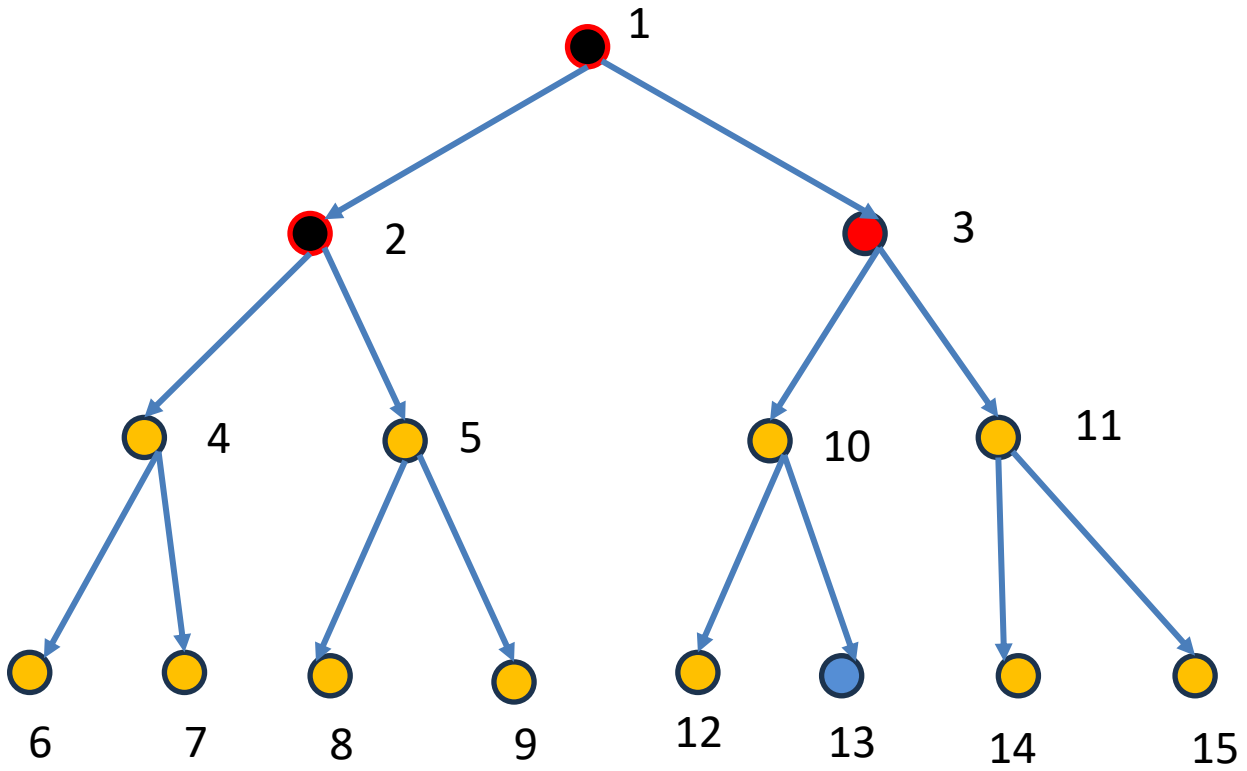


Blind Search: BFS

OPEN

CLOSE





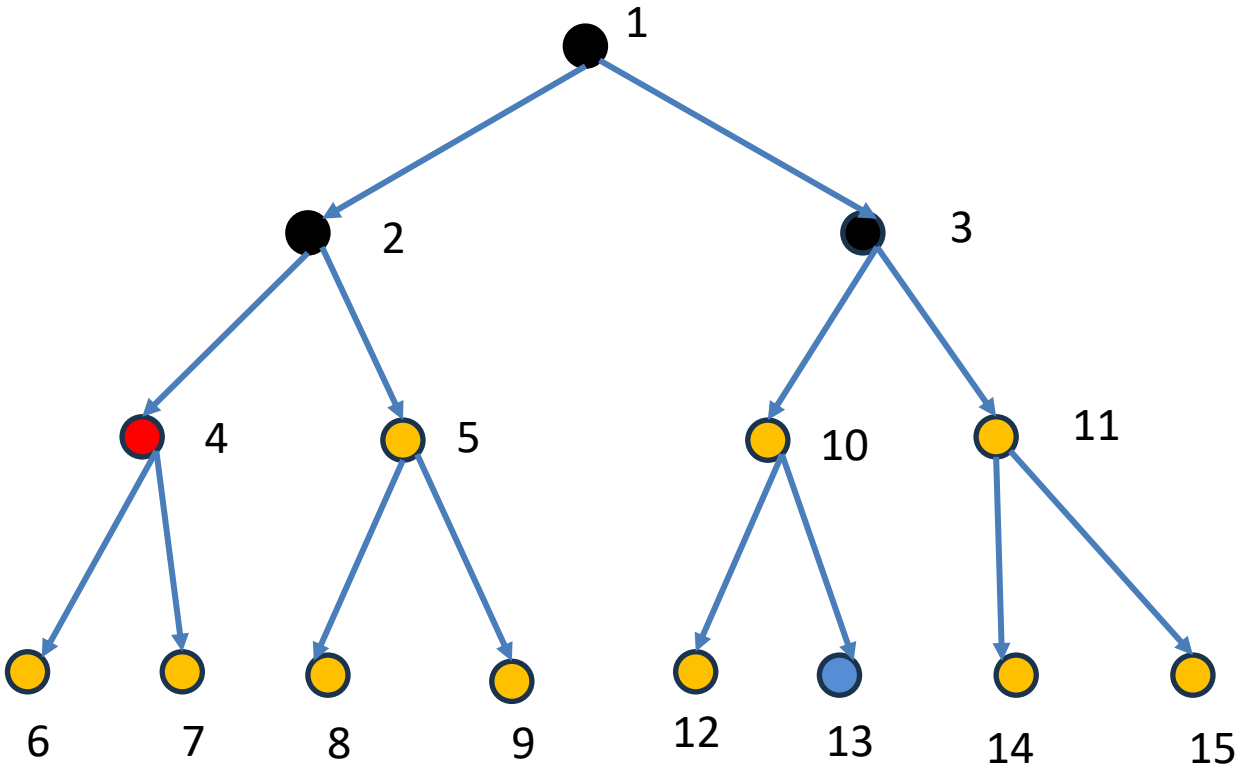
Blind Search: BFS

OPEN

7 6 11 10 5 4

CLOSE

1 2
3



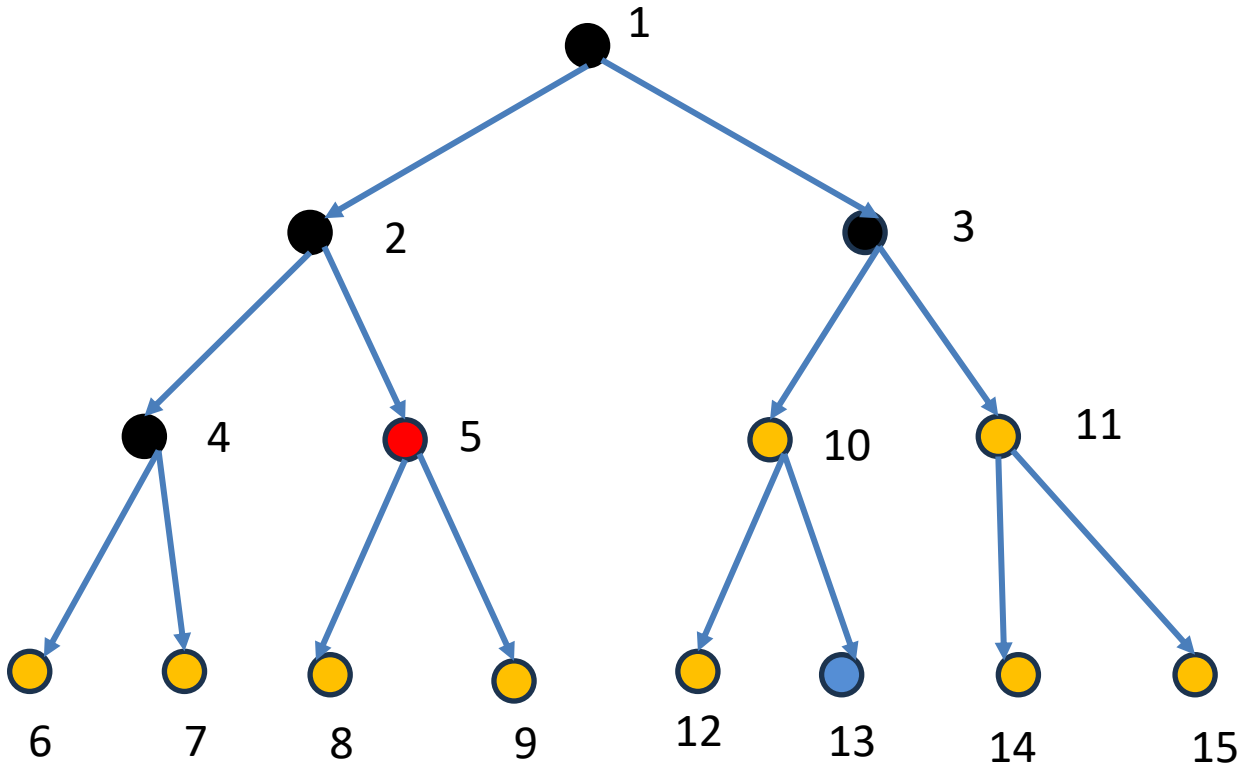
Blind Search: BFS

OPEN

9 8 7 6 11 10 5

CLOSE

1 2
3 4



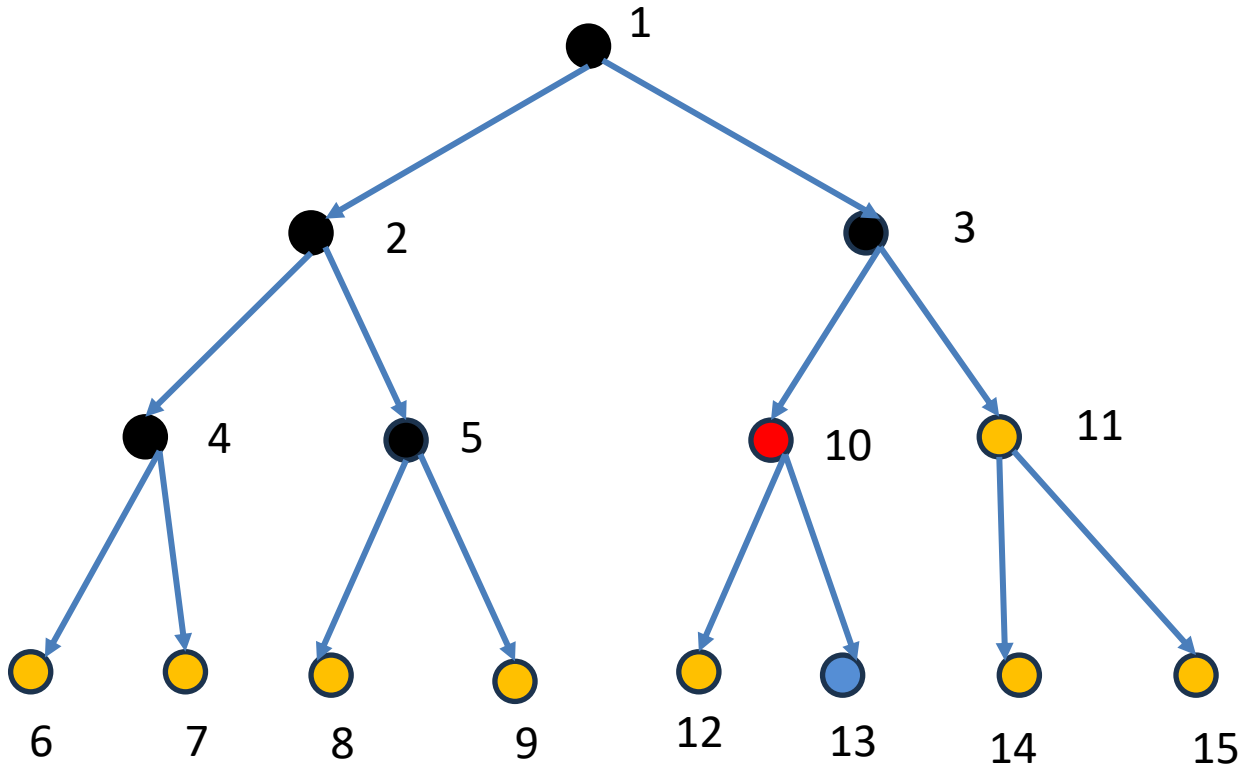
Blind Search: BFS

OPEN

13 12 9 8 7 6 11 10

CLOSE

1 2
3 4
5



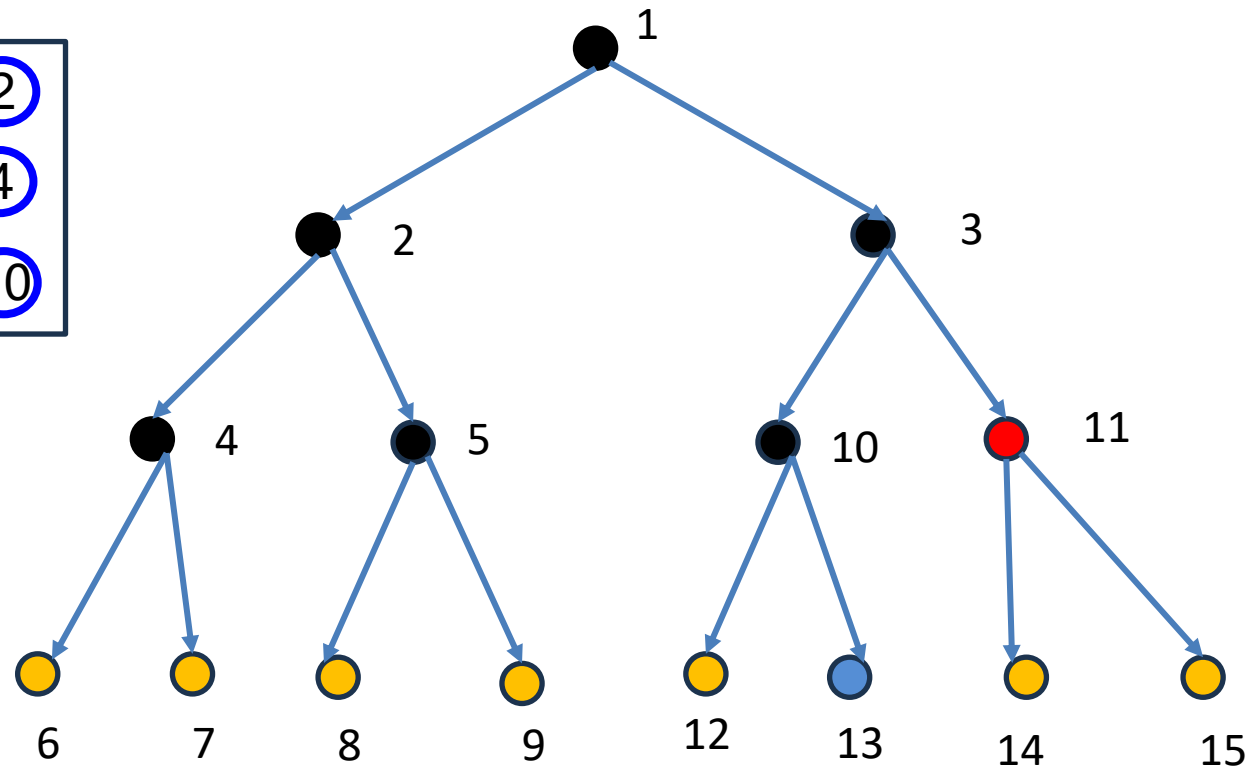
Blind Search: BFS

OPEN

15 14 13 12 9 8 7 6 11

CLOSE

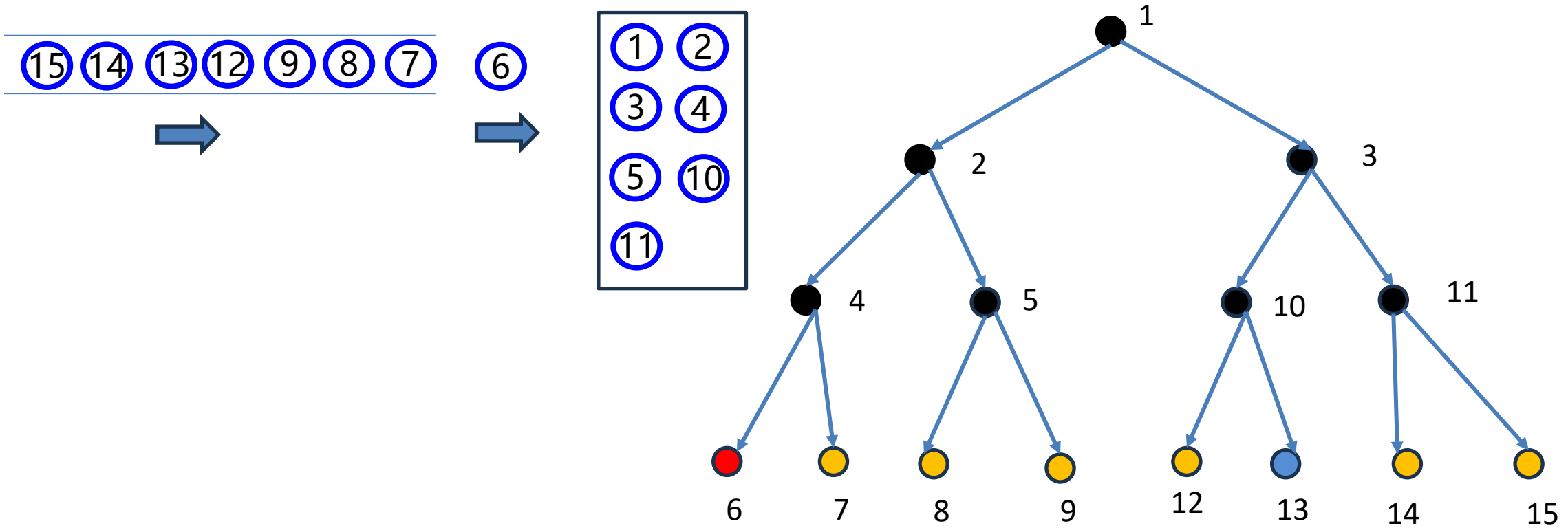
1 2
3 4
5 10



Blind Search: BFS

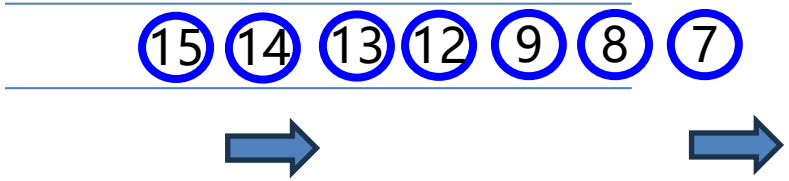
OPEN

CLOSE

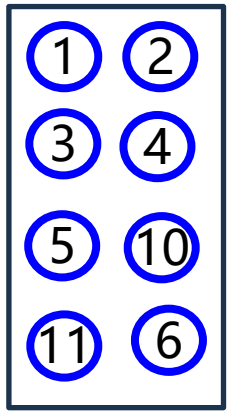


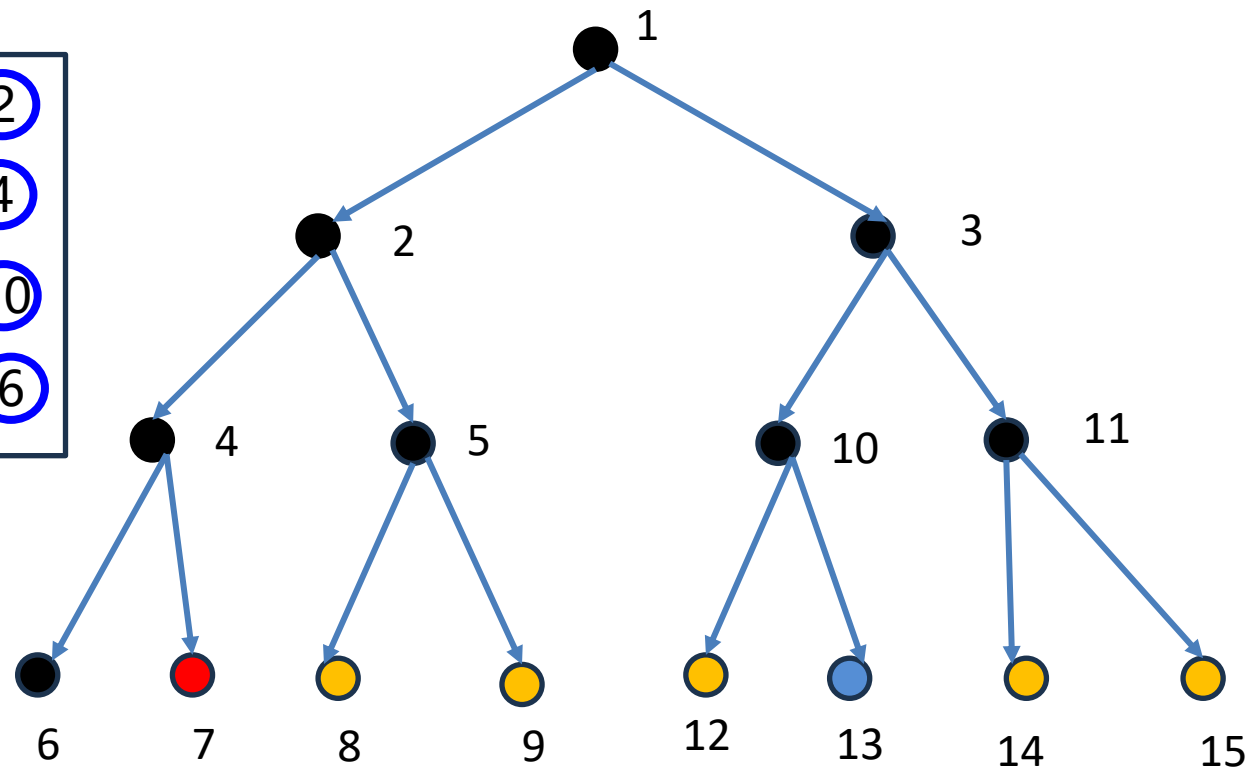
Blind Search: BFS

OPEN



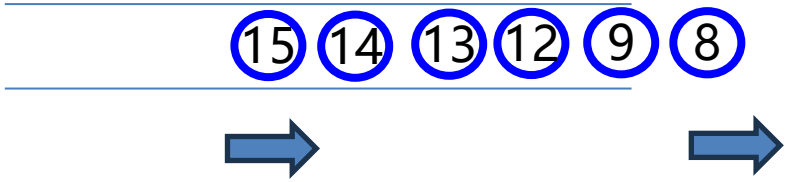
CLOSE



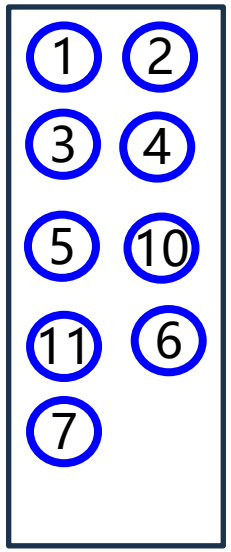


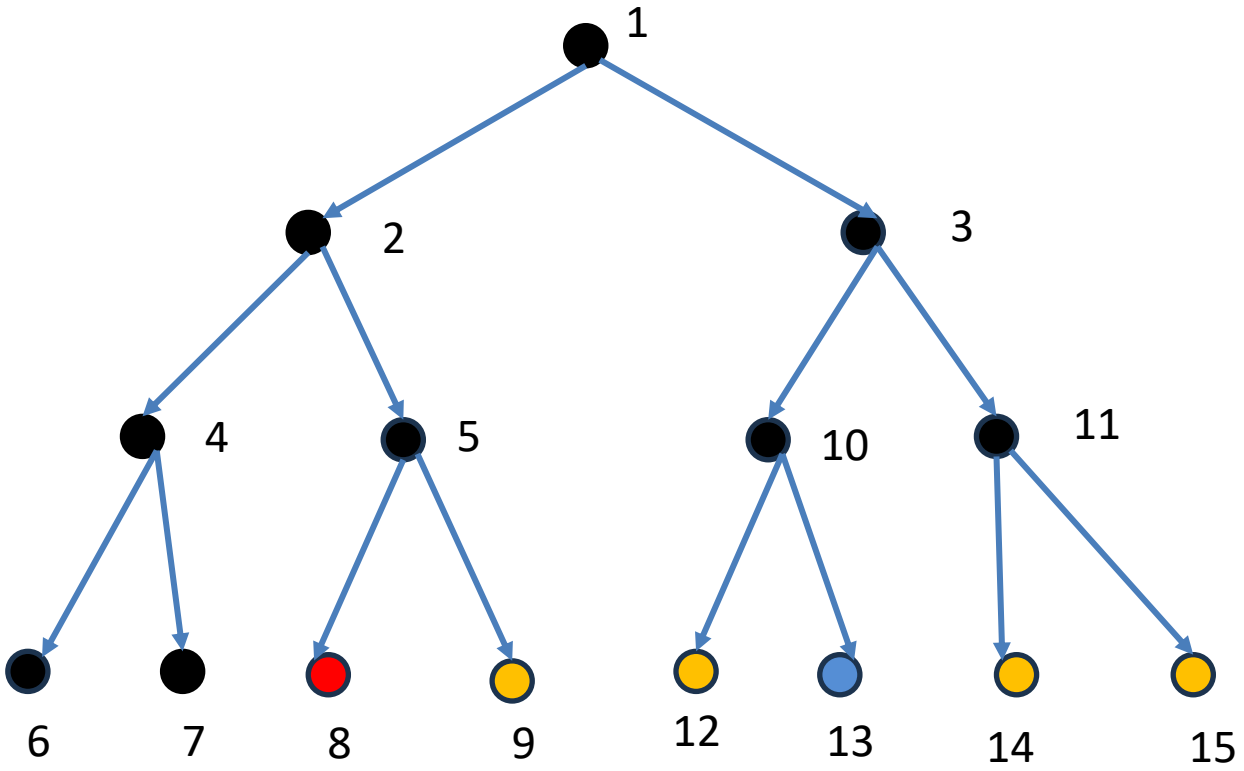
Blind Search: BFS

OPEN



CLOSE





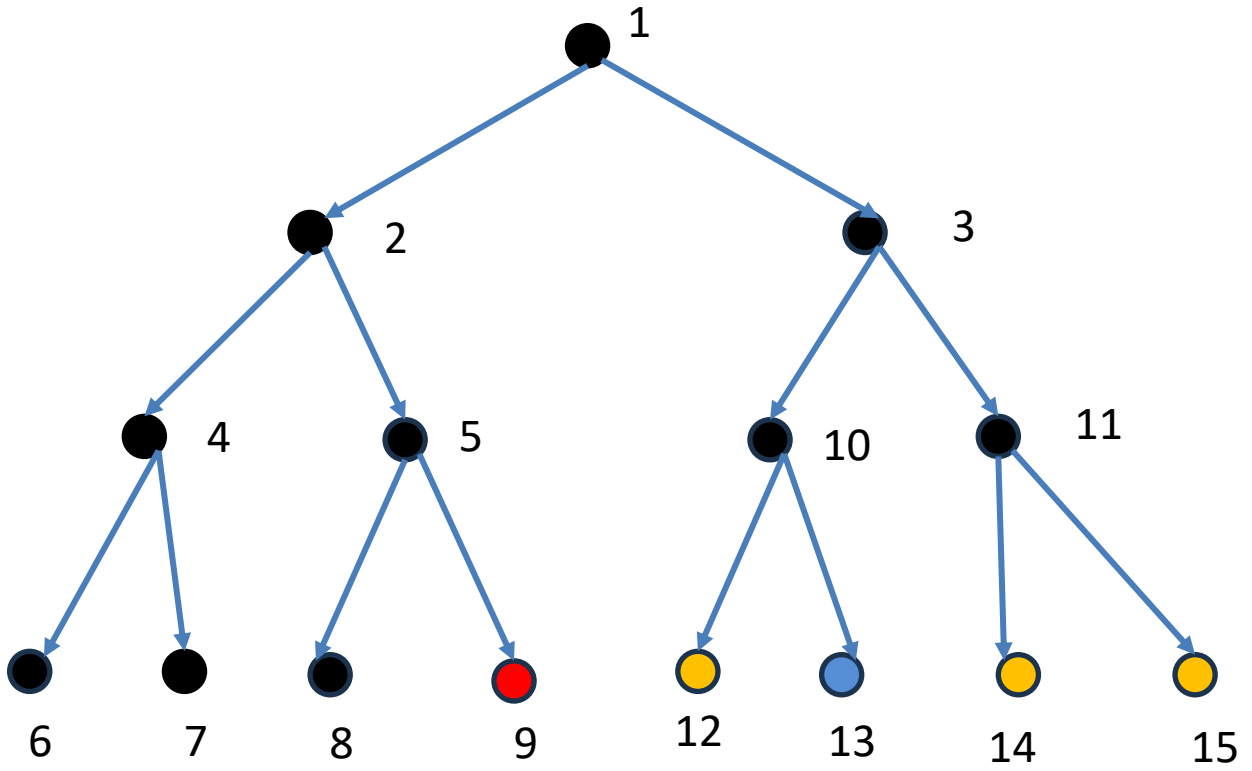
Blind Search: BFS

OPEN

