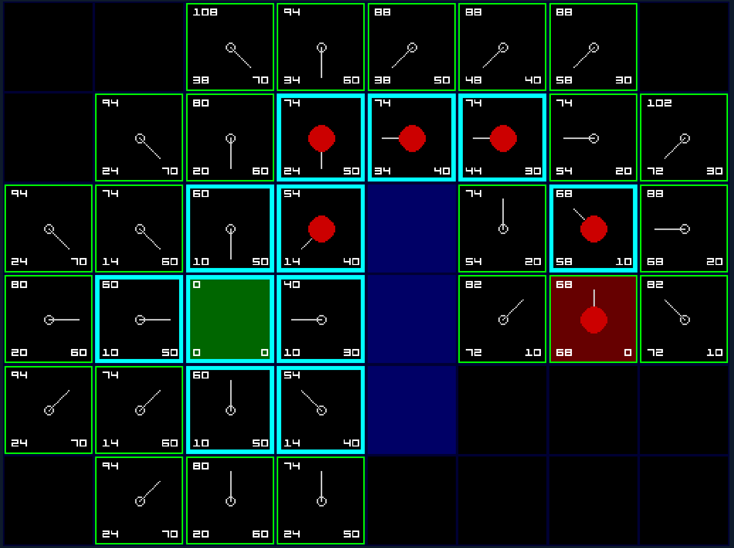
A* - Part 1



Canberra Institute of Technology

aie SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Lecture Contents

- Dijkstra’s Algorithm Review.
- What is a Heuristic?
- What is A*?
- F = G+H
- Picking a Heuristic
- A* Walkthrough

Dijkstra’s Algorithm Review

- Dijkstra’s algorithm is a modification of BFS that finds the shortest path from a node to all other nodes in a graph.

- It uses a priority queue to process the shortest distance node each time, therefore guaranteeing it traverses the shortest path.

What is a Heuristic?

- A heuristic is a guess.

- Sometimes, calculating a piece of information is computationally expensive. In these cases, there's often another way to calculate the answer that is:
  - Not guaranteed to be correct, but
  - Is significantly faster.
  - We're basically making an educated guess at the right answer without going to all the work to really figure it out.

- We call these 'guesses' Heuristics.

What is a Heuristic?

- Heuristics are used in many algorithms and in a lot of cases allow us to calculate a result that is almost as good as the real calculation (sometimes even the same) in a tiny fraction of the time.

What is A*?

- A* (pronounced A-Star) is a heuristic algorithm.
- A* is a modification of Dijkstra's algorithm.
  - It can typically be implemented by only changing one or two lines of code.
- It changes Dijkstra's so that instead of picking the node that's closest to the start to process each time, it picks the one that it thinks will get it closest to the goal.
  - The way A* figures out what node is closest is our heuristic

F = G + H

- If you remember from Dijkstra’s, the G score is the cost to travel from the starting node to the node we are currently traversing.

- For A* we add two new scores.
  - The H score is the heuristic, for each node we calculate how far away we think it is from the goal.
  - The F score – the final score for a node is the sum of the G and H scores

Canberra Institute of Technology

aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

F = G + H

- A* works exactly the same a Dijkstra’s except instead of running the priority queue off the G score, we run it off the F score.

- You could even say the Dijkstra’s is just a special case of A* where the H cost is always 0.

Picking a Heuristic

- The only thing left for us to implement A* is calculating the heuristic.

- There are a number of common heuristics to use. For pathfinding, the two most common are:
  - Distance
    - just plain old distance. How far away, as the bird flies, is the current node from the goal.
  - Manhattan distance
    - For 2D this is the difference in the x plus the difference in the y. This typically leads to better paths for grid based graphs without diagonal movement.

Picking a Heuristic

- A* isn’t just for movement, though.
- There are many applications of graphs beyond pathfinding
  - Finding the shortest number of moves to win a board game.
- For a board game, like chess, you could use the number of pieces caught as the heuristic.

