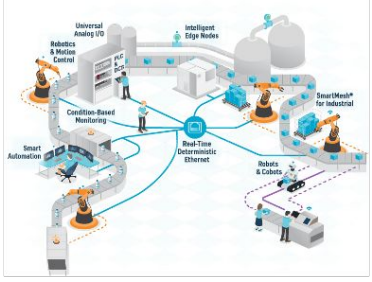
Kia Cooperative Systems Summer High School Outreach

Module 3

PI: Solmaz Kia
Graduate Students: Donipolo Ghimire
Mechanical and Aerospace Engineering Department
University of California Irvine
Summer 2021

Robots: realization of people's dream of building intelligent machines to perform tasks.



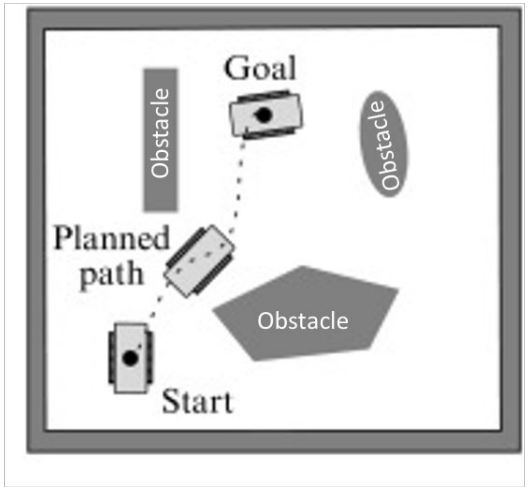
Why do we care about robot motion planning?

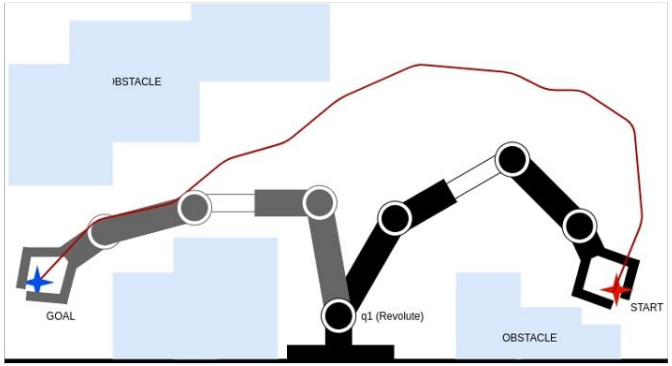
Regardless of the form of the robots or the task it must perform, robots must maneuver through the world.

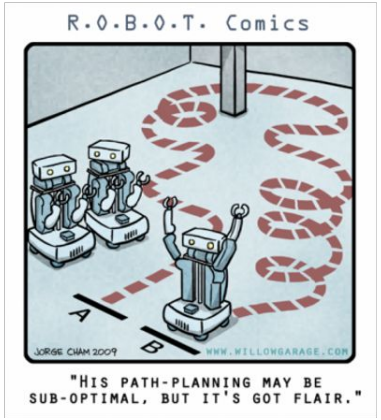
Motion planning is the problem of finding a robot motion from a start state to a goal state in a cluttered environment to achieve various goals while avoiding collisions.

In its simplest form, the motion planning problem is:

how to move a robot from a "start" location to a "goal" location avoiding obstacles.







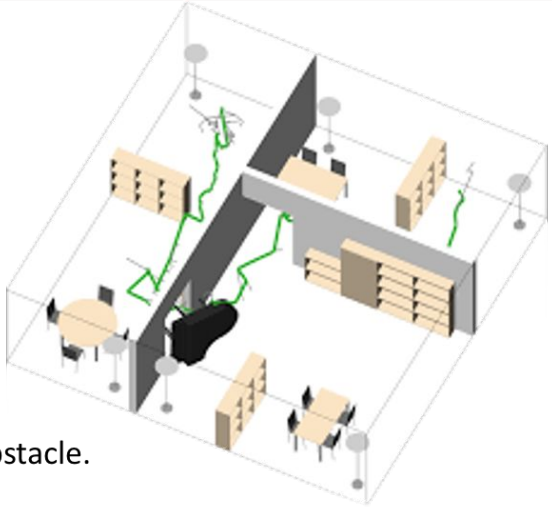
Motion Planning: Problem Formulation

The problem of motion planning can be stated as follows:

Given:

- ❑ A start pose of the robot
- ❑ A desired goal pose
- ❑ A geometric description of the robot
- ❑ A geometric description of the world

Find a path that moves the robot gradually from start to goal while never touching any obstacle.