15 14 13 12 9

CLOSE

1 2
3 4
5 10
11 6
7 8



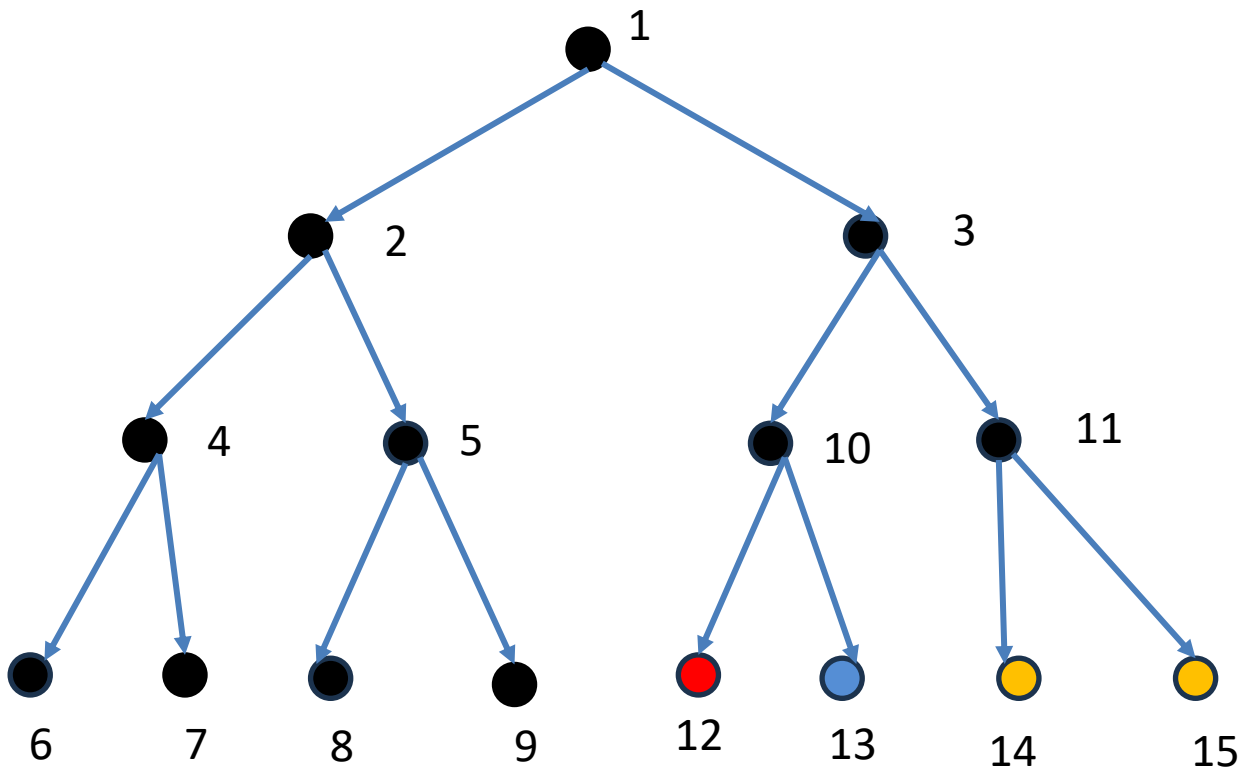
Blind Search: BFS

OPEN

15 14 13 12

CLOSE

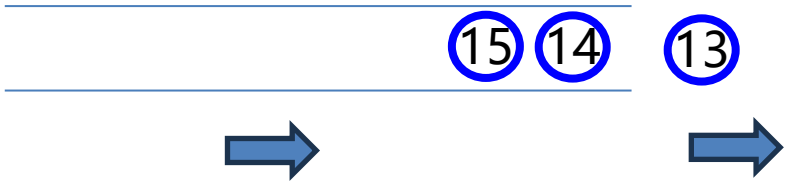
1 2
3 4
5 10
11 6
7 8
9

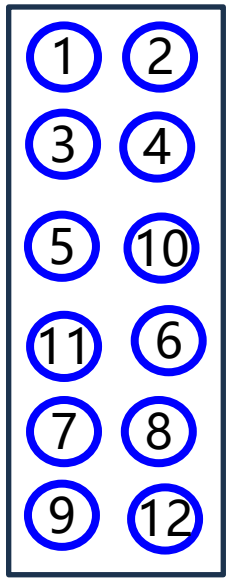


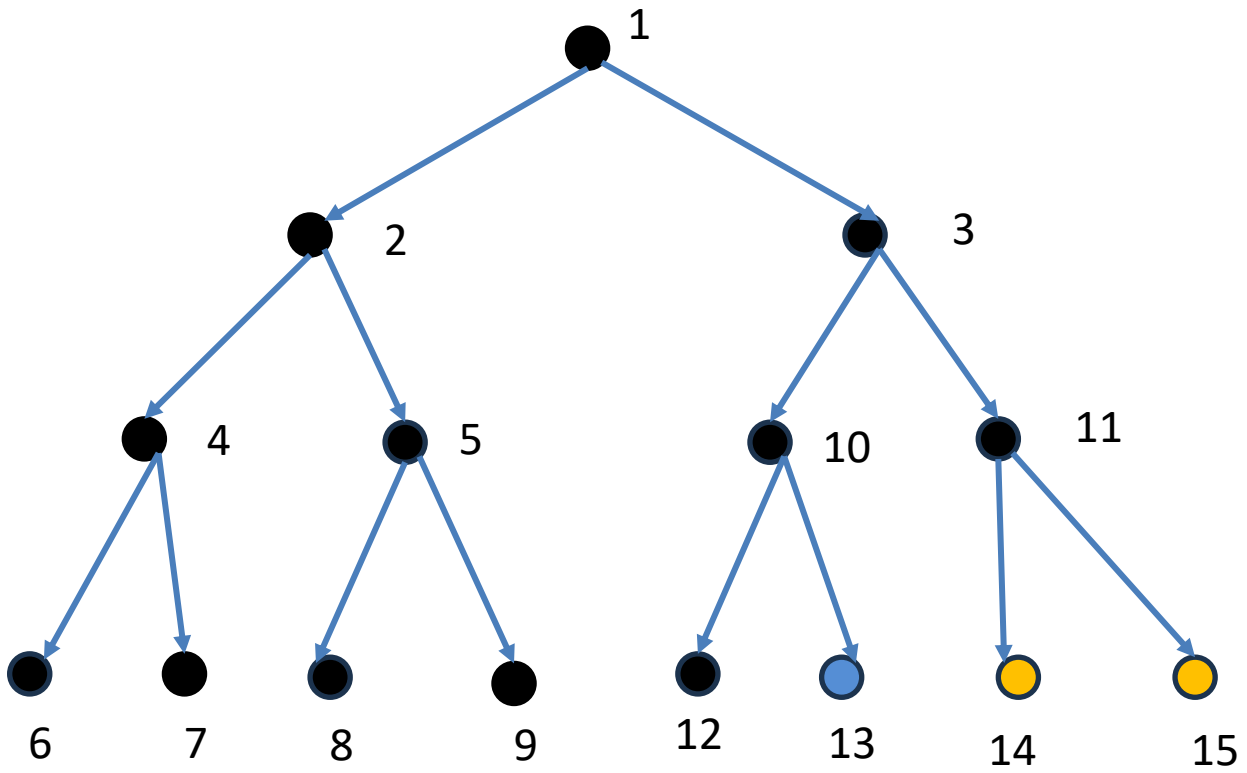
Blind Search: BFS

OPEN

CLOSE

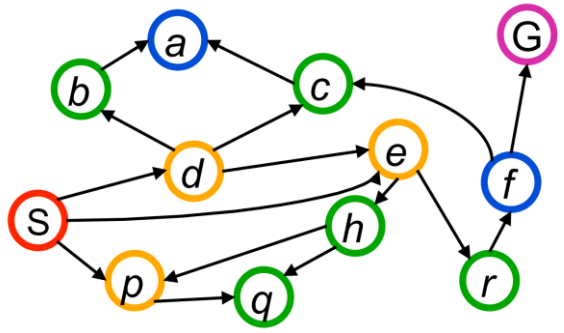


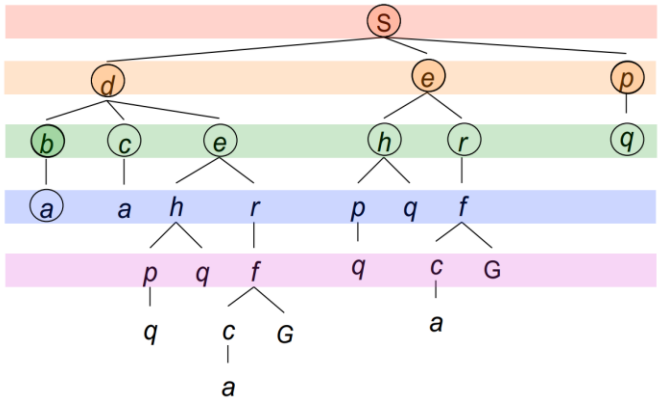




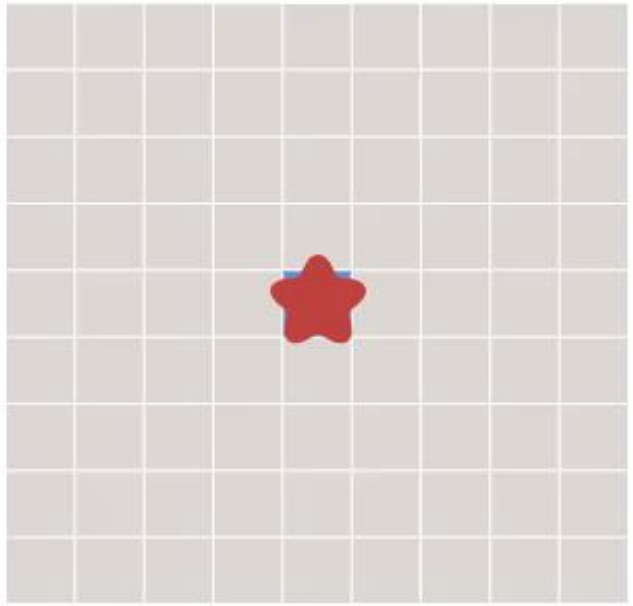
Breadth First Search (BFS)

- Strategy: Prioritize searching the shallowest node in the table





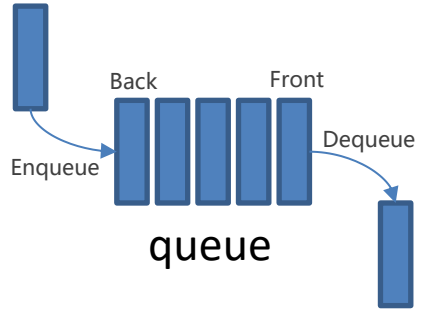
Breadth First Search (BFS)



Courtesy: Amit Patel's Introduction to A*, Stanford

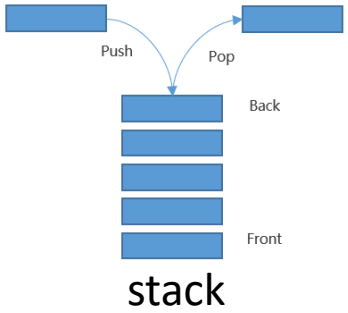
Blind Search: BFS & DFS

BFS:
first-in, first-out



Depth-First Search

Nodes are prioritized for expansion deeply until a certain depth limit is reached. If the goal is not found or expansion is not possible, backtrack to another node and continue expanding.



