COEN 5830, Fall 2024
Introduction to Robotics

Lecture 8

Path Planning Algorithms

Leopold Beuken (leopold.Beuken@Colorado.edu)

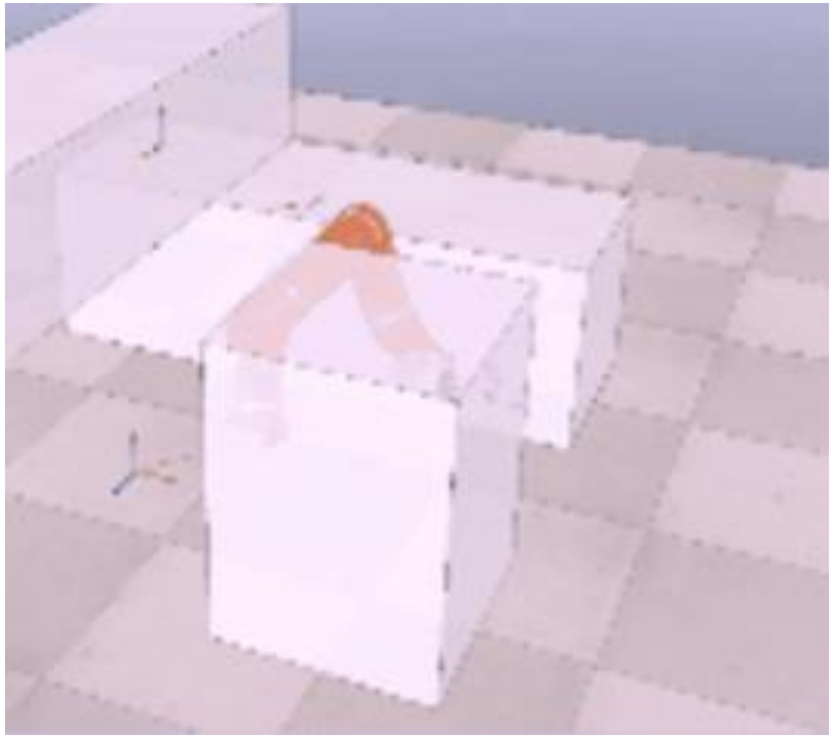
Thursday, 19/9/2024

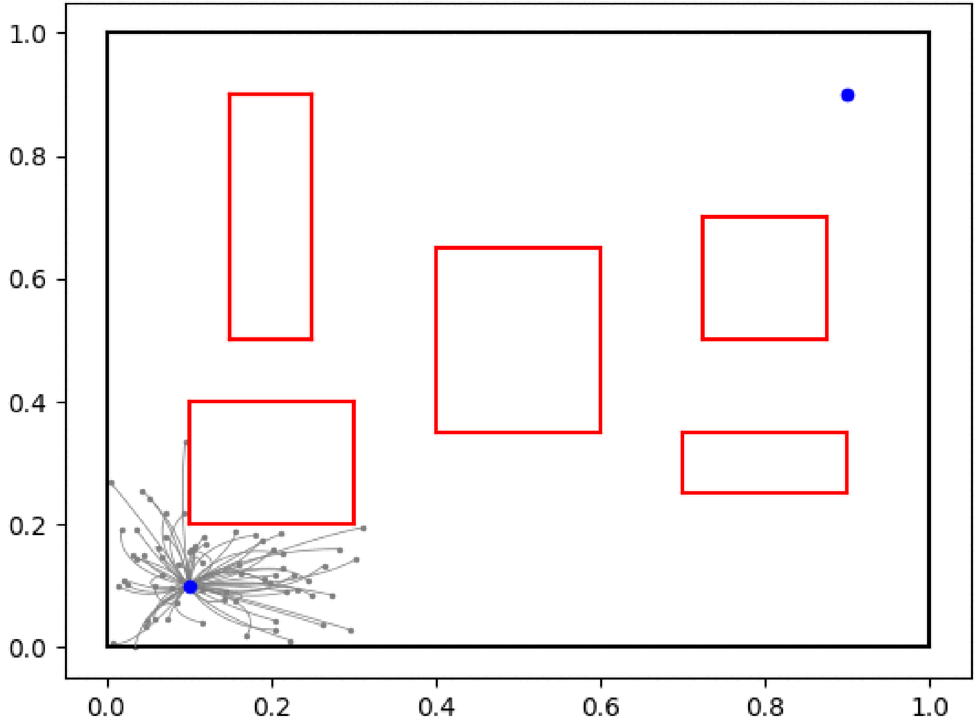
Administrative

- HW1 will be graded by end of next week
- HW2 will be released this weekend

What is path planning?

