Programming Abstractions

CS106X

Cynthia Lee

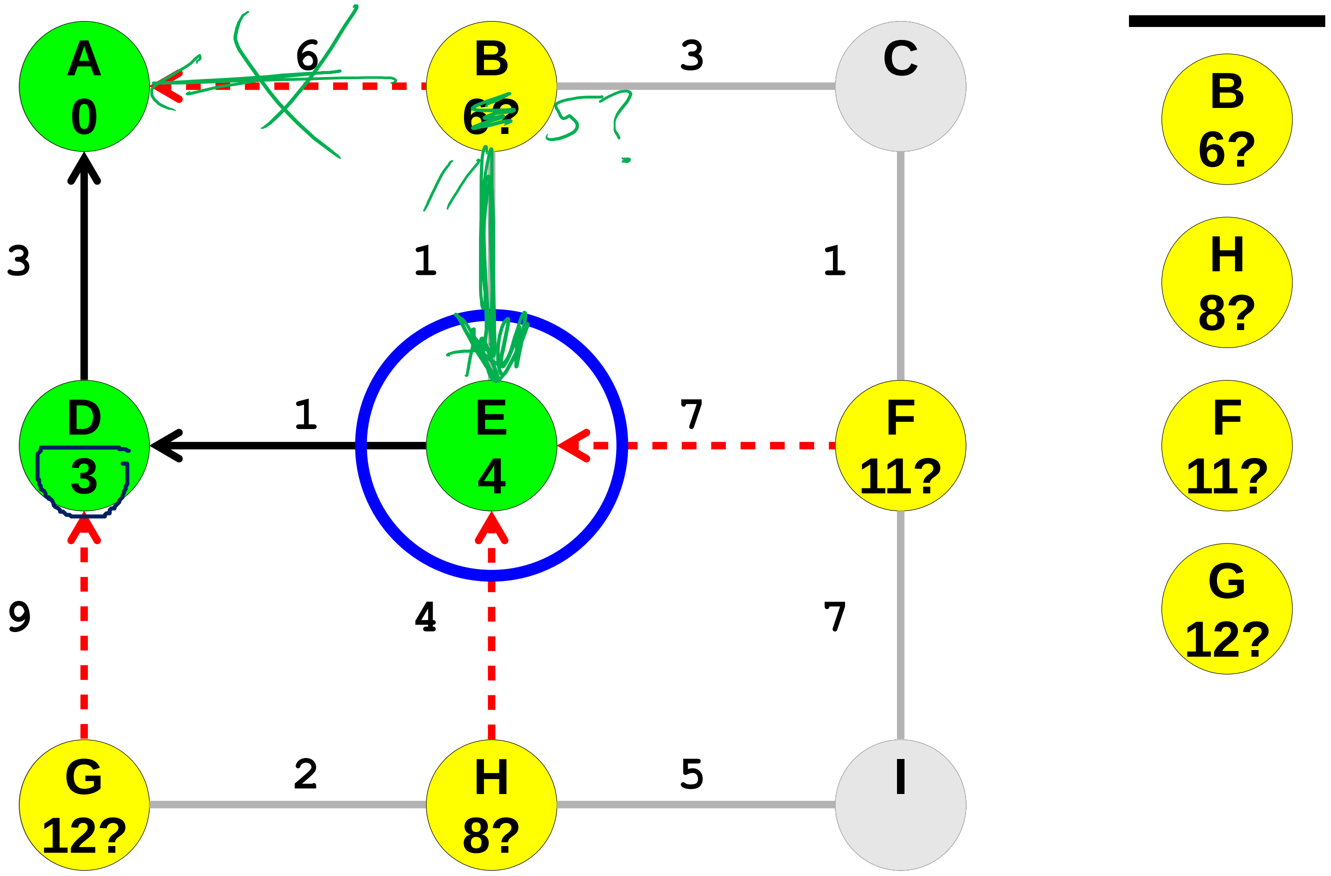
Graphs Topics

Graphs!

1. **Basics**
   - What are they? How do we represent them?
2. **Theorems**
   - What are some things we can prove about graphs?
3. **Breadth-first search on a graph**
   - Spoiler: just a very, very small change to tree version
4. **Dijkstra's shortest paths algorithm**
   - Spoiler: just a very, very small change to BFS
5. **A* shortest paths algorithm**
   - Spoiler: just a very, very small change to Dijkstra's
6. **Minimum Spanning Tree**
   - Kruskal's algorithm

Dijkstra's Algorithm

- Mark all nodes as gray.
- Mark the initial node ***s*** as yellow and at candidate distance **0**.
- Enqueue ***s*** into the priority queue with priority **0**.
- While not all nodes have been visited:
- Dequeue the lowest-cost node ***u*** from the priority queue.
- Color ***u*** green. The candidate distance ***d*** that is currently stored for node ***u*** is the length of the shortest path from ***s*** to ***u***.
- If ***u*** is the destination node ***t***, you have found the shortest path from ***s*** to ***t*** and are done.
- For each node ***v*** connected to ***u*** by an edge of length ***L***:
  - If ***v*** is gray:
    - Color ***v*** yellow.
    - Mark ***v***'s distance as ***d + L***.
    - Set ***v***'s parent to be ***u***.
    - Enqueue ***v*** into the priority queue with priority ***d + L***.
  - If ***v*** is yellow and the candidate distance to ***v*** is greater than ***d + L***:
    - Update ***v***'s candidate distance to be ***d + L***.
    - Update ***v***'s parent to be ***u***.
    - Update ***v***'s priority in the priority queue to ***d + L***.


