Graph Search, Part I

Howie Choset

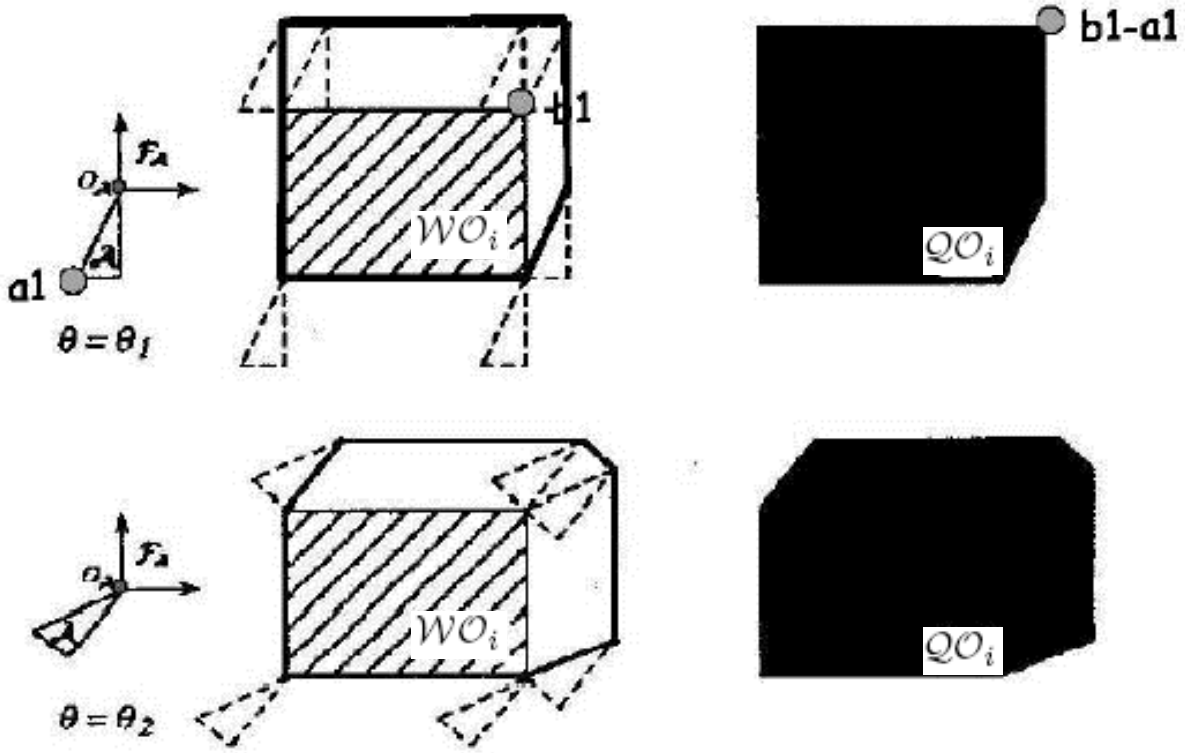
Introduction to Robotics

Happy President's Day

The Configuration Space

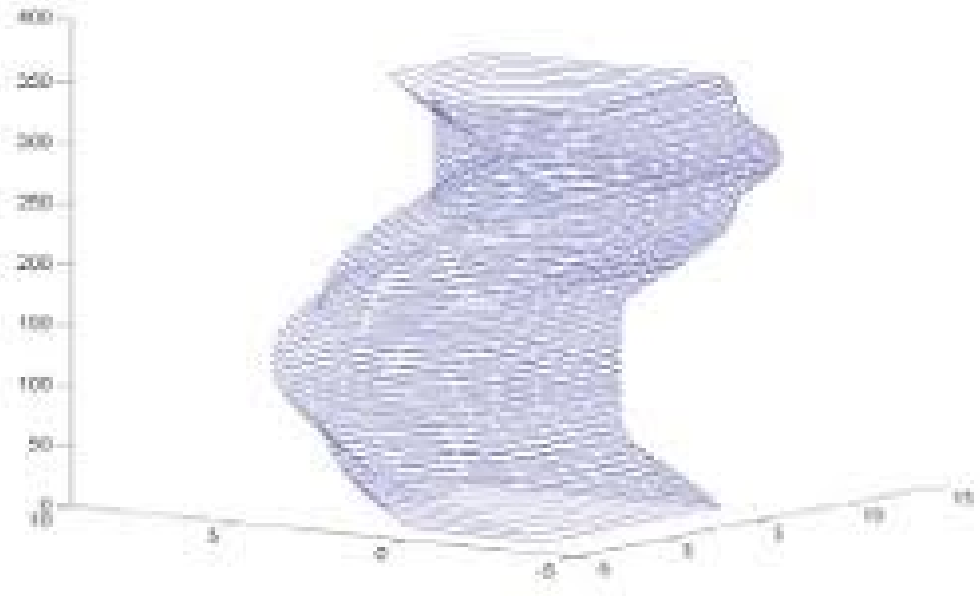
- What it is
  - A set of "reachable" areas constructed from knowledge of both the robot and the world
- How to create it
  - First abstract the robot as a point object. Then, enlarge the obstacles to account for the robot's footprint and degrees of freedom
  - In our example, the robot was circular, so we simply enlarged our obstacles by the robot's radius (*note the curved vertices*)

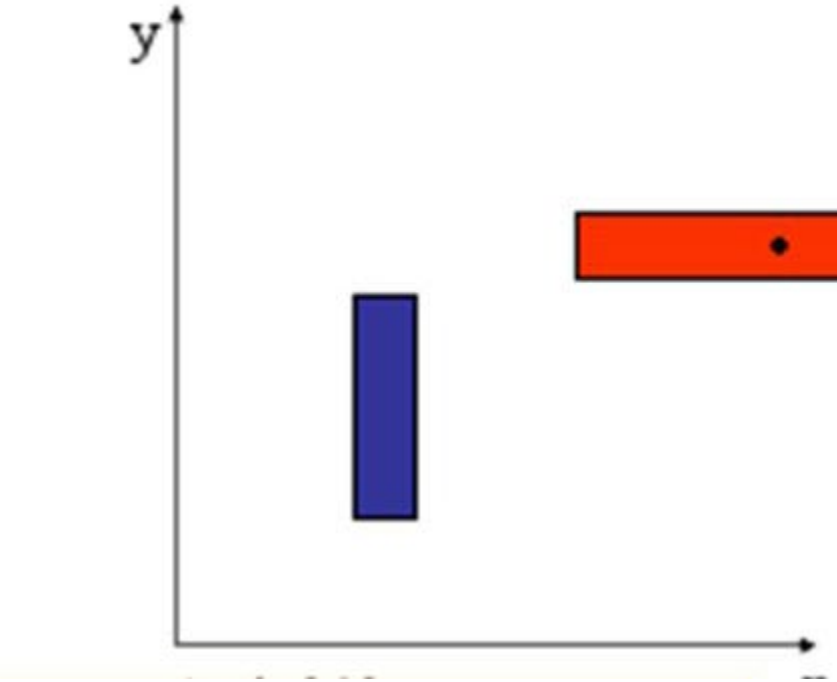
With Rotation: how much distance to rotate

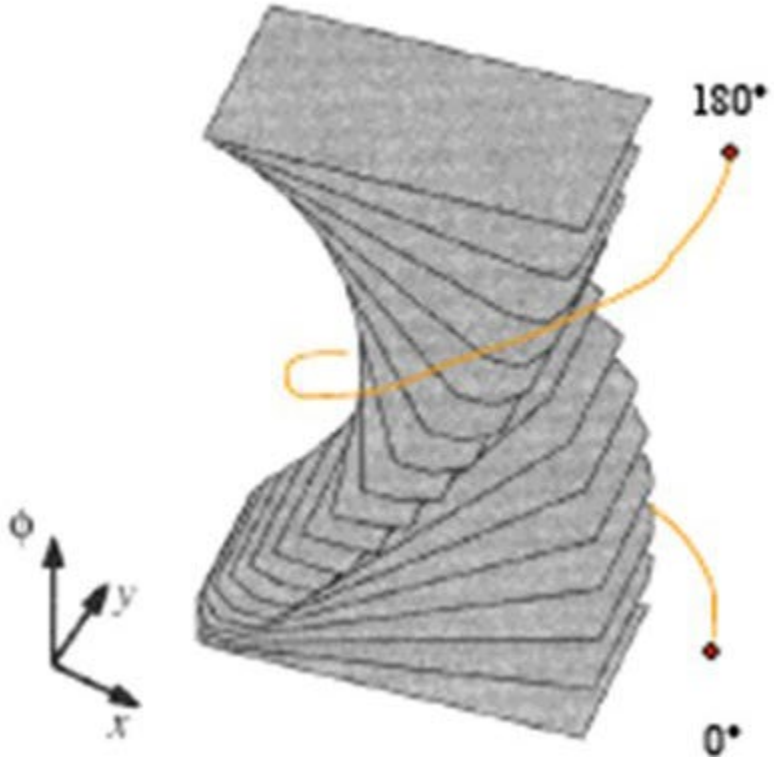


$$\mathcal{QO}_i = \{q \in \mathcal{Q} \mid R(q) \bigcap \mathcal{WO}_i \neq \emptyset\}.$$

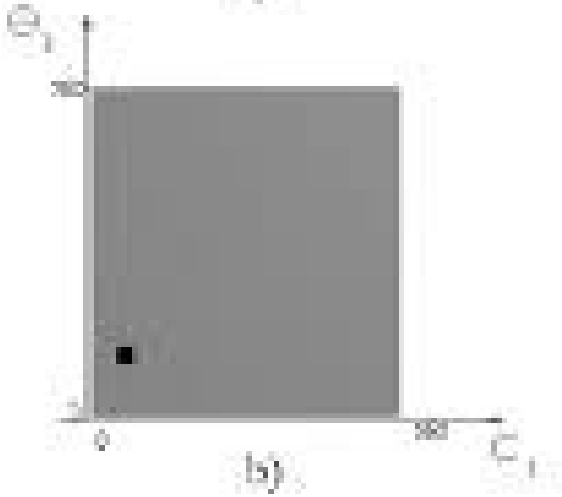
Pick a reference point…



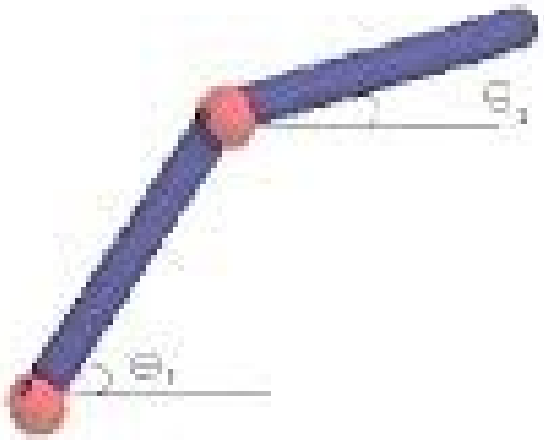

y


180°
0°
φ
y
x

c)



b)

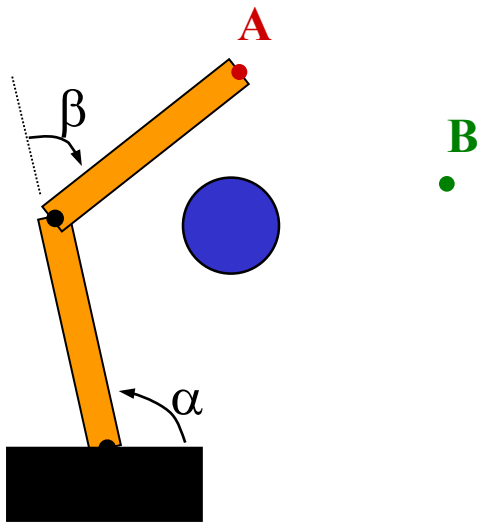


a)

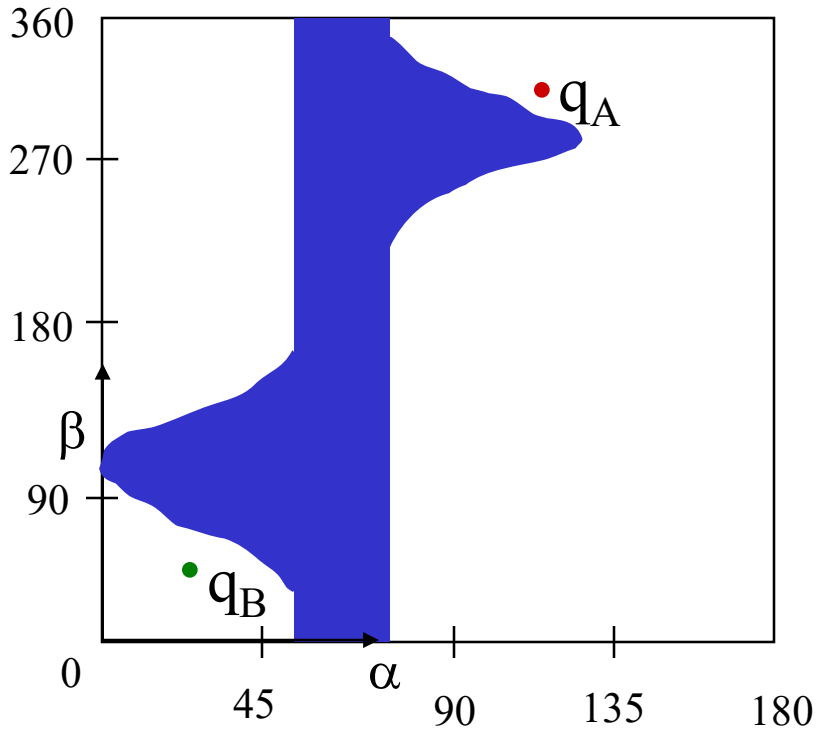
Configuration Space Obstacle

How do we get from A to B ?