Path planning is the problem of finding a **set of robot states** from a **start state** to a **goal state** that **avoids obstacles** in the environment and satisfies **other constraints**, such as joint limits or degree of freedom limitations.





Where do we explore next?

Where do we explore next?

Answer: It depends on the planning algorithm.

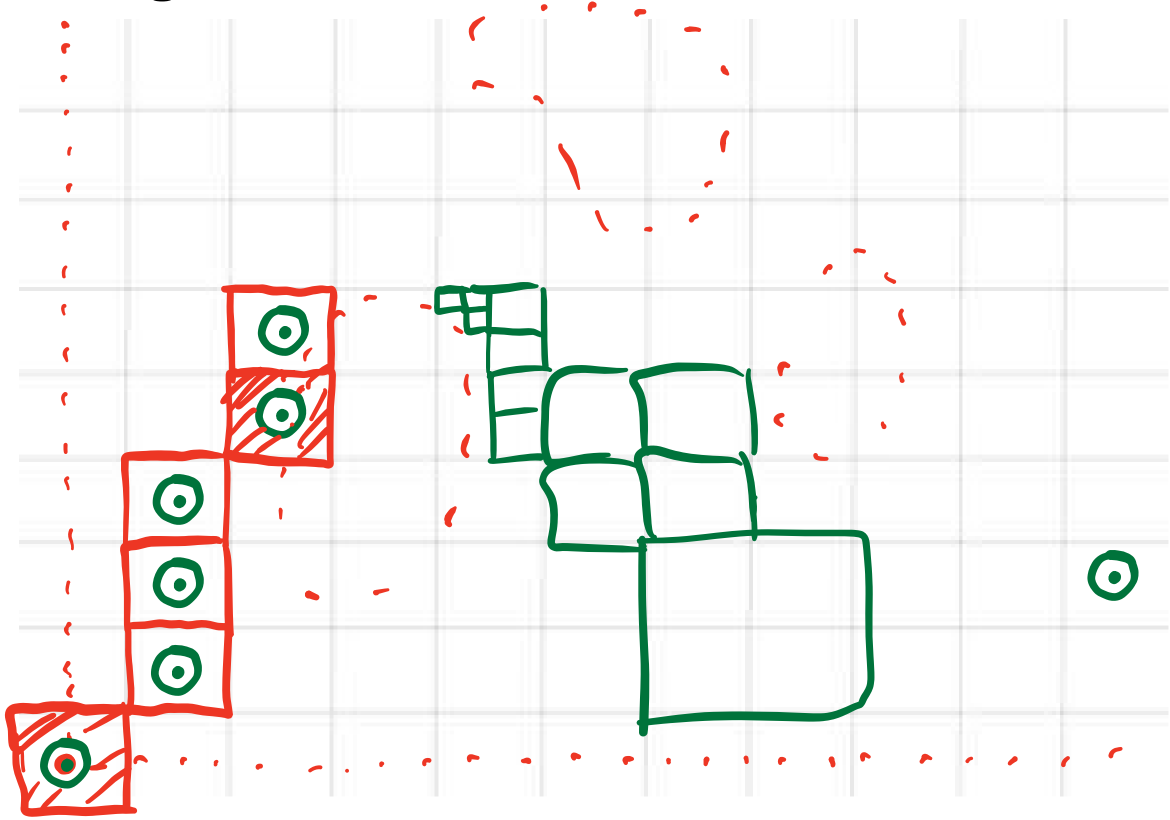
Assumptions

- We are **not concerned** with the **equations of motion (EOM)** of the robot, ie. the robot can move in any direction (up/down, left/right, diagonally) in the grid of cells

Discretization (for this class)

- Given:
  - List of **points** in configuration space that represent **obstacles**
  - Circular robot with specified radius, r