A
0
6
B
6?
3
C
3
1
1
D
3
1
E
4
7
F
11?
9
4
7
G
12?
2
H
8?
5
I
B
6?
H
8?
F
11?
G
12?

Dijkstra's Algorithm

- Split nodes apart into three groups:

Green nodes, where we already have the shortest path;

Gray nodes, which we have never seen; and

Yellow nodes that saw just long enough to enqueue, but we still need to process.

- Dijkstra's algorithm works as follows:
- Mark all nodes gray except the start node, which is yellow and has cost 0.
- Until no yellow nodes remain:
  - Choose the yellow node with the lowest total cost.
  - Mark that node green.
  - Mark all its gray neighbors yellow and with the appropriate cost.
  - Update the costs of all adjacent yellow nodes by considering the path through the current node.

HOMEWORK: An Important Note

- The version of Dijkstra's algorithm I have just described is ***not*** the same as the version described in the course reader.
- This version is more complex than the book's version, but is faster.
- THIS IS THE VERSION YOU MUST USE ON YOUR TRAILBLAZER ASSIGNMENT!

How Dijkstra's Works

- Situation:
  - Dijkstra's algorithm works by incrementally computing the shortest path to intermediary nodes in the graph *in case* they prove to be useful.
- Problem:
  - No big-picture conception of how to get to the destination – the algorithm explores outward in all directions, “in case.”
- Implication:
  - Most of these explored nodes will end up being in completely the wrong direction.
- **Need:**
  - **Could we give the algorithm a “hint” of which direction to go?**

A* and Dijkstra's

Close cousins

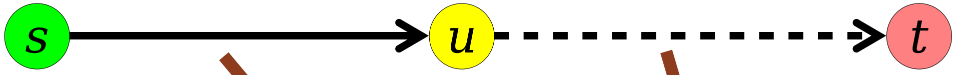
Heuristics

- In the context of graph searches, a **heuristic function** is a function that guesses the distance from some known node to the destination node.
- The guess doesn't have to be correct, but it should try to be as accurate as possible.
- Examples: For Google Maps, a heuristic for estimating distance might be the straight-line “as the crow flies” distance.

Admissible Heuristics

- A heuristic function is called an **admissible heuristic** if it never overestimates the distance from any node to the destination.
- In other words:
    - ***predicted-distance ≤ actual-distance***

Why Heuristics Matter

- We can modify Dijkstra's algorithm by introducing heuristic functions.
- Given any node *u*, there are two associated costs:
- 

- The actual distance from the start node *s*.
- The heuristic distance from u to the end node *t*.
- Key idea: Run Dijkstra's algorithm, but use the following priority in the priority queue:
    - ***priority(u)* = *distance(s, u)* + *heuristic(u, t)***
- This modification of Dijkstra's algorithm is called the **A* search algorithm**.

A* Search

- As long as the heuristic is admissible (and satisfies one other technical condition), A* will always find the shortest path from the source to the destination node.
- Can be *dramatically* faster than Dijkstra's algorithm.
- Focuses work in areas likely to be productive.
- Avoids solutions that appear worse *until* there is evidence they may be appropriate.