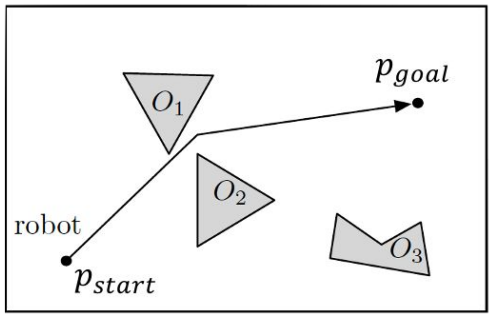


https://youtu.be/HdfAzUXvmOQ

This problem is sometimes referred to as the “move from A to B” or the “**piano movers problem**”
(how do you move a complex object like a piano in an environment with lots of obstacles, like a house).

Motion Planning: Workspace

a robot described by a moving point (that is, the robot has zero size).

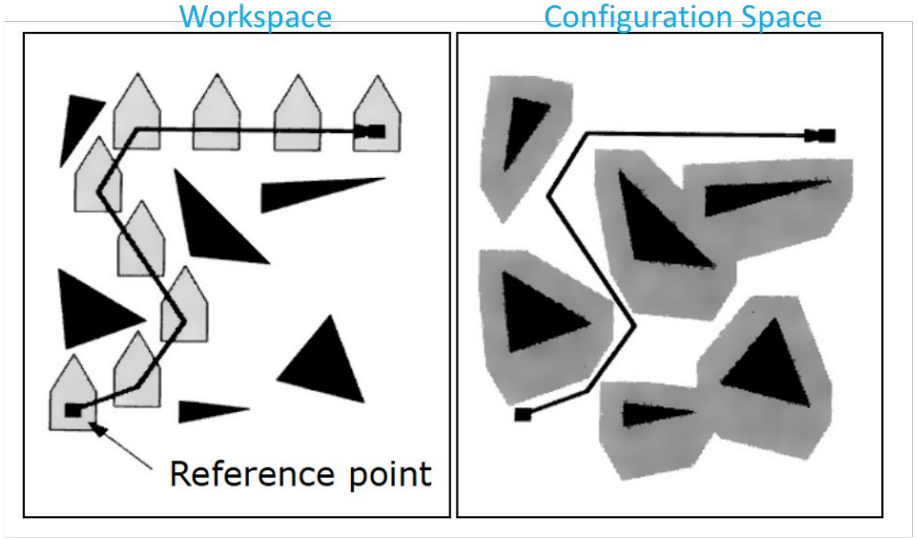


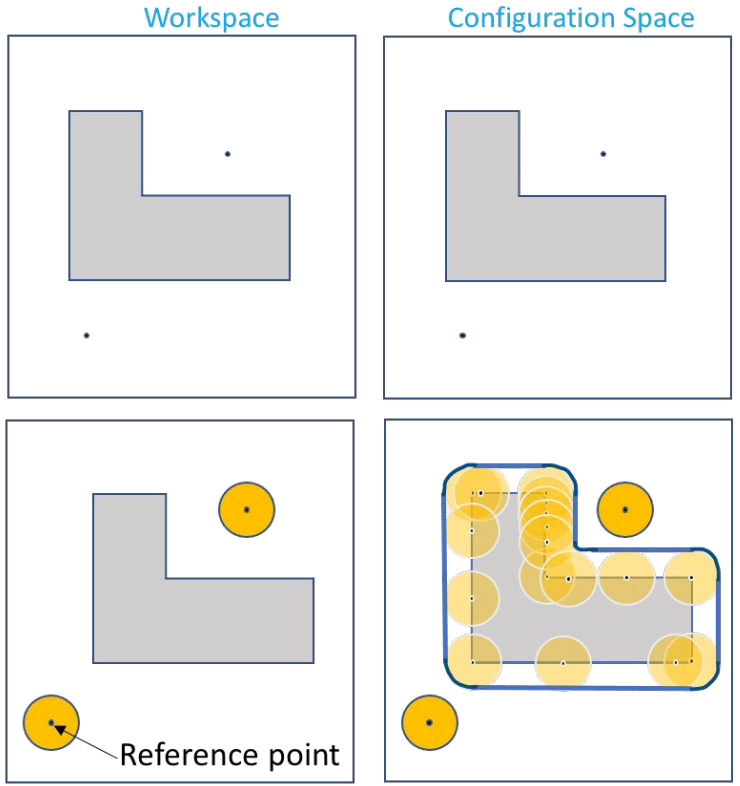