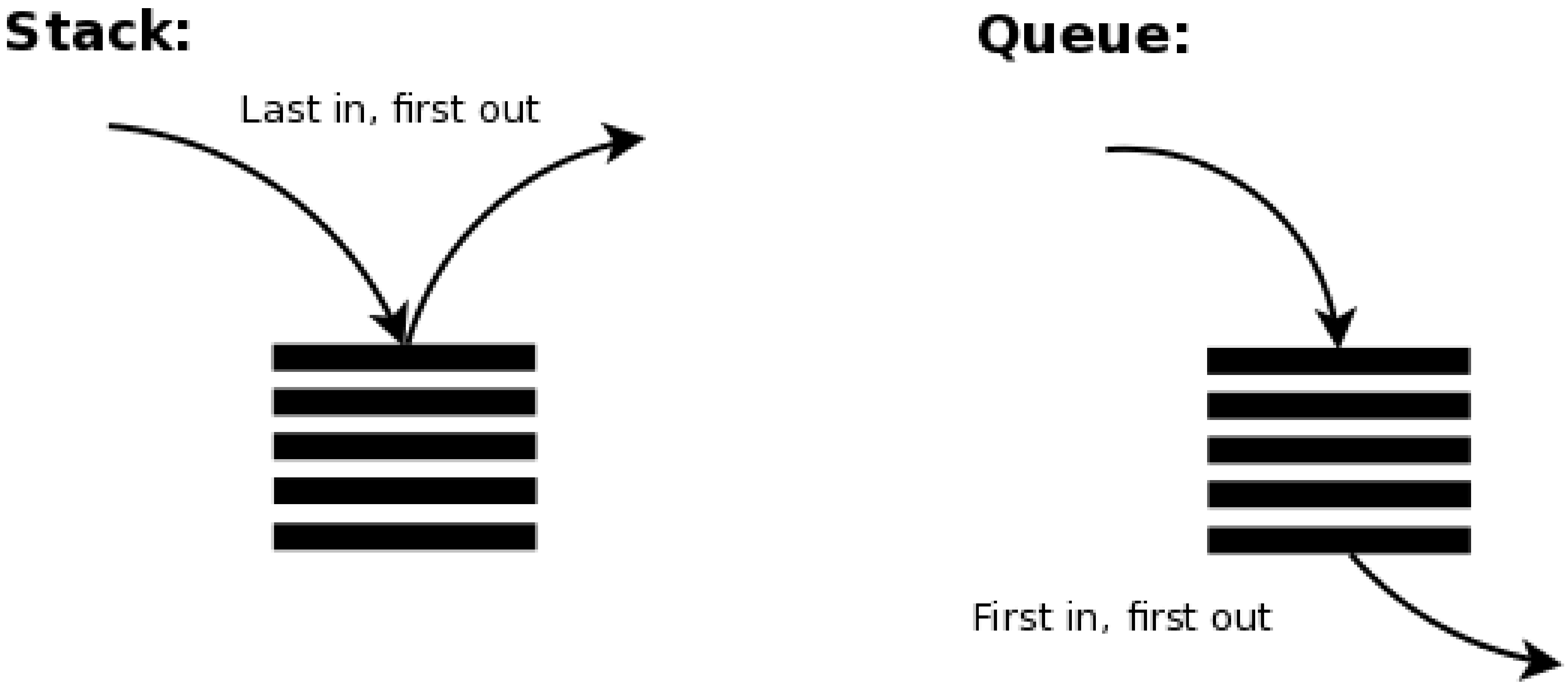
- Task:
  - Create discretization grid of traversable areas



Graphs and Trees

- **Search-based** planners represent the **C-space as a graph** through discretization
- A graph consists of a collection of **nodes** and **edges**
- Edges **connect** two nodes.
- **Nodes** typically represent **robot states**, while edges indicate the ability **to move between nodes without collision**
- Edges are often **weighted** by the **cost** to **move** from one node to another
- A **tree** is a **directed** graph (edges can only move in one direction) with **no cycles** and each node has **at most one parent**