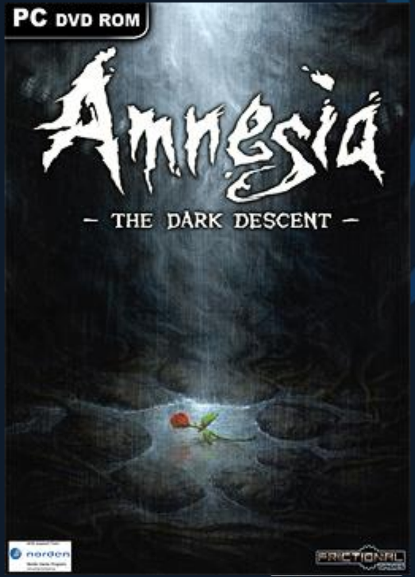
Picking a Heuristic

- One of the best things about A* is that we can easily modify how we calculate the heuristic as the game runs.
- We can add in modifiers based on how many enemies are in a given area to make agents avoid it.
- Or make distances to health pickups or powerups modify the heuristic score so that agents will naturally pick them up on their way to the goal node.

Canberra Institute of Technology
aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

Picking a Heuristic

- One of the best examples of this is Amnisia: The Dark Decent
- The monster tries to path as close to the player as possible, but is actively avoiding being able to see the player.
- This leads to many high tension sequences where the player hides under a table, or behind a box or in a cupboard with the monster walking past but avoids sequences where the player is repeatedly found which would be frustrating and pull the player out of the game.

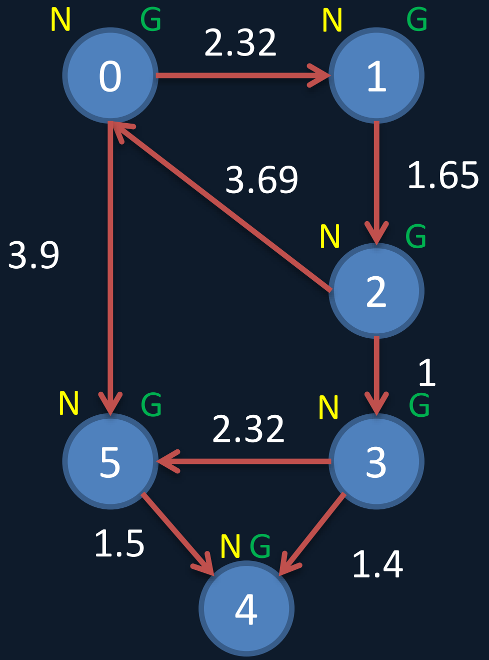


aie
SPECIALIST EDUCATORS IN GAMES, ANIMATION & FILM VFX

A* Walkthrough

- The algorithm for A* is once again, very similar to Dijkstra's Algorithm.
  - Set all Ns to null, set all Gs to infinity
  - Push start node onto priority queue, set its N to itself and its G to 0
  - While queue not empty
    - Get the current node off the end of the queue and remove it.
    - Mark it as traversed
    - If the current node is the goal node, return.
    - Loop through its edges
      - If end node not traversed
        - » Calculate current node's G + the edge cost + **heuristic of end node**
        - » If cost is less than existing F cost in end node
          - Set end node's N to the current node
          - Set end node's F to the current node's G + the edge cost + **heuristic of end node**
          - If end node not in the queue
            - Push end node onto the queue

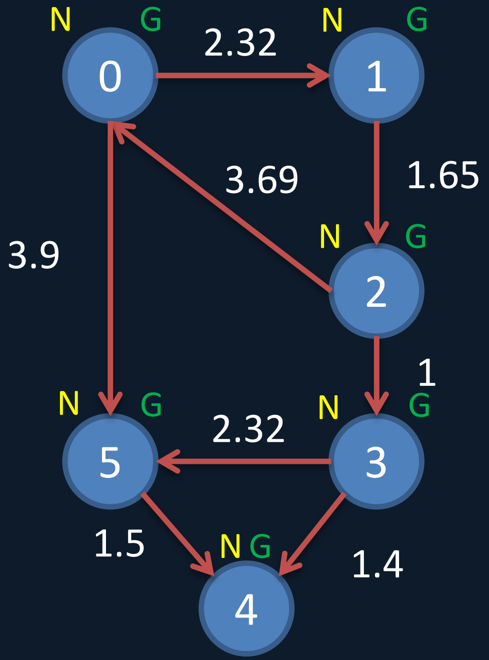
A* Walkthrough



- Just like when we looked at Dijkstra’s, we’re going to travel from node 0 to node 4.