Reference *configuration*



An obstacle in the robot's workspace

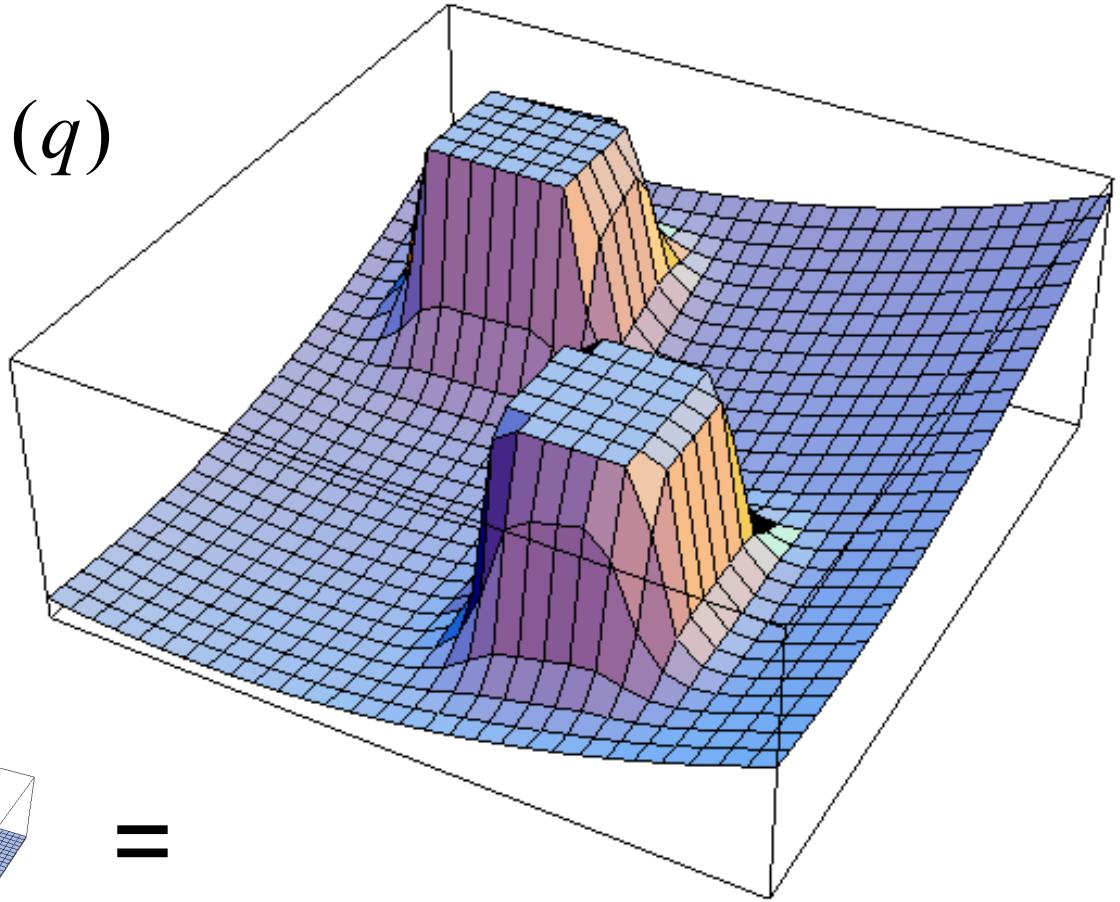


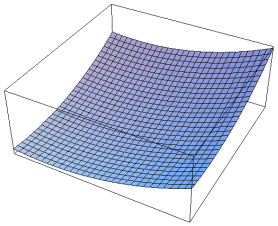
The C-space representation of this obstacle…

Total Potential Function

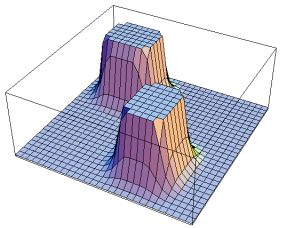
$$U(q) = U_{\text{att}}(q) + U_{\text{rep}}(q)$$

$$F(q) = -\nabla U(q)$$





+

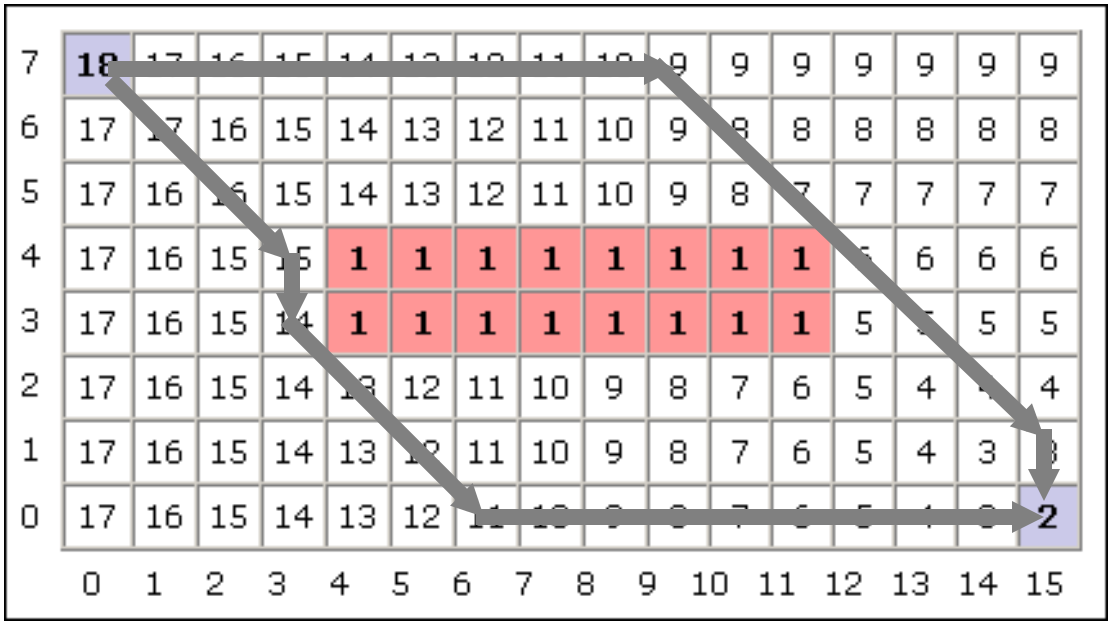


=

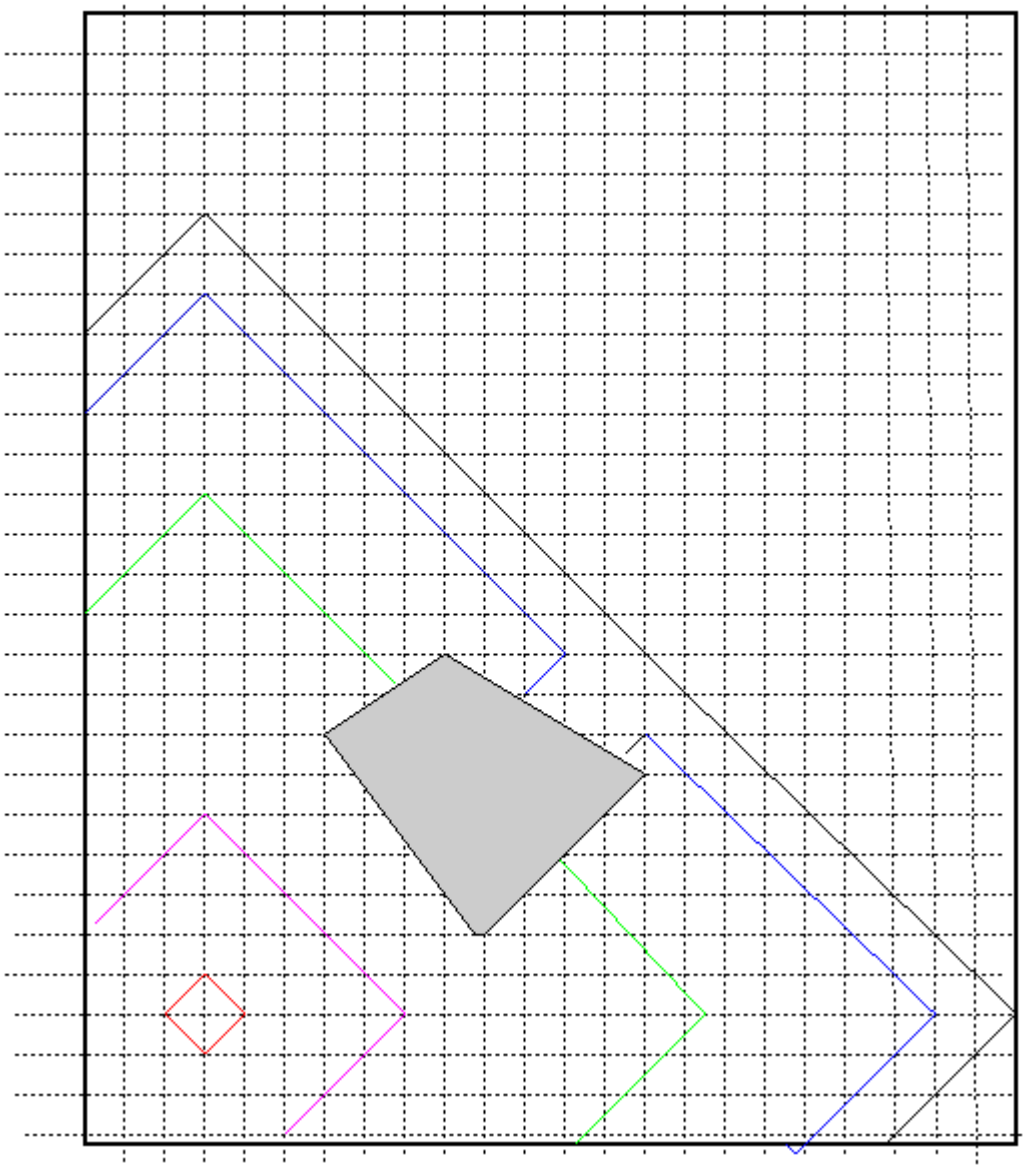
The Wavefront, Now What?