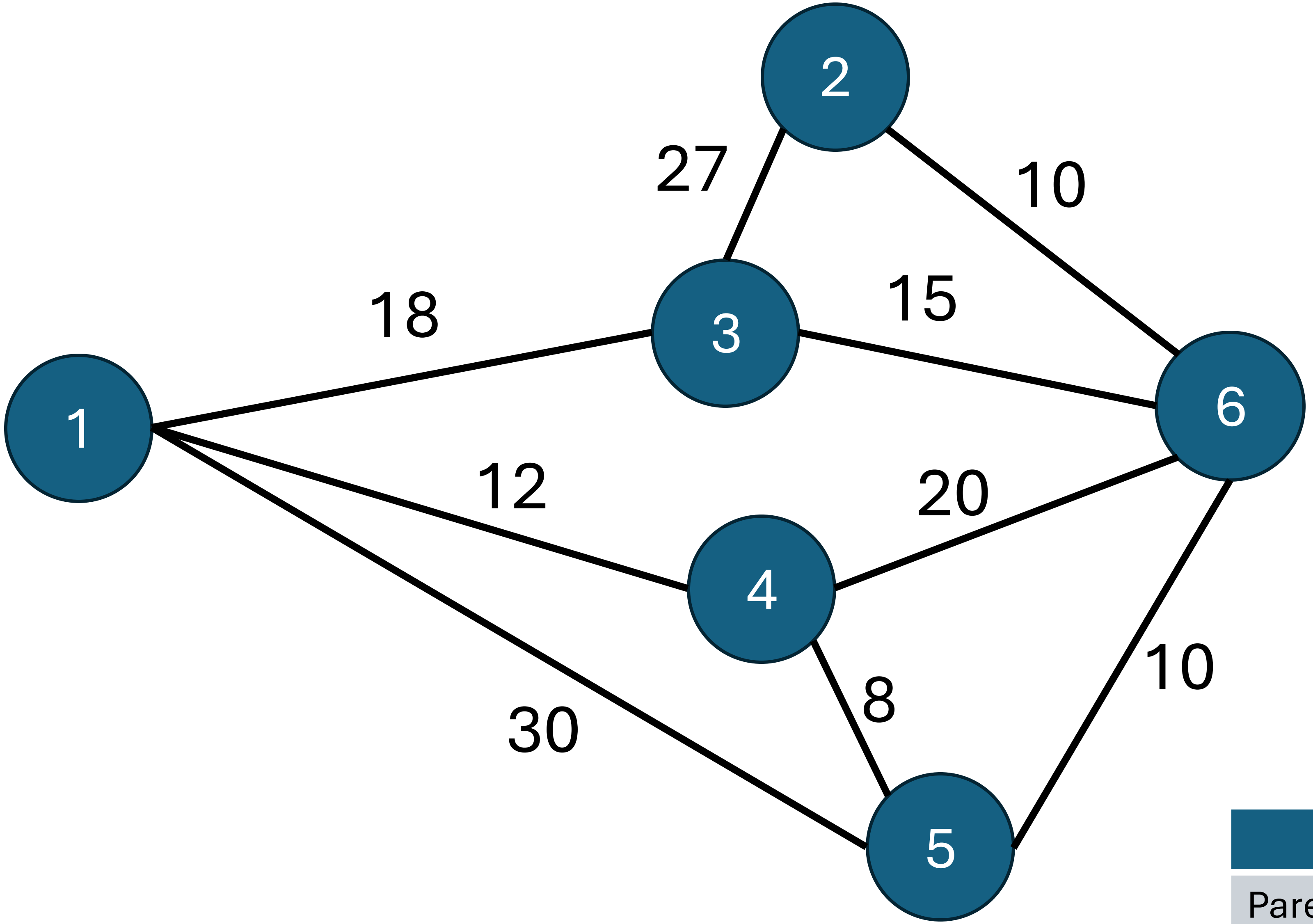
Stack vs Queue



Depth-First Search

- Search **as far down one tree branch** as possible before backtracking and searching the next available branch.
- Uses a stack structure where the **newest unexplored node** is explored

Depth-First Search Example (Generic Graph)