- However, I’ve replaced all the line costs with their actual lengths.

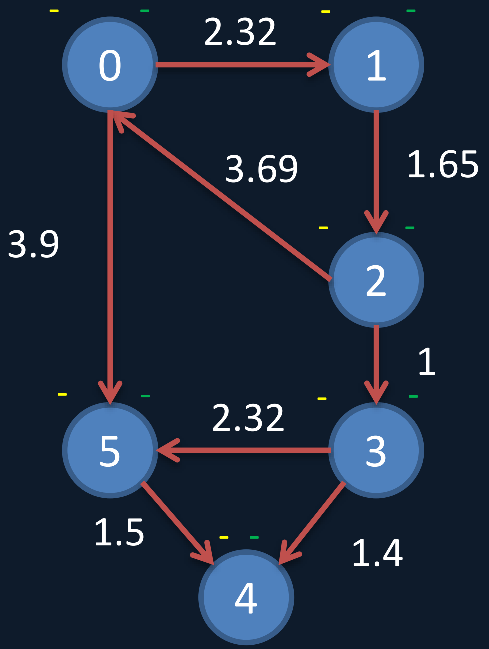
A* Walkthrough



Priority Queue:

●	= untraversed
●	= traversed

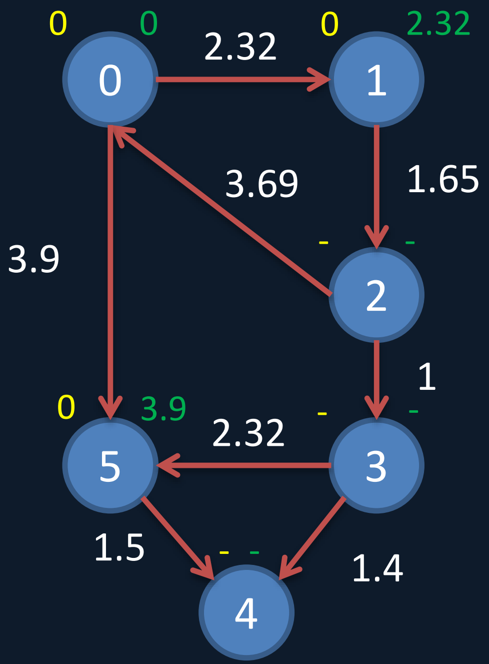
A* Walkthrough



Priority Queue:

●	= untraversed
●	= traversed

A* Walkthrough



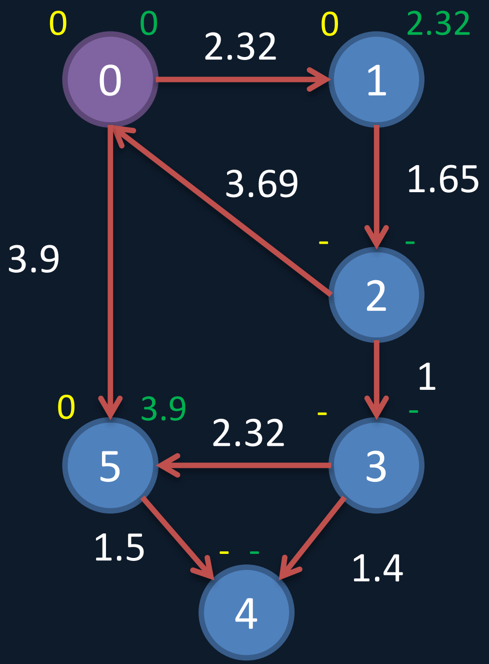
Priority Queue:

0

● = untraversed

● = traversed

A* Walkthrough

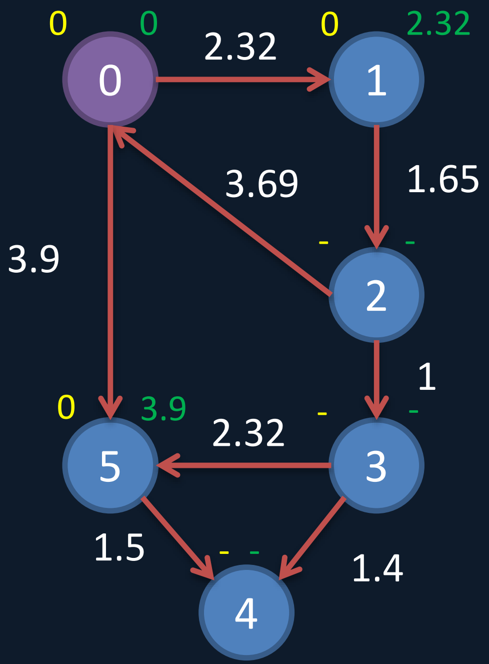


Priority Queue:

● = untraversed

● = traversed

A* Walkthrough



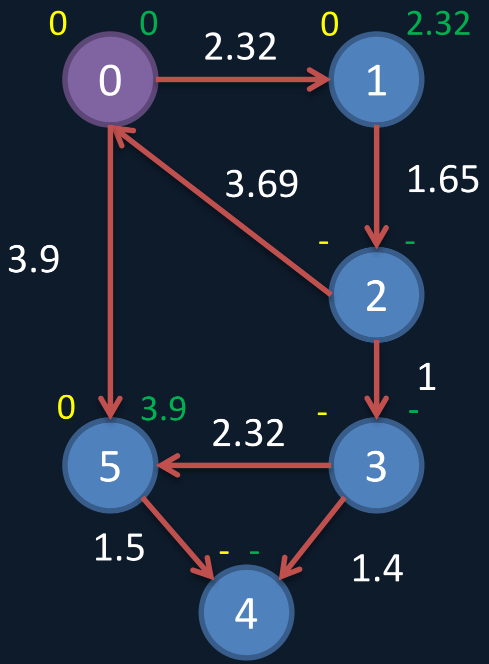
Priority Queue:

1

● = untraversed

● = traversed

A* Walkthrough

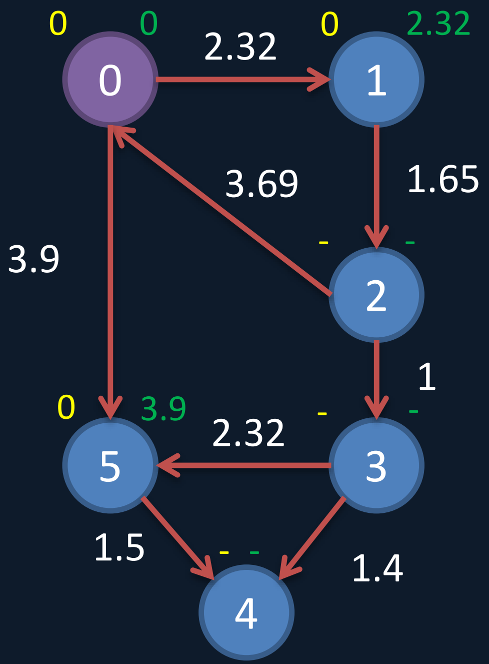


Priority Queue:

1
5

🔵	= untraversed
🟣	= traversed

A* Walkthrough



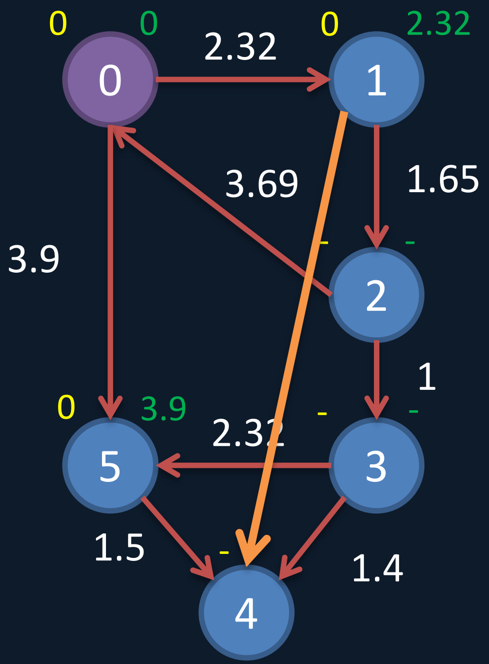
Priority Queue:

1
5

●	= untraversed
●	= traversed

- So far, the algorithm has been identical to Dijkstra's.
- However, instead of picking the node with the smallest G score from the priority queue (Node 1), we pick the node with the lowest F score.