- To find the shortest path, according to your metric, simply always move toward a cell with a lower number
  - The numbers generated by the Wavefront planner are roughly proportional to their distance from the goal

Two possible shortest paths shown



This is really a Continuous Solution



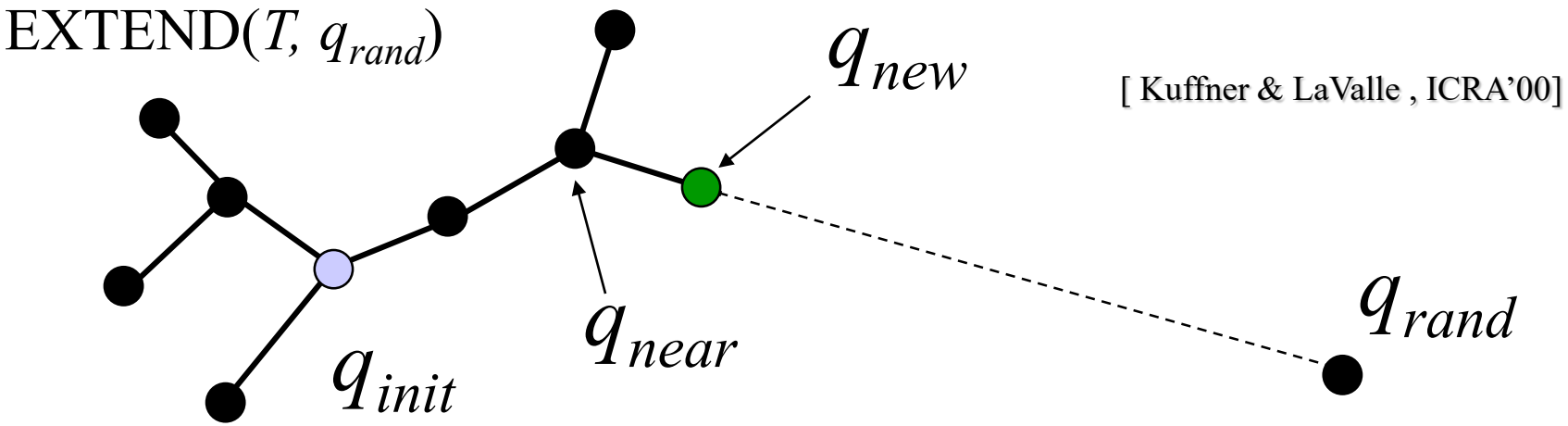
Not pixels

Waves bend

L1 distance

Path Planning with RRTs (Rapidly-Exploring Random Trees)

```
BUILD_RRT (q_init) {
  T.init(q_init);
  for k = 1 to K do
    q_rand = RANDOM_CONFIG();
    EXTEND(T, q_rand)
}
```



[Kuffner & LaValle , ICRA'00]

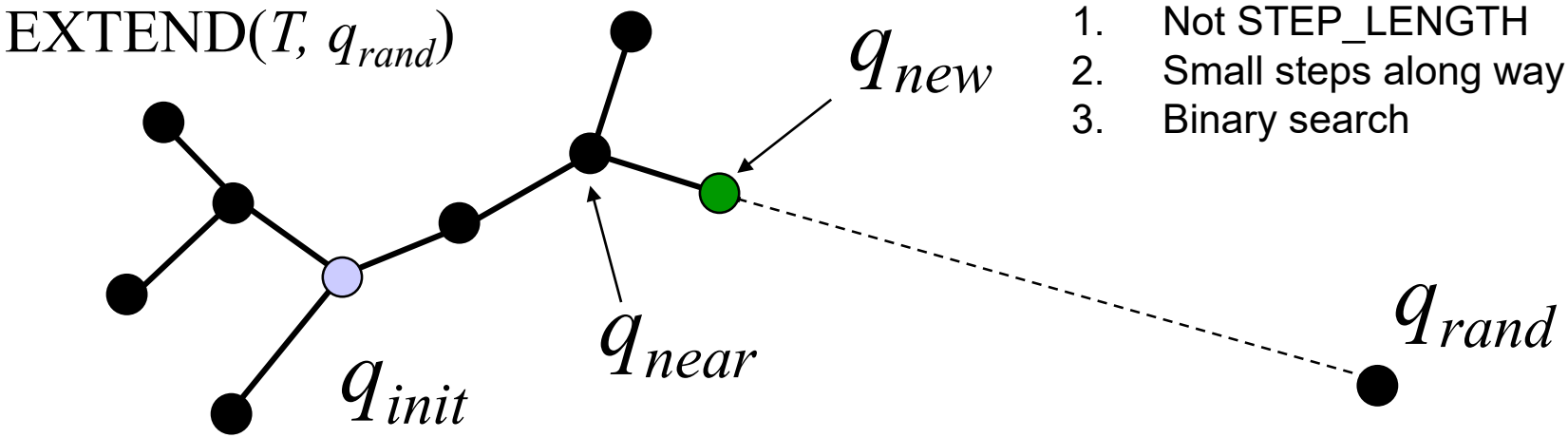
Path Planning with RRTs (Some Details)

```
BUILD_RRT (q_init) {
  T.init(q_init);
  for k = 1 to K do
    q_rand = RANDOM_CONFIG();
    EXTEND(T, q_rand)
}
```

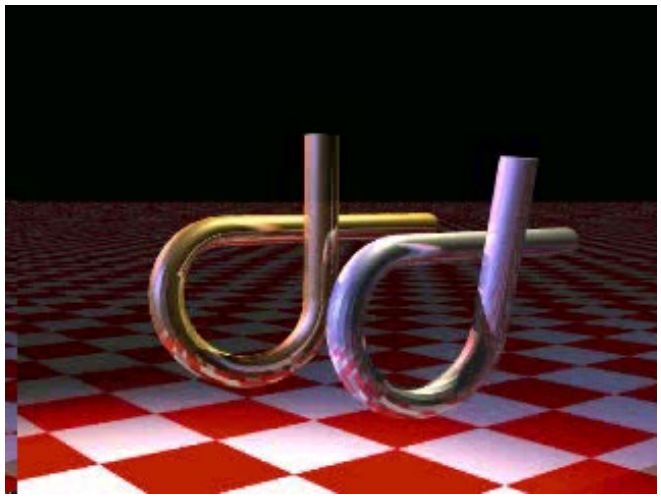
STEP_LENGTH: How far to sample
1. Sample just at end point
2. Sample all along
3. Small Step

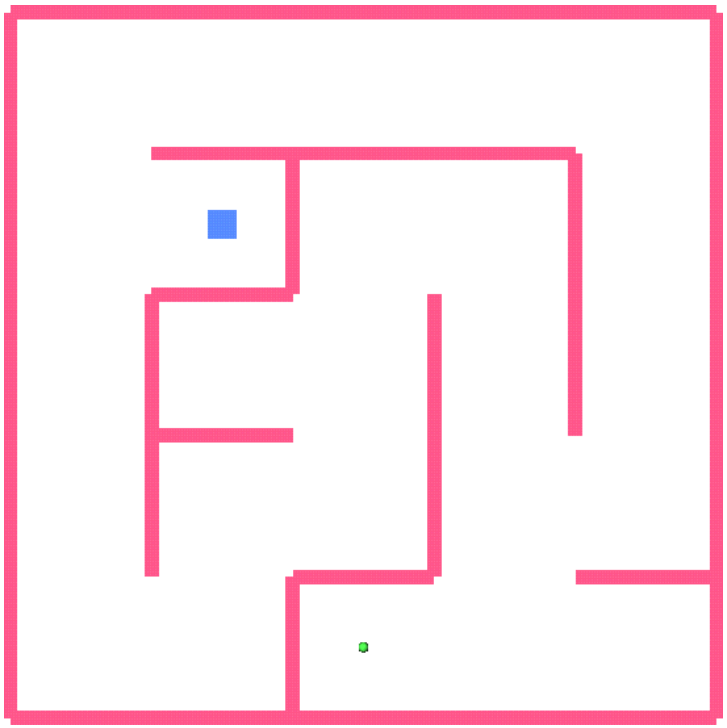
Extend returns
1. Trapped, cant make it
2. Extended, steps toward node
3. Reached, connects to node

STEP_SIZE
1. Not STEP_LENGTH
2. Small steps along way
3. Binary search

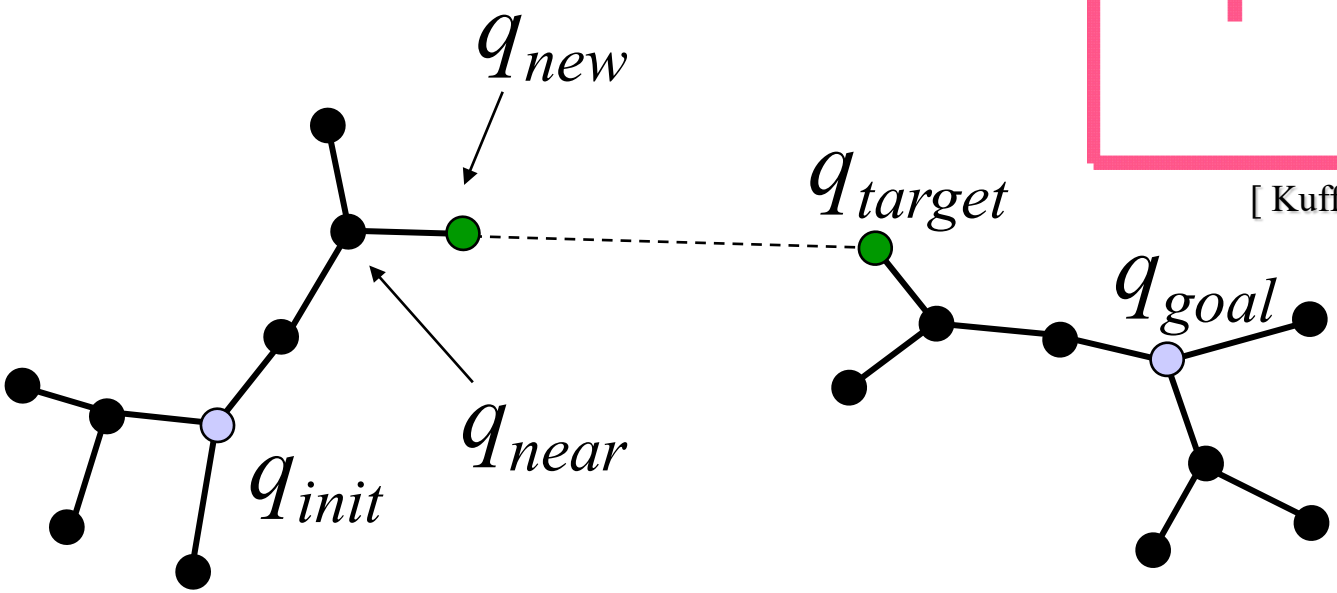


Grow two RRTs towards each other





[Kuffner, LaValle ICRA '00]



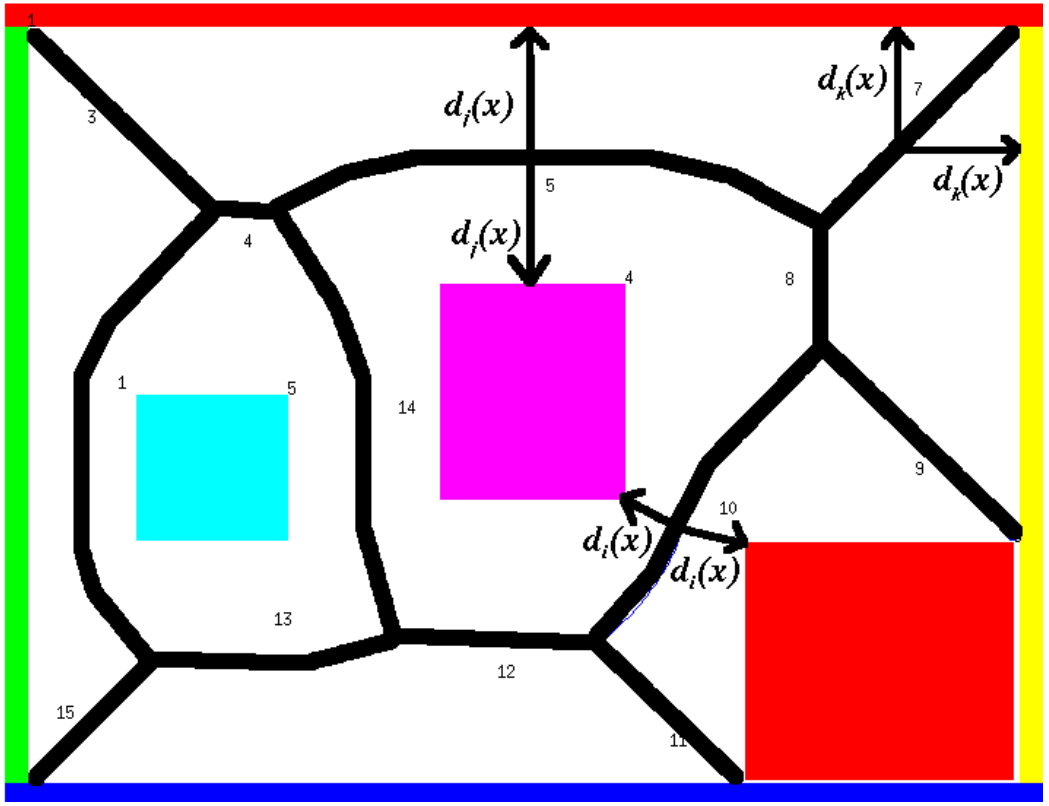
Map-Based Approaches: Roadmap Theory

- Properties of a roadmap:
  - Accessibility: there exists a collision-free path from the start to the road map
  - Departability: there exists a collision-free path from the roadmap to the goal.
  - Connectivity: there exists a collision-free path from the start to the goal (on the roadmap).