OPEN

CLOSED

	1	2	3	4	5	6
Parent Node						

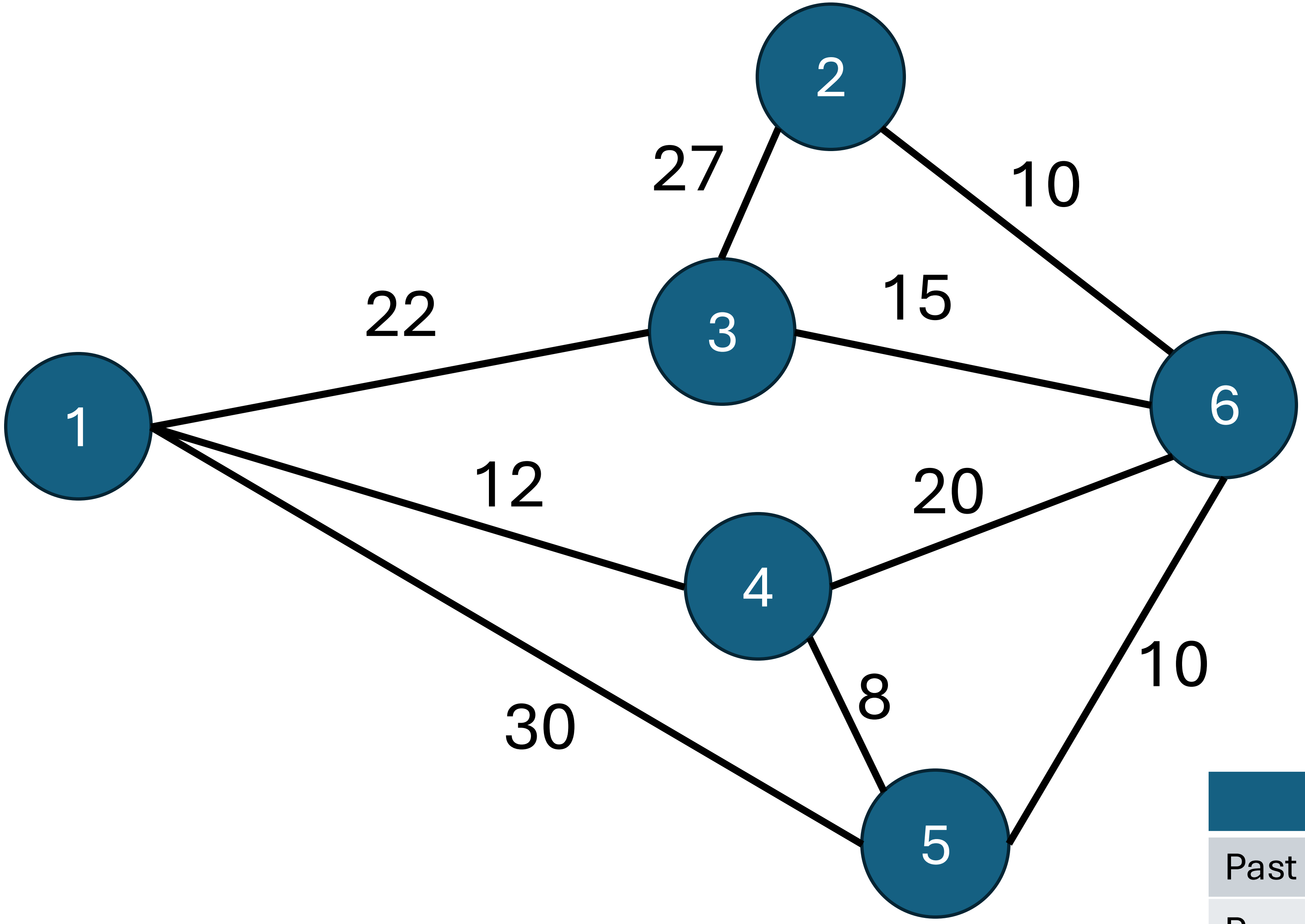
Depth-First Search Pseudocode

```
open_set, closed_set = dict(), dict()
open_set[self.calc_index(start_node)] = start_node

while True:
    if len(open_set) == 0:
        break
    current_node = open_set[-1]

    if current_node = goal_node:
        goal_node.parent = current_node.parent
        goal_node.cost = current_node.cost
        break

    for motion in allowed_motions:
        node = create_node(motion)
        if node in closed_set:
            continue
        if node not valid:
            continue
        if node not in open_set:
            open_set += node
        else:
            if node.cost < open_set[node].cost:
                open_set[node] = node

    del open_set[current_node]
    closed_set += current_node

calculate_final_path()
```