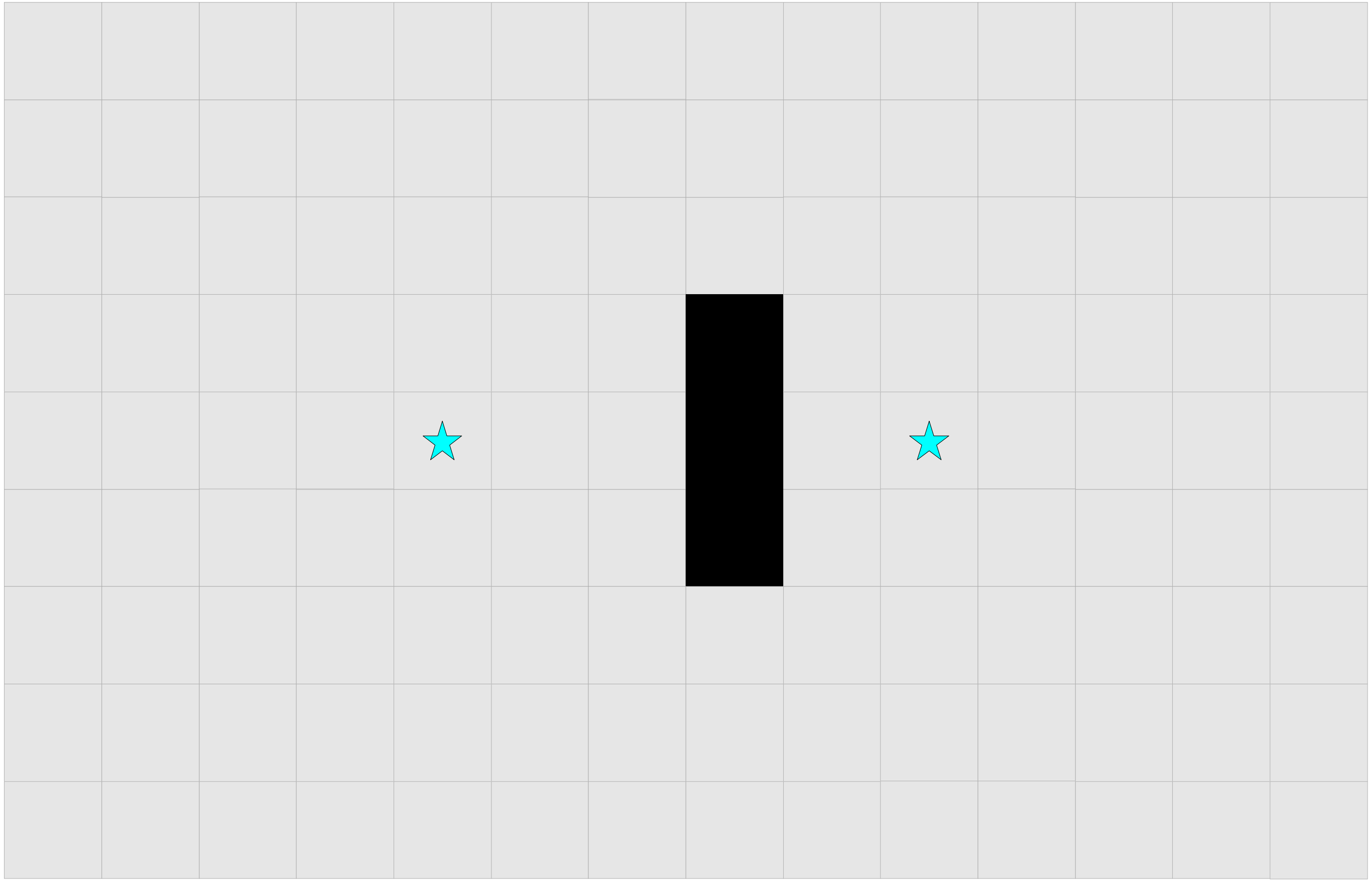
Dijkstra's Algorithm

- Mark all nodes as gray.
- Mark the initial node ***s*** as yellow and at candidate distance **0**.
- Enqueue ***s*** into the priority queue with priority **0**.
- While not all nodes have been visited:
- Dequeue the lowest-cost node ***u*** from the priority queue.
- Color ***u*** green. The candidate distance ***d*** that is currently stored for node ***u*** is the length of the shortest path from ***s*** to ***u***.
- If ***u*** is the destination node ***t***, you have found the shortest path from ***s*** to ***t*** and are done.
- For each node ***v*** connected to ***u*** by an edge of length ***L***:
  - If ***v*** is gray:
    - Color ***v*** yellow.
    - Mark ***v***'s distance as ***d* + *L***.
    - Set ***v***'s parent to be ***u***.
    - Enqueue ***v*** into the priority queue with priority ***d* + *L***.
  - If ***v*** is yellow and the candidate distance to ***v*** is greater than ***d* + *L***:
    - Update ***v***'s candidate distance to be ***d* + *L***.
    - Update ***v***'s parent to be ***u***.
    - Update ***v***'s priority in the priority queue to ***d* + *L***.