- **a roadmap exists ⇔ a path exists**
- **Examples of Roadmaps**
  - **Generalized Voronoi Graph (GVG)**
  - **Visibility Graph**

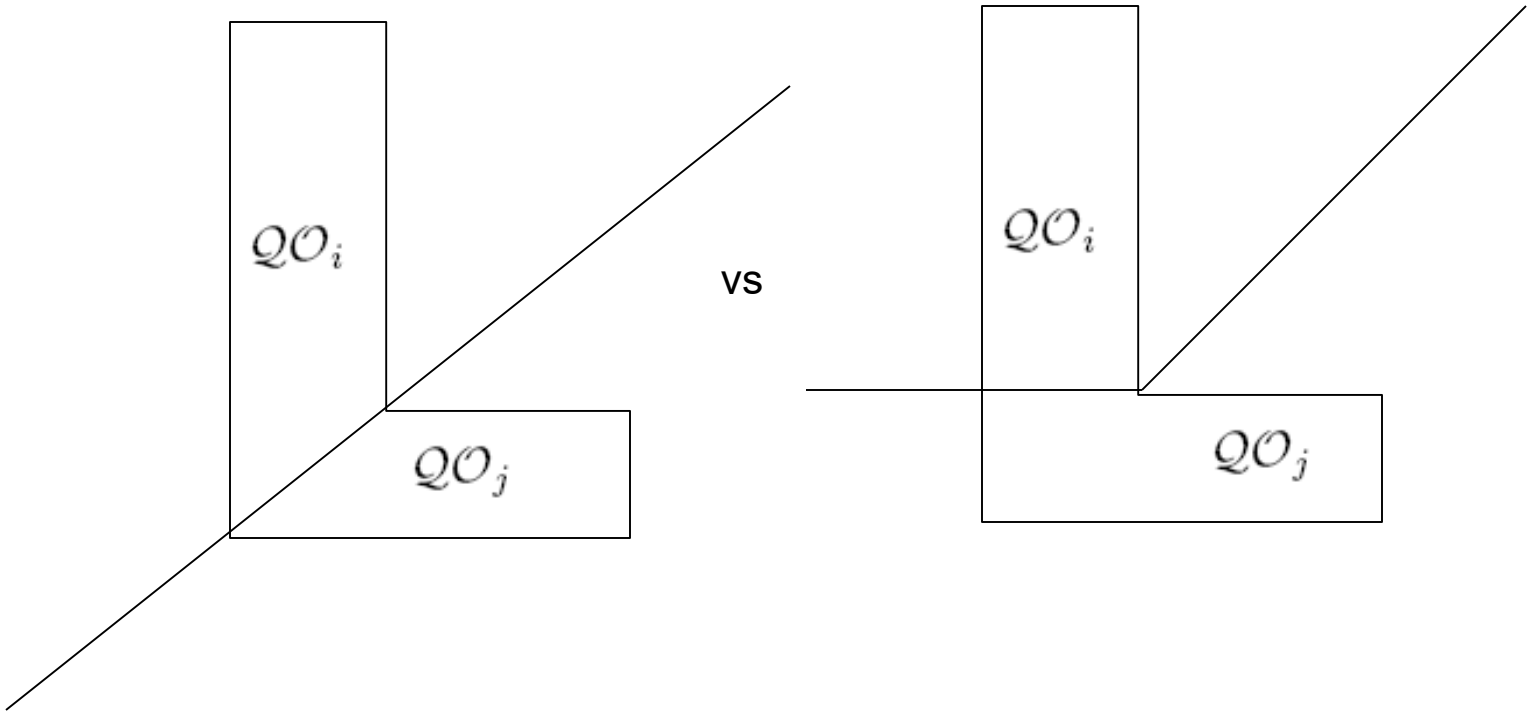
Roadmap: GVG

- A GVG is formed by paths equidistant from the two closest objects
- *Remember "spokes", start and goal*

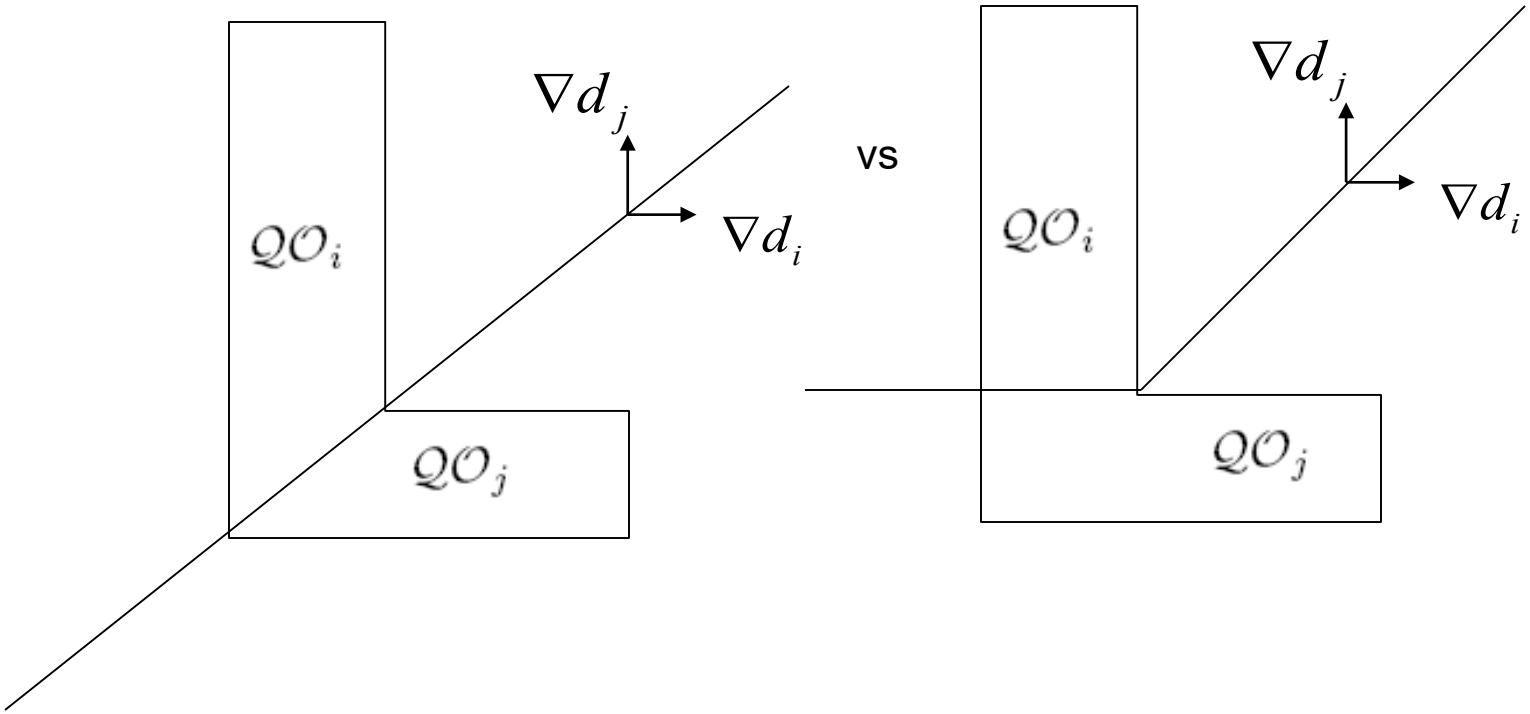


- This generates a very safe roadmap which avoids obstacles as much as possible

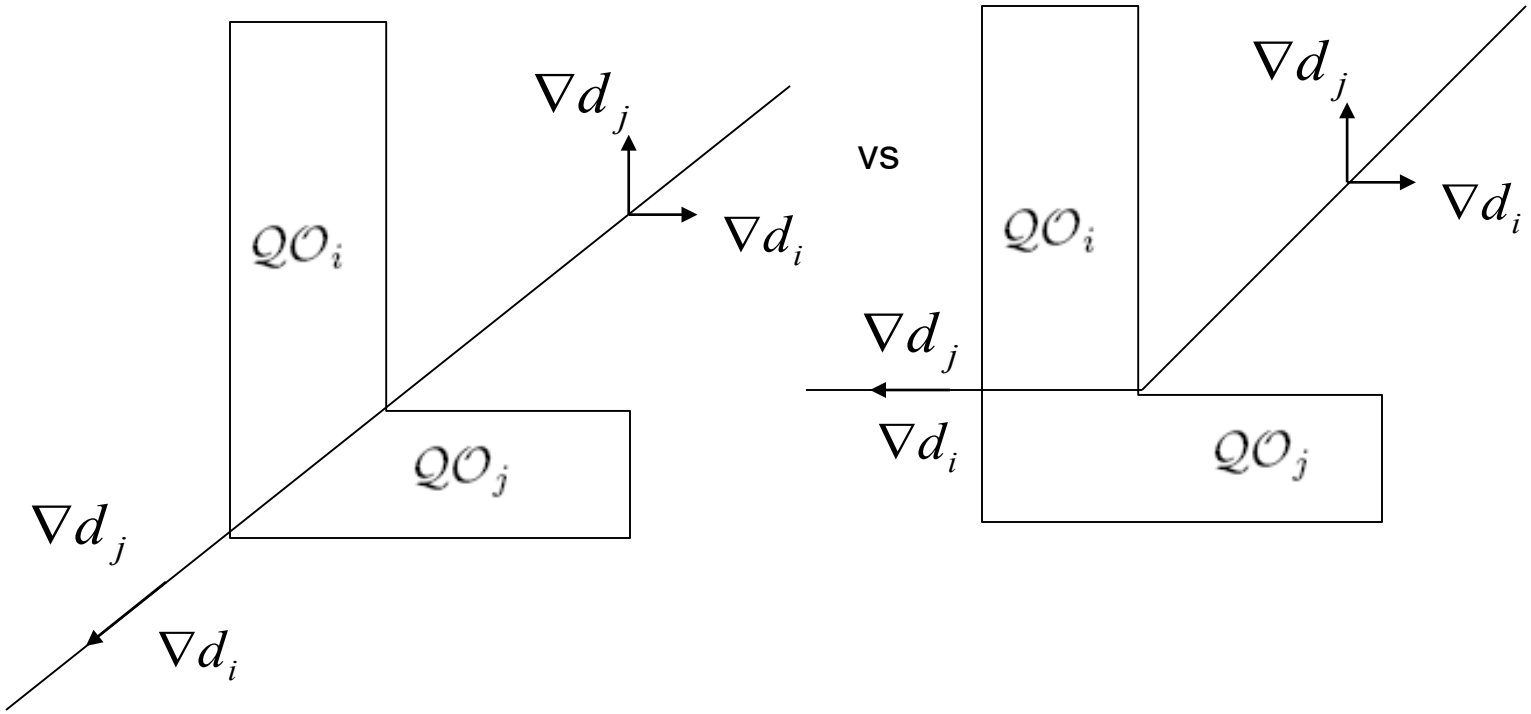
What about concave obstacles?



What about concave obstacles?