Dijkstra’s Algorithm

- Explore branches connected to the node with the **lowest total cost**. Keep exploring until the next node to be explored is the goal node.
- Considered a **variant** of a **breadth-first search**

Dijkstra Example (Generic Graph)



OPEN

CLOSED

	1	2	3	4	5	6
Past Cost						
Parent Node						

Dijkstra Pseudocode

```
open_set, closed_set = dict(), dict()
open_set[self.calc_index(start_node)] = start_node

while True:
    if len(open_set) == 0:
        break
    current_node = min(open_set.cost)
    if current_node = goal_node:
        goal_node.parent = current_node.parent
        goal_node.cost = current_node.cost
        break

    for motion in allowed_motions:
        node = create_node(motion)
        if node in closed_set:
            continue
        if node not valid:
            continue
        if node not in open_set:
            open_set += node
        else:
            if node.cost < open_set[node].cost:
                open_set[node] = node

    del open_set[current_node]
    closed_set += current_node

calculate_final_path()
```

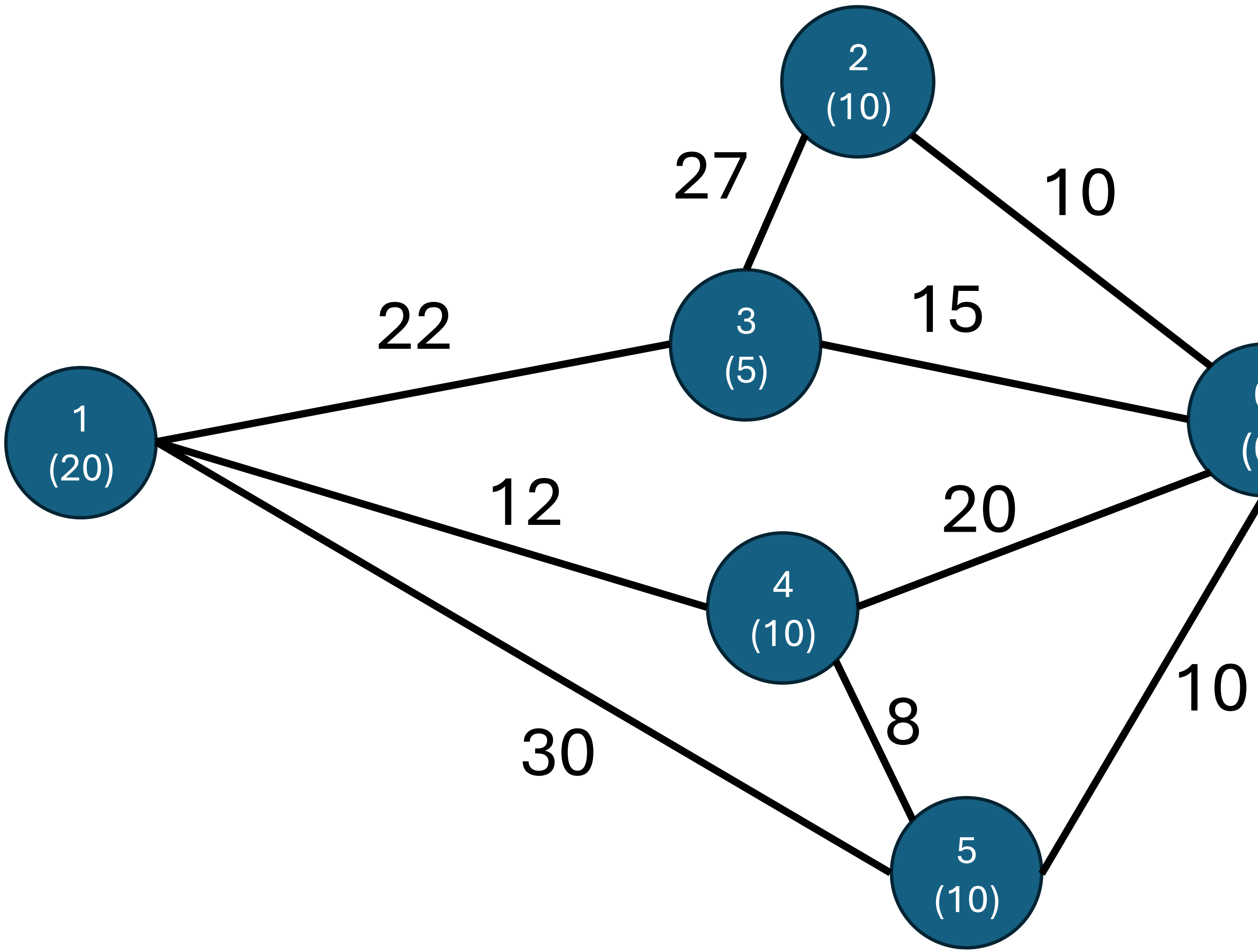
A* Search

- Operates in the same way as Dikstra's algorithm, but with an **added heuristic measure**
- The **heuristic** is an **estimate** of how far away the goal node is from any particular node.