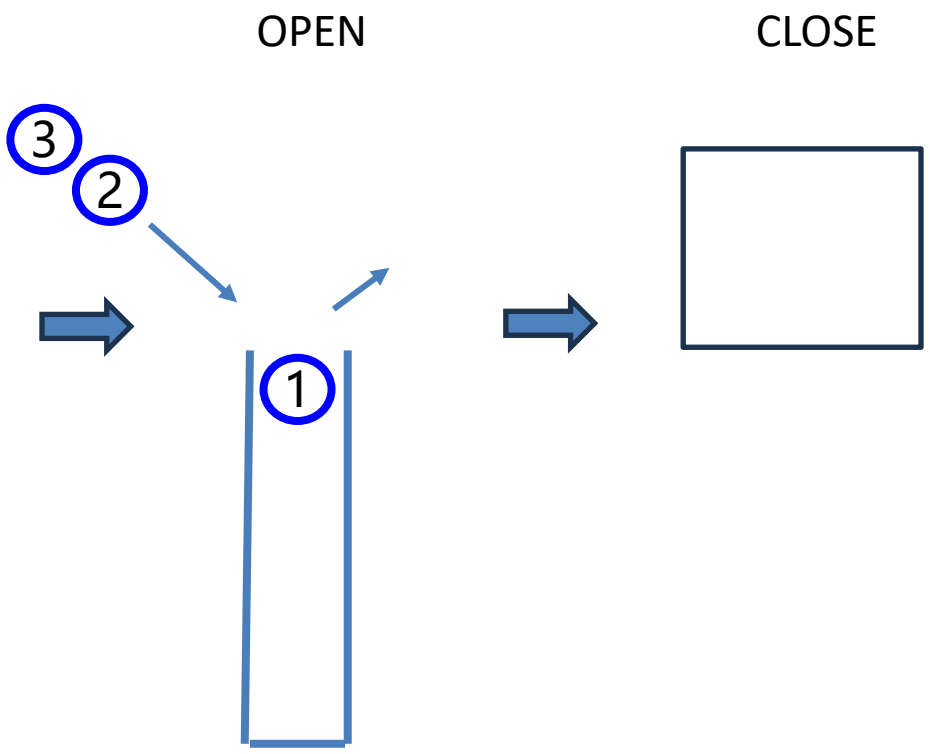
Requires a depth limit, backtracking control, and saves space, not optimal solution.

Data structures used in the search:

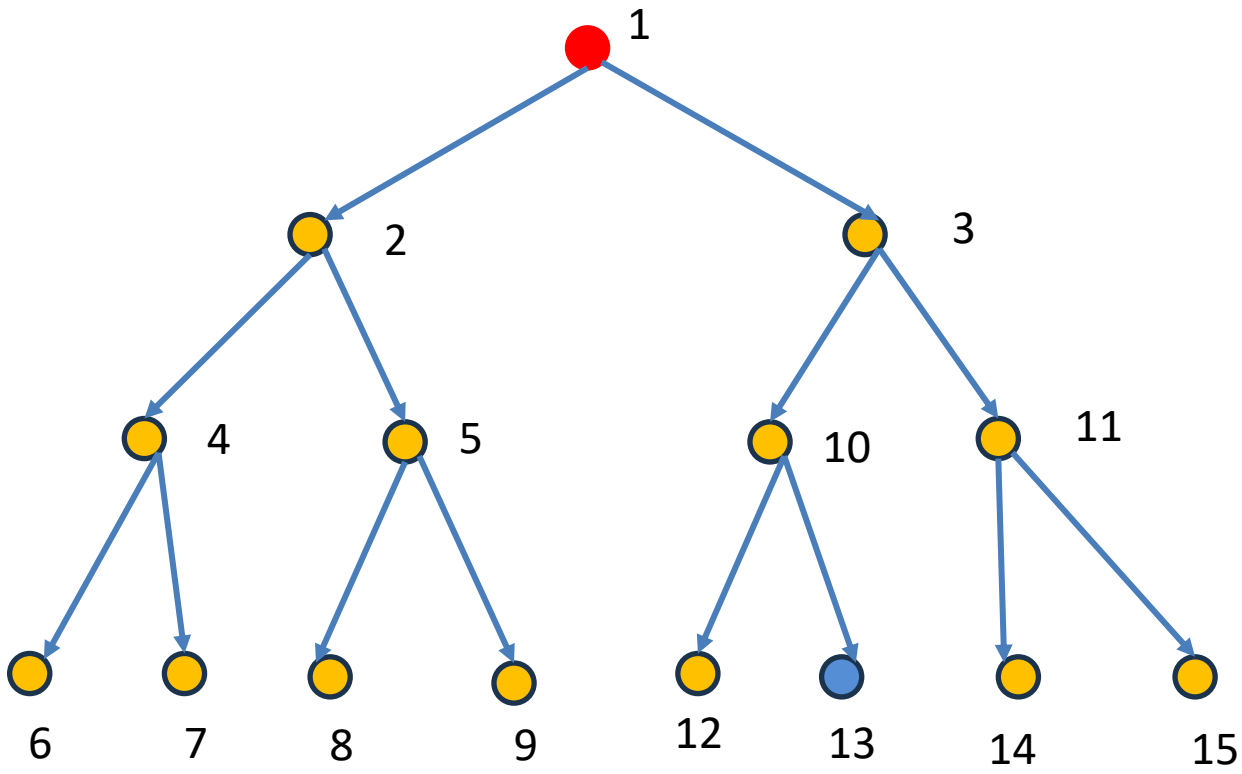
Open Table A last-in, first-out stack that stores nodes to be expanded.

Close Table Stores nodes that have already been expanded.