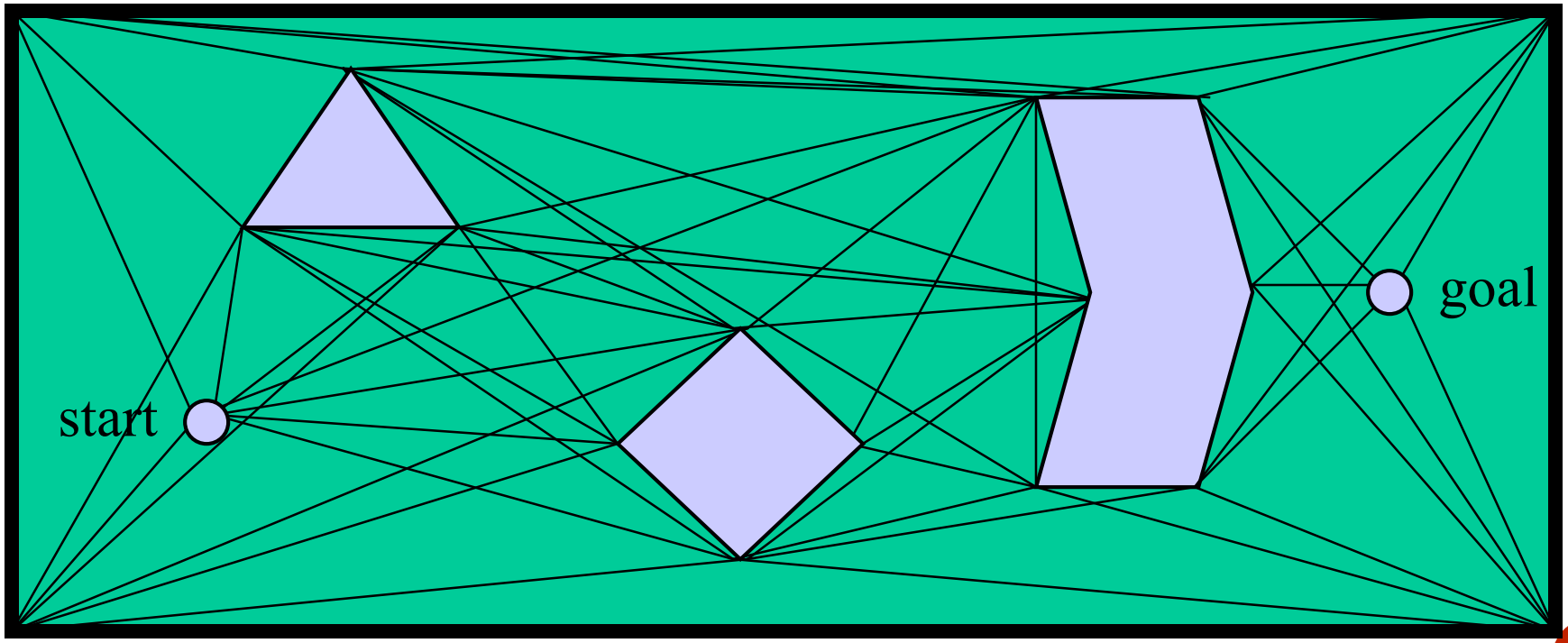


What about concave obstacles?



The Visibility Graph (Done)

- Repeat until you're done.



Visibility Graph Overview

- Start with a map of the world, draw lines of sight from the start and goal to every "corner" of the world and vertex of the obstacles, not cutting through any obstacles.
- Draw lines of sight from every vertex of every obstacle like above. Lines along edges of obstacles are lines of sight too, since they don't pass through the obstacles.
- If the map was in Configuration space, each line potentially represents part of a path from the start to the goal.

Graph Search

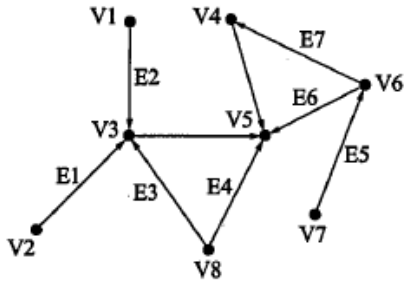
Howie Choset

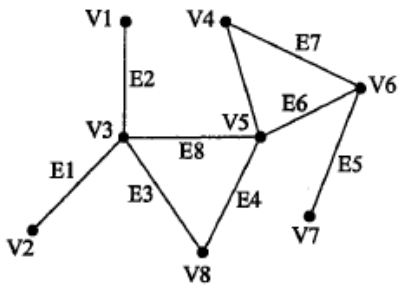
16-311

Outline

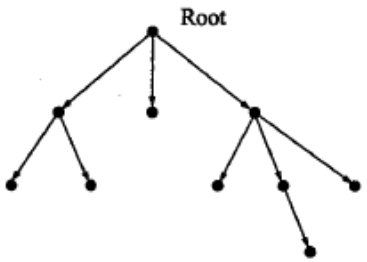
- Overview of Search Techniques
- A* Search

Graphs

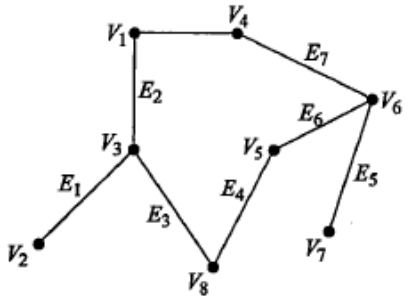




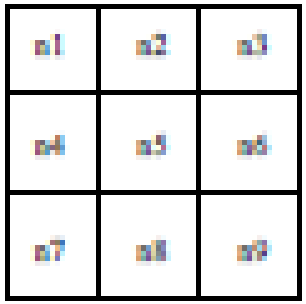
Collection of Edges and Nodes (Vertices)

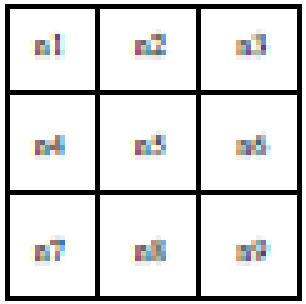


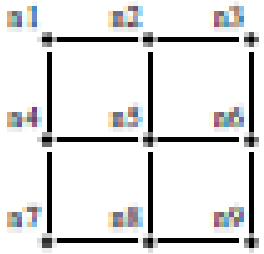
A tree

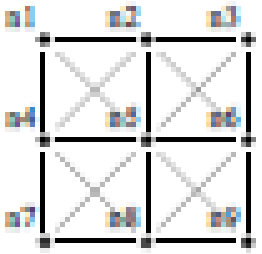


Grids

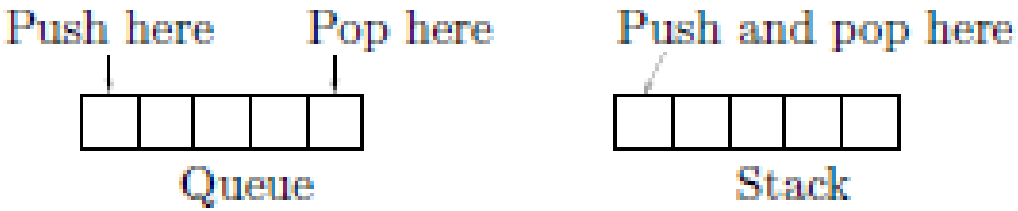

n1
n2
n3
n4
n5
n6
n7
n8
n9


n1
n2
n3
n4
n5
n6
n7
n8
n9


n1
n2
n3
n4
n5
n6
n7
n8
n9


n1
n2
n3
n4
n5
n6
n7
n8
n9

Stacks and Queues

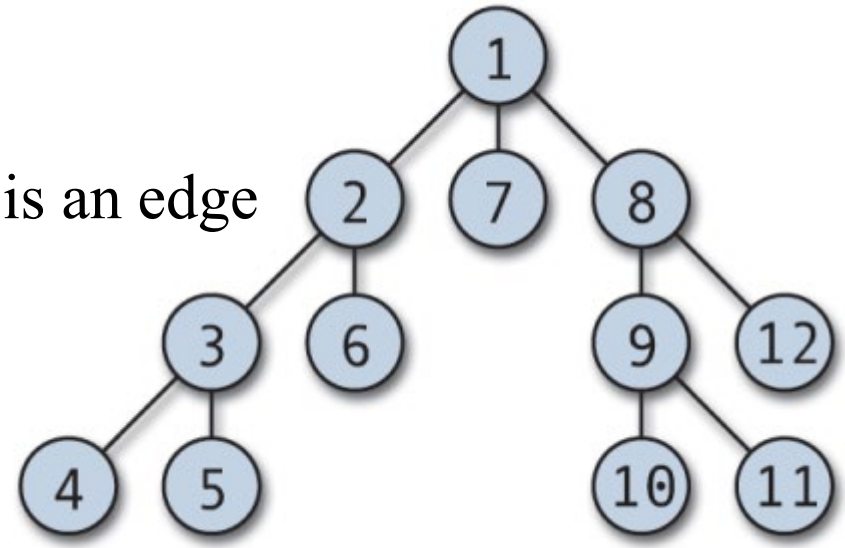


Stack: First in, Last out (FILO)

Queue: First in, First out (FIFO)

Depth First Search

```
algorithm  dft(x)
    visit(x)
    FOR each y such that (x,y) is an edge
        IF y was not visited yet
            THEN dft(y)
```



Copied from wikipedia

Worst case performance

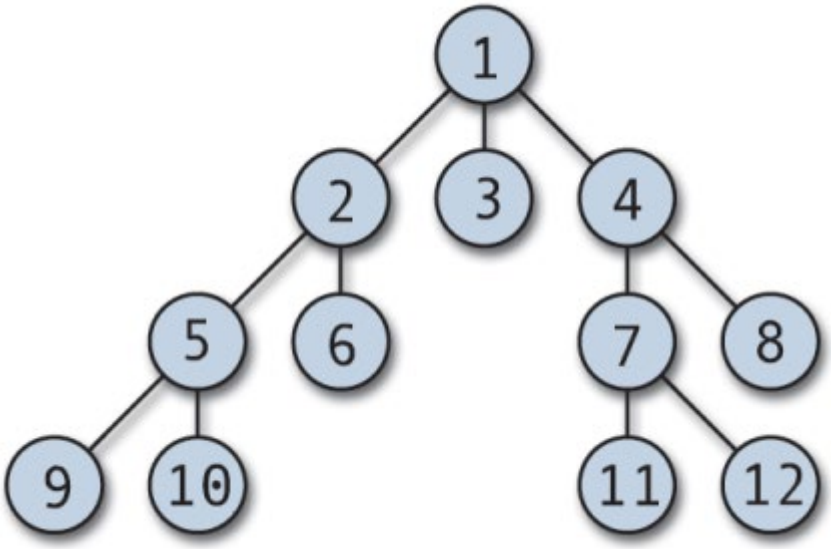
$O(|V| + |E|)$ for explicit graphs traversed without repetition,

Worst case space complexity

$O(|V|)$ if entire graph is traversed without repetition, O(longest path length searched) for implicit graphs without elimination of duplicate nodes

Breadth First Search

```
visit(start node)
queue <- start node
WHILE queue is nor empty D(
        x <- queue
        FOR each y such that (
                and y has not b(
            visit(y)
           queue <- y
           END
END
```



wikipedia

http://www.cse.ohio-state.edu/~gurari/course/cis680/cis680Ch14.html

Wavefront Planner: A BFS

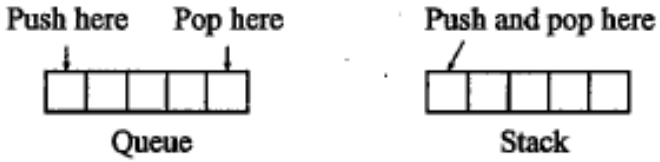
7	**18**	17	16	15	14	13	12	11	10	9	9	9	9	9	9	9
6	17	17	16	15	14	13	12	11	10	9	8	8	8	8	8	8
5	17	16	16	15	14	13	12	11	10	9	8	7	7	7	7	7
4	17	16	15	15	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	6	6	6	6
3	17	16	15	14	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	5	5	5	5
2	17	16	15	14	13	12	11	10	9	8	7	6	5	4	4	4
1	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	3
0	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	**2**
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

