Pathfinding Algorithms

Mersad Hassanjani

@AICUP2021

OUTLINE

+Introduction

+Dijkstra’s algorithm

+A* algorithm

Introduction

+ Agent movement is one of the greatest challenges in the design of realistic Artificial Intelligence (AI) in computer game
+ Finding a path from any coordinate in the game world to another is pathfinding responsibility
+ Pathfinding inevitably leads to a drain on CPU resources especially if the algorithm wastes valuable time searching for a path that turns out not to

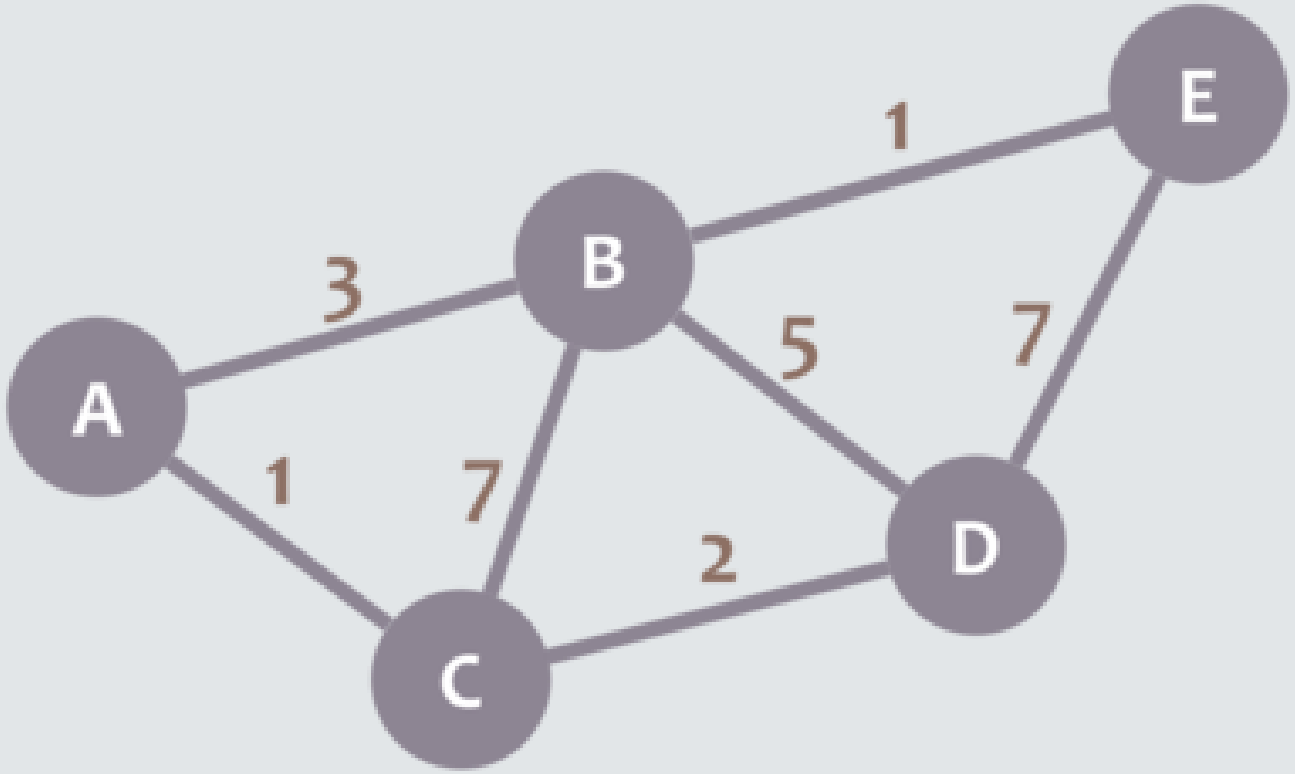
Dijkstra's algorithm

+ Lets us prioritize which paths to explore
+ Allows you to calculate the shortest path between one node (you pick which one) and every other node in the graph
+ When movement costs vary, we use it instead of Breadth First Search
+ Let's see an example!

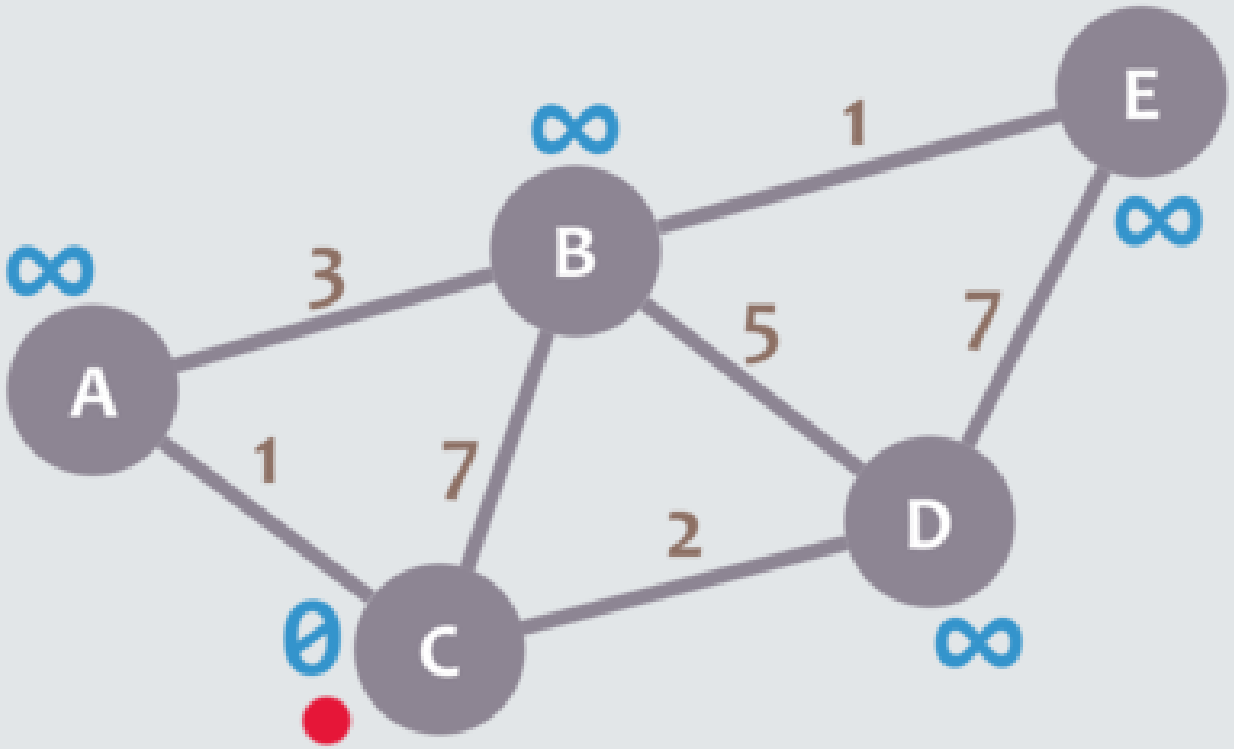
Pseudo code

1. Mark your selected initial node with a current distance of 0 and the rest with infinity.
2. Set the non-visited node with the smallest current distance as the current node C.
3. For each neighbor N of your current node C: add the current distance of C with the weight of the edge connecting C-N. If it's smaller than the current distance of N, set it as the new current distance of N.
4. Mark the current node C as visited.
5. If there are non-visited nodes, go to step 2.