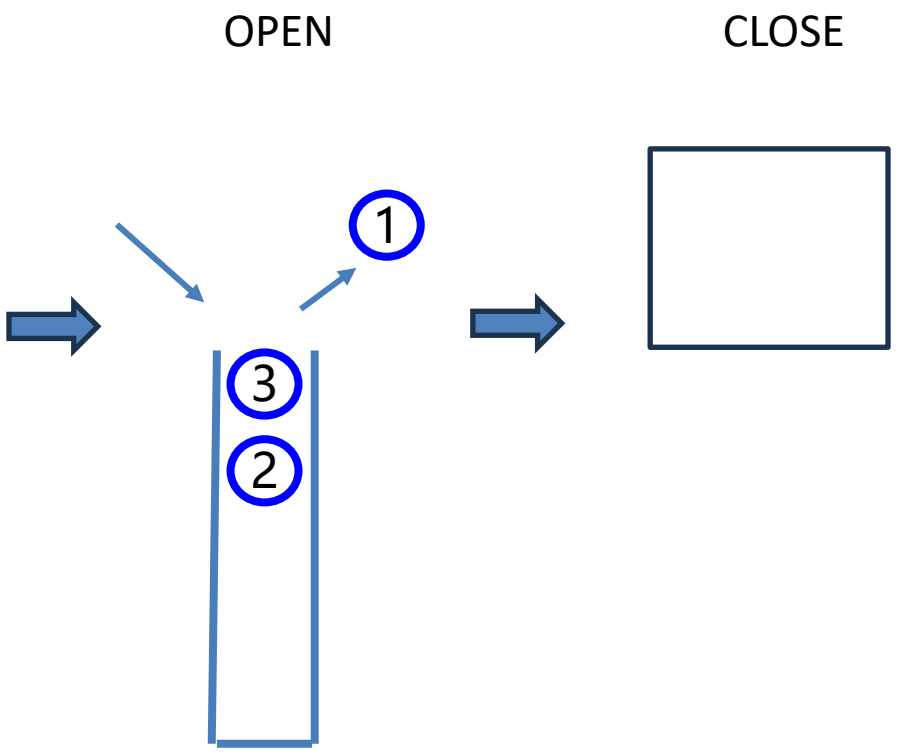
Blind Search: DFS



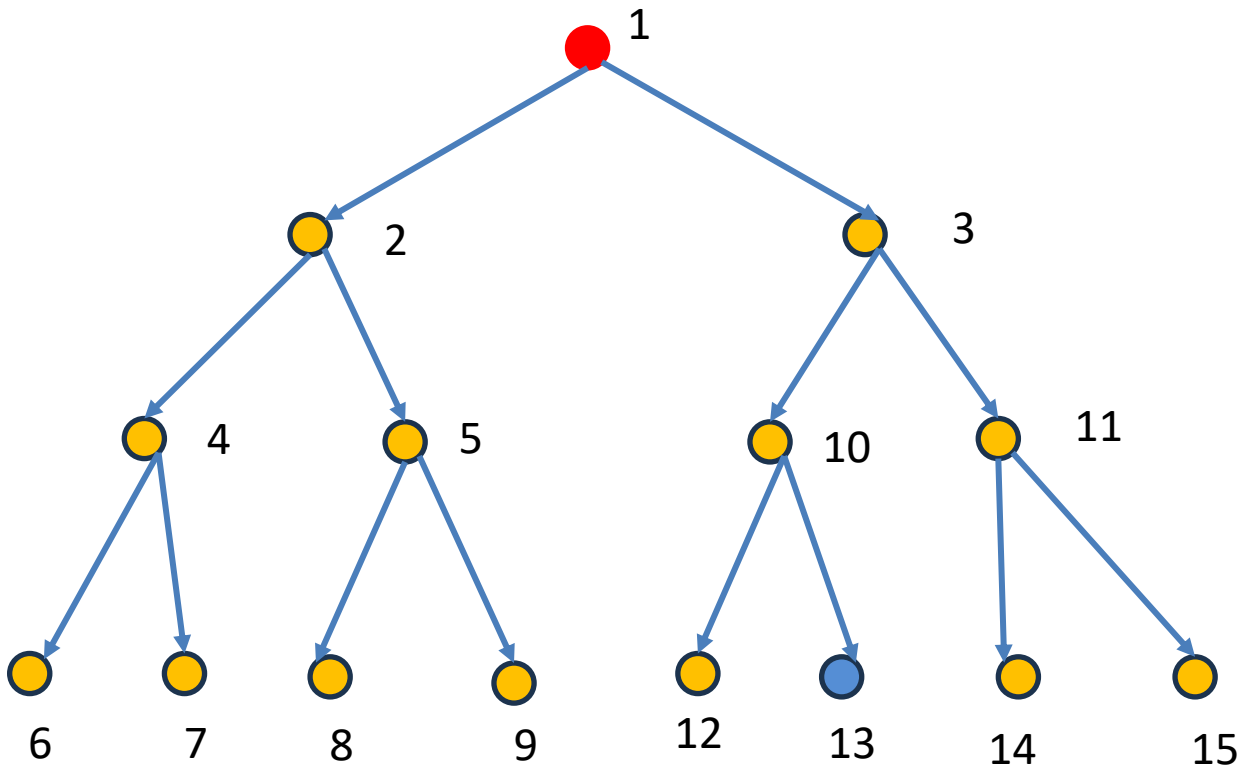
last-in, first-out



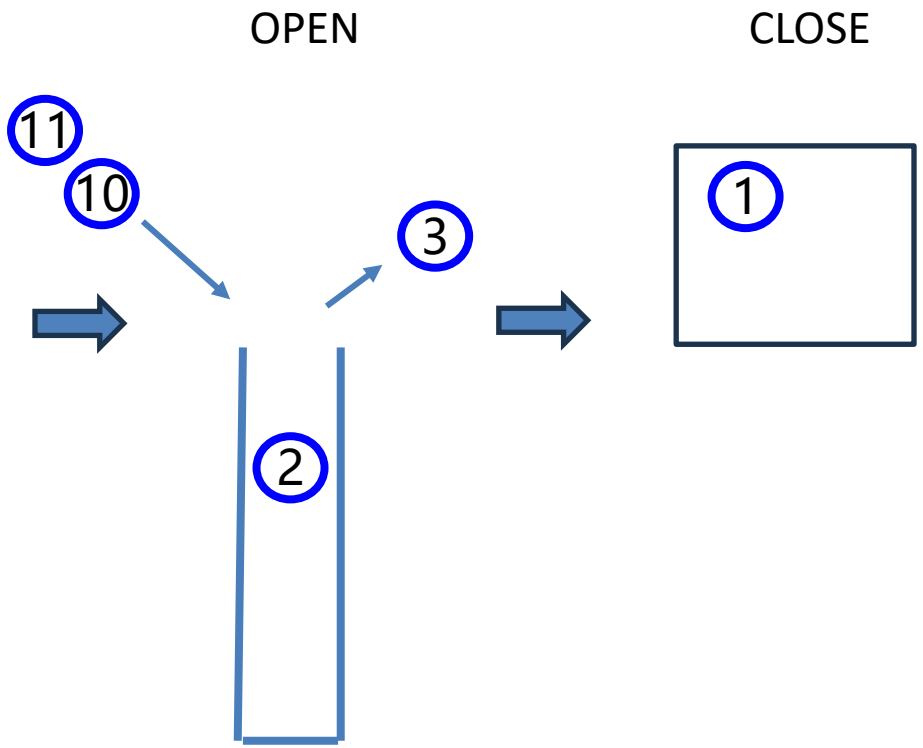
Blind Search: DFS

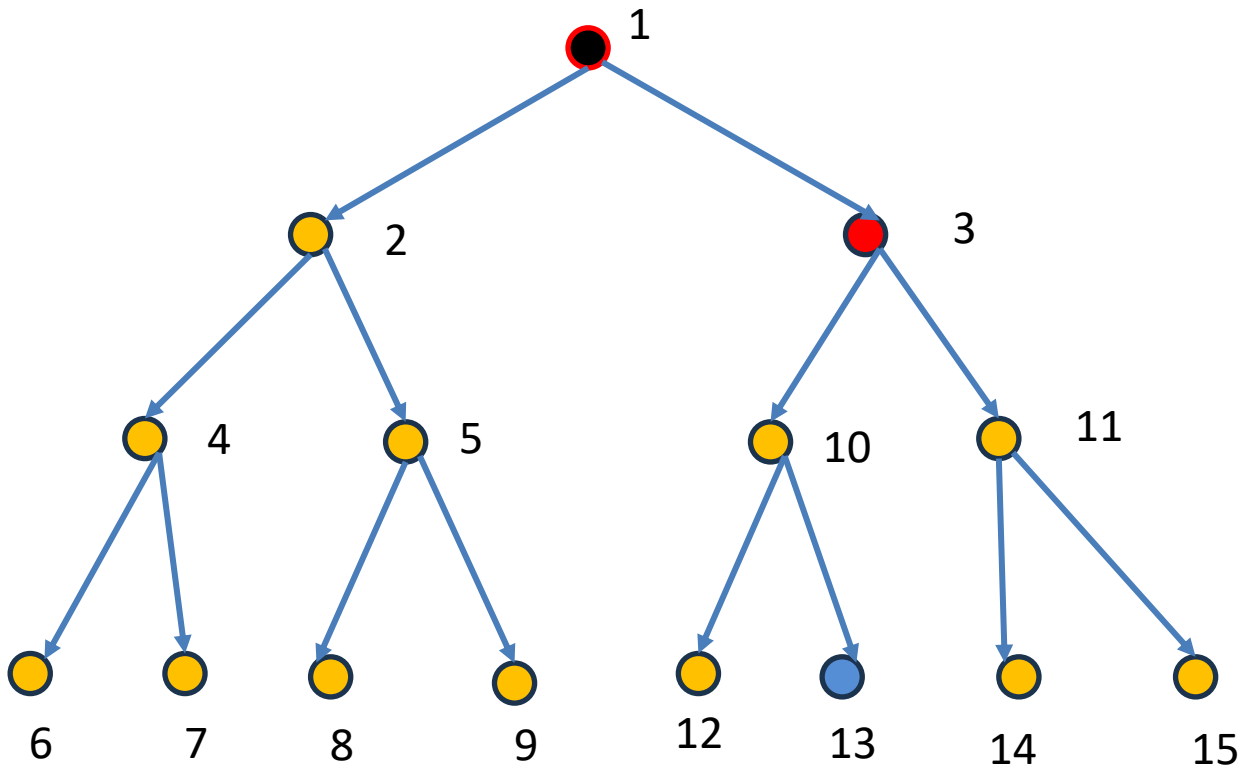


last-in, first-out

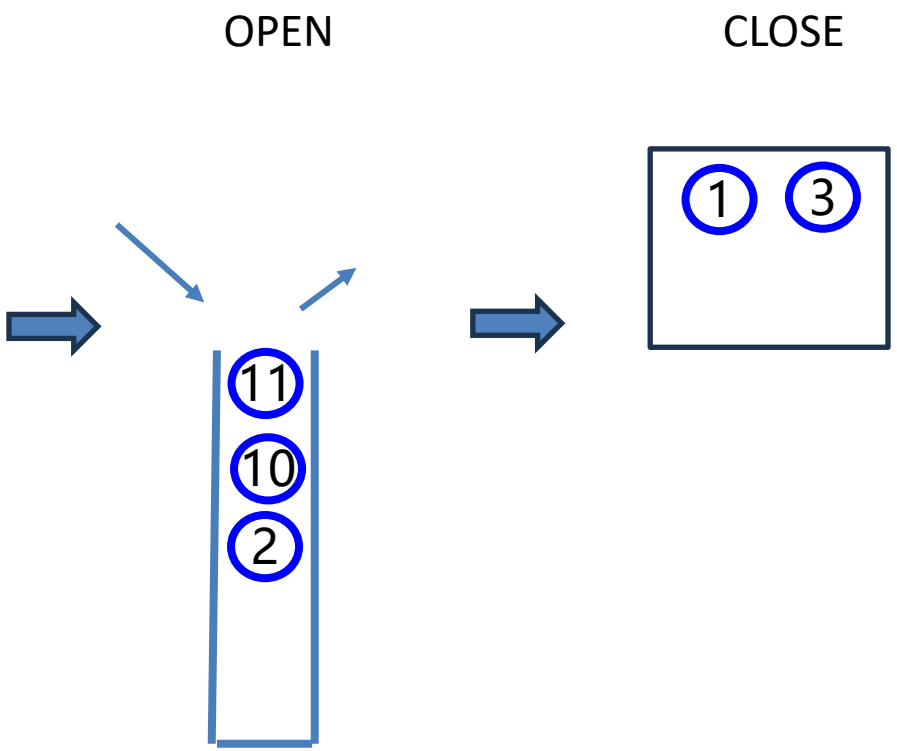


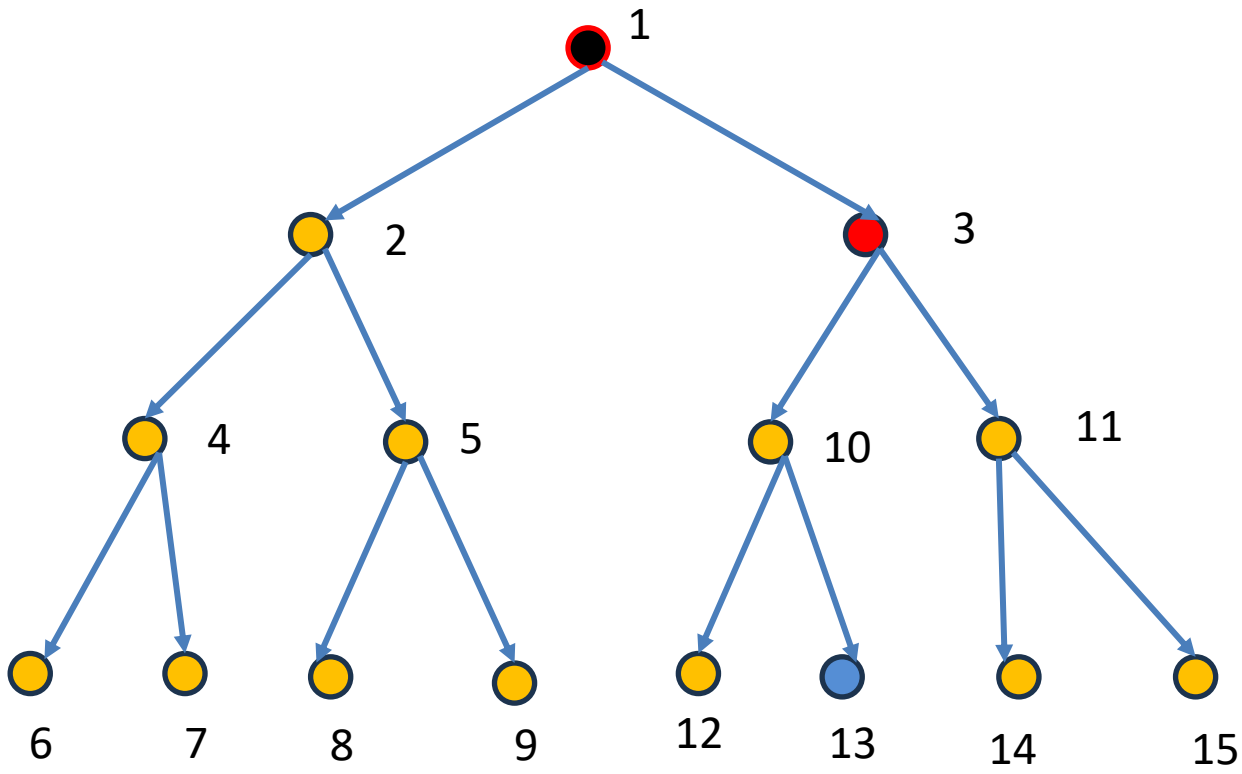
Blind Search: DFS



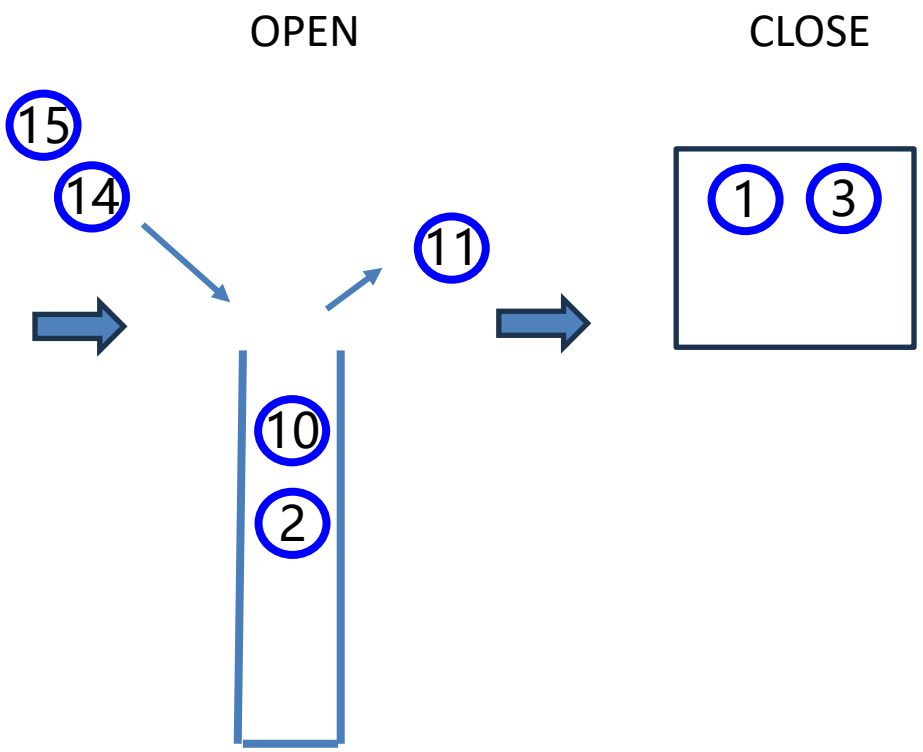


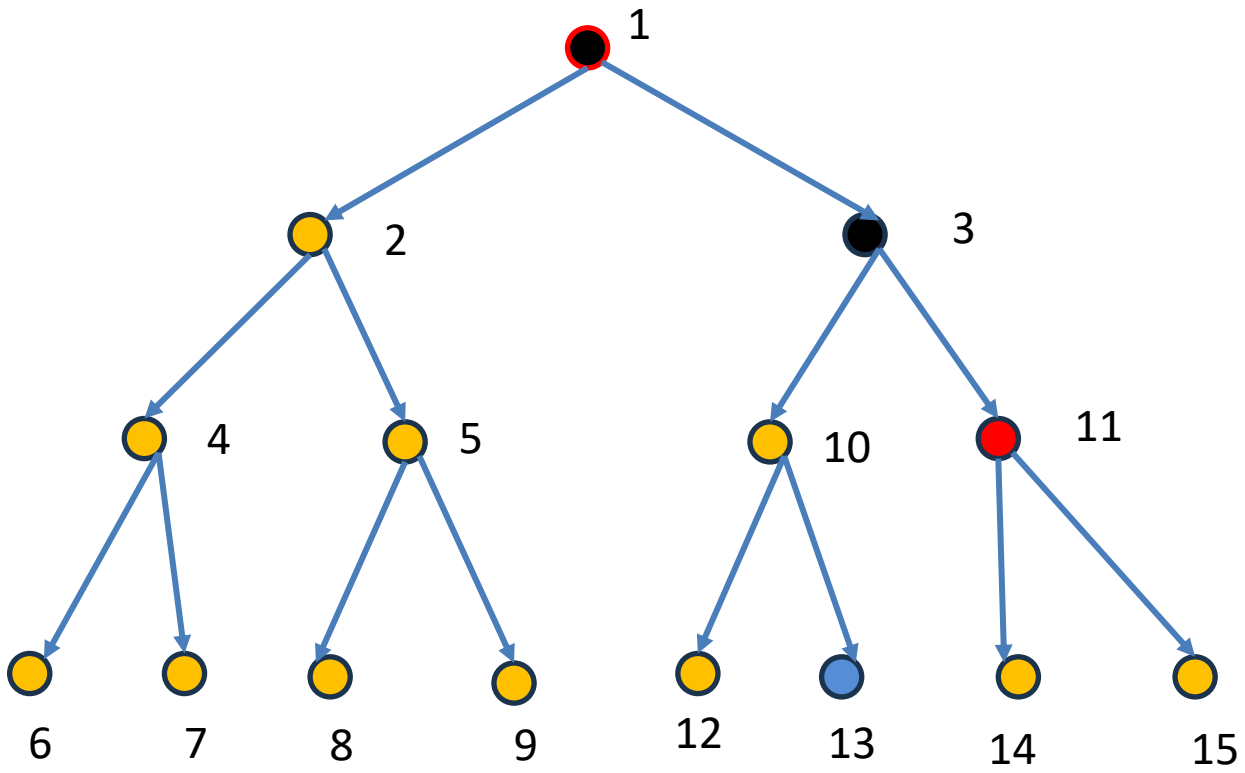
Blind Search: DFS



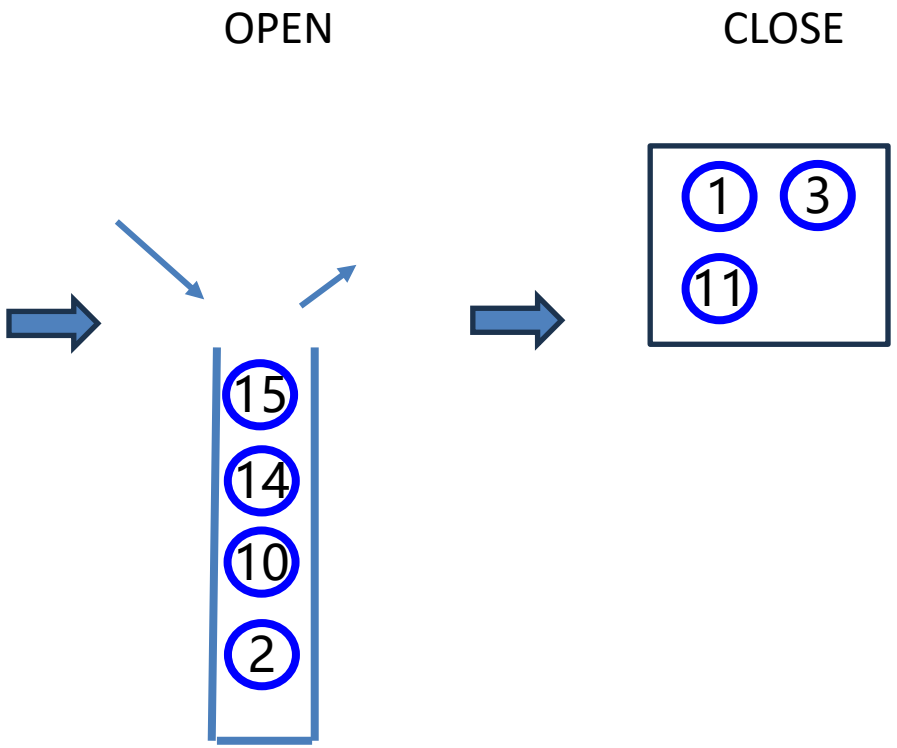


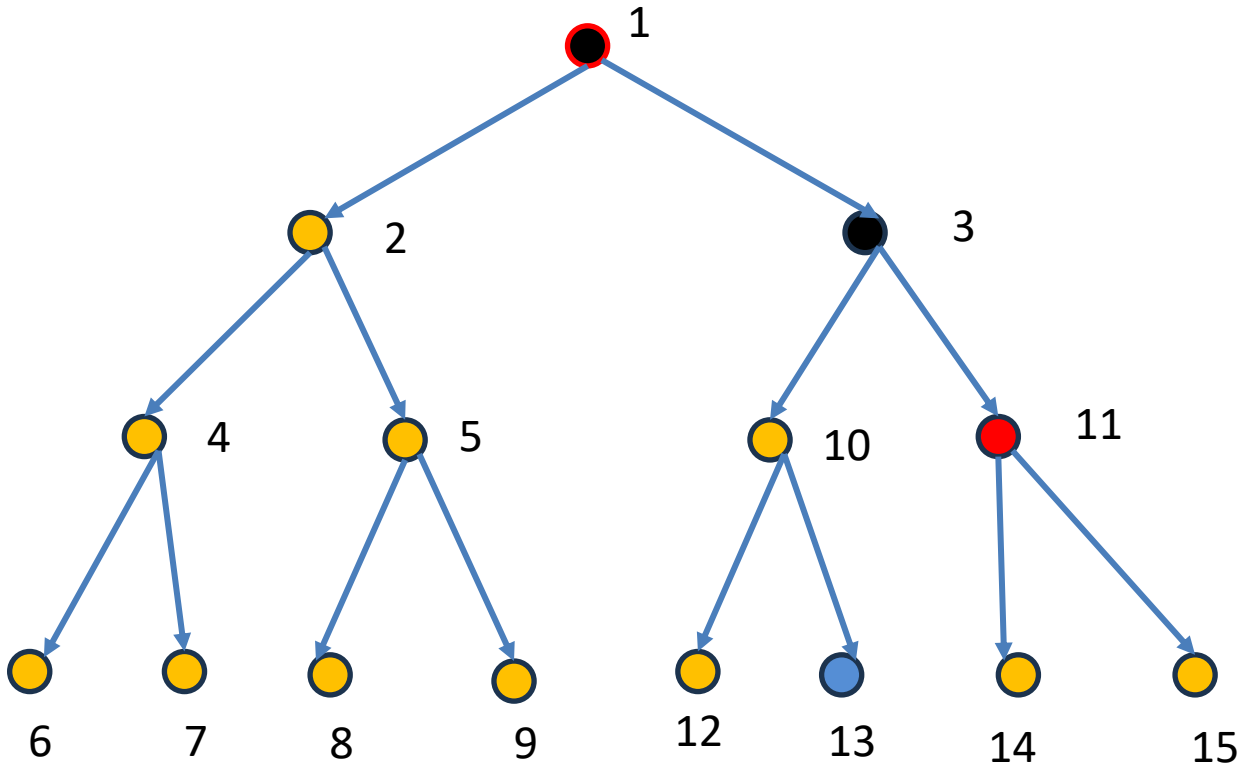
Blind Search: DFS



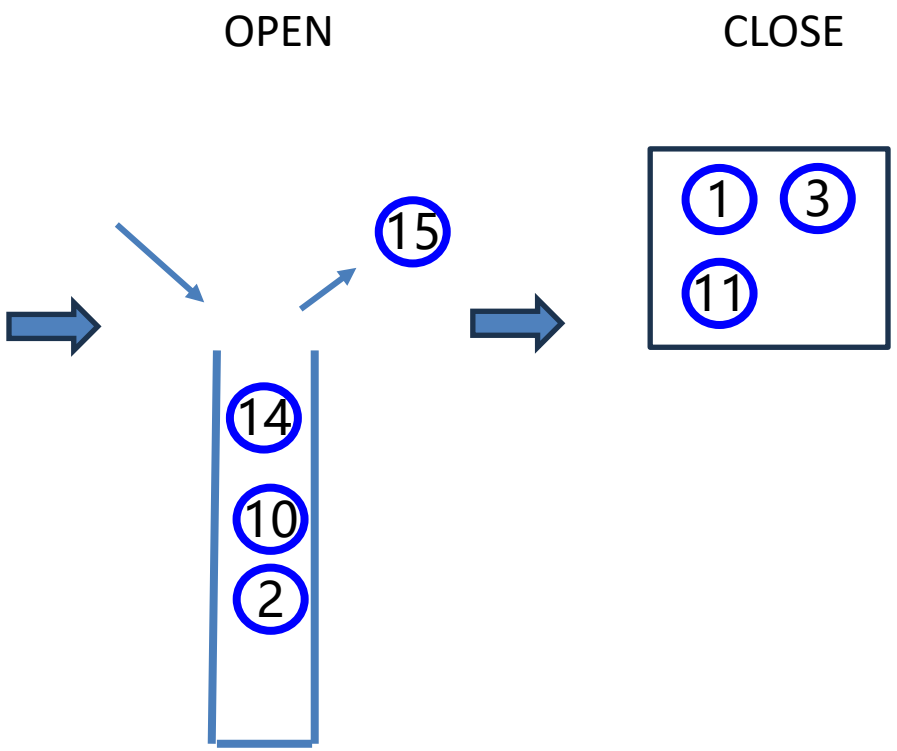


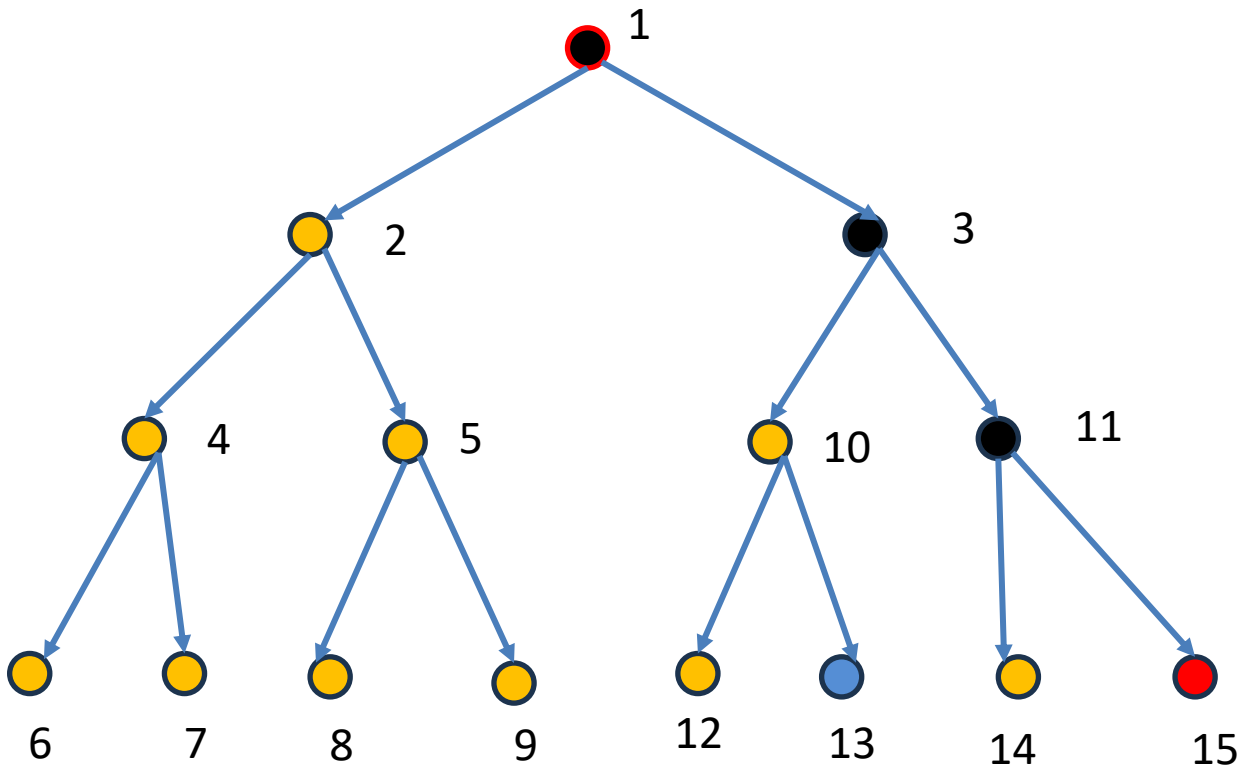
Blind Search: DFS



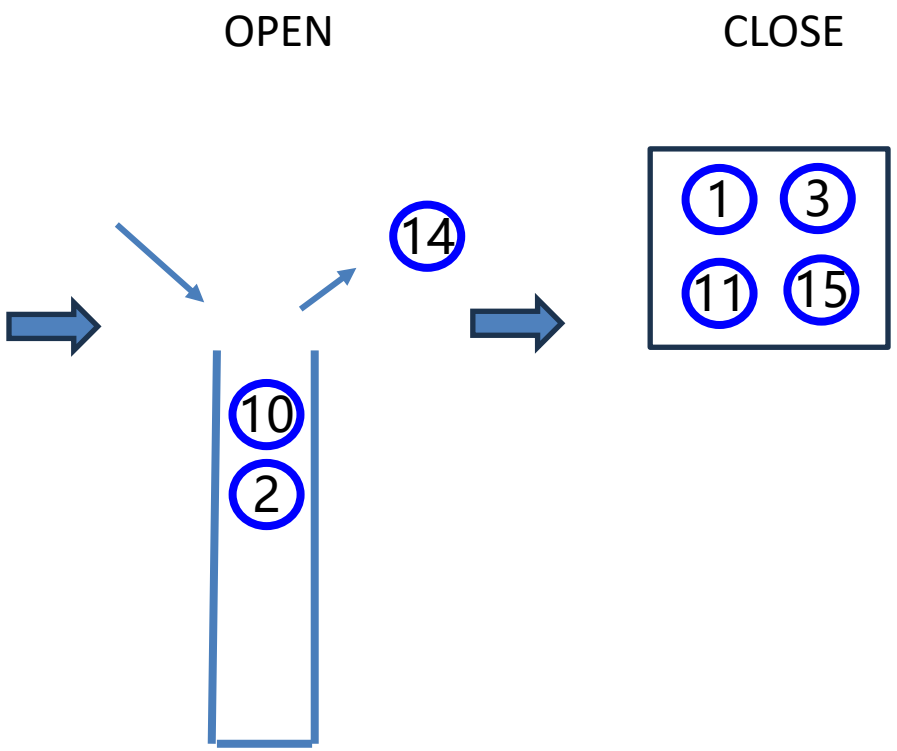


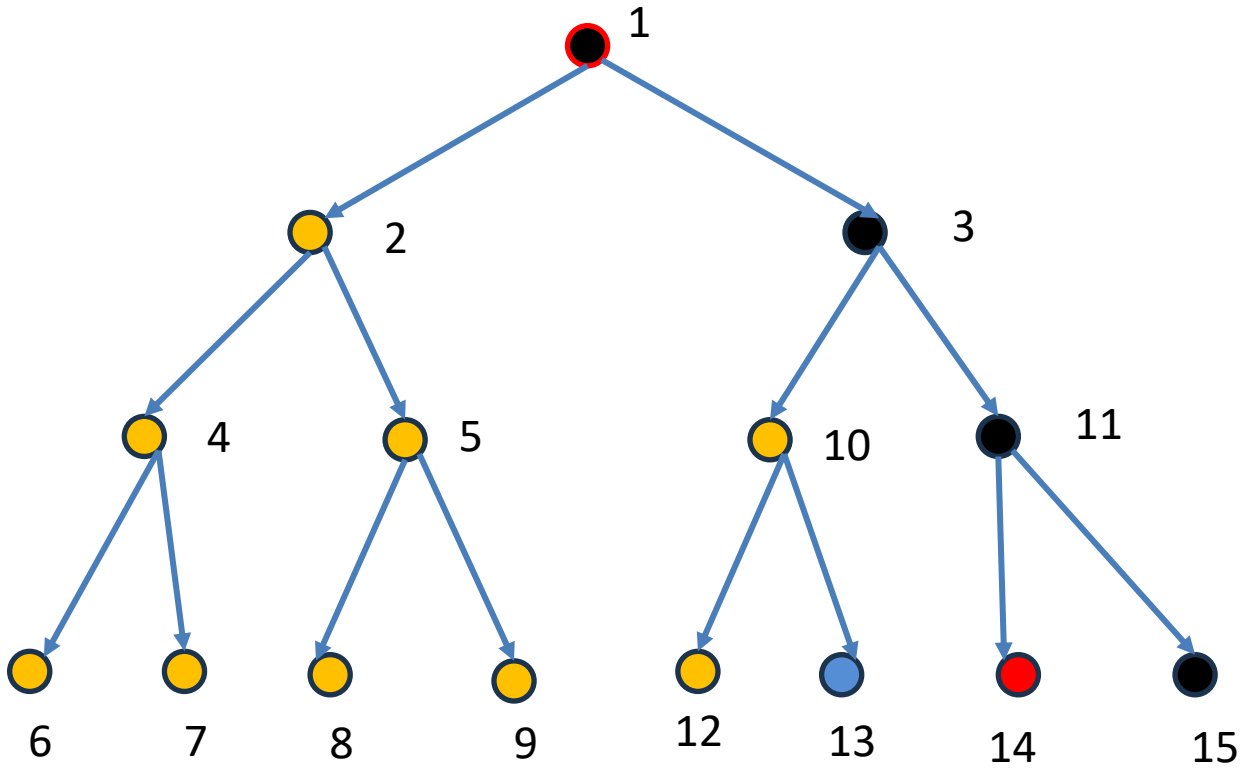
Blind Search: DFS