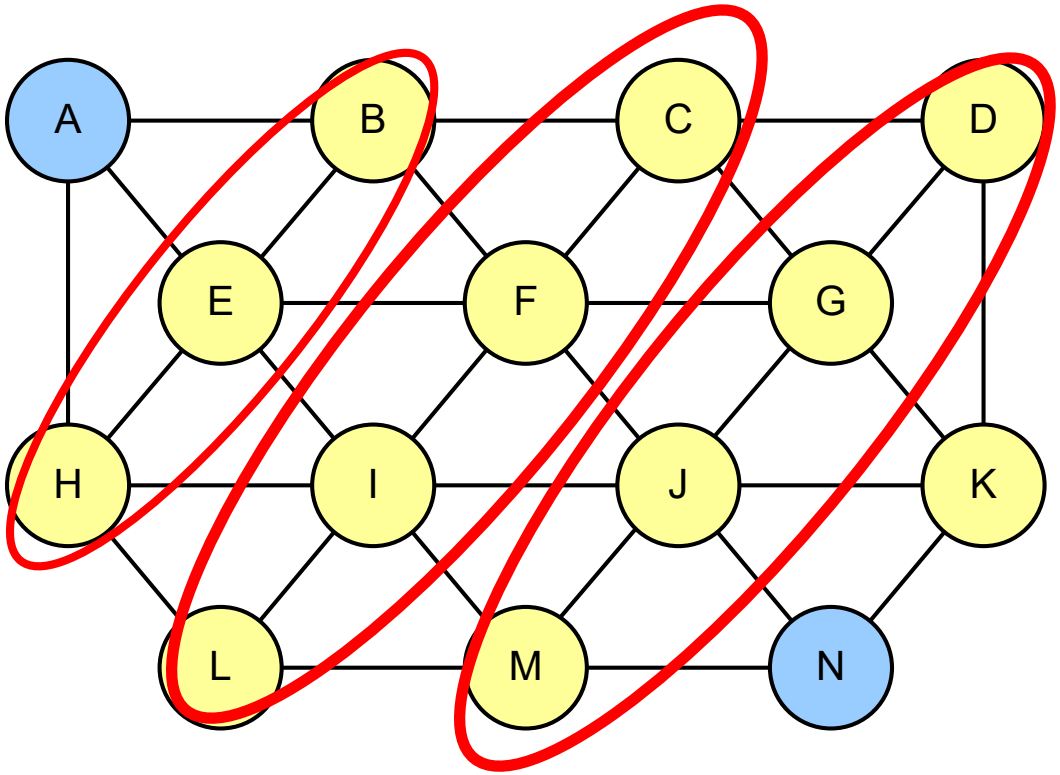
Search

- Uninformed Search
  - Use no information obtained from the environment
  - Blind Search: BFS (Wavefront), DFS
- Informed Search
  - Use evaluation function
  - More efficient
  - Heuristic Search: A*, D*, etc.

Uninformed Search

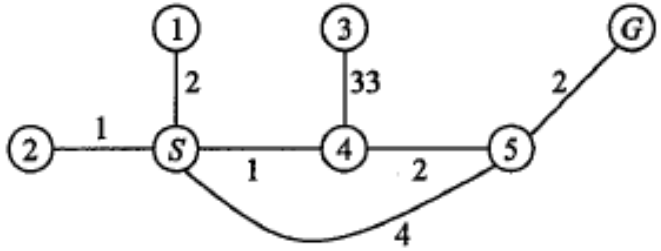
Graph Search from A to N





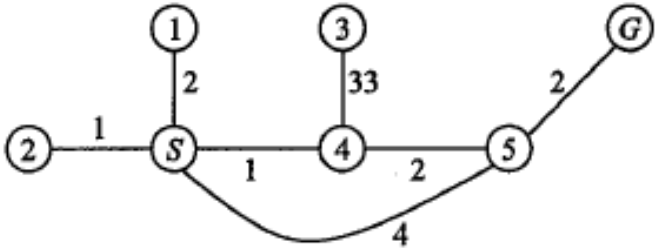
BFS

Dijkstra's Search: f(n) = g(n)



Pop lowest f first

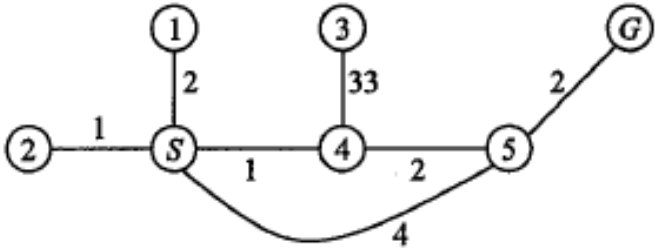
Dijkstra's Search: f(n) = g(n)



Pop lowest f first

1. O = {S}

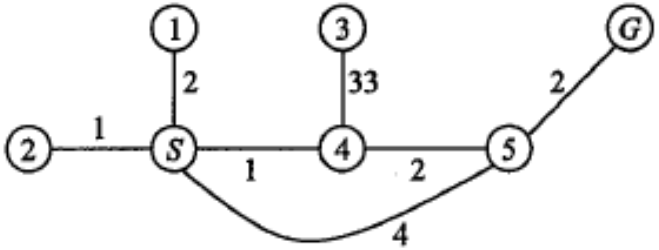
Dijkstra's Search: f(n) = g(n)



Pop lowest f first

1. O = {S}
2. O = {1, 2, 4, 5}; C = {S} (1,2,4,5 all back point to S)

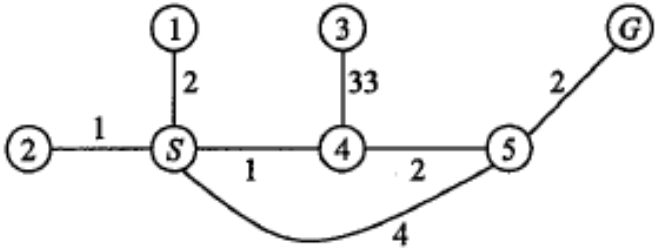
Dijkstra's Search: f(n) = g(n)



Pop lowest f first

1. O = {S}
2. O = {1, 2, 4, 5}; C = {S} (1,2,4,5 all back point to S)
3. O = {1, 4, 5}; C = {S, 2} (there are no adjacent nodes not in C)

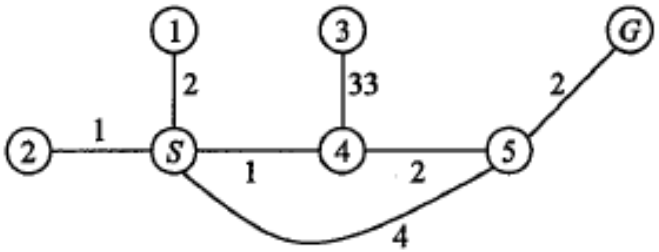
Dijkstra's Search: f(n) = g(n)



Pop lowest f first

1. O = {S}
2. O = {1, 2, 4, 5}; C = {S} (1,2,4,5 all back point to S)
3. O = {1, 4, 5}; C = {S, 2} (there are no adjacent nodes not in C)
4. O = {1, 5, 3}; C = {S, 2, 4} (1, 2, 4 point to S; 5 points to 4)

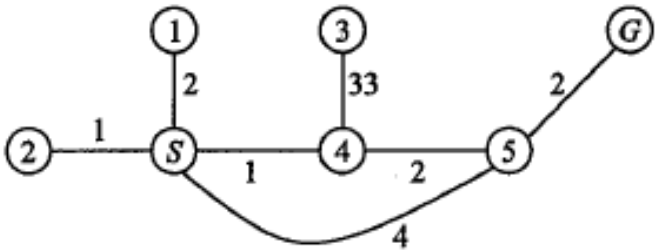
Dijkstra's Search: f(n) = g(n)



Pop lowest f first

1. O = {S}
2. O = {1, 2, 4, 5}; C = {S} (1,2,4,5 all back point to S)
3. O = {1, 4, 5}; C = {S, 2} (there are no adjacent nodes not in C)
4. O = {1, 5, 3}; C = {S, 2, 4} (1, 2, 4 point to S; 5 points to 4)
5. O = {5, 3}; C = {S, 2, 4, 1}

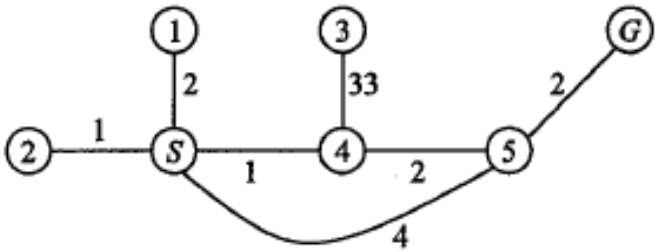
Dijkstra's Search: f(n) = g(n)



Pop lowest f first

1. O = {S}
2. O = {1, 2, 4, 5}; C = {S} (1,2,4,5 all back point to S)
3. O = {1, 4, 5}; C = {S, 2} (there are no adjacent nodes not in C)
4. O = {1, 5, 3}; C = {S, 2, 4} (1, 2, 4 point to S; 5 points to 4)
5. O = {5, 3}; C = {S, 2, 4, 1}
6. O = {3, G}; C = {S, 2, 4 1,5} (goal points to 5 which points to 4 which points to S)

Dijkstra's Search: f(n) = g(n)



Pop lowest f first

1. O = {S}
2. O = {1, 2, 4, 5}; C = {S} (1,2,4,5 all back point to S)
3. O = {1, 4, 5}; C = {S, 2} (there are no adjacent nodes not in C)
4. O = {1, 5, 3}; C = {S, 2, 4} (1, 2, 4 point to S; 5 points to 4)
5. O = {5, 3}; C = {S, 2, 4, 1}
6. O = {3, G}; C = {S, 2, 4 1,5} (goal points to 5 which points to 4 which points to S)
7. O = {3}; C = {S, 2, 4, 1,5} (Path found because Goal was popped)

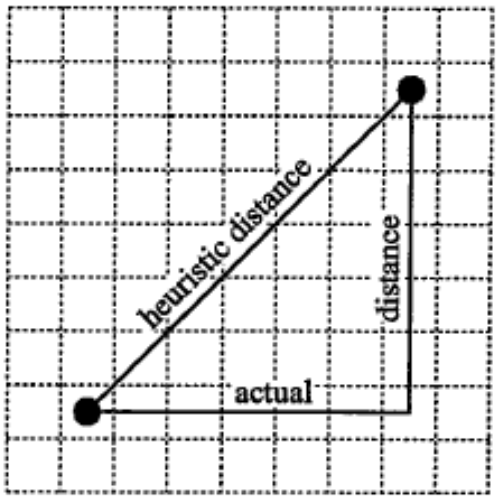
Informed Search: A*

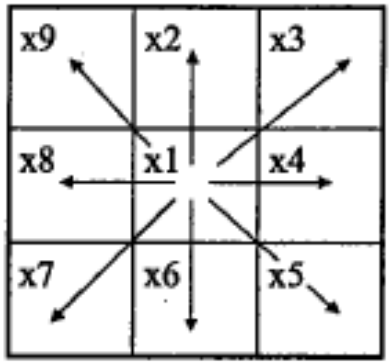
Notation

- $n \rightarrow$ node/state
- $c(n_1,n_2) \rightarrow$ the length of an edge connecting between n_1 and n_2
- $b(n_1) = n_2 \rightarrow$ backpointer of a node n_1 to a node n_2.