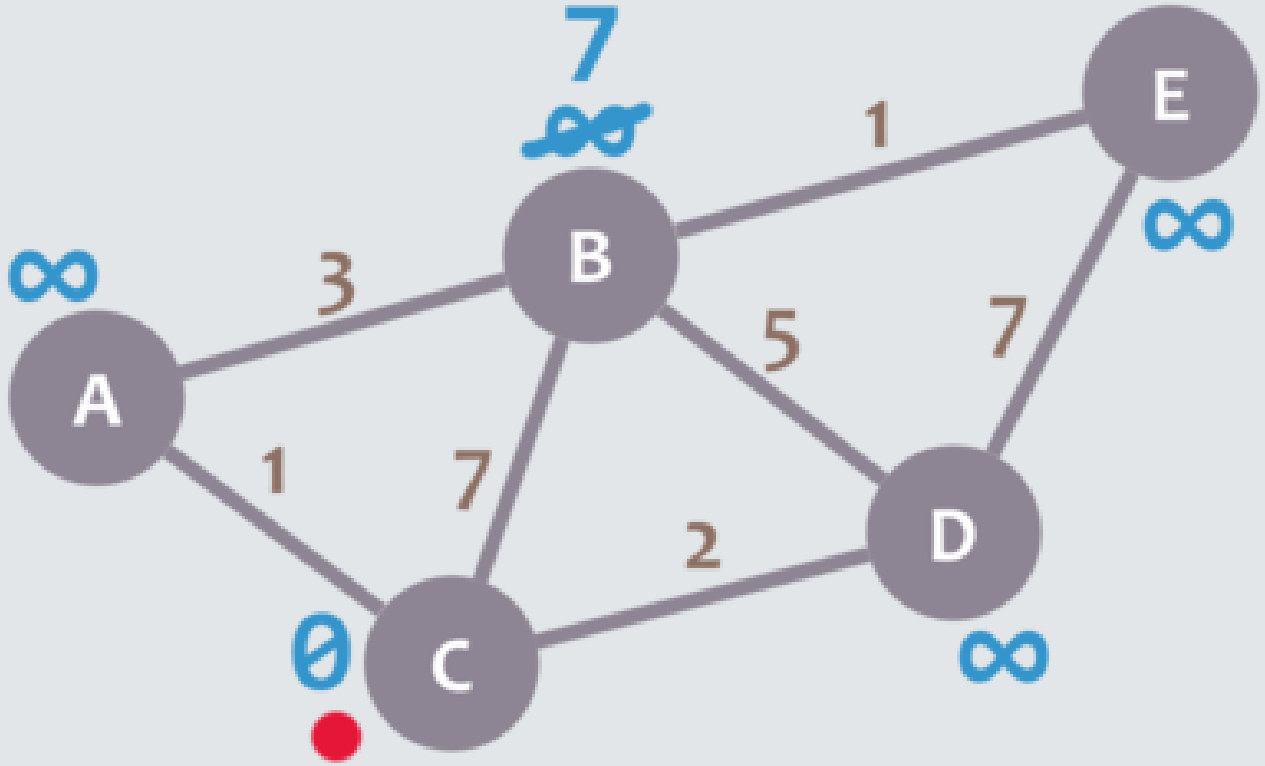
Dijkstra Example



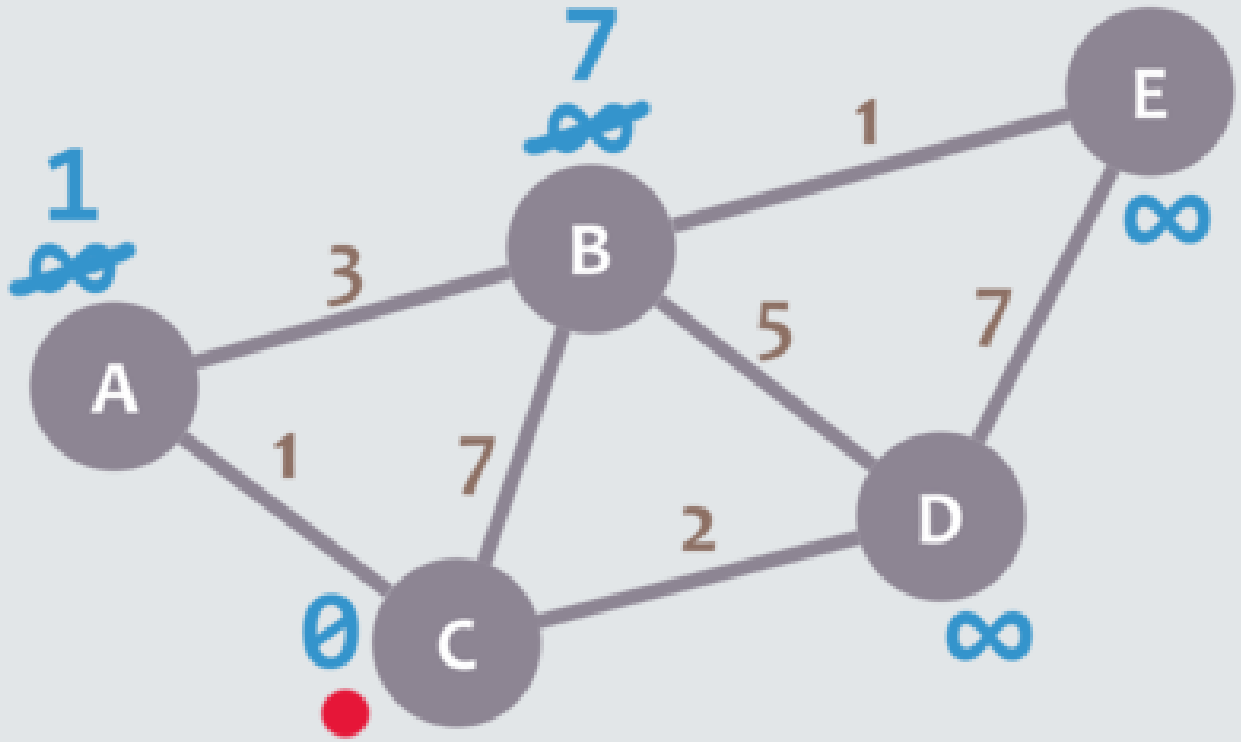
Dijkstra Example



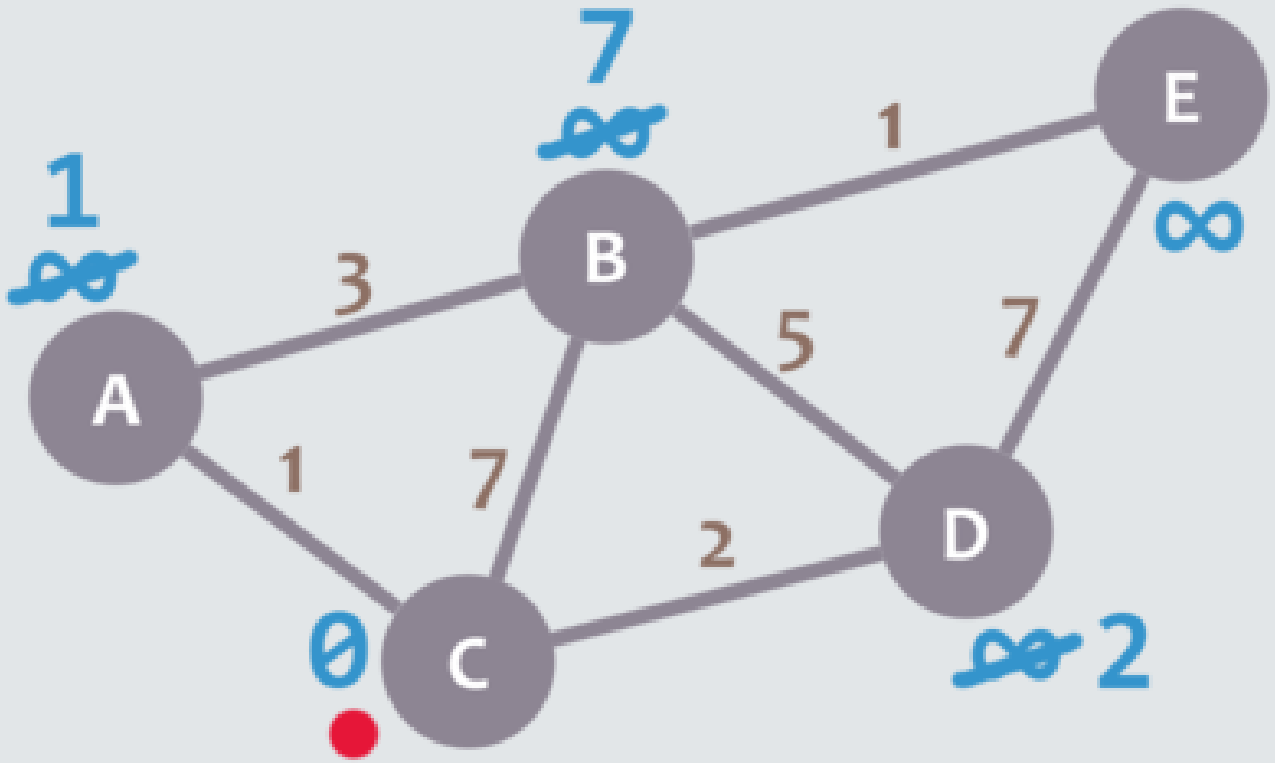
Dijkstra Example



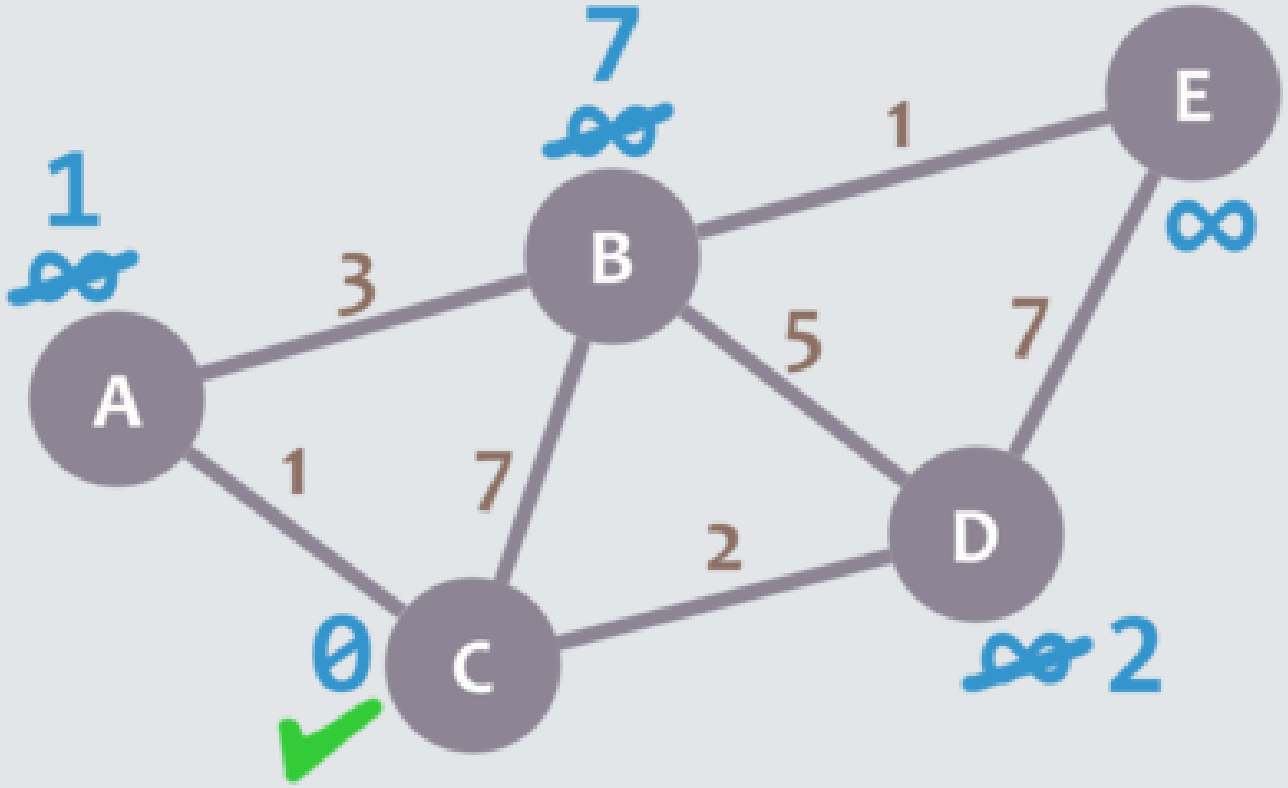
Dijkstra Example



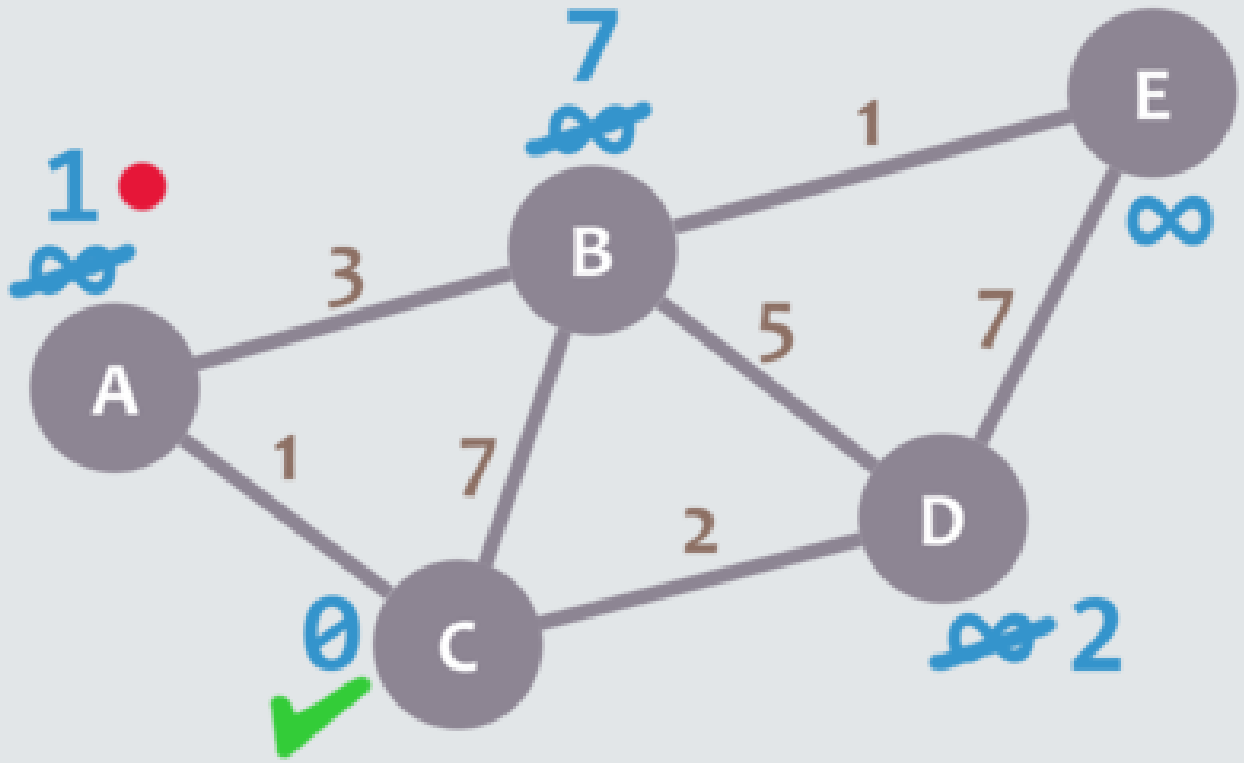
Dijkstra Example



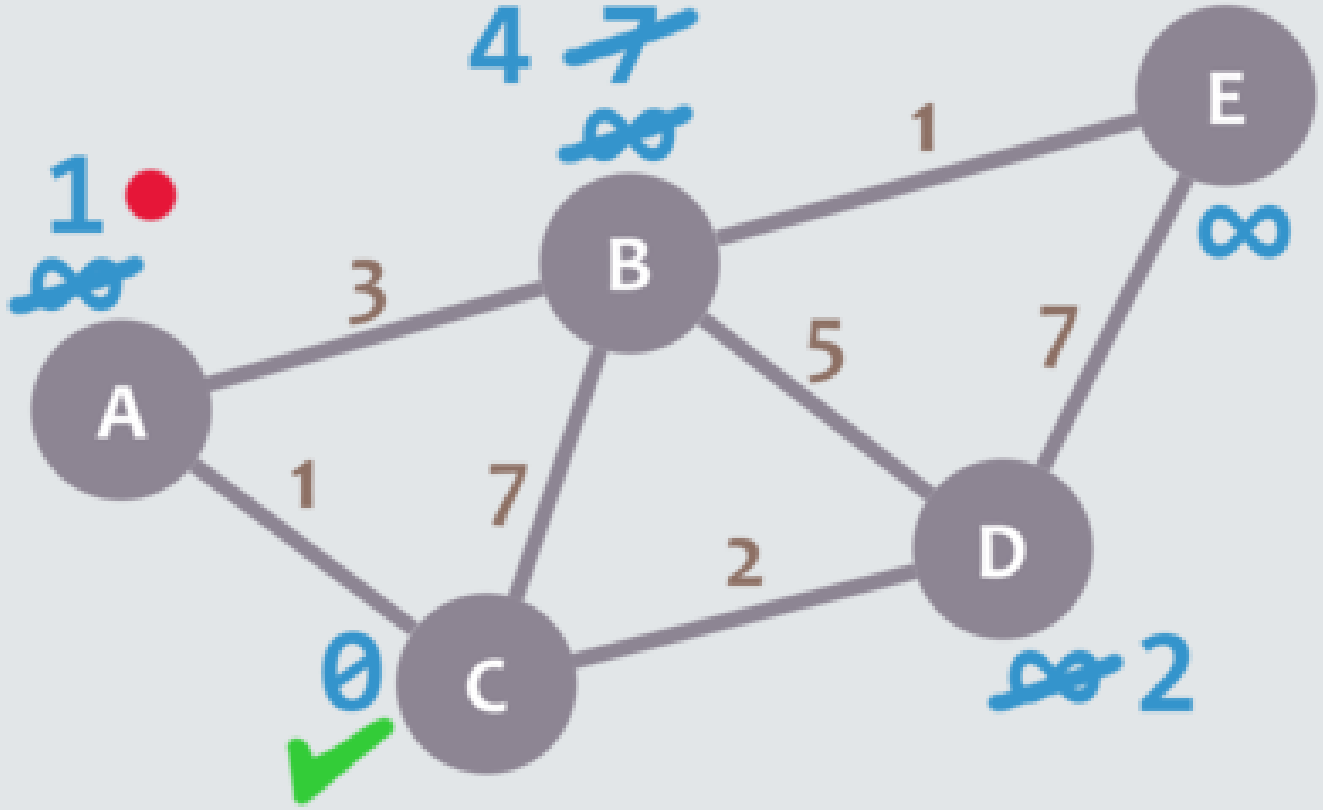
Dijkstra Example



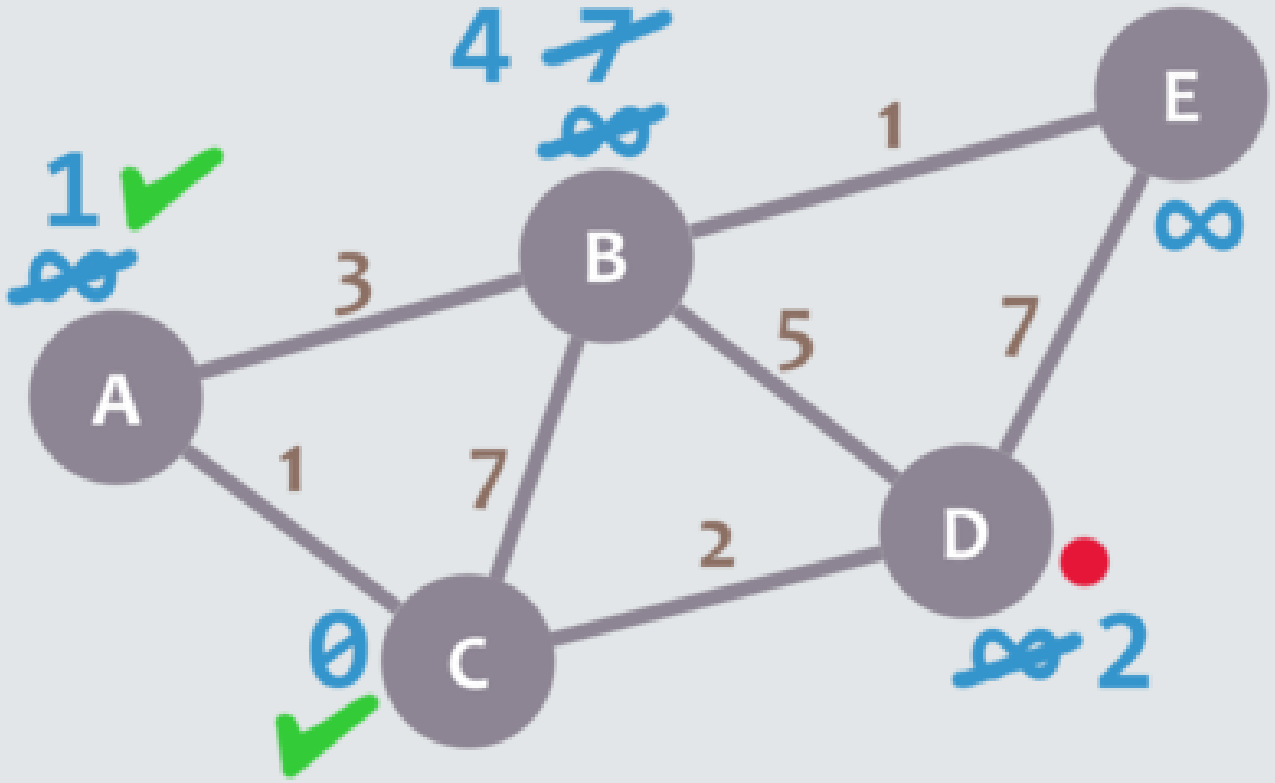
Dijkstra Example



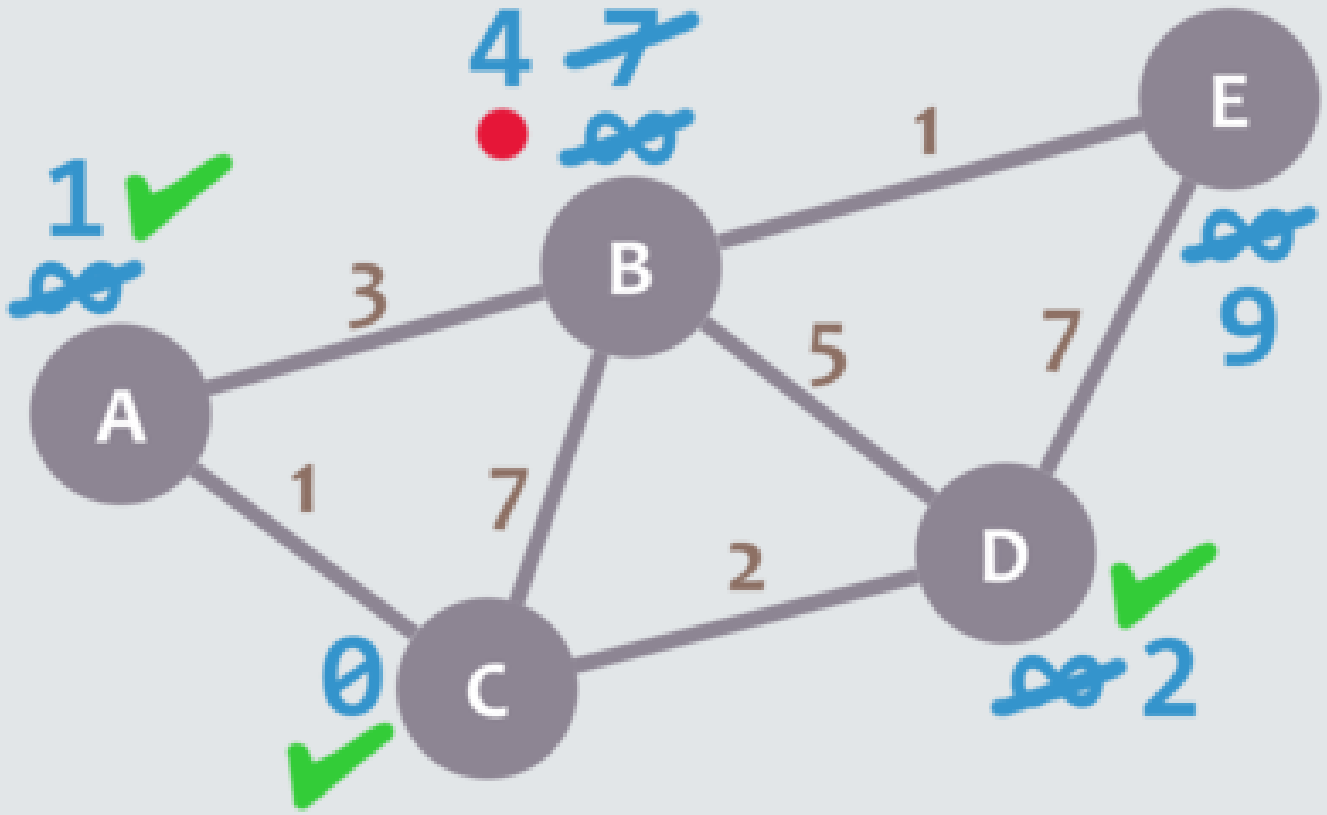
Dijkstra Example



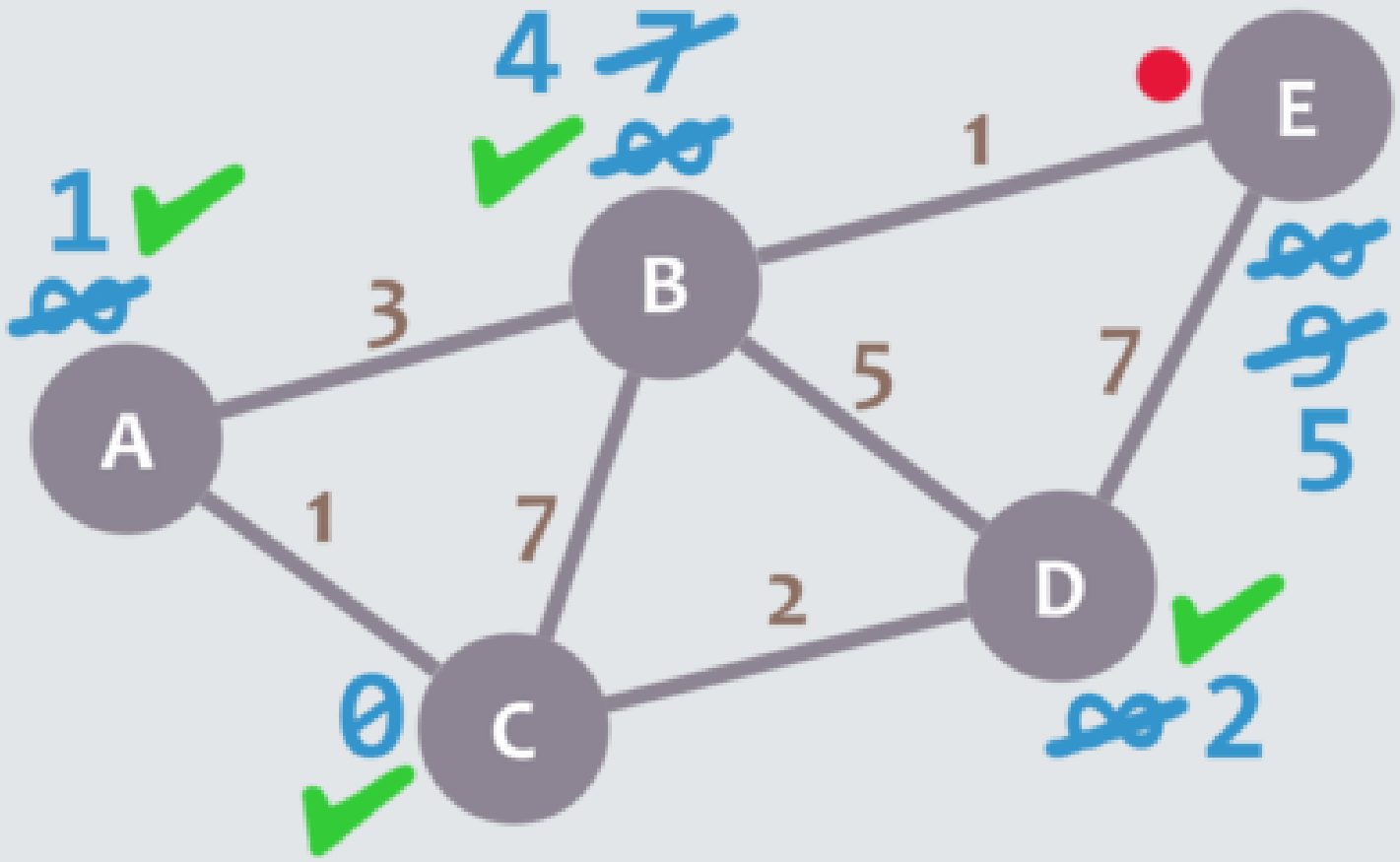
Dijkstra Example



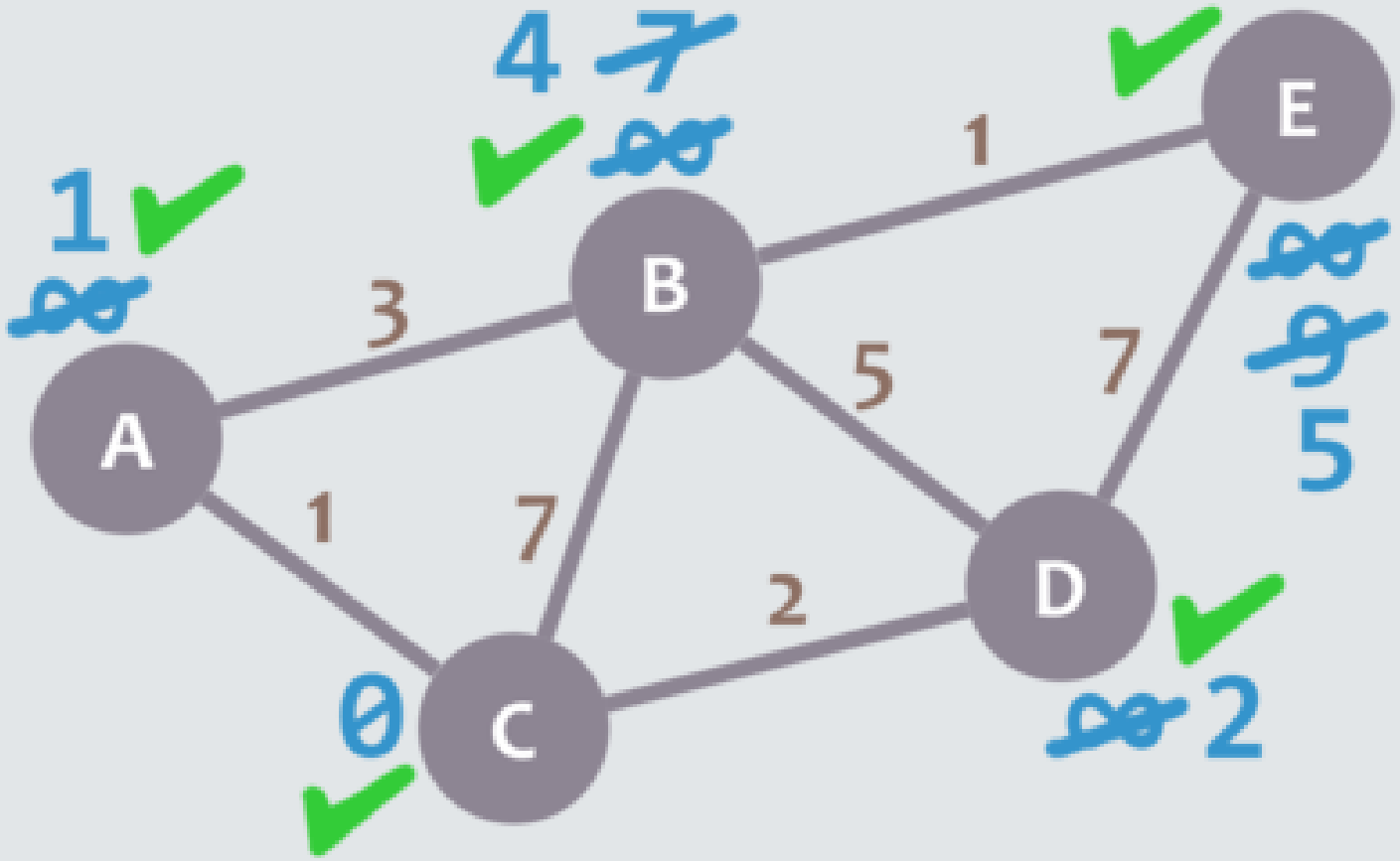