A* Walkthrough



Priority Queue:

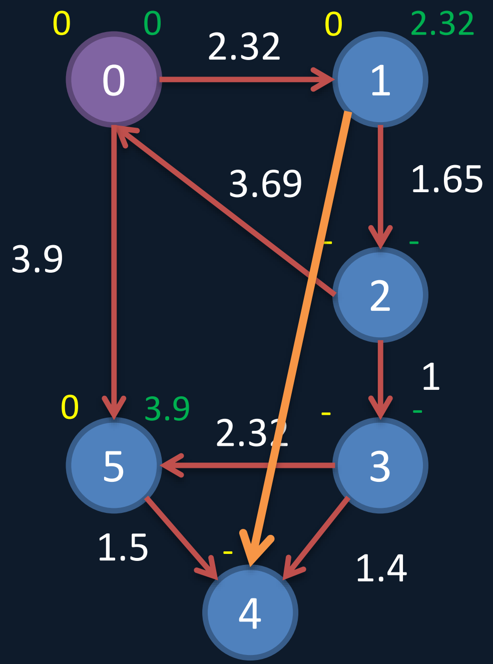
1
5

🔵 = untraversed

🟣 = traversed

- F score for Node 1 = 2.32 + H

A* Walkthrough



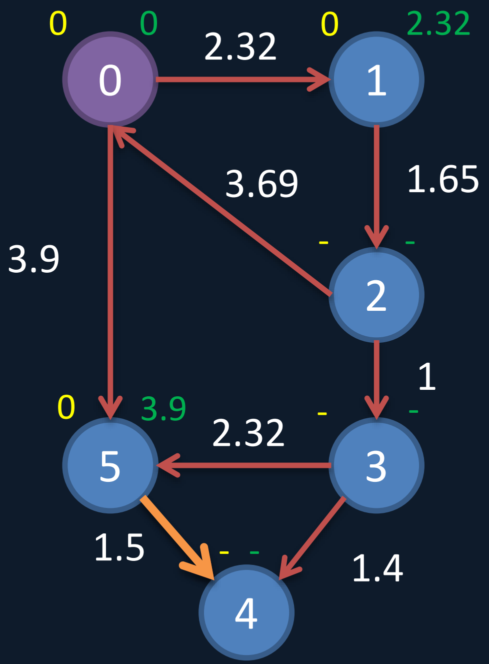
Priority Queue:

1
5

🔵	= untraversed
🟣	= traversed

- F score for Node 1 = 2.32 + H
- H = 6.08
- F = 2.32 + 6.08
- F = 8.4

A* Walkthrough



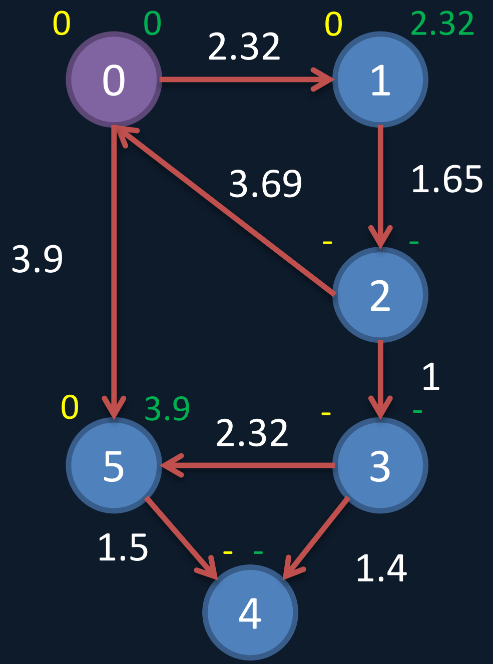
Priority Queue:

1
5

●	= untraversed
●	= traversed

- F score for Node 5 = 3.9 + H
- H = 1.5
- F = 3.9 + 1.5
- F = 5.4

A* Walkthrough



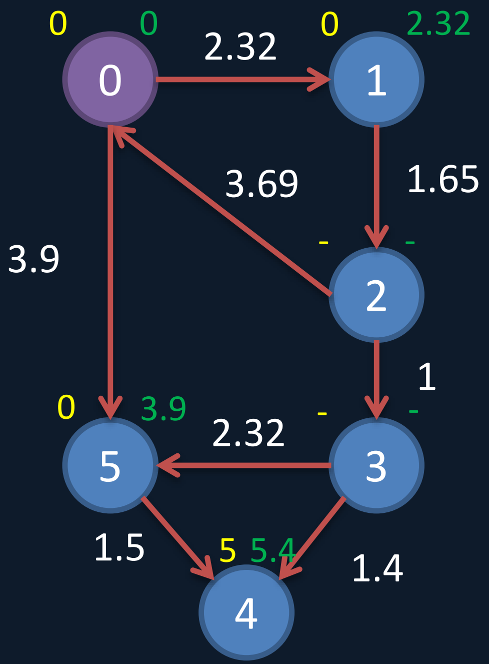
Priority Queue:

1
5

●	= untraversed
●	= traversed

- F score for Node 1 = 8.4
- F score for Node 5 = 5.4

A* Walkthrough

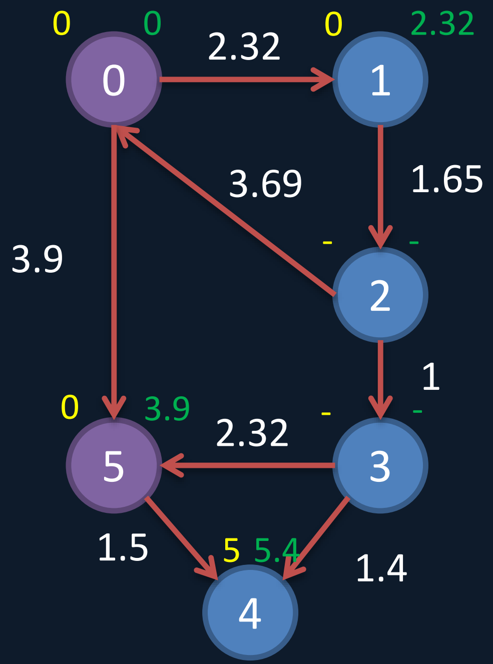


Priority Queue:

1
5

🔵	= untraversed
🟣	= traversed

A* Walkthrough

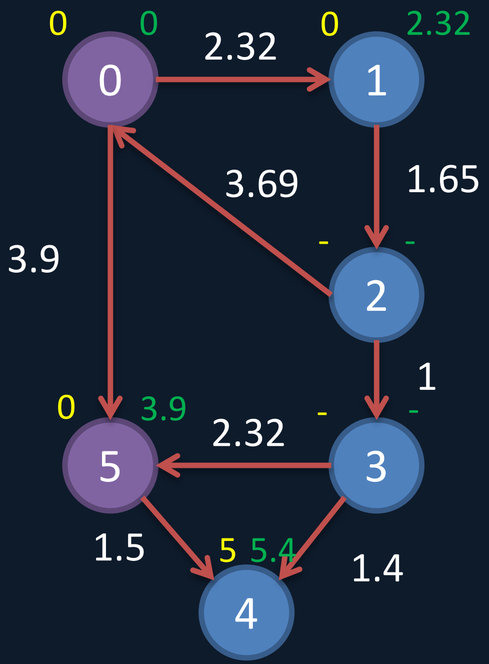


Priority Queue: 1

● = untraversed

● = traversed

A* Walkthrough



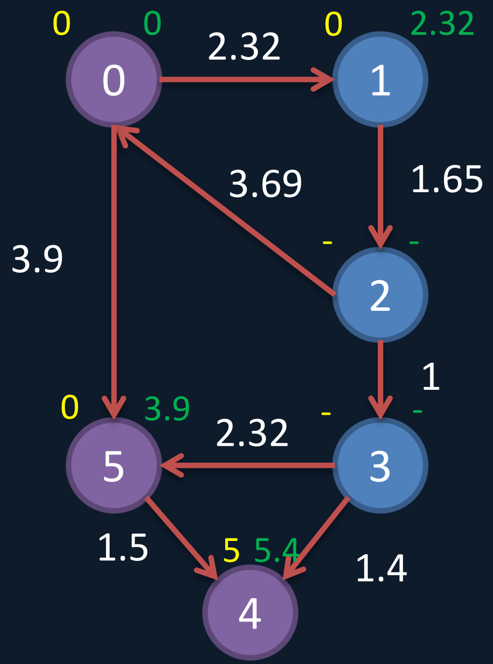
Priority Queue:

1
4

● = untraversed

● = traversed

A* Walkthrough



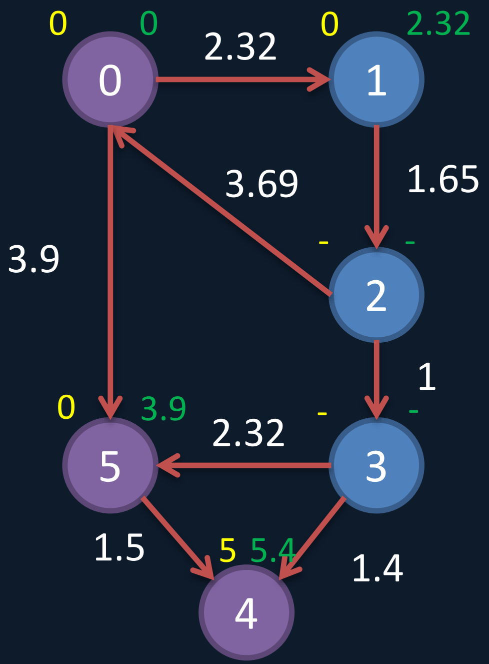
Priority Queue:

1

● = untraversed

● = traversed

A* Walkthrough



Priority Queue: 1

🟦 = untraversed

🟪 = traversed

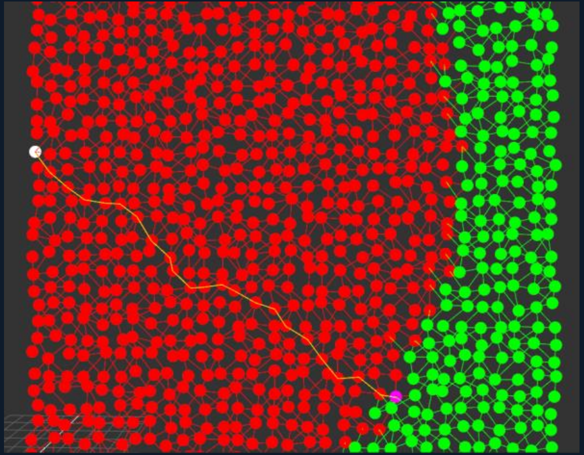
- Node 4 is the goal node so we exit.

A* Walkthrough

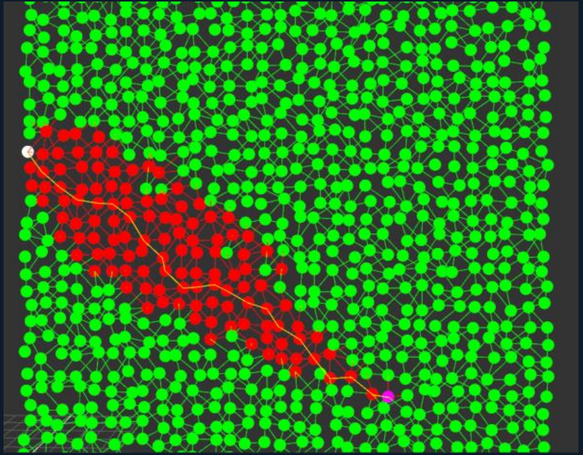
- In this example, note that the only nodes A* even looked at were the nodes for the final path.

- The biggest advantage of A* over Dijkstra's is that, given a good heuristic, it will run significantly faster.

The Advantage of A*



Dijkstra's



A*

Questions?