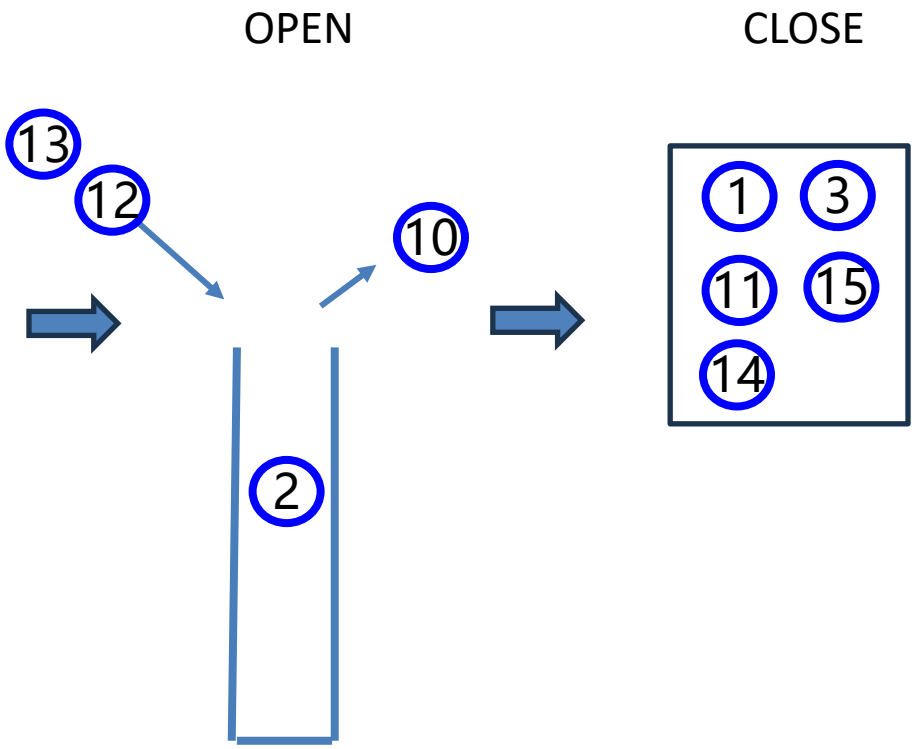


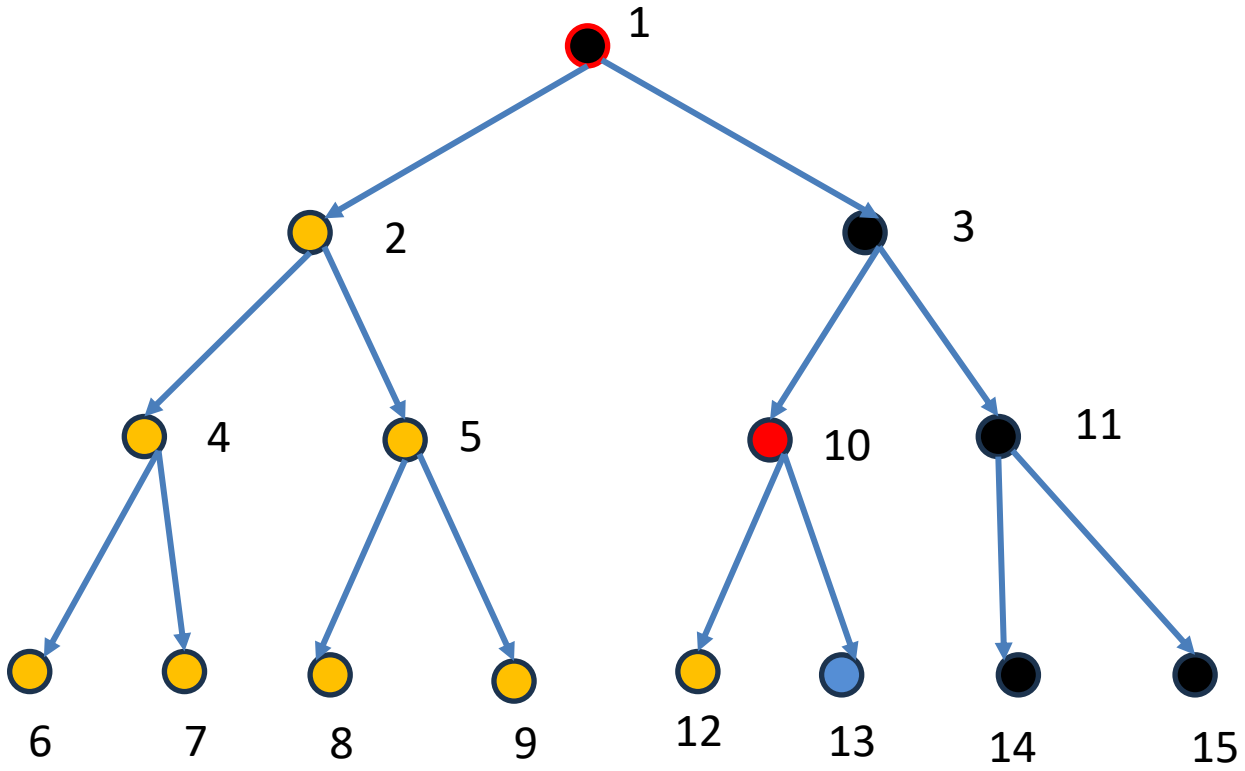
Blind Search: DFS



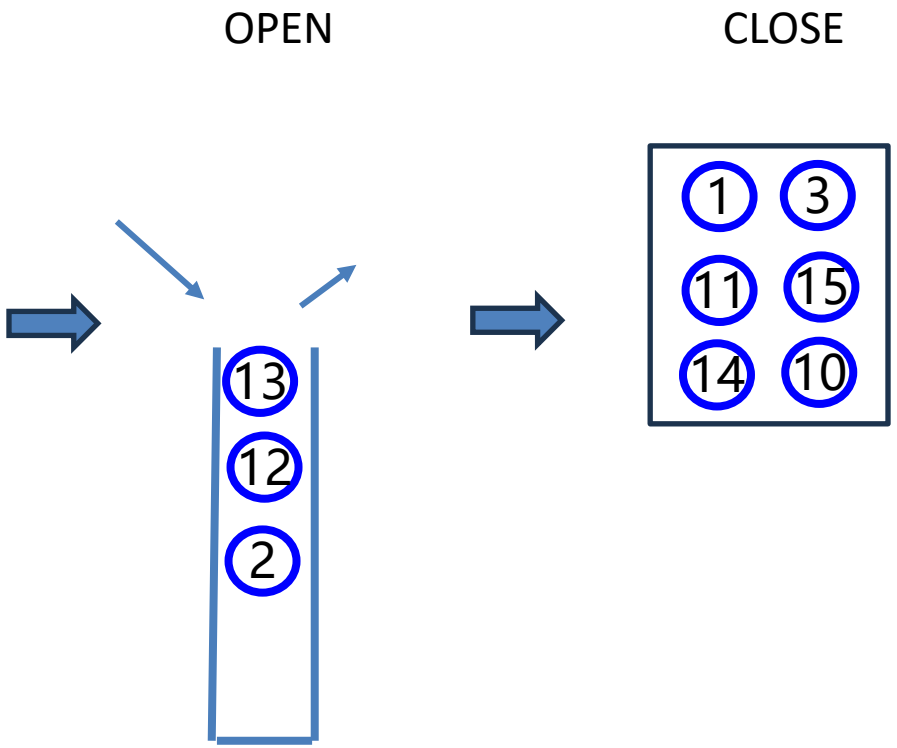


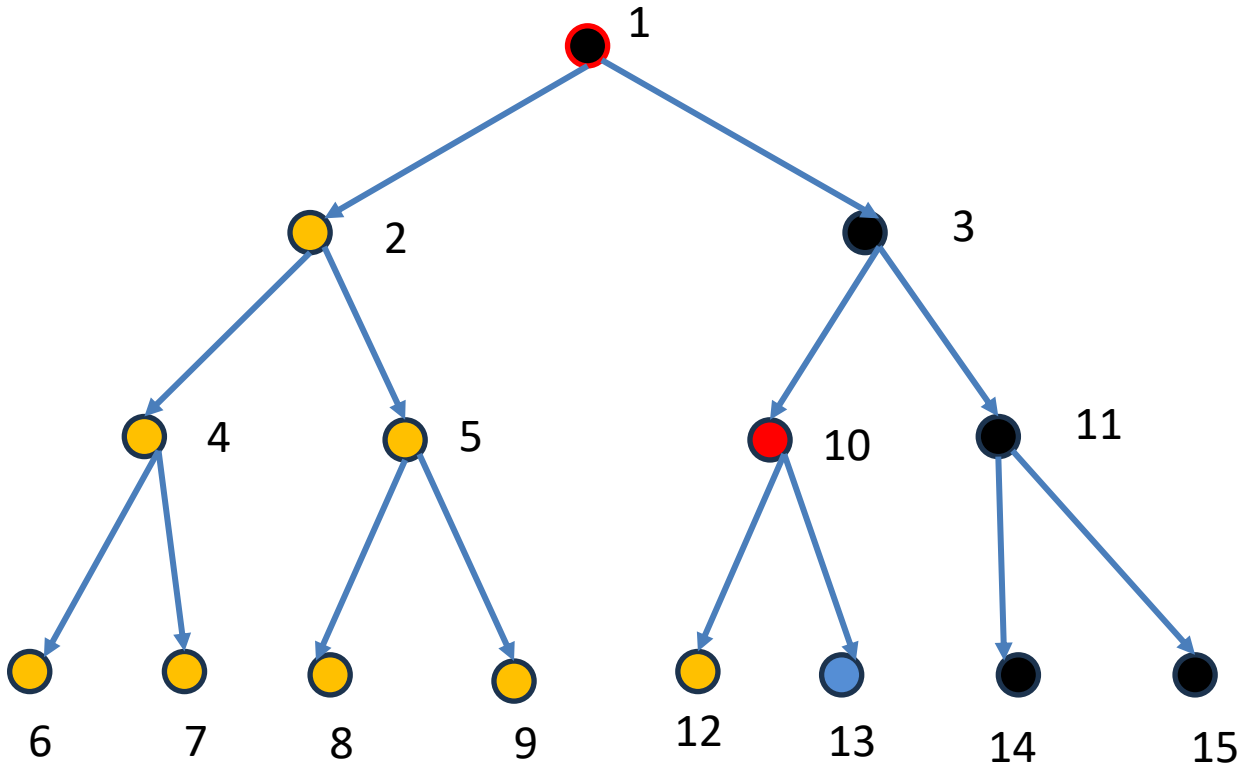
Blind Search: DFS



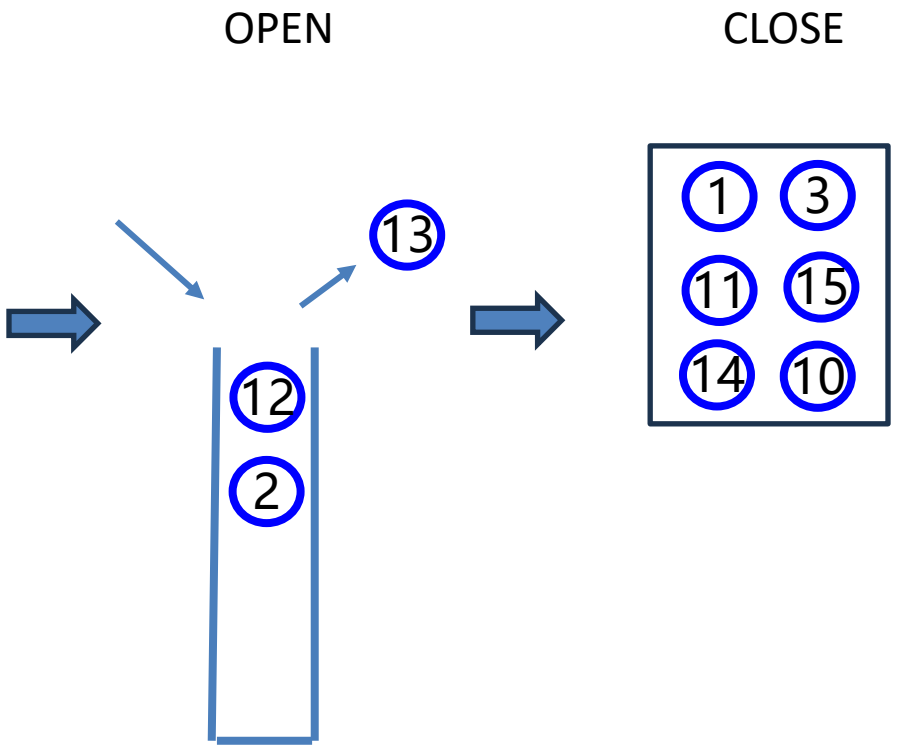


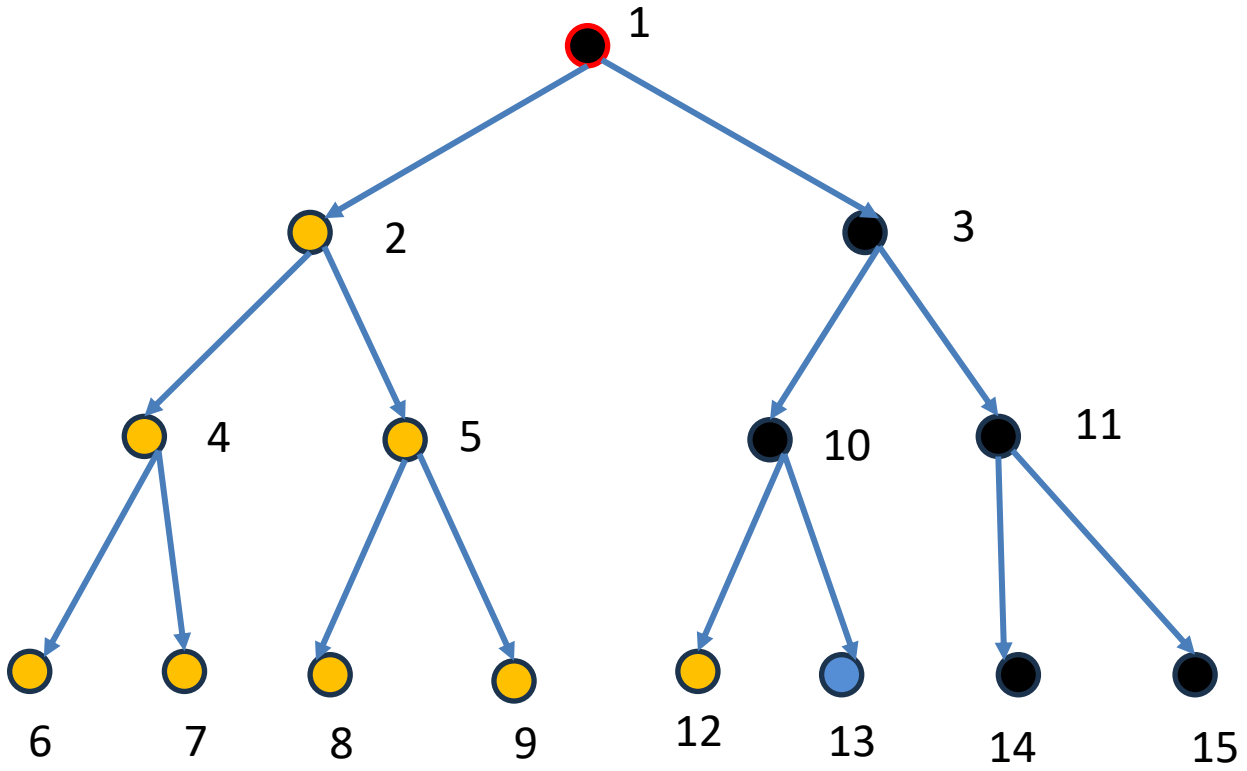
Blind Search: DFS





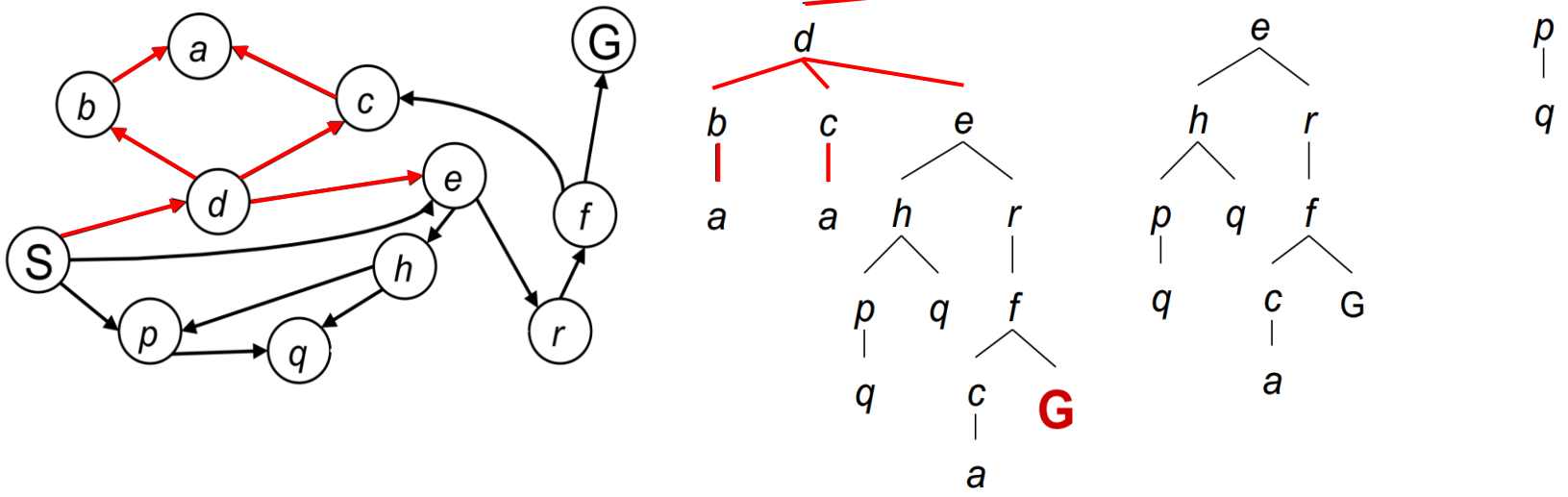
Blind Search: DFS



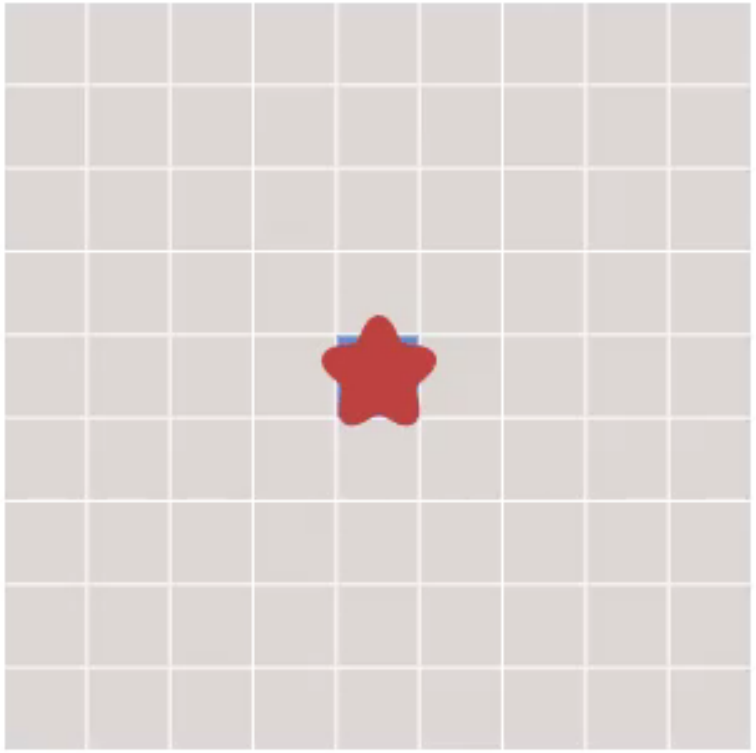


Depth First Search (DFS)

- Strategy: remove / expand the deepest node in the open table

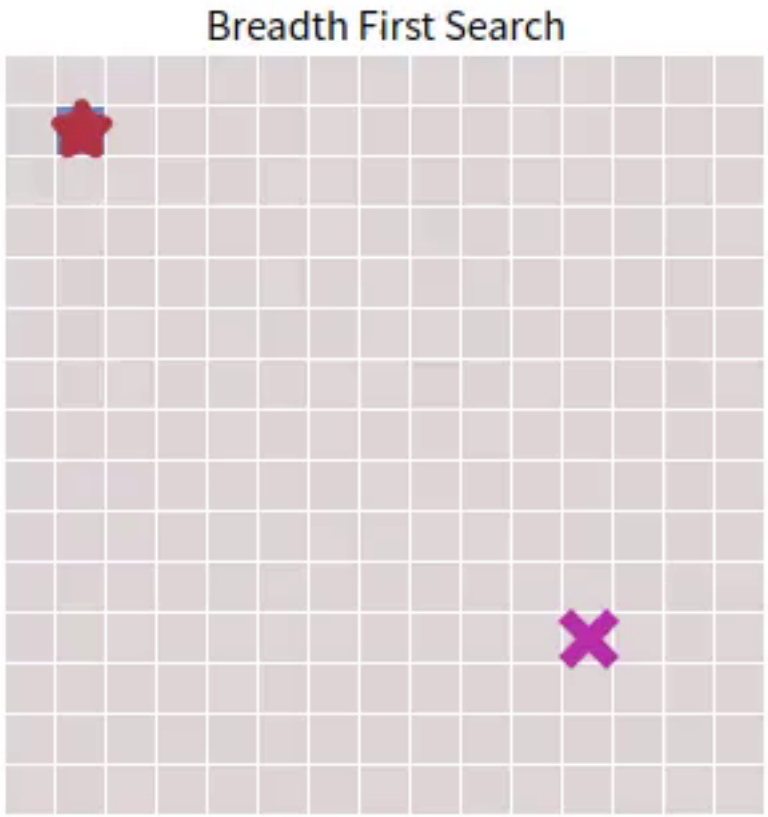


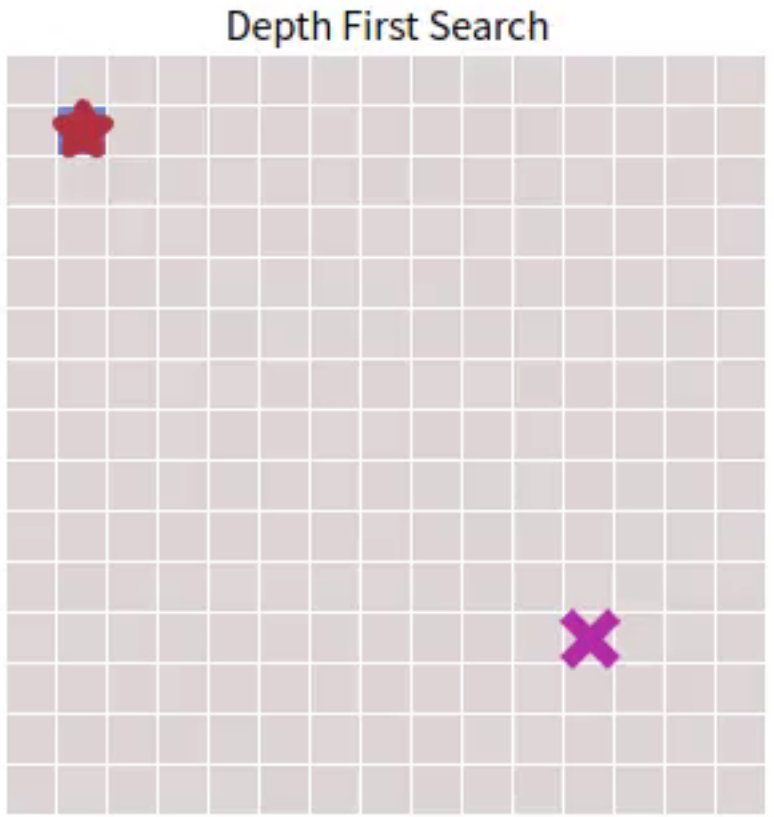
Depth First Search (DFS)



Courtesy: Amit Patel's Introduction to A*, Stanford

BFS vs. DFS

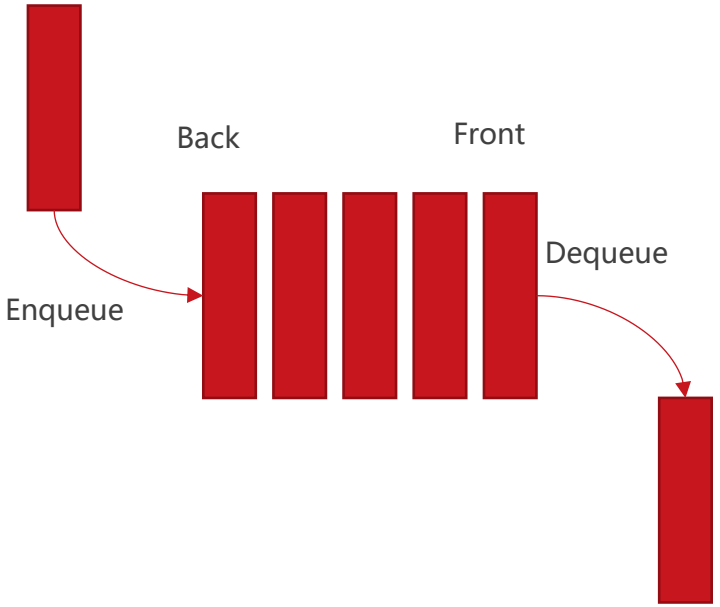




Graph Traversal