- A workspace $W \subset R^2$ or R^3, often just a rectangle;
- Some obstacles $O_1, O_2, \cdots, O_n$;
- A start point p_{start} and a goal point p_{goal};

free workspace: $W_{free} = W \backslash (O_1 \cup O_2 \cup \cdots \cup O_n)$: the set of points in W that are outside all obstacles.

Motion Planning: Configuration space

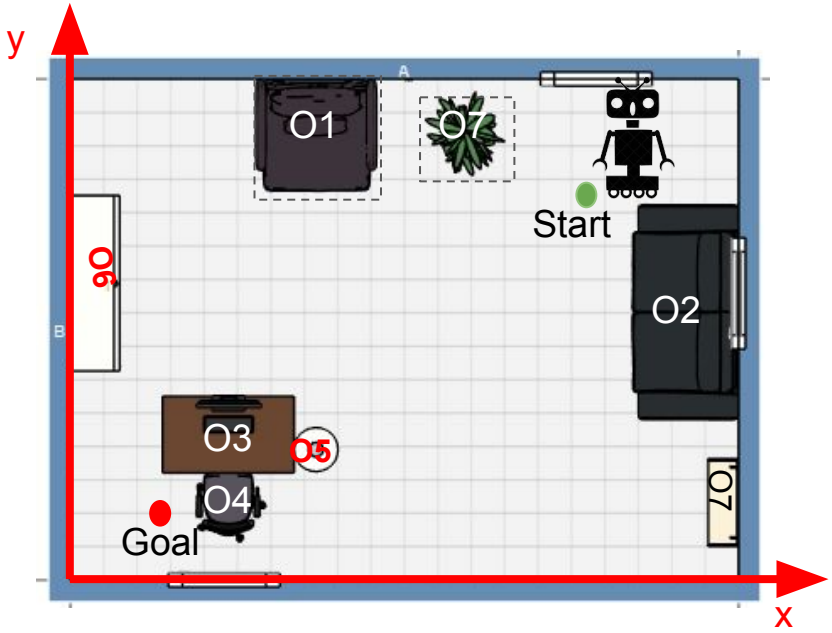
- Q-space is obtained by sliding the robot along the edge of the obstacle regions