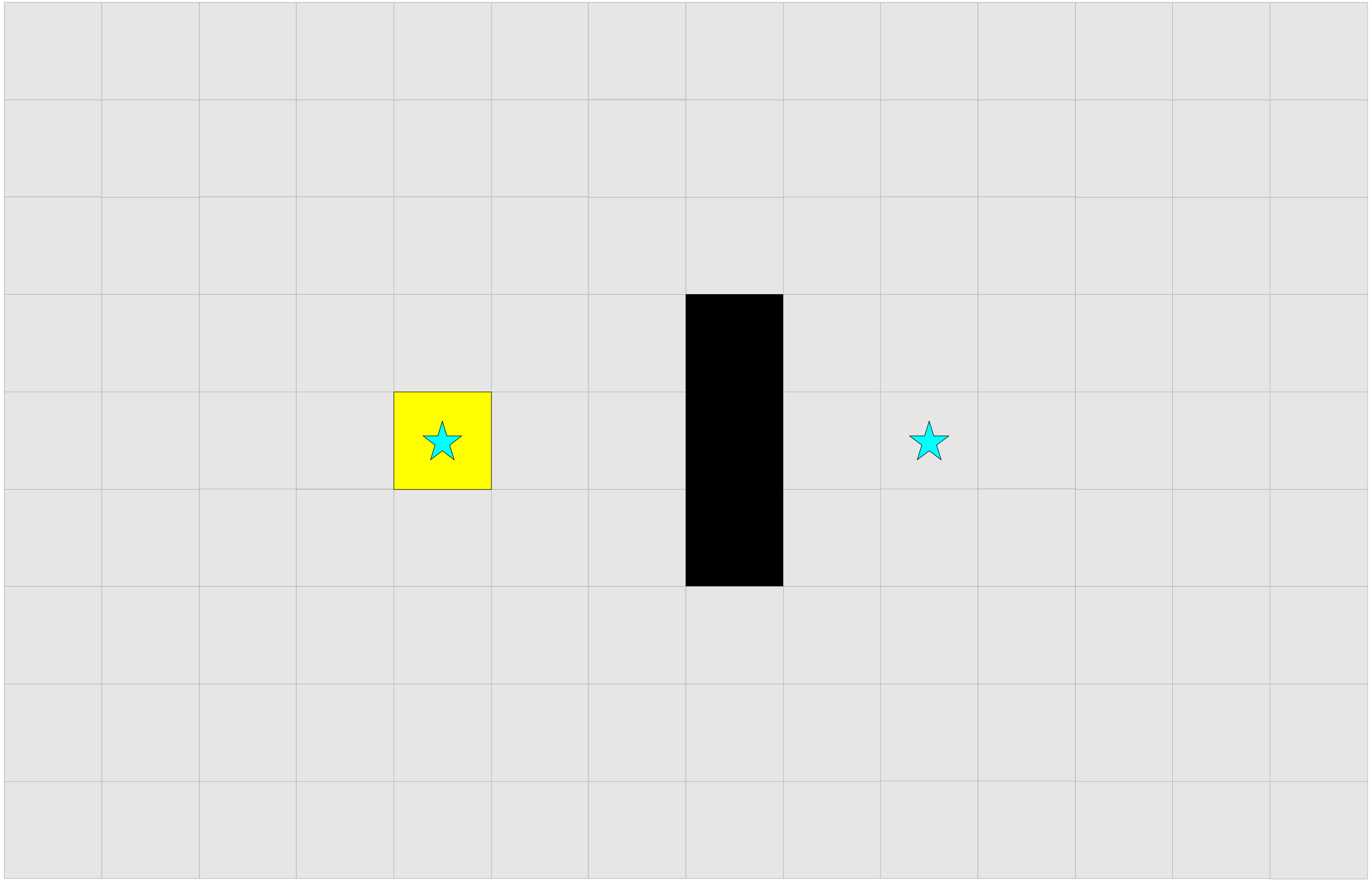
A* Search

- Mark all nodes as gray.
- Mark the initial node *s* as yellow and at candidate distance **0**.
- Enqueue *s* into the priority queue with priority **h(s,t)**.
- While not all nodes have been visited:
- Dequeue the lowest-cost node *u* from the priority queue.
- Color *u* green. The candidate distance *d* that is currently stored for node *u* is the length of the shortest path from *s* to *u*.
- If *u* is the destination node *t*, you have found the shortest path from *s* to *t* and are done.
- For each node *v* connected to *u* by an edge of length *L*:
  - If *v* is gray:
    - Color *v* yellow.
    - Mark *v*'s distance as *d* + *L*.
    - Set *v*'s parent to be *u*.
    - Enqueue *v* into the priority queue with priority *d* + *L* + h(v,t).
  - If *v* is yellow and the candidate distance to *v* is greater than *d* + *L*:
    - Update *v*'s candidate distance to be *d* + *L*.
    - Update *v*'s parent to be *u*.
    - Update *v*'s priority in the priority queue to *d* + *L* + h(v,t).

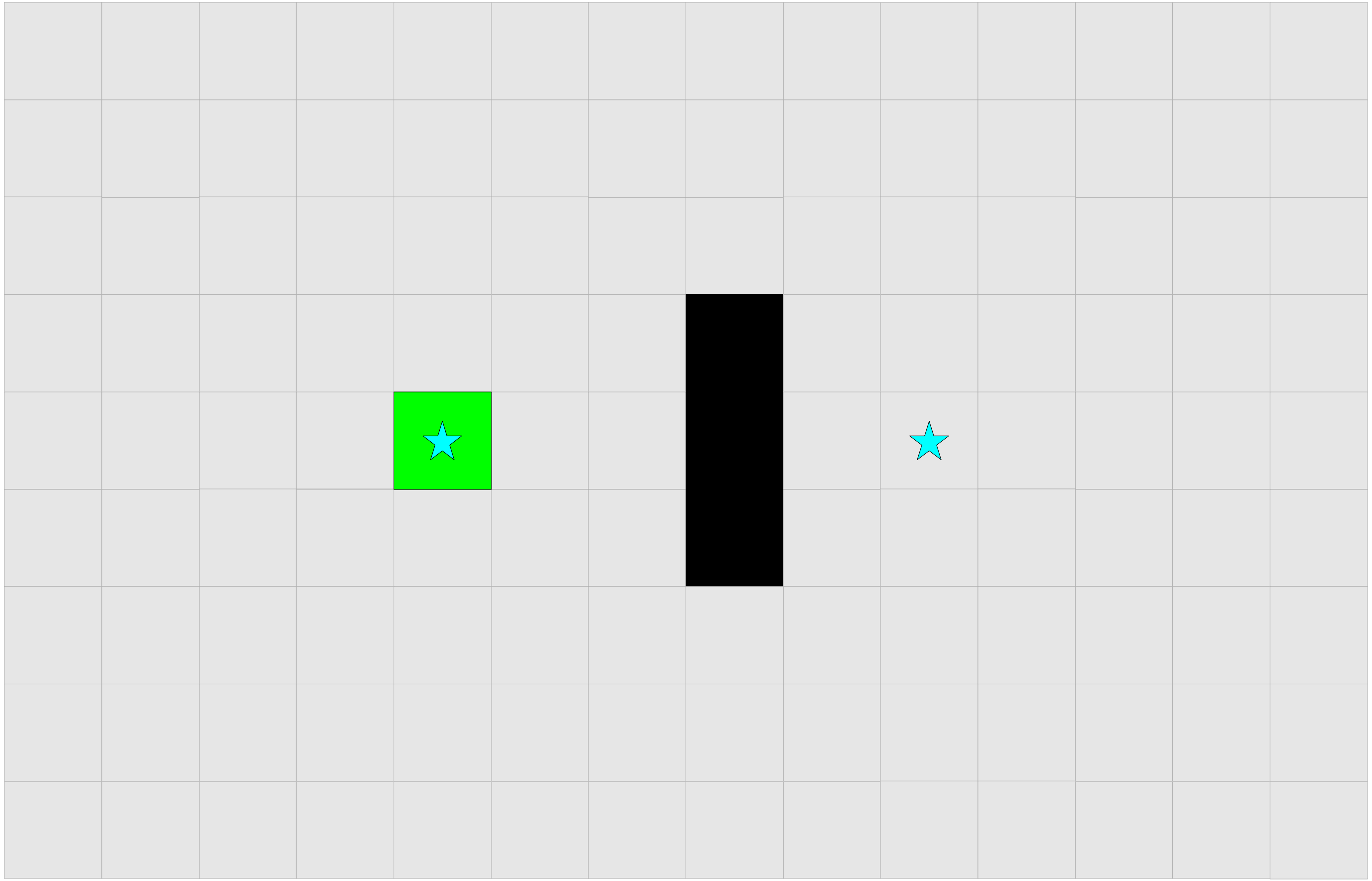
A* on two points where the heuristic is slightly misleading due to a wall blocking the way