- There are 2 requirements for the heuristic function:
  - **Always optimistic** (estimated remaining path length is less than actual path length). This estimate serves as a lower bound on the cost to go.
  - Simple and easy to evaluate

A* Pseudocode

```
open_set, closed_set = dict(), dict()
open_set[self.calc_index(start_node)] = start_node
while True:
    if len(open_set) == 0:
        break
    current_node = min(open_set.cost + calc_heuristic(open_set))
    if current_node = goal_node:
        goal_node.parent = current_node.parent
        goal_node.cost = current_node.cost
        break

    for motion in allowed_motions:
        node = create_node(motion)
        if node in closed_set:
            continue
        if node not valid:
            continue
        if node not in open_set:
            open_set += node
        else:
            if node.cost < open_set[node].cost:
                open_set[node] = node
    del open_set[current_node]
    closed_set += current_node

calculate_final_path()
```

A* Example (Generic Graph)



OPEN

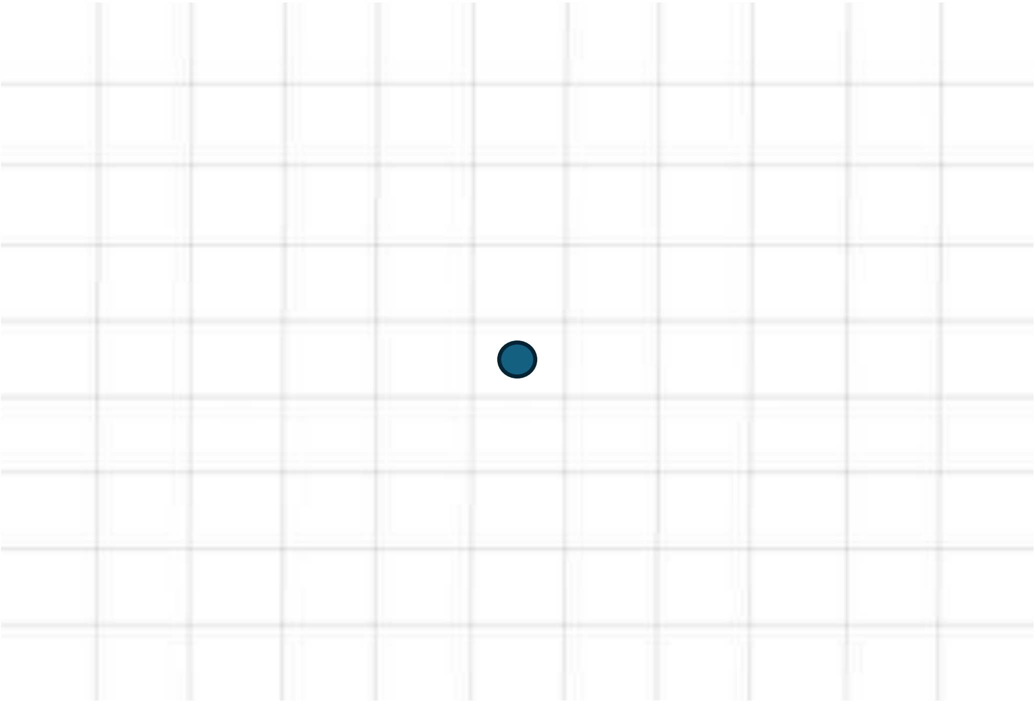
CLOSED

	1	2	3	4	5	6
Past Cost						
Heuristic						
Est Total Cost						
Parent Node						

DFS vs Dijkstra (and A*)

Exploration patterns

DFS



Dijkstra (and A*)