Informed Search: A*

- Evaluation function, $f(n) = g(n) + h(n)$
- Operating cost function, $g(n)$
  - Actual operating cost having been already traversed
- Heuristic function, $h(n)$
  - Information used to find the promising node to traverse
  - Admissible → never overestimate the actual path cost





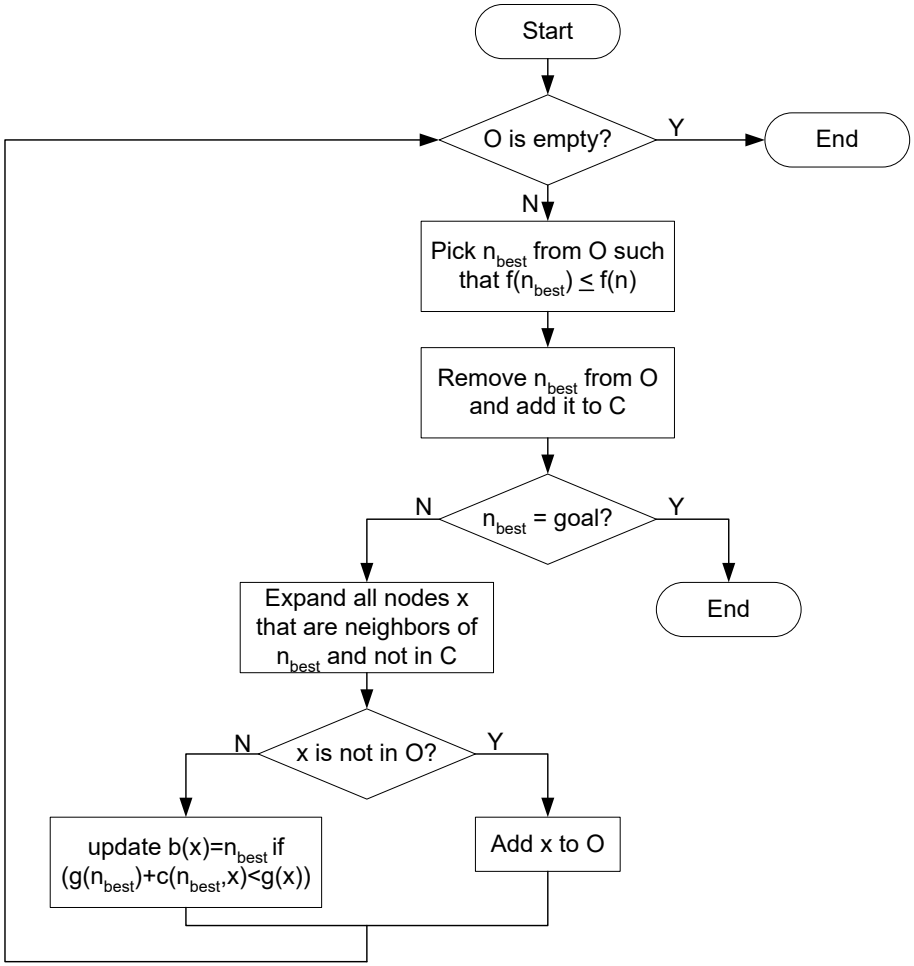
c(x1, x2) = 1
c(x1, x9) = 1.4

c(x1, x8) = 10000, if x8 is in obstacle, x1 is a free cell

c(x1,x9) = 10000.4, if x9 is in obstacle, x1 is a free cell

Cost on a grid

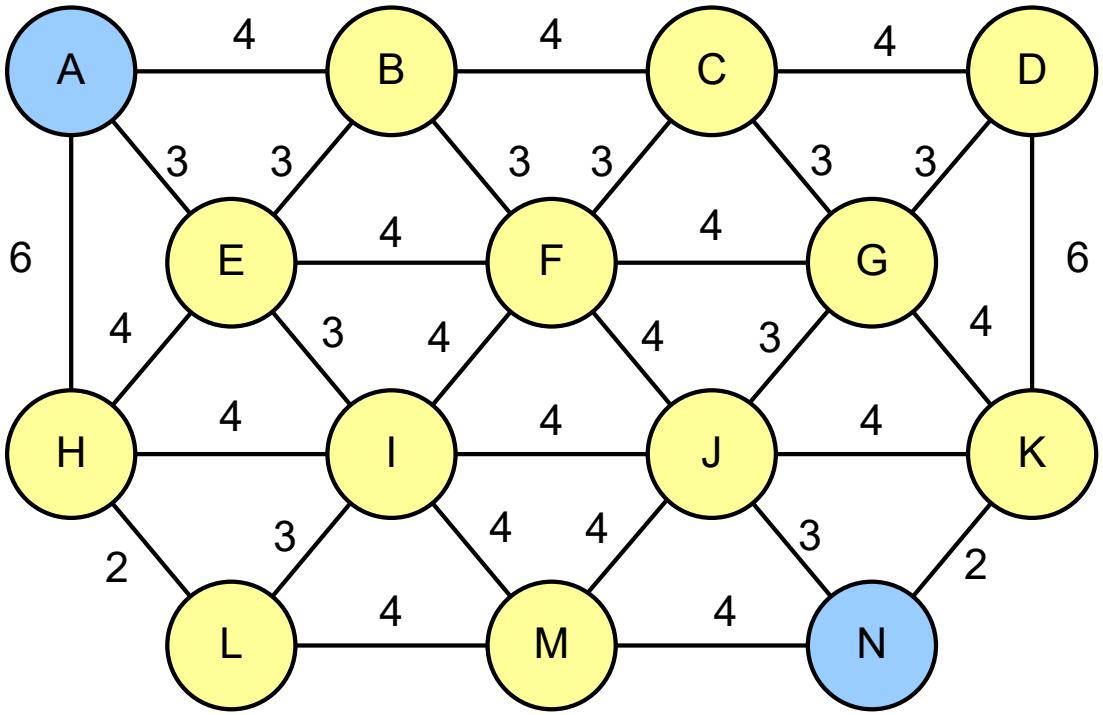
A*: Algorithm



The search requires 2 lists to store information about nodes

1) **Open list (O)** stores nodes for expansions
2) **Closed list (C)** stores nodes which we have explored

A*: Example (1/6)

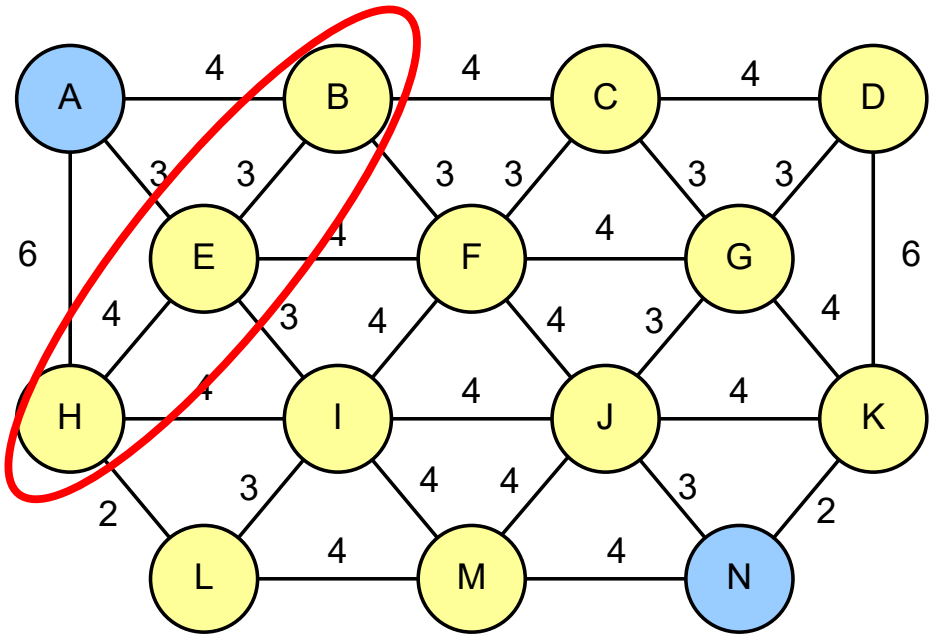


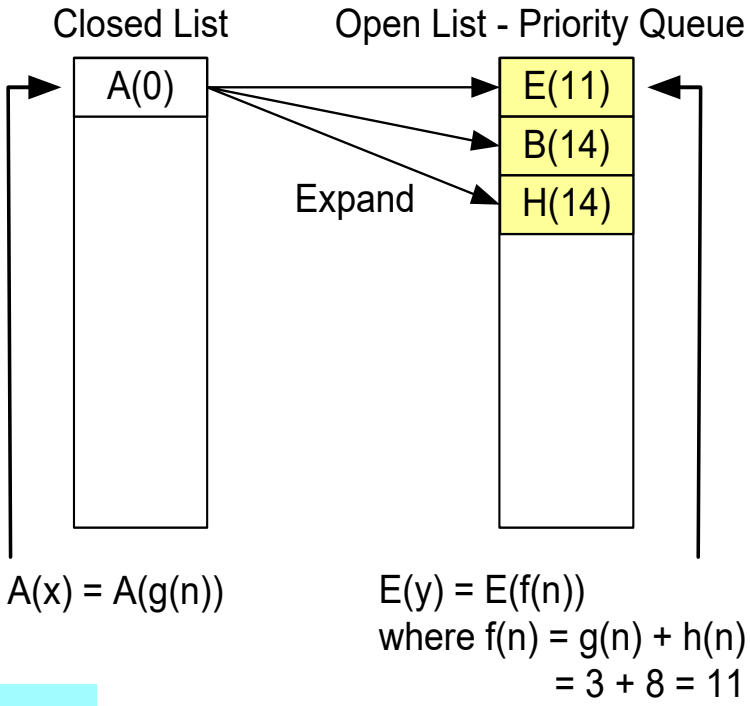
Heuristics

A = 14	H = 8
B = 10	I = 5
C = 8	J = 2
D = 6	K = 2
E = 8	L = 6
F = 7	M = 2
G = 6	N = 0

Legend: operating cost

A*: Example (2/6)



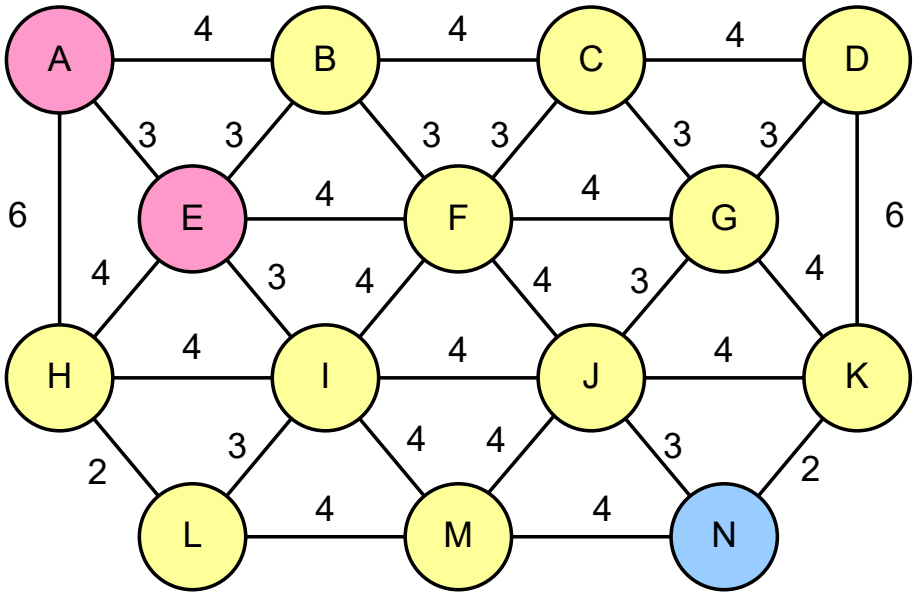


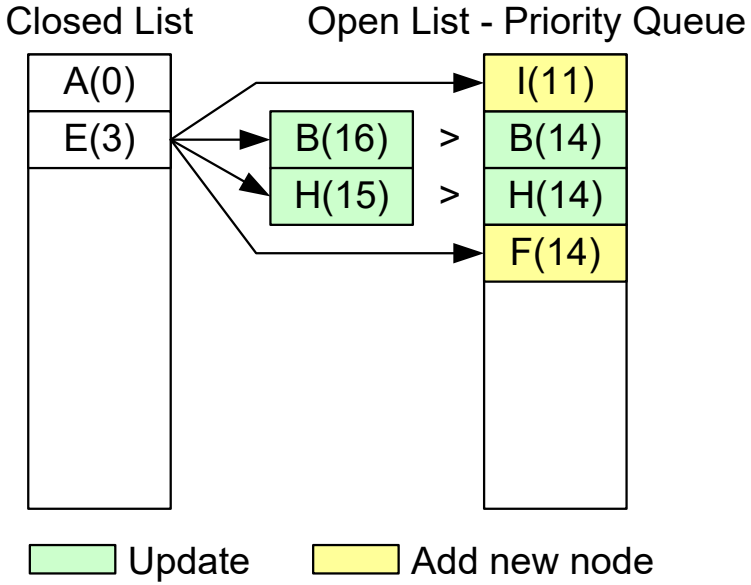
Heuristics

A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6

H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

A*: Example (3/6)