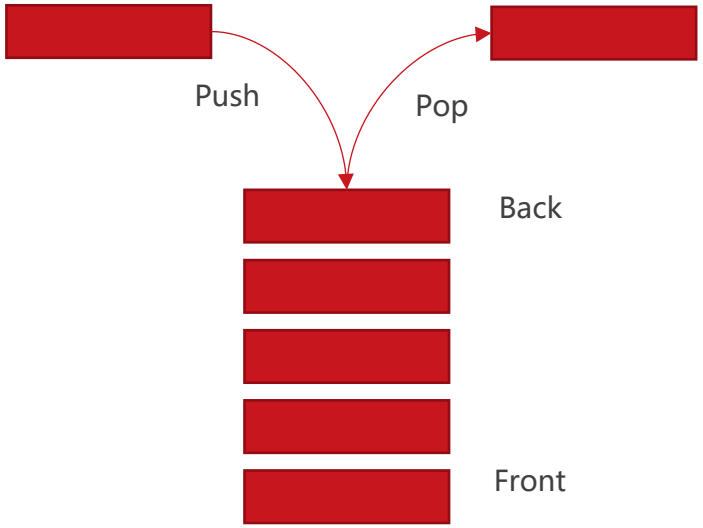
- Breadth First Search (BFS) vs. Depth First Search (DFS)

BFS uses “first in first out”



This is a queue

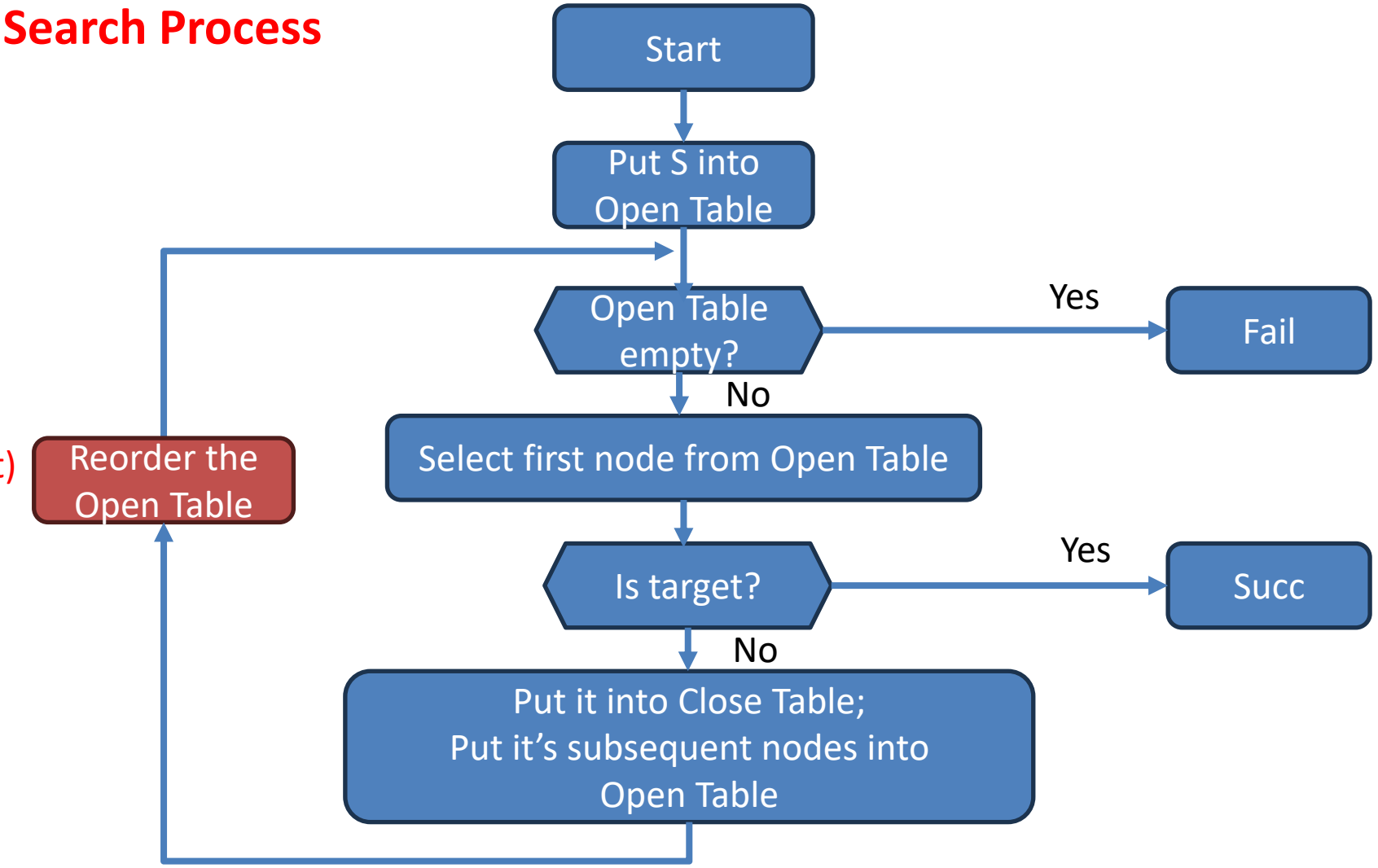
DFS uses “last in first out”



This is a stack

Search Problem

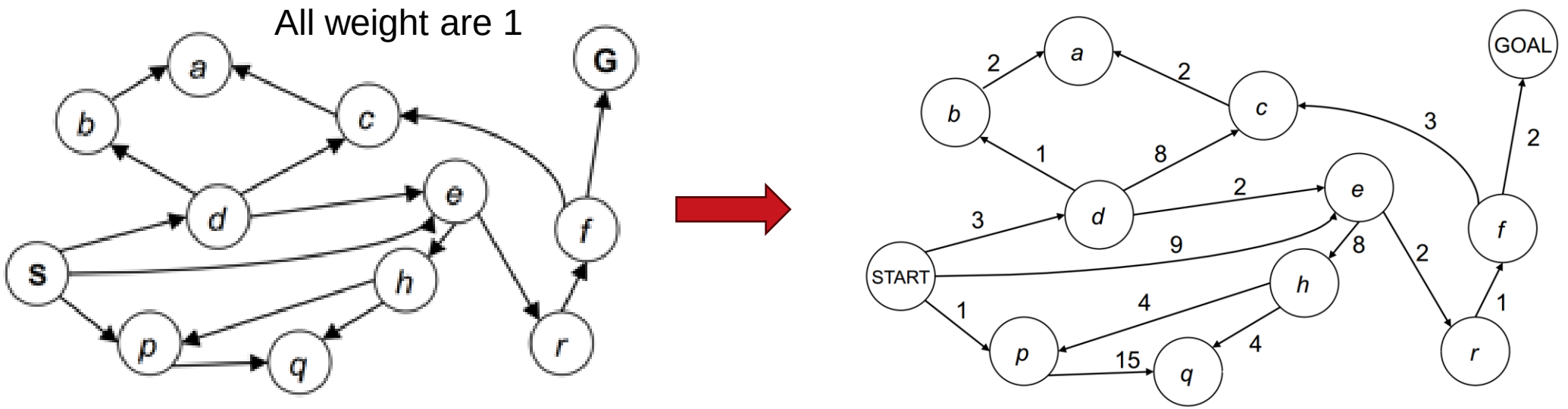
Flowchart of Graph Search Process



Different strategy:
BFS -> Queue (first in, first out)
DFS-> Stack (last in, first out)

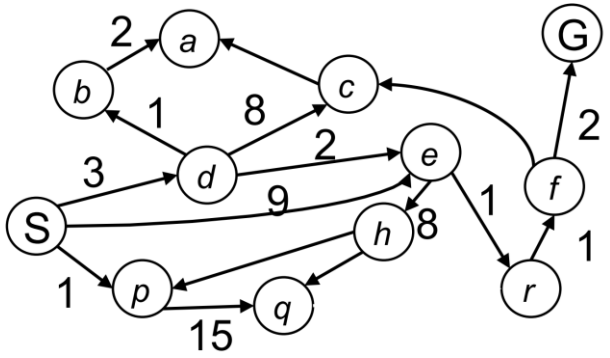
Dijkstra’s Algorithm

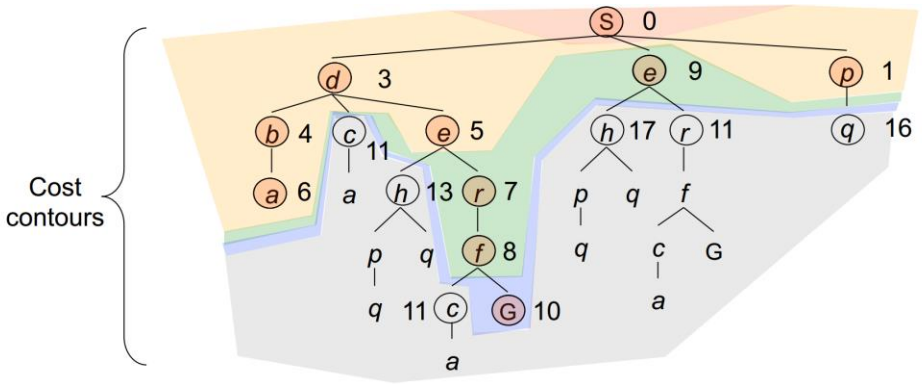
- A practical search problem has a cost “C” from a node to its neighbor
  - Length, time, energy, etc.
- When all weight are 1, BFS finds the optimal solution
- For general cases, how to find the least-cost path as soon as possible?