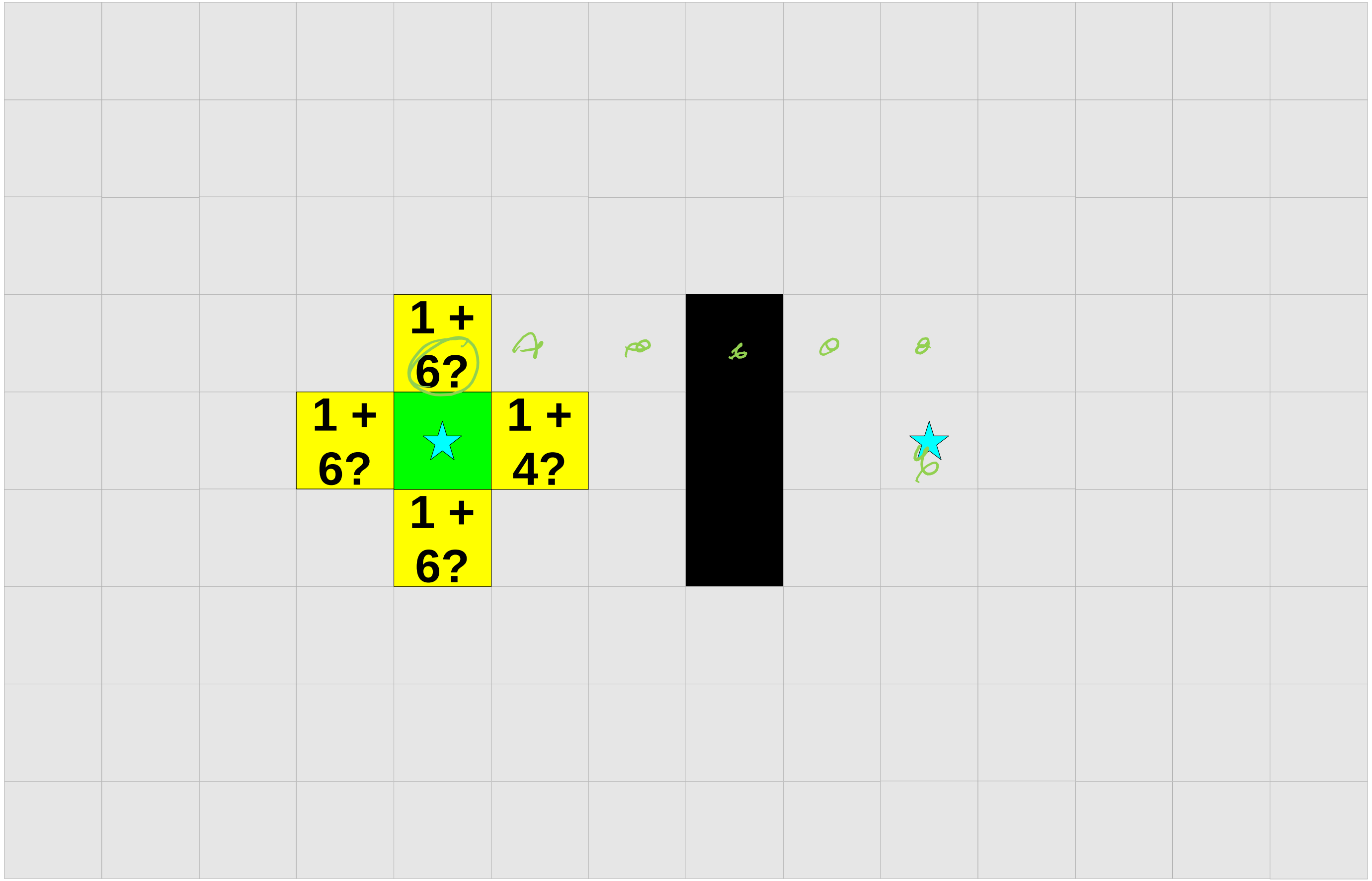
A* starts with start node yellow, other nodes grey.