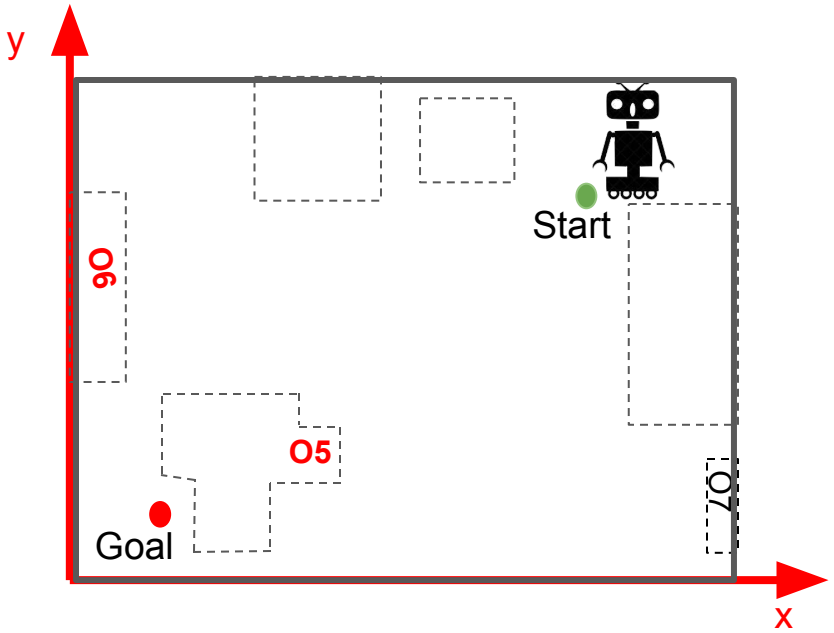


Q-space is obtained by "blowing the edge of the obstacles up" by the robot radius

How to represent my problem in the best way possible in Python



A map depicting different obstacles

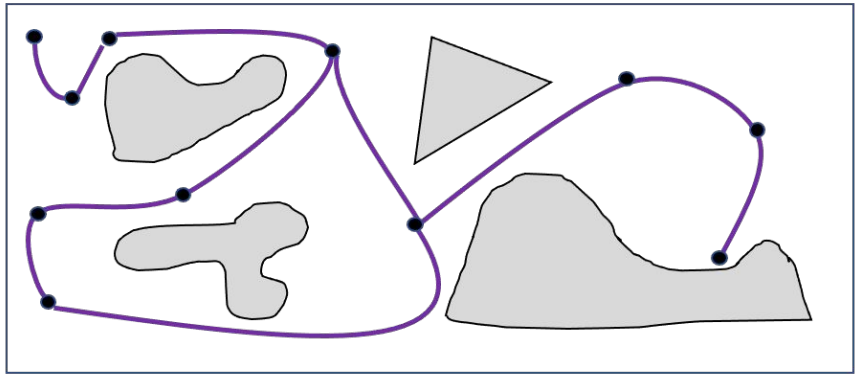


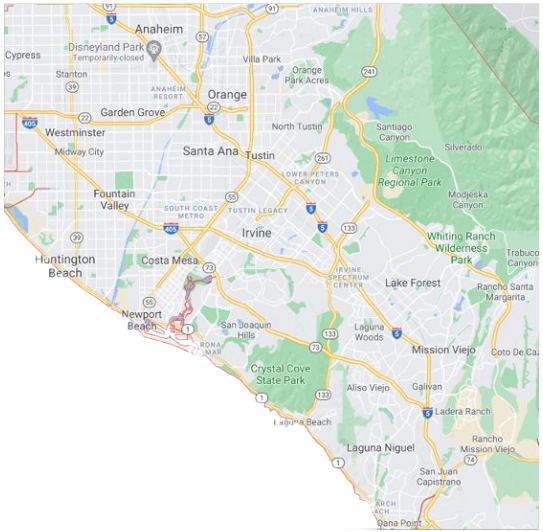
A map depicting the obstacles fitted by polygon boundaries

Roadmaps

A roadmap is a collection of locations in the configuration space along with paths connecting them.

- With each path, we associate a positive weight that represents a cost for traveling along that path, for example, the path length or the travel time.
- Think of a roadmap as a weighted graph $G = (V, E, w)$, where w is a function that assigns the weight (e.g., path length) to each edge in E.





Roadmaps

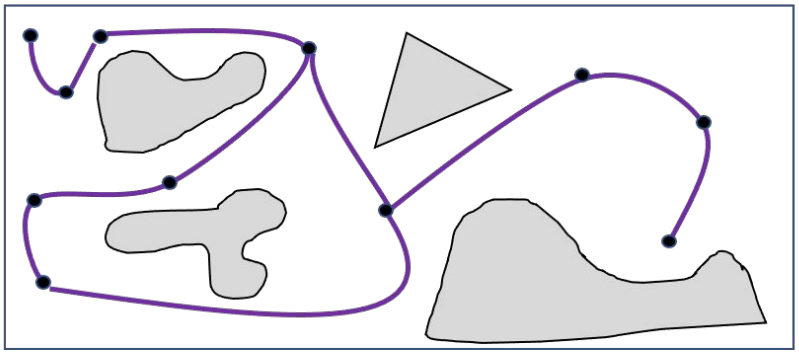
roadmap may have the following properties:

i. the roadmap is accessible if, for each point $q_{start} \in Q_{free}$, there is an easily computable path from q_{start} to some node in the roadmap,

ii. similarly, the roadmap is departable if, for each point $q_{goal} \in Q_{free}$, there is an easily computable path from some node in the roadmap to q_{goal}, and

iii. the roadmap is connectivity-preserving if, given a connected free configuration space (i.e., any two configurations in Q_{free} are connected by a path in Q_{free}), then any two locations of the roadmap are connected by a path in the roadmap,

iv. the roadmap is efficient with factor $\delta \geq 1$ if, for any two locations in the roadmap, say u and v, the path length from u to v along edges of the roadmap is no longer than δ times the length of the shortest path from u to v in the environment



The notions of accessibility and departability are not fully specified as they depend upon the notion of "easily computable path."

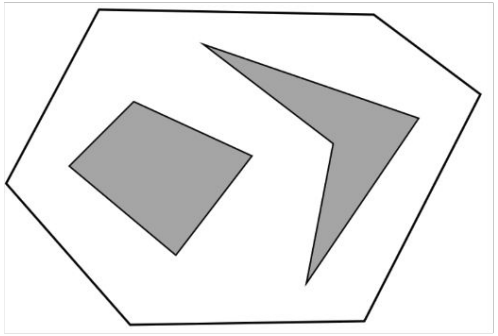
Motion Planning Using Visibility Graph

Visibility roadmaps: the visibility graph $G = (V, E, w)$, is defined as

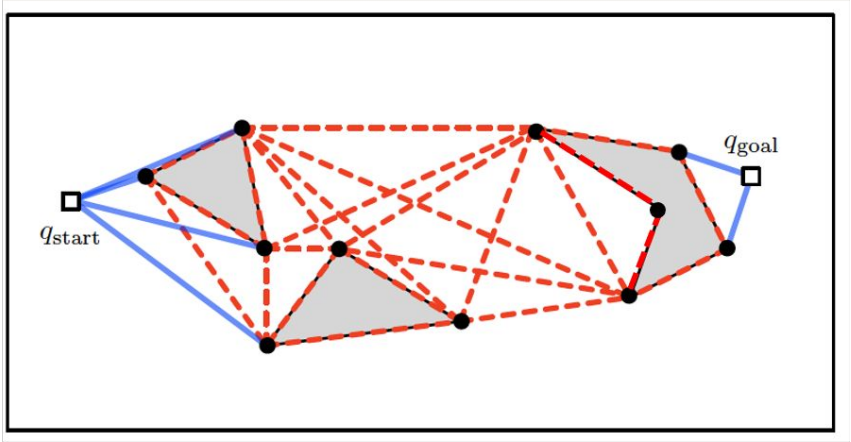
i. the nodes V of the visibility graph are all the vertices of the polygons $O_1, \cdots, O_n$,

ii. the edges E of the visibility graph are all pairs of vertices that are visibly connected. That is, given $u, v \in V$, we add the edge $\{u, v\}$ to the edge set E if the straight-line segment between u and v is not in collision with any obstacle, and

iii. the weight of an edge $\{u, v\}$ is given by the length of the segment connecting u and v.



environments with polygonal obstacles.