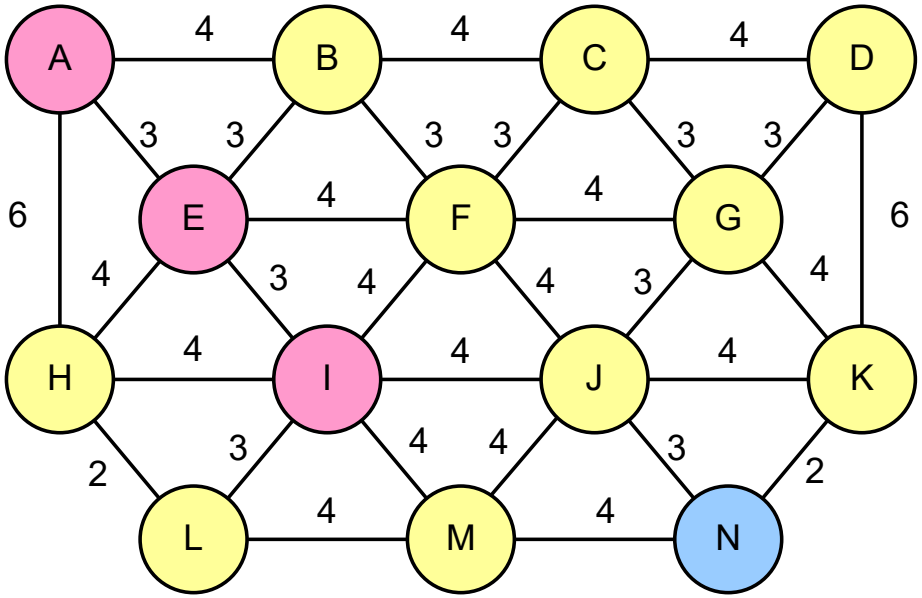
Heuristics

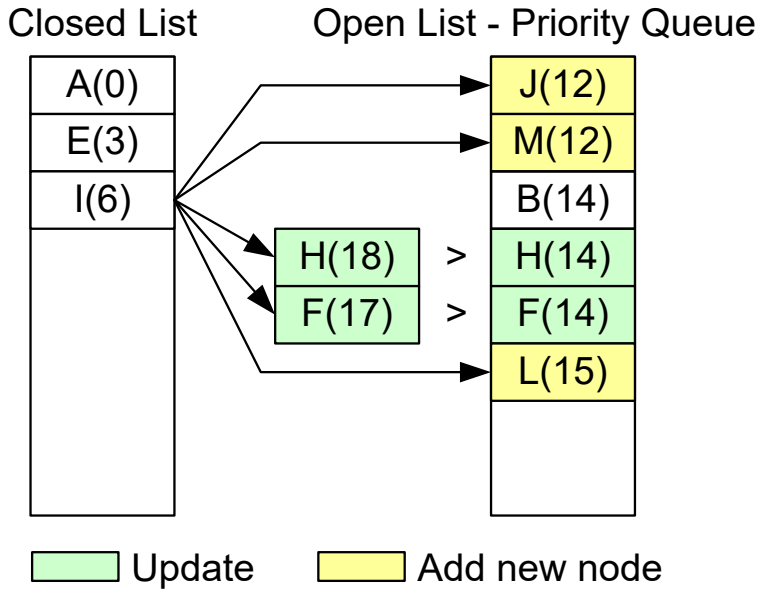
A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6

H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

Since A → B is smaller than A → E → B, the f-cost value of B in an open list needs not be updated

A*: Example (4/6)



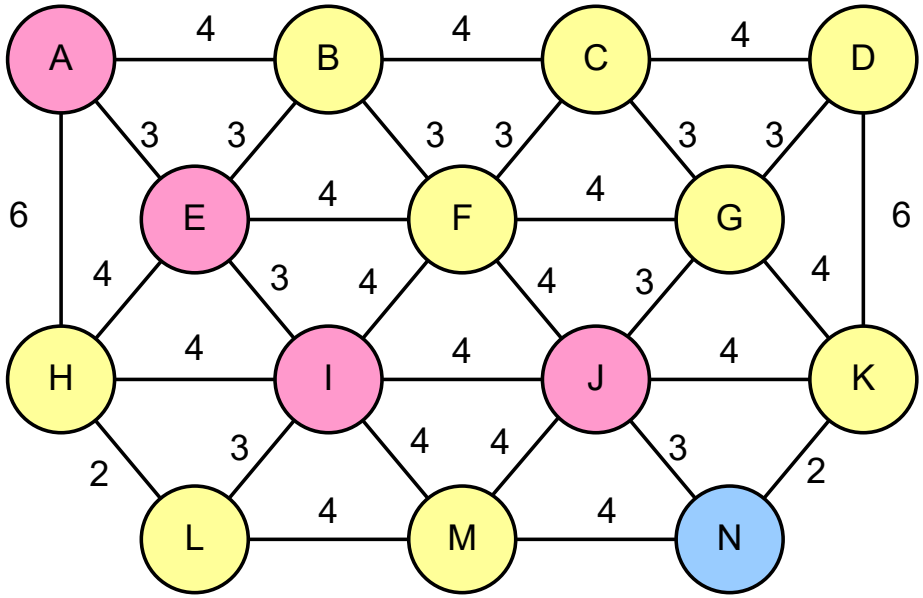


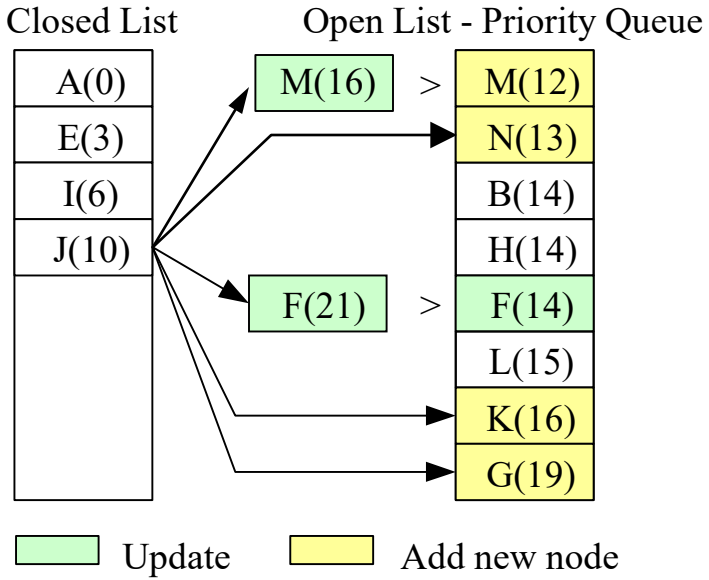
Heuristics

A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6

H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

A*: Example (5/6)



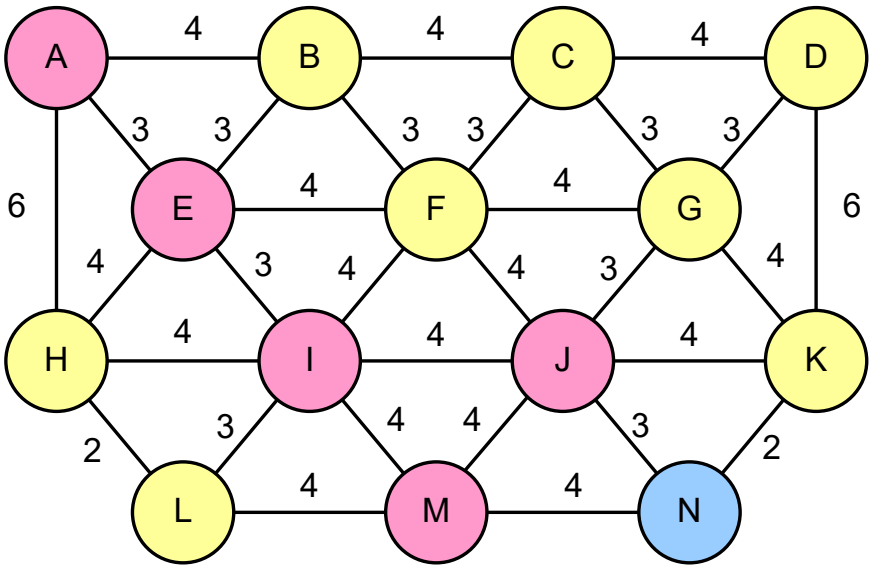


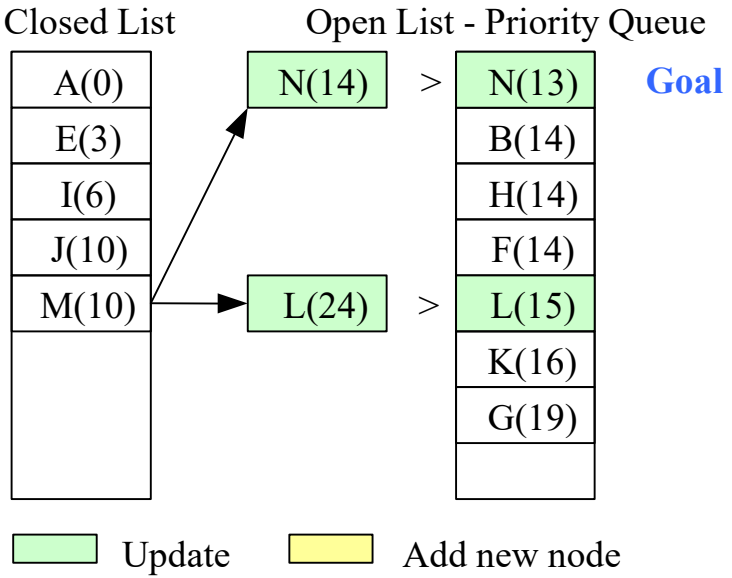
Heuristics

A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6

H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

A*: Example (6/6)





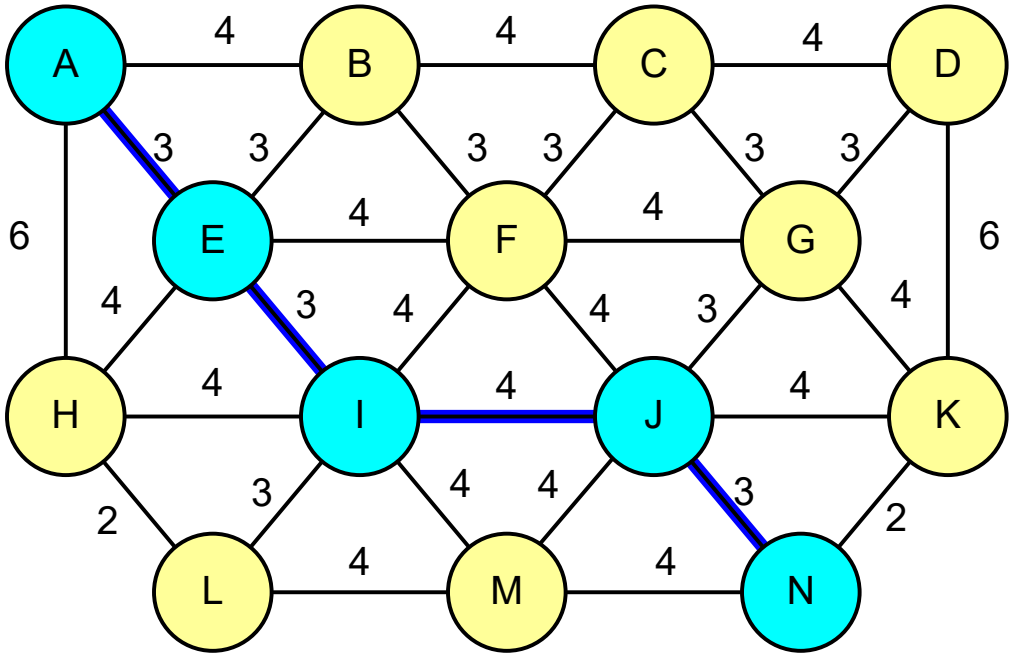
Heuristics

A = 14, B = 10, C = 8, D = 6, E = 8, F = 7, G = 6

H = 8, I = 5, J = 2, K = 2, L = 6, M = 2, N = 0

Since the path to N from M is greater than that from J, **the optimal path to N is the one traversed from J**

A*: Example Result



Generate the path from the goal node back to the start node through the back-pointer attribute

Cool Animation

- https://qiao.github.io/PathFinding.js/visual/