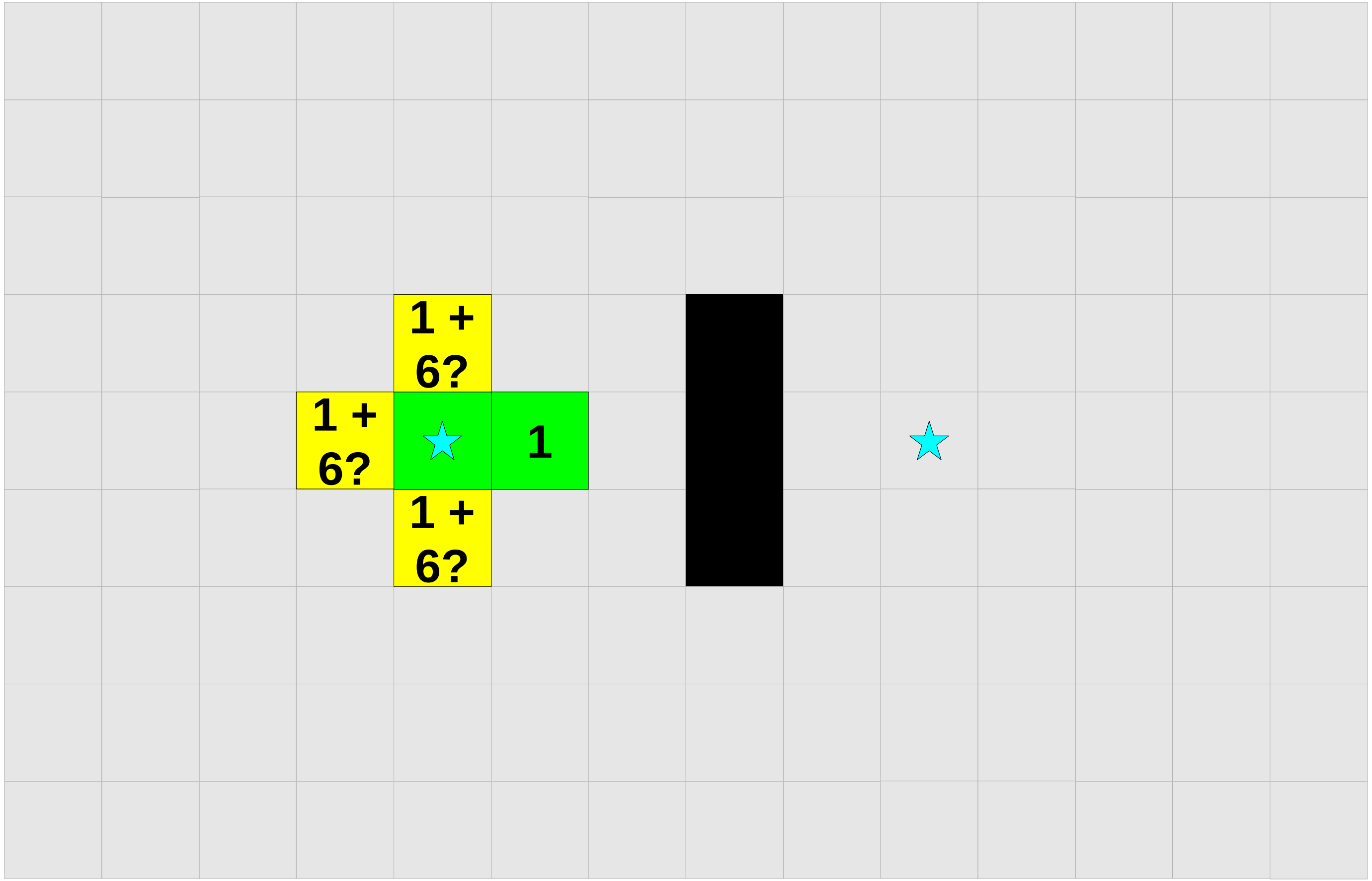
A*: dequeue start node, turns green.