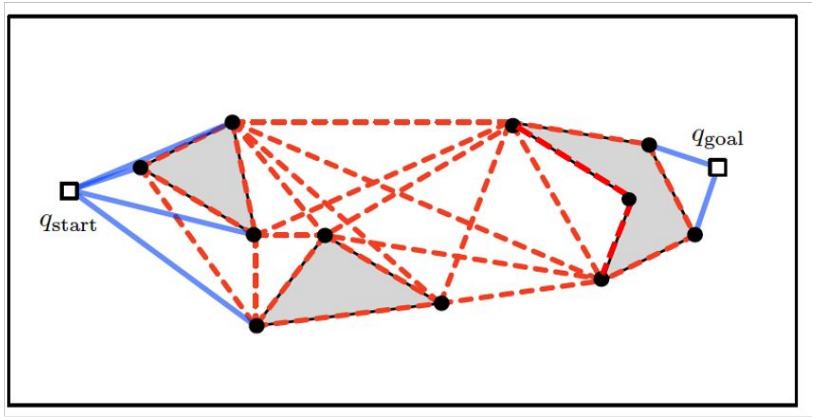


Motion Planning Using Visibility Graph

Properties of visibility roadmap:

- If the free configuration space is connected, then the visibility graph is connected, departable, and accessible.

- The shortest path from start to goal is a path in the visibility graph. Hence, the roadmap obtained via the visibility graph is optimally efficient, in the sense that the efficiency factor δ is 1.



Theorem 5.1 (Shortest paths through polygonal obstacles) Consider a configuration space with polygonal configuration space obstacles.
Any shortest path in the free configuration space between q_{start} and q_{goal}

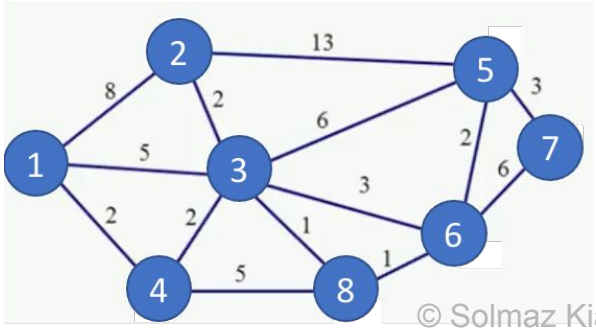
(i) consists of only straight line segments, and

(ii) has segments whose endpoints are either the start location q_{start}, the goal location q_{goal}, or an obstacle vertex (or, more precisely, a convex obstacle vertex if start and goal locations are not non-convex vertices).

Dijkstra's Shortest Path Algorithm

1. Mark all nodes unvisited. Create a set of all the unvisited nodes called the *unvisited set*.
2. Assign to every node a tentative distance value: set it to zero for the initial node and to infinity for all other nodes. Set the initial node as current.
3. For the current node, consider all of its unvisited neighbours and calculate their *tentative* distances through the current node. Compare the newly calculated *tentative* distance to the current assigned value and assign the smaller one. For example, if the current node *A* is marked with a distance of 6, and the edge connecting it with a neighbour *B* has length 2, then the distance to *B* through *A* will be 6 + 2 = 8. If B was previously marked with a distance greater than 8 then change it to 8. Otherwise, the current value will be kept.
4. When we are done considering all of the unvisited neighbours of the current node, mark the current node as visited and remove it from the *unvisited set*. A visited node will never be checked again.
5. If the destination node has been marked visited (when planning a route between two specific nodes) or if the smallest tentative distance among the nodes in the *unvisited set* is infinity (when planning a complete traversal; occurs when there is no connection between the initial node and remaining unvisited nodes), then stop. The algorithm has finished.
6. Otherwise, select the unvisited node that is marked with the smallest tentative distance, set it as the new "current node", and go back to step 3.

Source:
https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm



Shortest paths in weighted graphs via Dijkstra's algorithm

The minimum-weight path between two nodes, also called the shortest path in a weighted graph, is a path of minimum weight between the two nodes

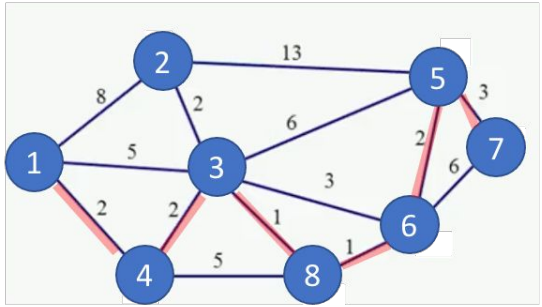
Unvisited set of nodes	distance	parent
1		
2		
3		
4		
5		
6		
7		
8		

Template to implement the algorithm

Result

nodes	distance	parent
1	0	1
2	6	3
3	4	4
4	2	1
5	8	6
6	6	8
7	11	5
8	5	3

Find the shortest path from 1 to 7.