Dijkstra Example



Dijkstra Example



Dijkstra Example



Some notes

- If you only need the path between two specific nodes, you can stop the algorithm as soon as you mark your second node as visited.
- Sometimes, there are several minimum paths between two nodes (different paths with the same weights). If you wish, you can keep track of all those variants: in step 3, if there is a tie between the calculated value and the current distance, save both the old current path and the new one. This complicates the tracking, but it may be useful to you.
- If you finish the algorithm because there are not unvisited nodes left but there are nodes which minimum distance is still infinity, those nodes don't have any valid path to the original node.

A* Algorithm

- Dijkstra's Algorithm works well to find the shortest path, but it wastes time exploring in directions that aren't promising
- Greedy Best First Search explores in promising directions, but it may not find the shortest path
- The A* algorithm uses *both* the actual distance from the start and the estimated distance to the goal

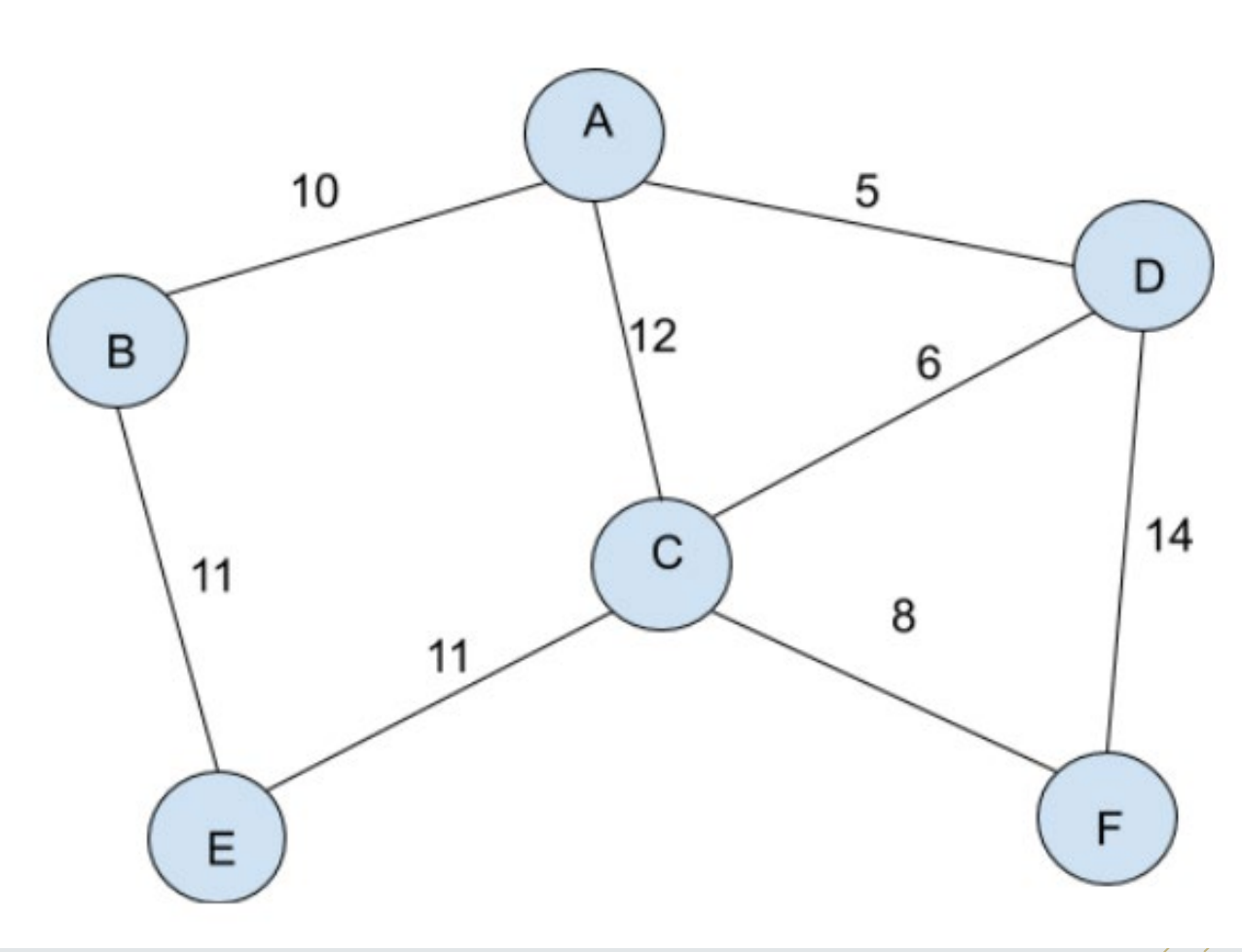
A* parameters

- **g** : the cost of moving from the initial cell to the current cell. Basically, it is the sum of all the cells that have been visited since leaving the first cell.
- **h** : also known as the heuristic value, it is the estimated cost of moving from the current cell to the final cell. The actual cost cannot be calculated until the final cell is reached. Hence, h is the estimated cost. We must make sure that there is never an over estimation of the cost.
- **f** : it is the sum of g and h. So, f = g + h

Pseudo Code

1. Define a list OPEN. Initially, OPEN consists solely of a single node, the start node **S**.
2. If the list is empty, return failure and exit.
3. Remove node **n** with the smallest value of **f(n)** from OPEN and move it to list CLOSED. If node **n** is a goal state, return success and exit.
4. Expand node **n**.
5. If any successor to **n** is the goal node, return success and the solution by tracing the path from goal node to **S**. Otherwise, go to Step 6.
6. For each successor node, Apply the evaluation function **f** to the node. If the node has not been in either list, add it to OPEN.
7. Go back to Step 2.