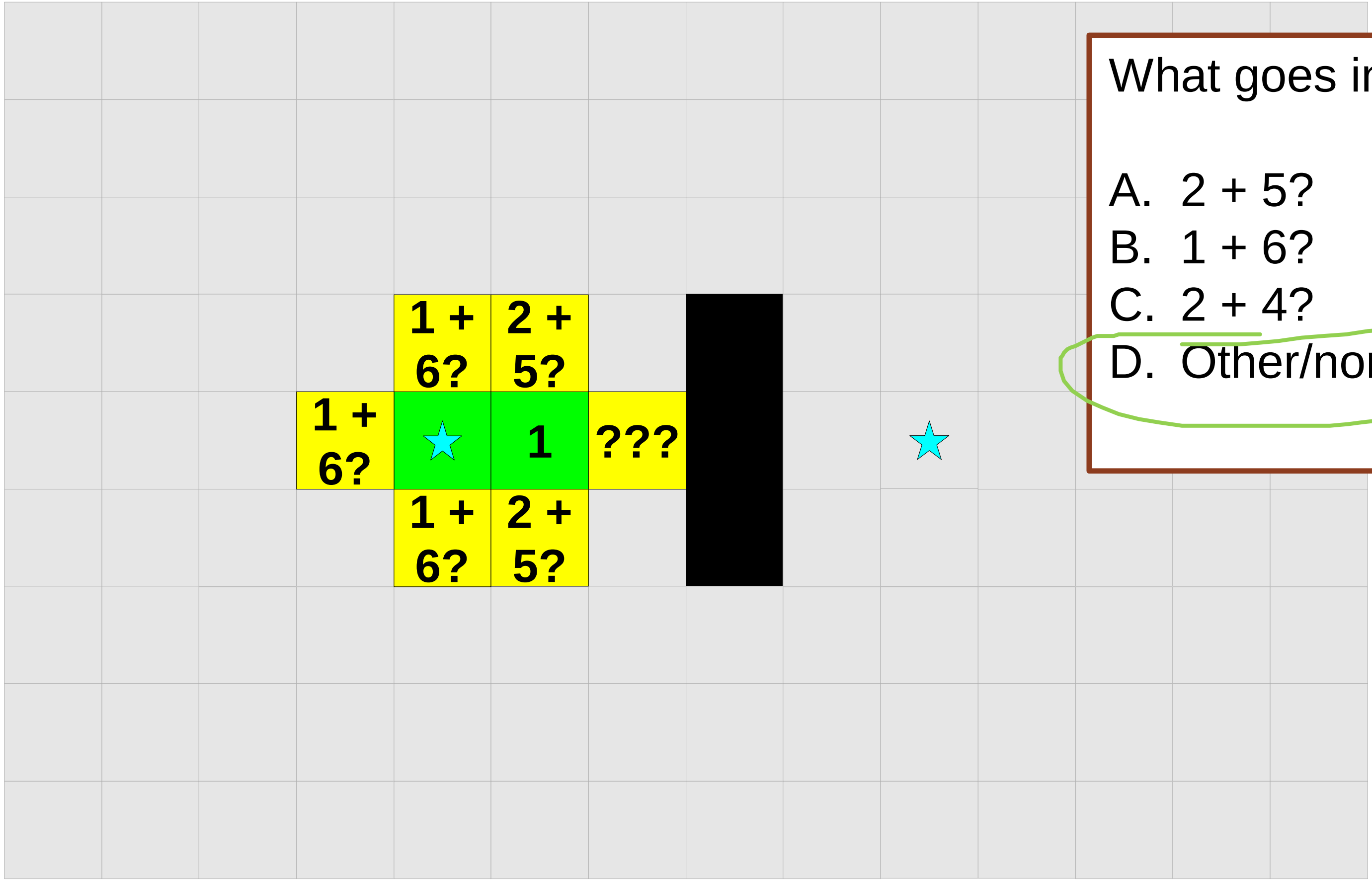


A*: enqueue neighbors with candidate distance + heuristic distance as the priority value.



A*: dequeue min-priority-value node.