Dijkstra’s Algorithm

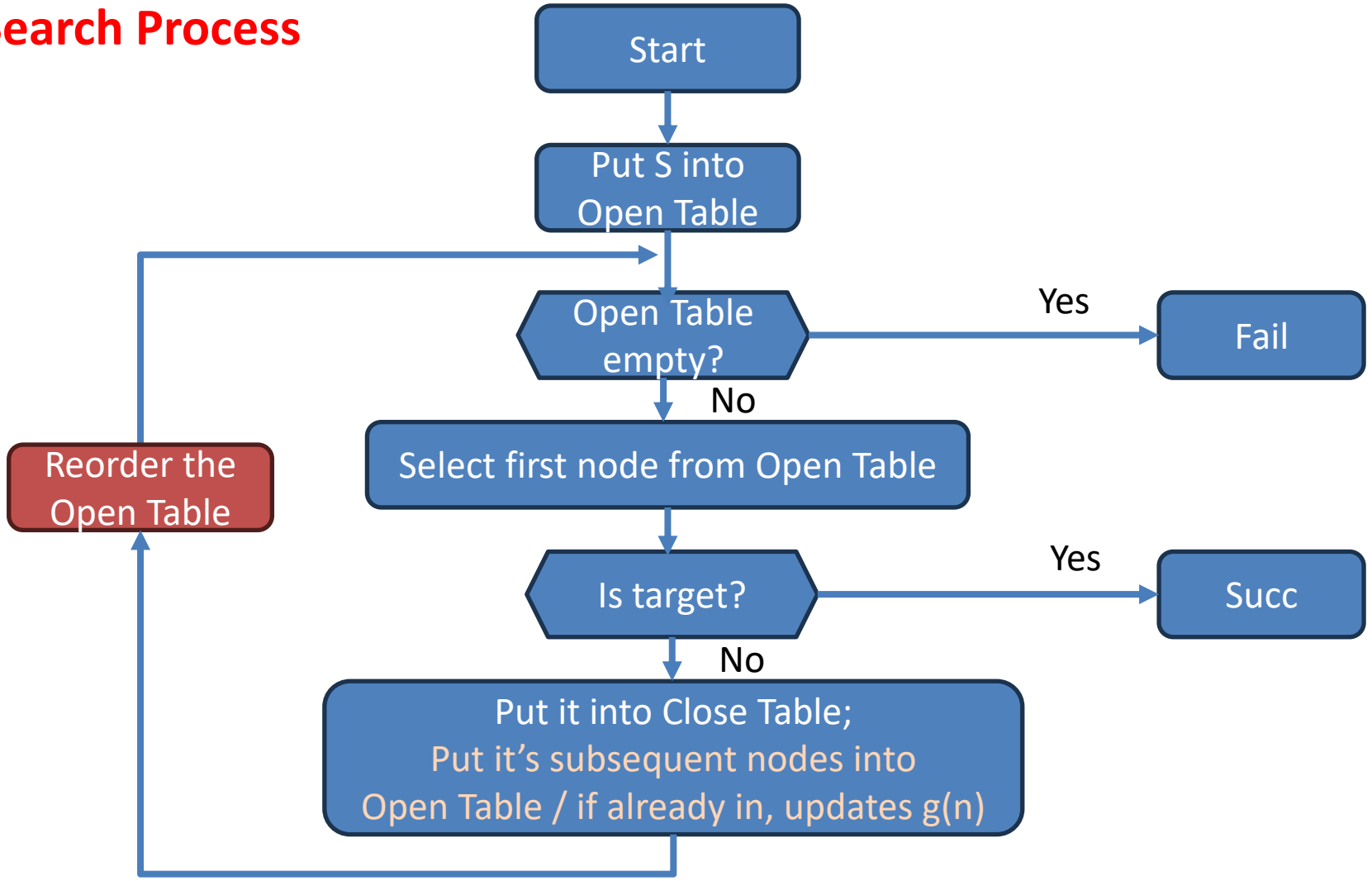
- Strategy: Prioritize searching the node with cheapest accumulated cost, g(n)
  - g(n): The current best estimates of the accumulated cost from the start state to node “n”





Search Problem

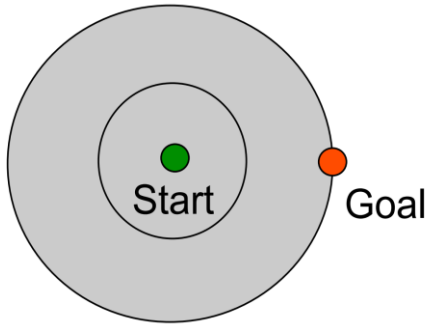
Flowchart of Graph Search Process

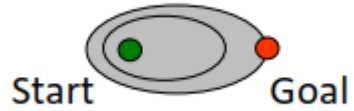


Different strategy:
BFS -> first in, first out;
DFS-> last in, first out;
Dijkstra ->cheapest accumulated cost g(n)

priority_queue

Blind Search v.s. Heuristic Search



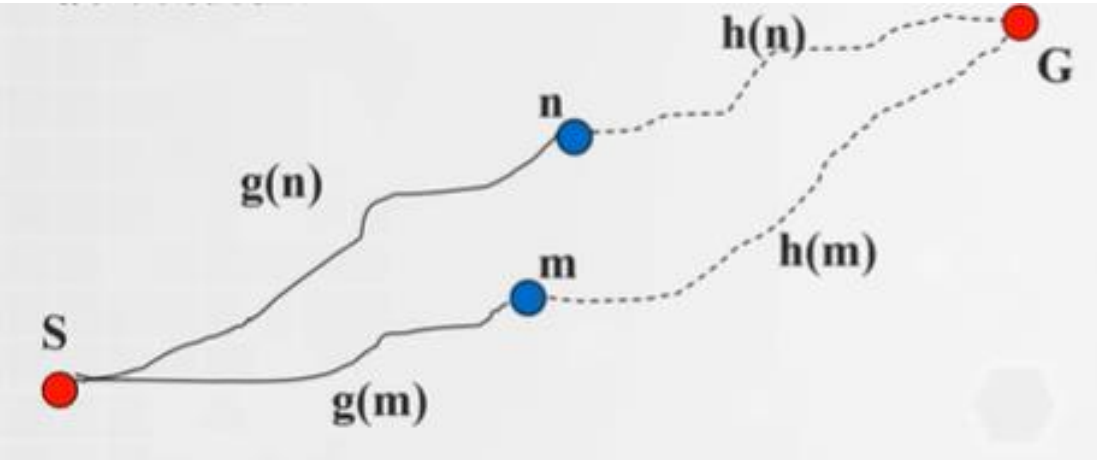


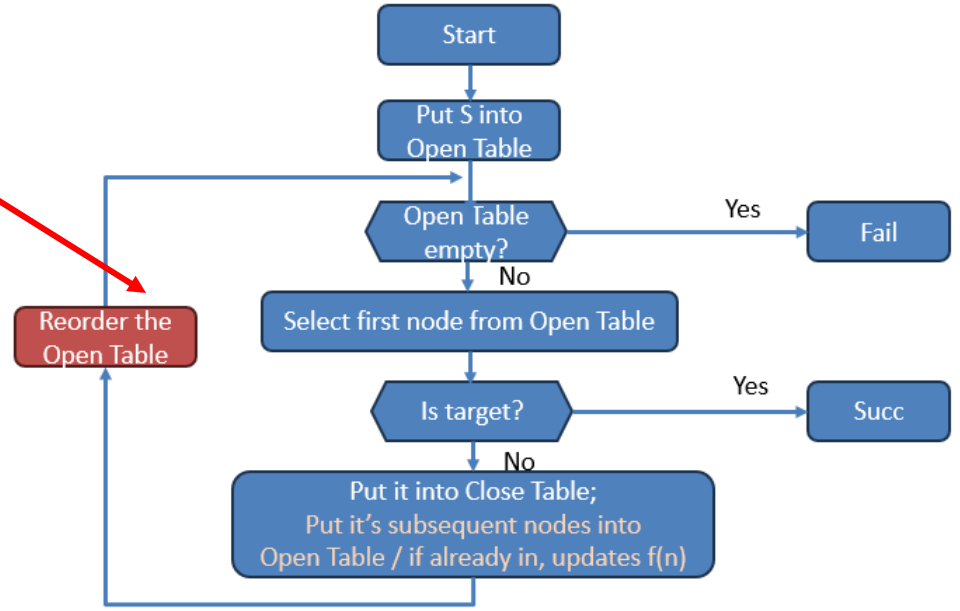
Content

A algorithm

Evaluation (cost) function *f(n)=g(n)+h(n)*, represents the estimated cost of passing through node x.

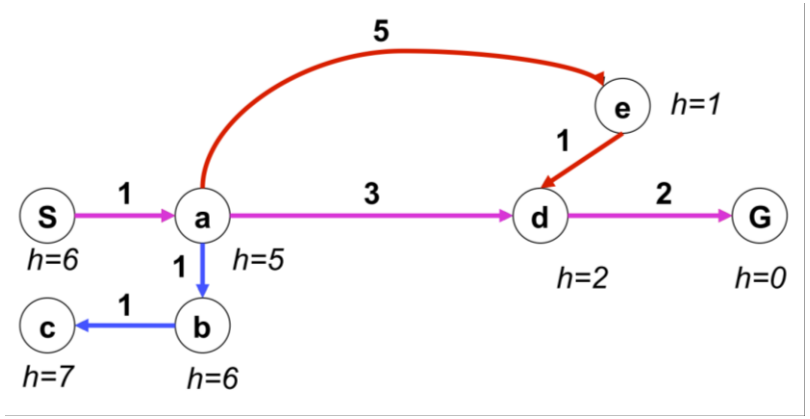
- *g(n)*: accumulated cost, cost from start point to point n
- *h(n)*: future cost, heuristic cost from n to goal point





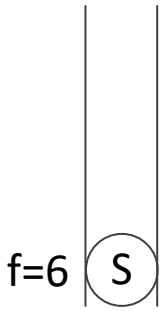
A Example

f(n)=g(n)+h(n)

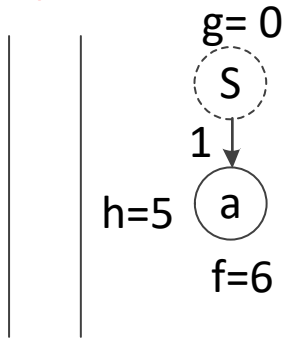


`priority_queue`

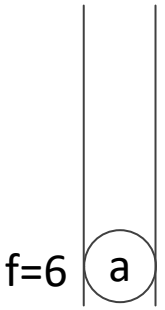
Initial queue



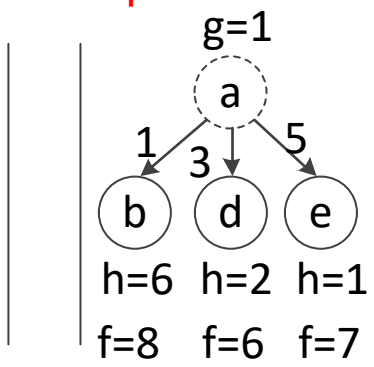
1st dequeue & expand



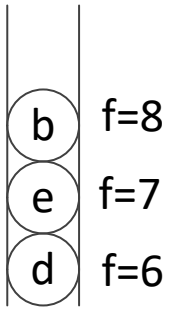
1st enqueue



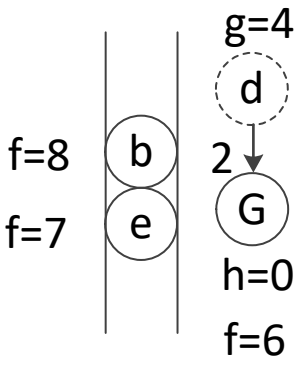
2nd dequeue & expand



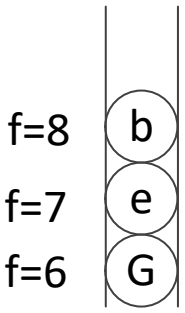
2nd enqueue



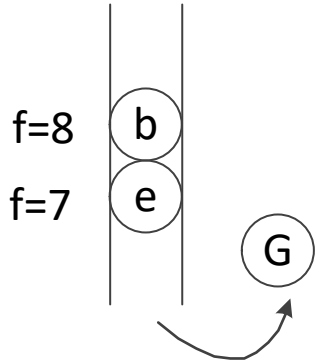
3rd dequeue & expand



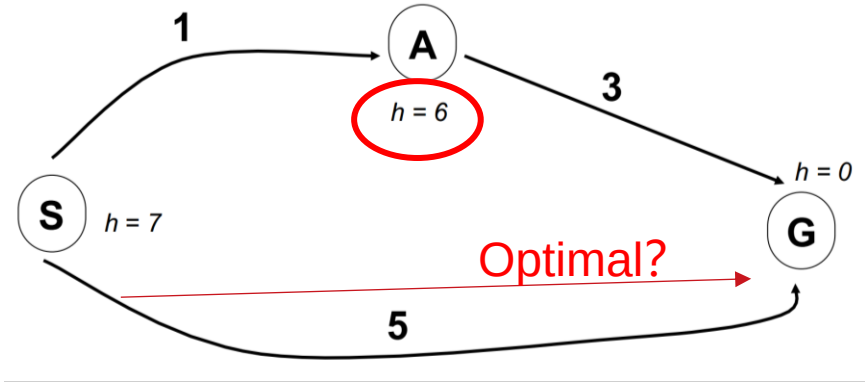
3rd enqueue

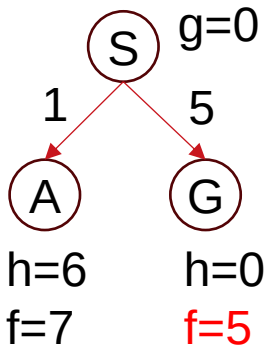


4th dequeue



A Optimality

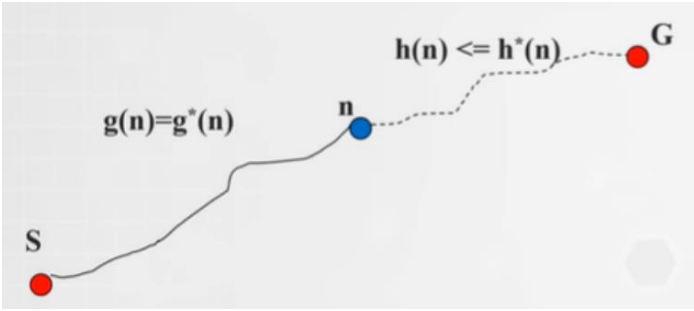




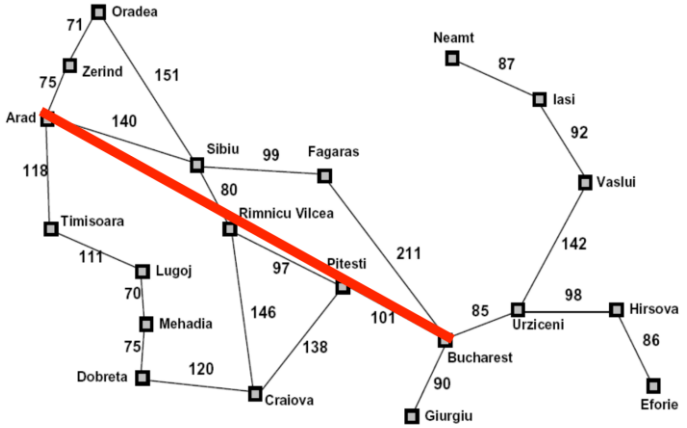
- What went wrong?
- For node A: heuristic value (estimated least cost to goal) > real
- We need the estimate to be less than actual least cost to goal (i.e. goal cost) for all nodes!

A -> A*

- For A*:
  - h(n) <= h*(n) for all node “n”, where h*(n) is the true least cost to goal from node “n”
  - h(n) is a conservative (保守的) estimation
  - If there is a solution, the A* algorithm will always find the optimal solution.



- How to define a good h(n), that h(n) <=h*(n)
- Example:

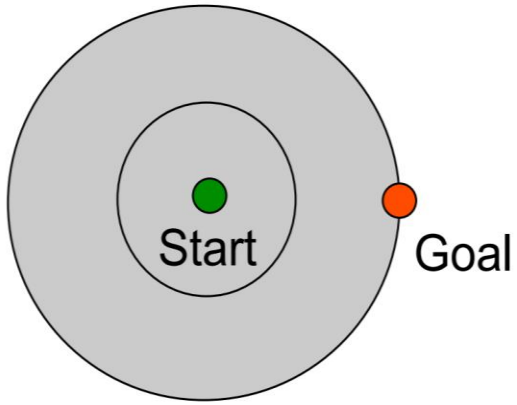


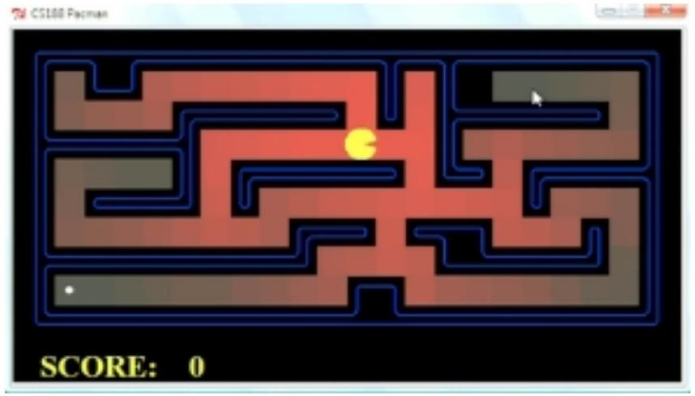
Euclidean distance (L2 norm)

The shortest distance between two points is a straight line. <=h*(n)

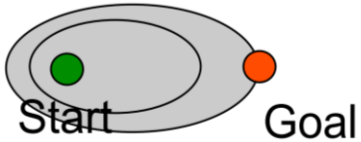
Dijkstra’s VS A*

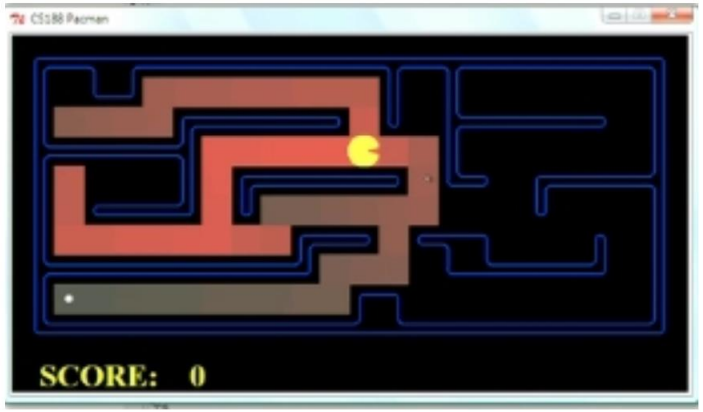
- Dijkstra’s algorithm expanded in all directions





- A* expands mainly towards the goal

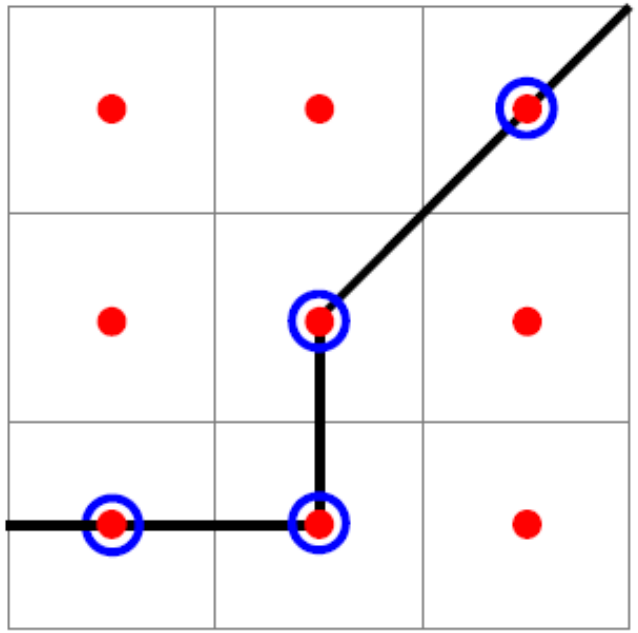


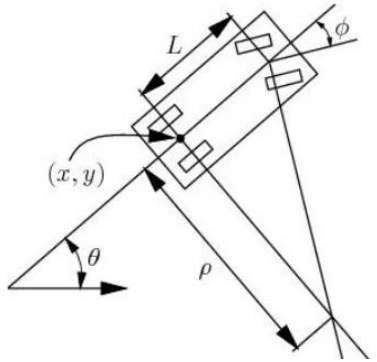


Content

Hybrid A*

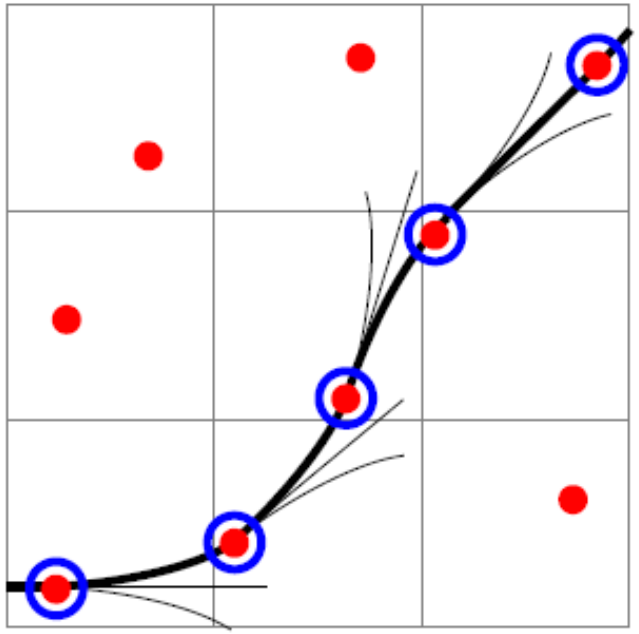
BFS / A*



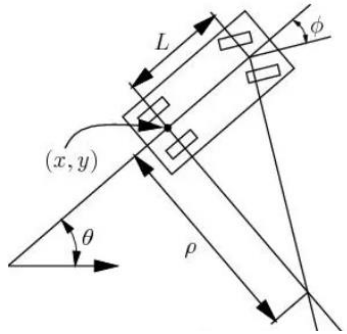


$$\dot{x} = v\cos(\theta)$$
$$\dot{y} = v\sin(\theta)$$
$$\dot{\theta} = \omega = \frac{v\tan(\varphi)}{L}$$