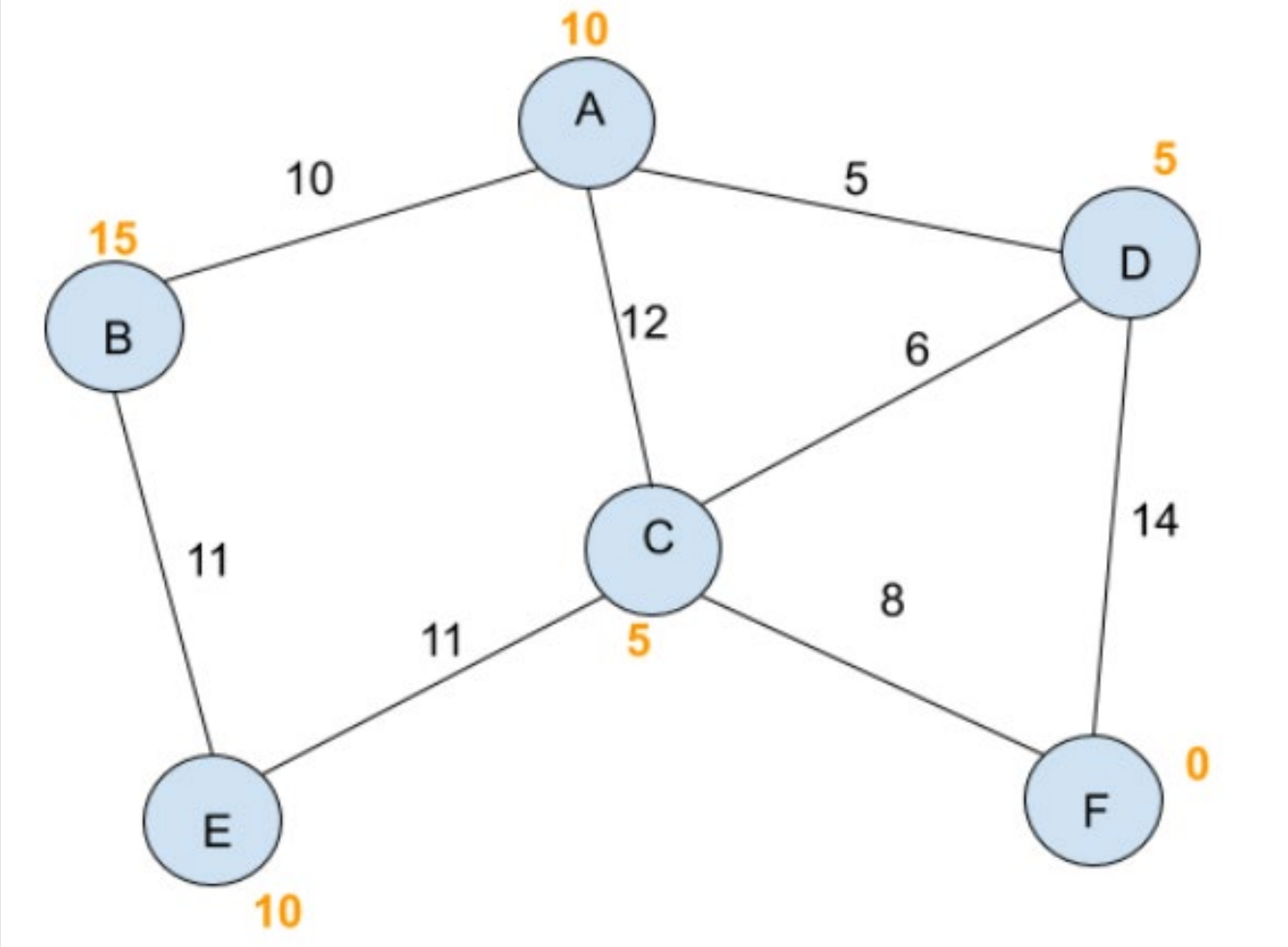
A* example



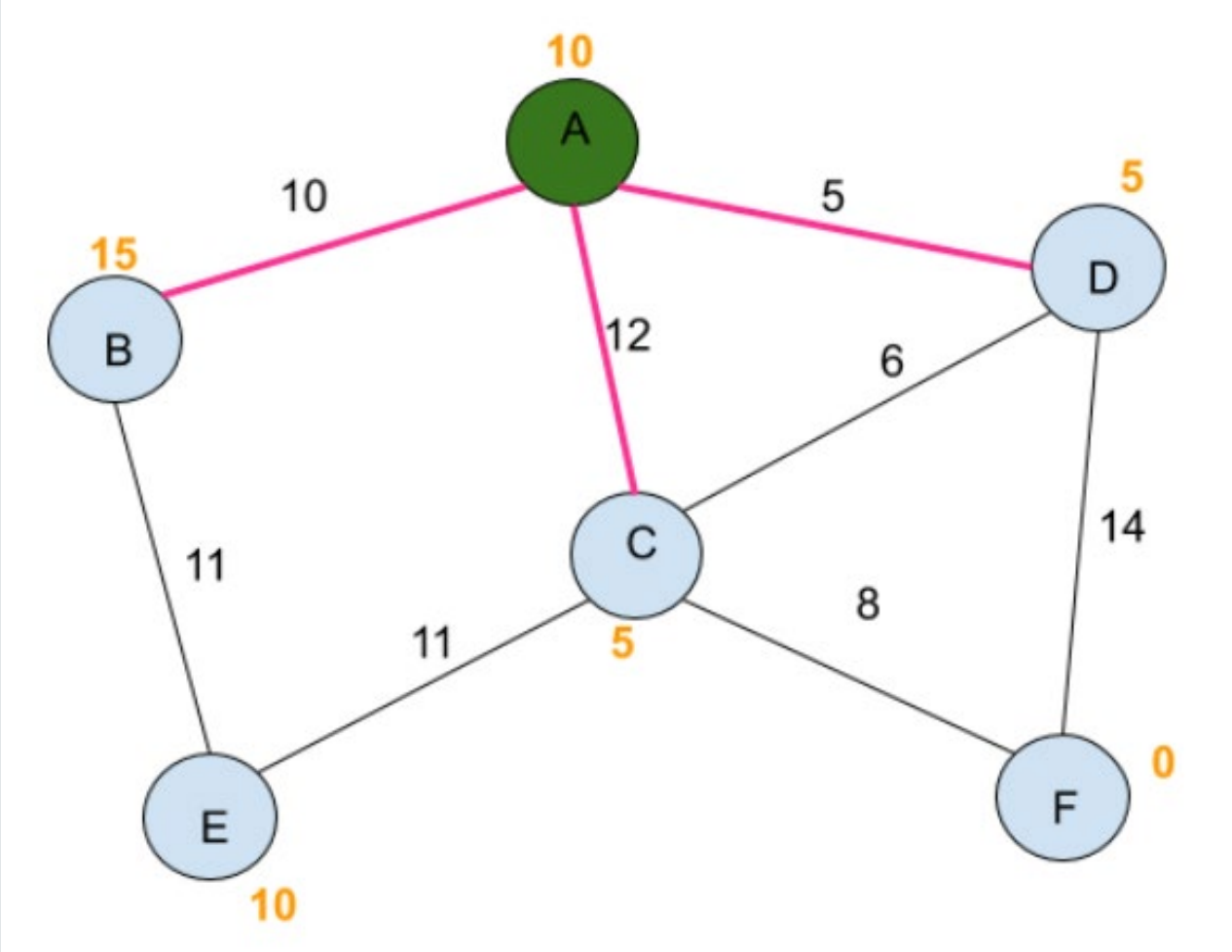
A* example

Node	Estimated cost
A	10
B	15
C	5
D	5
E	10
F	0

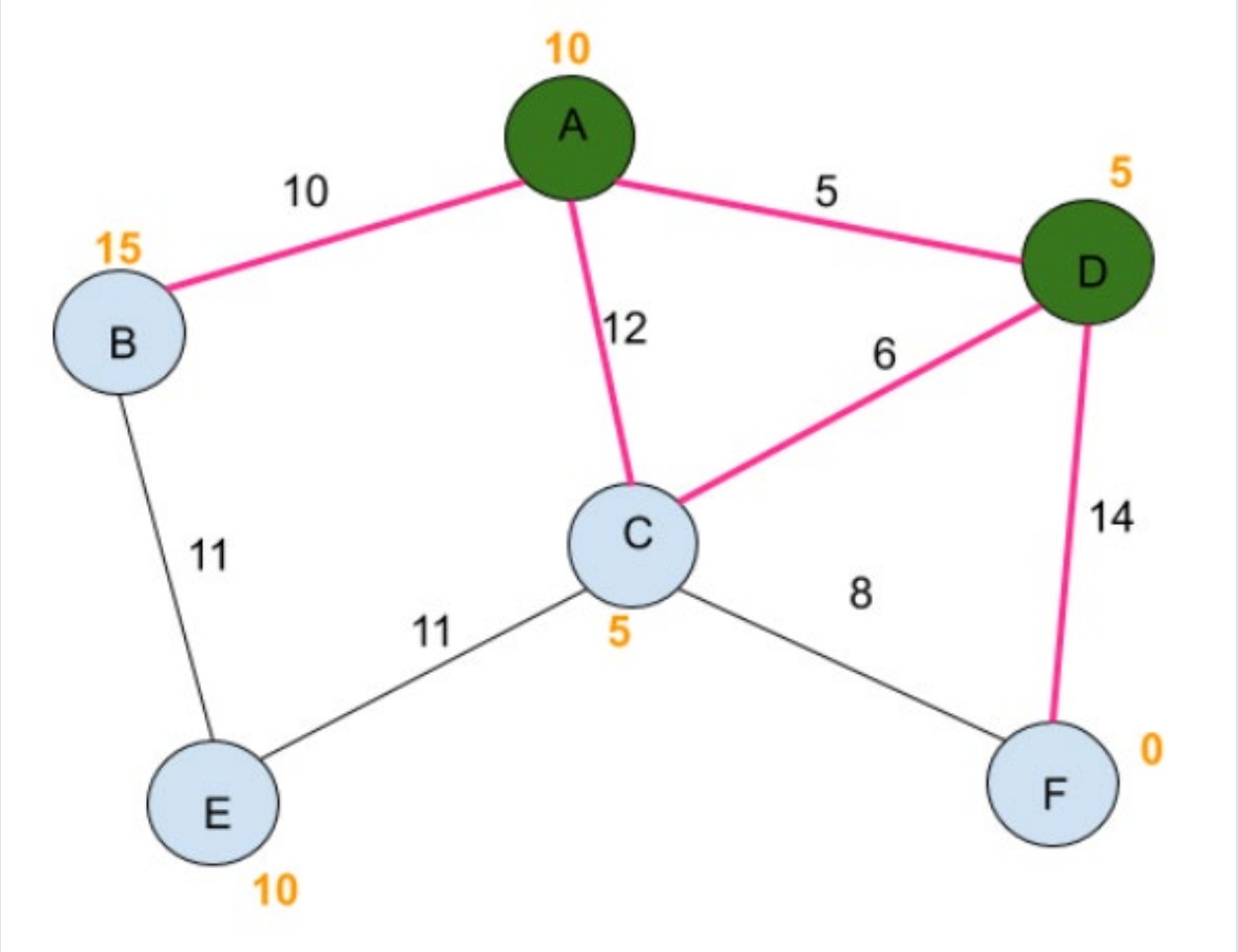
A* example



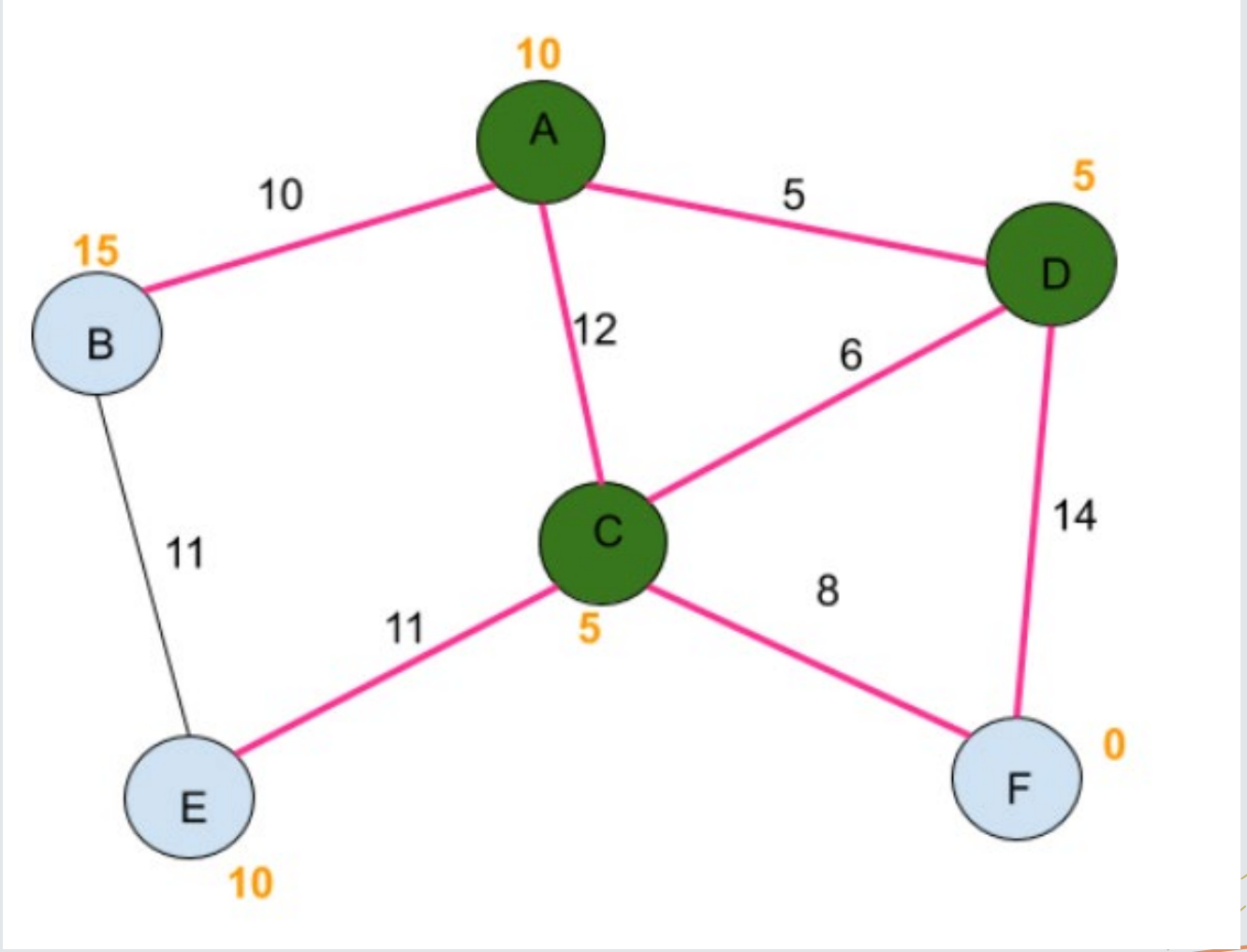
A* example



A* example



A* example



Some notes

+ **Dijkstra's** algorithm will **always** find the shortest path between the start node and a target node. In addition, Dijkstra's algorithm will find the shortest path between the start node and every other node in the graph. However, the algorithm can be **inefficient** as it will 'waste time' evaluating routes that could be ignored.
+ The **A*** algorithm is more **efficient** than Dijkstra's algorithm. By using a heuristic function to provide an estimate of the cost of the path between each node and the target node, it can make better choices about the next path to take and will find the shortest path faster.
+ The greatest challenge in selecting A* is the need for a **good heuristic function**. The time it takes to provide the heuristic must not cancel out any time savings in the process of pathfinding. In addition, the heuristic must not overestimate the cost of the path. If h(n) is always lower than (or equal to) the cost of moving from n to the target node, then A* is guaranteed to find the shortest path.

Check In Tomorrow for implementations!

Thank You!