Shortest path from 1 to 7:
1->4>3->8->6>5->7
(distance=11)

Shortest paths in weighted graphs via Dijkstra's algorithm

The minimum-weight path between two nodes, also called the shortest path in a weighted graph, is a path of minimum weight between the two nodes

Dijkstra's algorithm

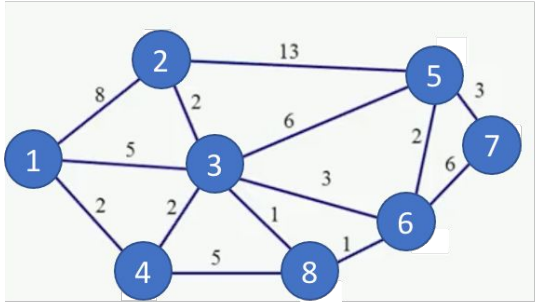
Input: a weighted graph G and a start node v_{start}
Output: the `parent` pointer and `dist` values for each node in the graph G

// Initialization of distance and parent pointer for each node

```
for each node v in G :
    dist(v) := +∞
    parent(v) := NONE
dist(v_start) := 0
parent(v_start) := SELF
Q := set of all nodes in G
```

// Main loop to grow the tree and update distance estimates

```
while Q is not empty :
    find node v in Q with smallest dist(v)
    remove v from Q
    for each node w in Q connected to v by an edge :
        if dist(w) > dist(v) + weight(v, w) :
            dist(w) := dist(v) + weight(v, w)
            parent(w) := v
return parent values and dist values for all nodes v
```



extract-path algorithm

Input: a goal node v_{goal}, and the `parent` values
Output: a path from v_{start} to v_{goal}

```
create an array P := [v_goal]
set u := v_goal
while parent(u) ≠ SELF :
    u := parent(u)
    insert u at the beginning of P
return P
```

- The shortest path tree as {parent(u), u} for each node u for which parent(u) is different from $+\infty$.
- Given a goal node v_{goal} we can use the parent values to reconstruct the sequence of nodes on the shortest path from v_{start} to v_{goal} using the The extract-path algorithm.

Reading/Watching Assignment

Youtube videos

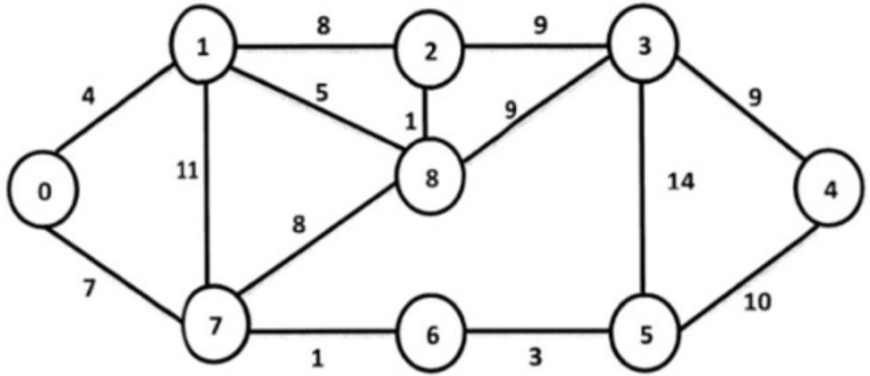
- https://youtu.be/pVfj6mxhdMw
- https://youtu.be/GazC3A4OQTE
- https://youtu.be/W4Jh0hlfd-o

Reading:

- https://brilliant.org/wiki/dijkstras-short-path-finder/
- https://medium.com/@nicholas.w.swift/easy-dijkstras-pathfinding-324a51eeb0
- https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm

Practice Problem

Find the shortest path from node 0 to all the other nodes.



Sponsors

UCI Center for Educational Partnerships