Hybrid A*



Hybrid A*

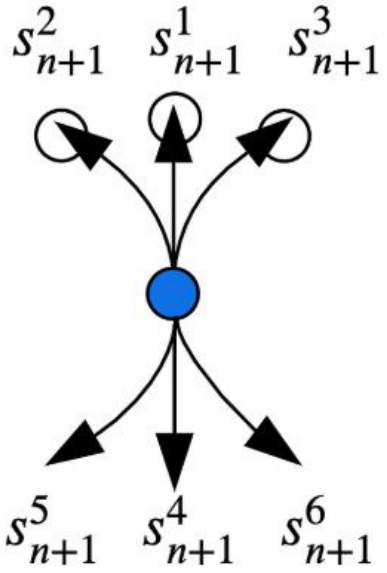


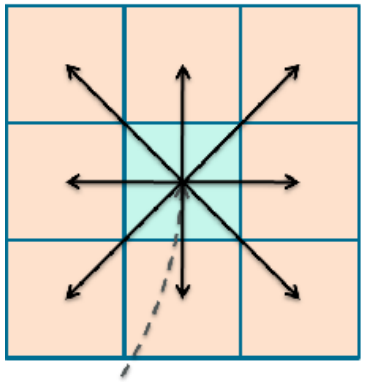
$$\dot{x} = v\cos(\theta)$$
$$\dot{y} = v\sin(\theta)$$
$$\dot{\theta} = \omega = \frac{v\tan(\varphi)}{L}$$

Expend neighbors

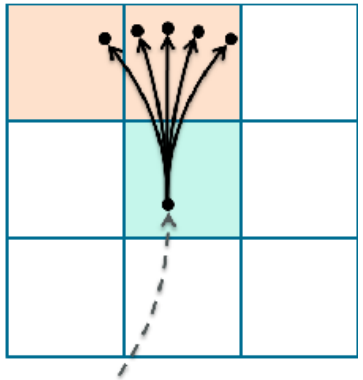
- Discrete sampling:

$$v \in [-1,1]$$
$$\varphi \in \left[-\frac{\pi}{4}, 0, \frac{\pi}{4}\right]$$





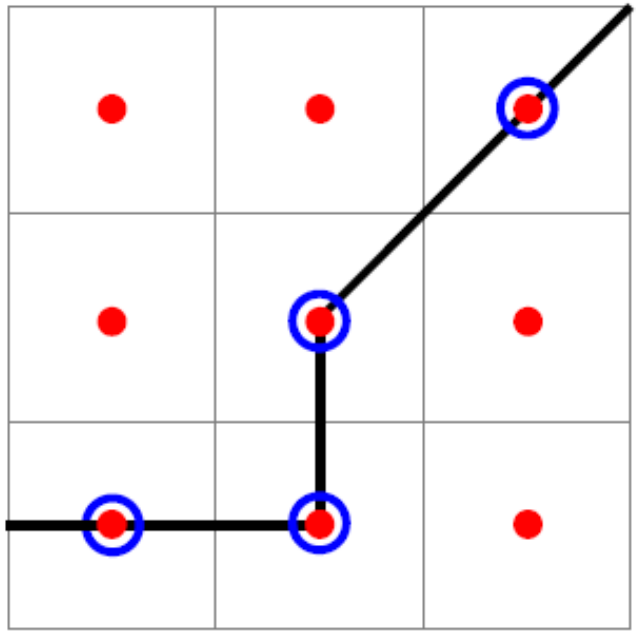
(a) regular A*



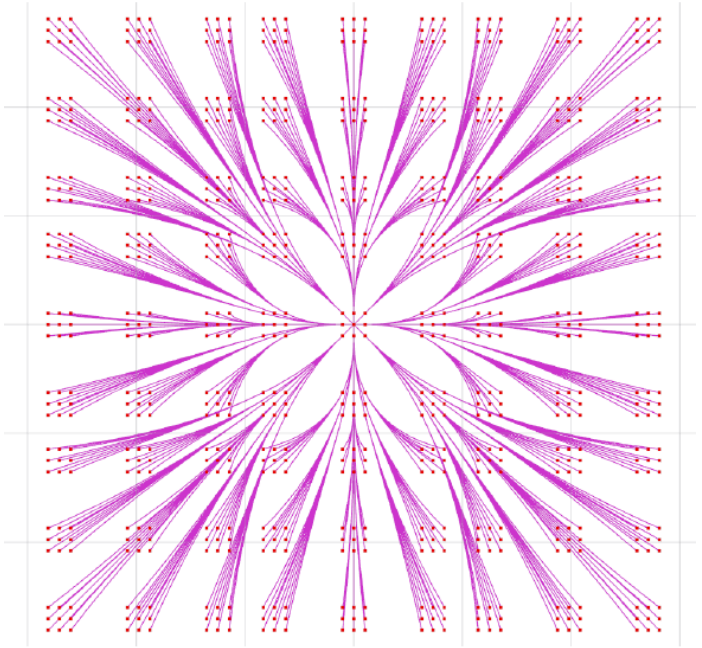
(b) Hybrid A*

Hybrid A*

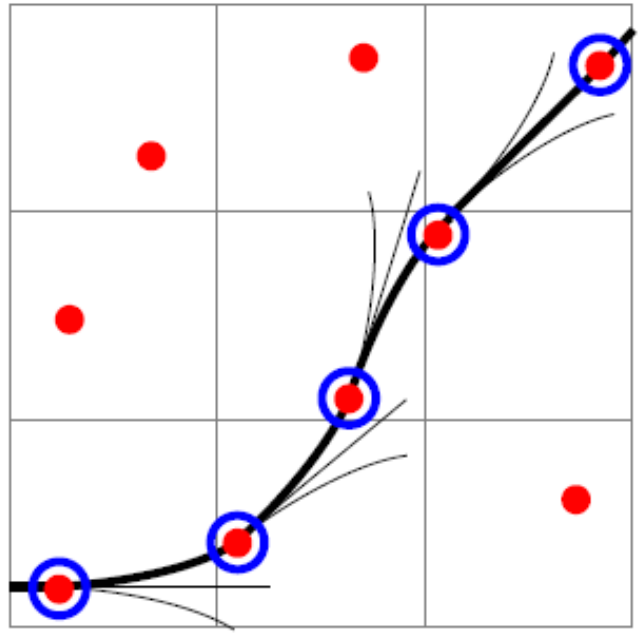
A*



Motion model

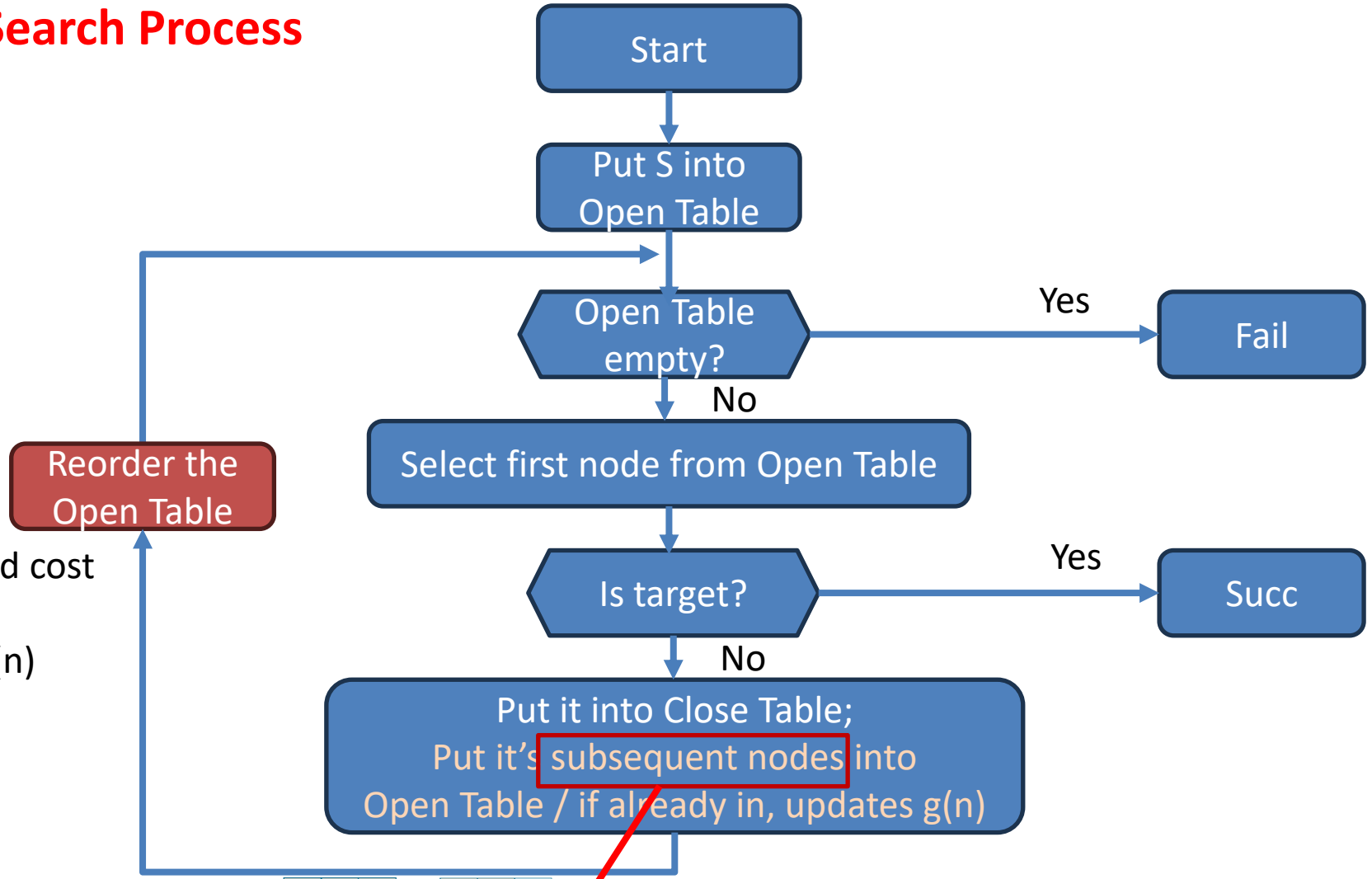


Hybrid A*



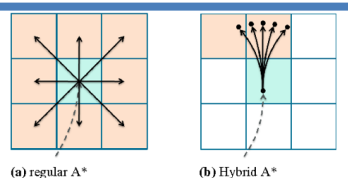
Search Flow

Flowchart of Graph Search Process



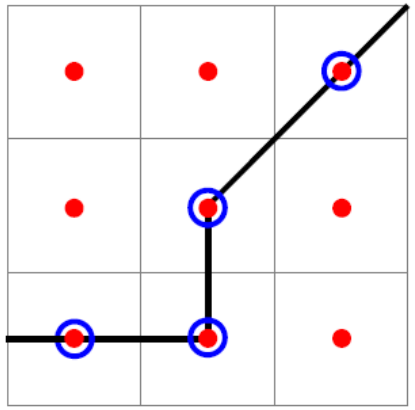
Different strategy:
BFS -> first in, first out;
DFS-> last in, first out;
Dijkstra ->cheapest accumulated cost g(n)
A* ->cheapest cost f(n)=g(n)+h(n)
Hybrid A*->cheapest cost f(n)=g(n)+h(n)

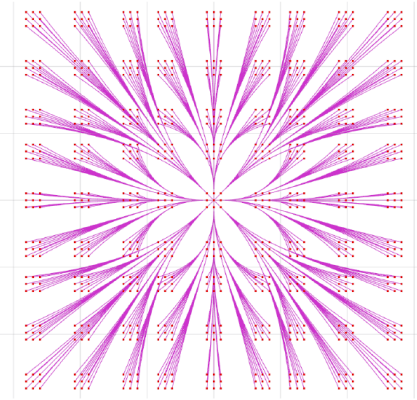
priority_queue



Define subsequent nodes with motion model

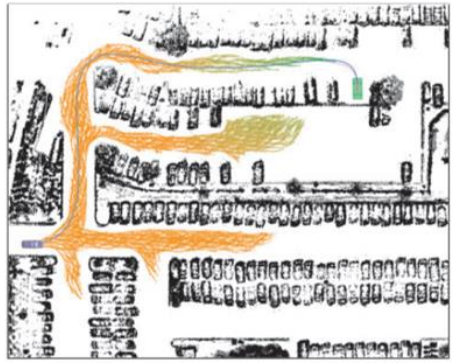
Tricks for Hybrid A*

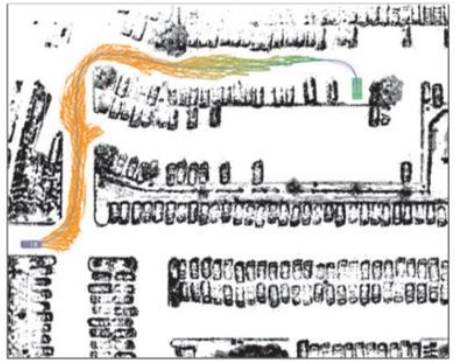




Problems: low efficiency

- The search space becomes larger
  - $(x, y)\ ->\ (x, y, \theta)$
  - Limited grid -> infinite node





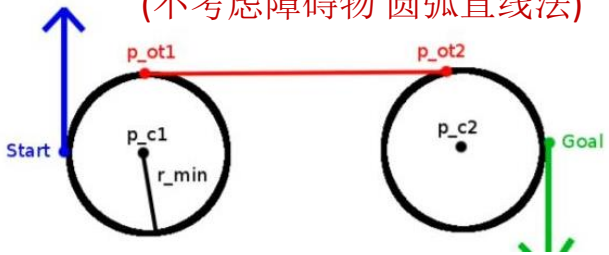
Hybrid A*

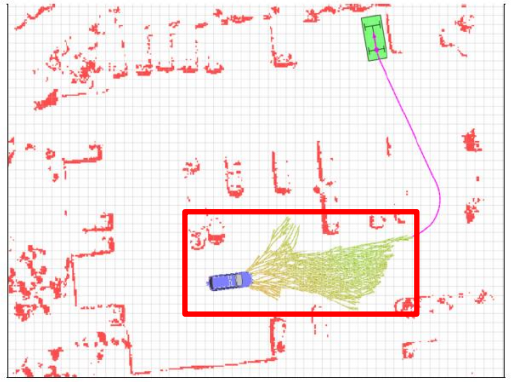
Hybrid A* based on shortest path

Tricks

- Choose smart heuristic function
  - Take BFS/A* result as heuristic function
- Mixed method
  - hybrid A* + RS/Dubins曲线

(不考虑障碍物 圆弧直线法)





Content

Assignment

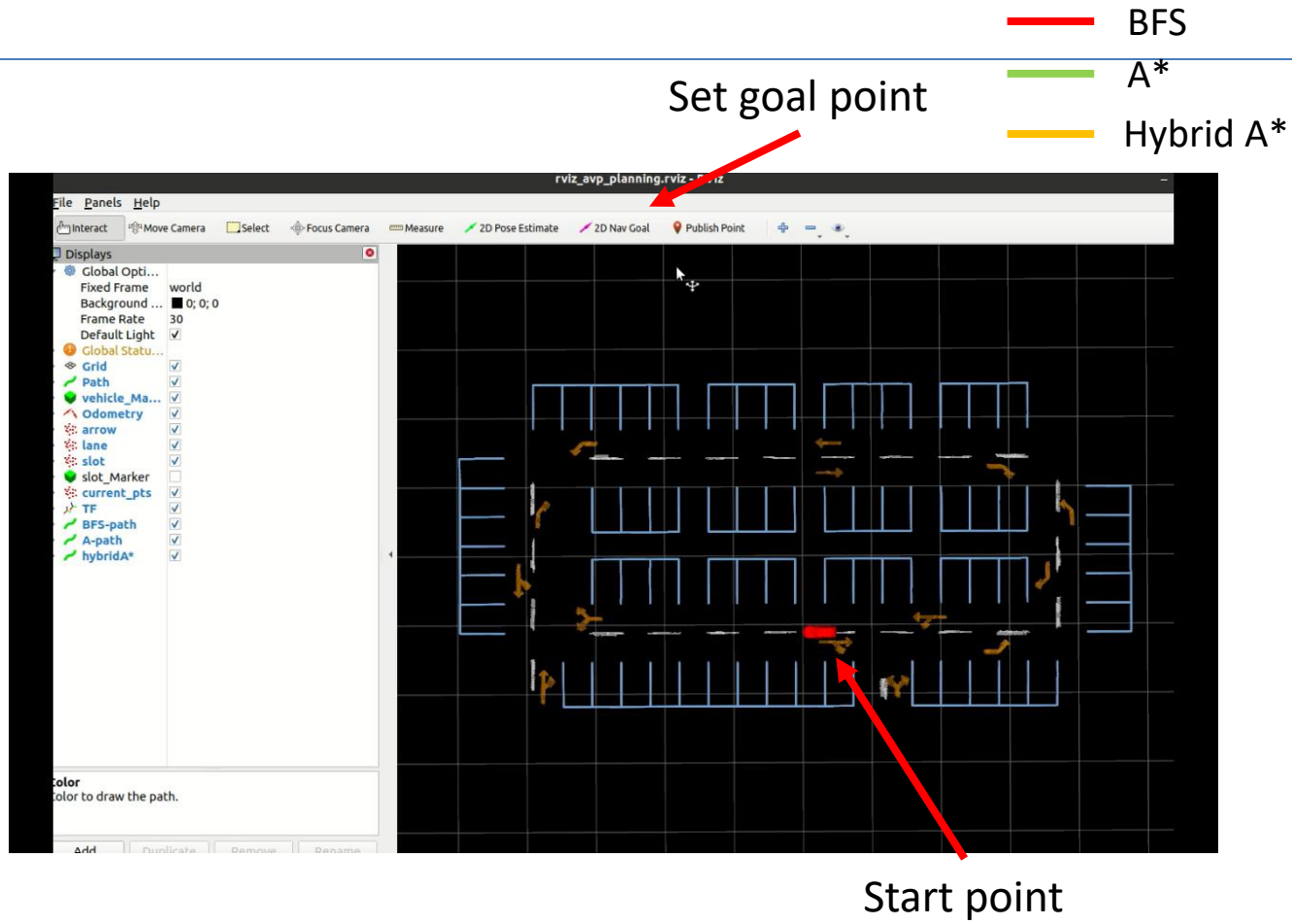
BFS/A*/Hybrid A* in ROS

- Complete functions in *planner.cpp*
  - Bfs()
  - AStar()
  - HybridAStar()

- catkin_make your workspace
  - rosrun avp_planning avp_planning

- Click the 2D Nav Goal on RVIZ, set an end point on the map

You will see the path from different algorithms on the map.



深蓝学院
shenlanxueyuan.com

感谢聆听！

Thanks for Listening