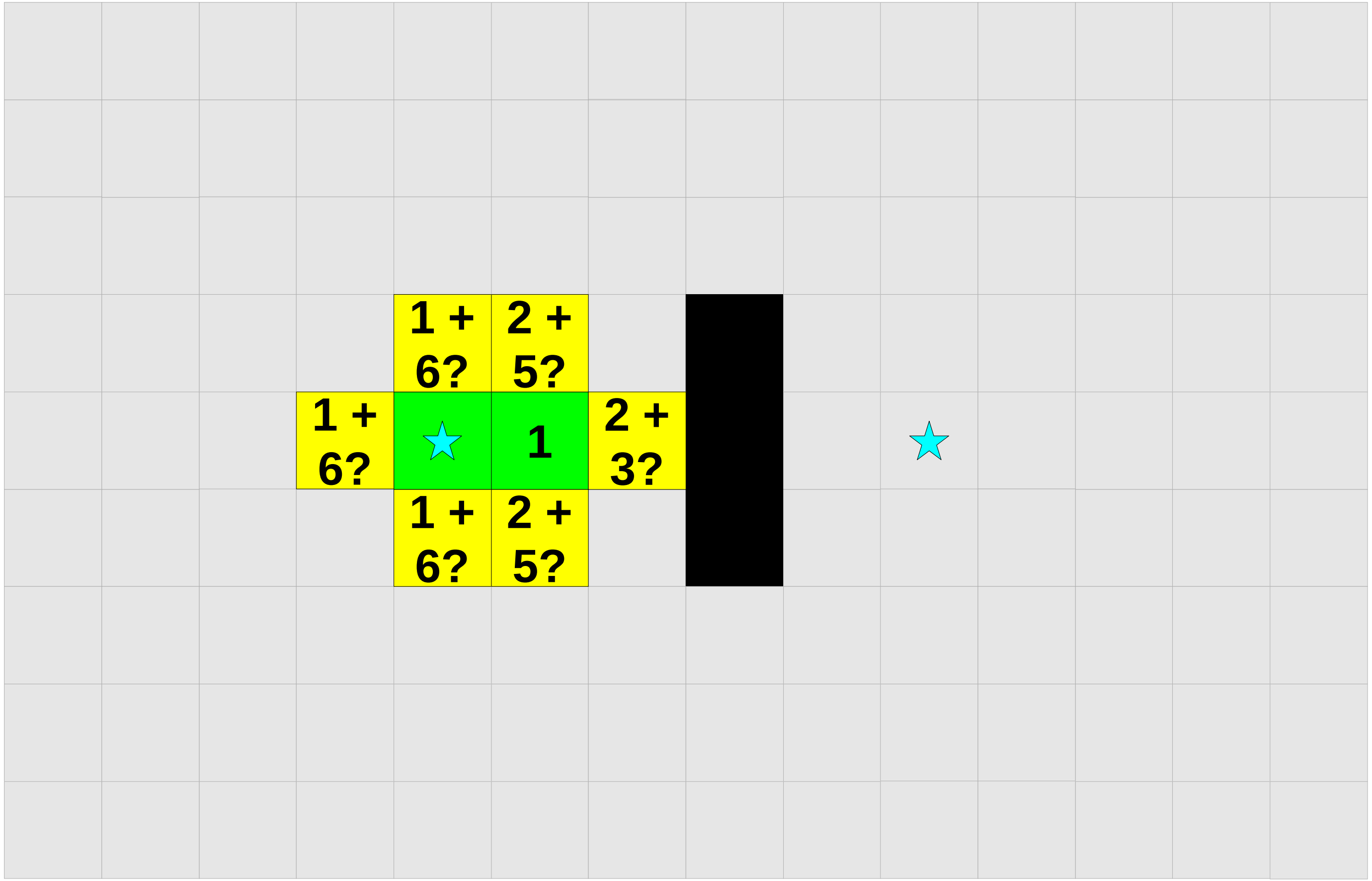


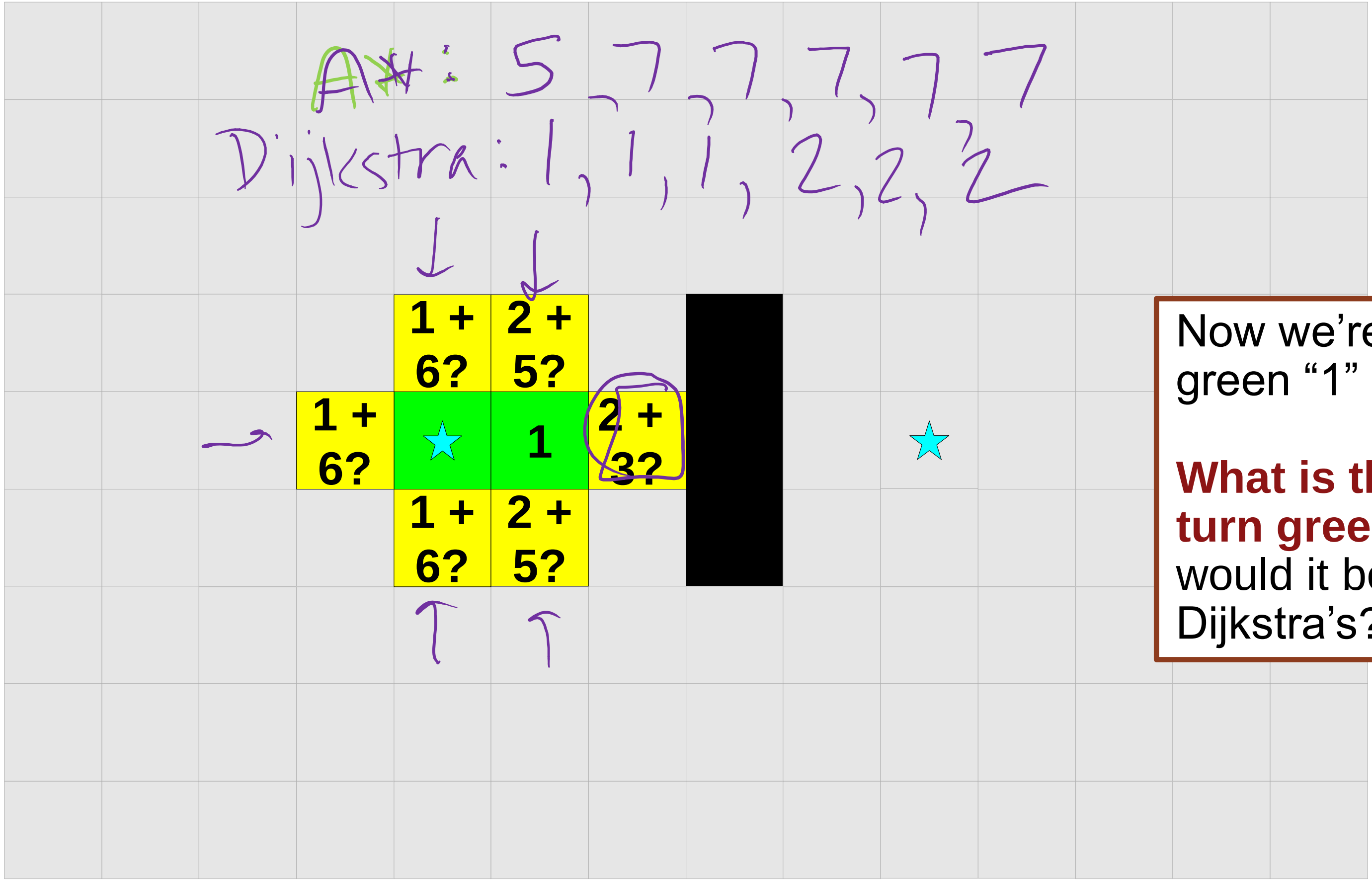


What goes in the ??? ?

A. 2 + 5?
B. 1 + 6?
C. 2 + 4?
D. Other/none/more

A*: enqueue neighbors.

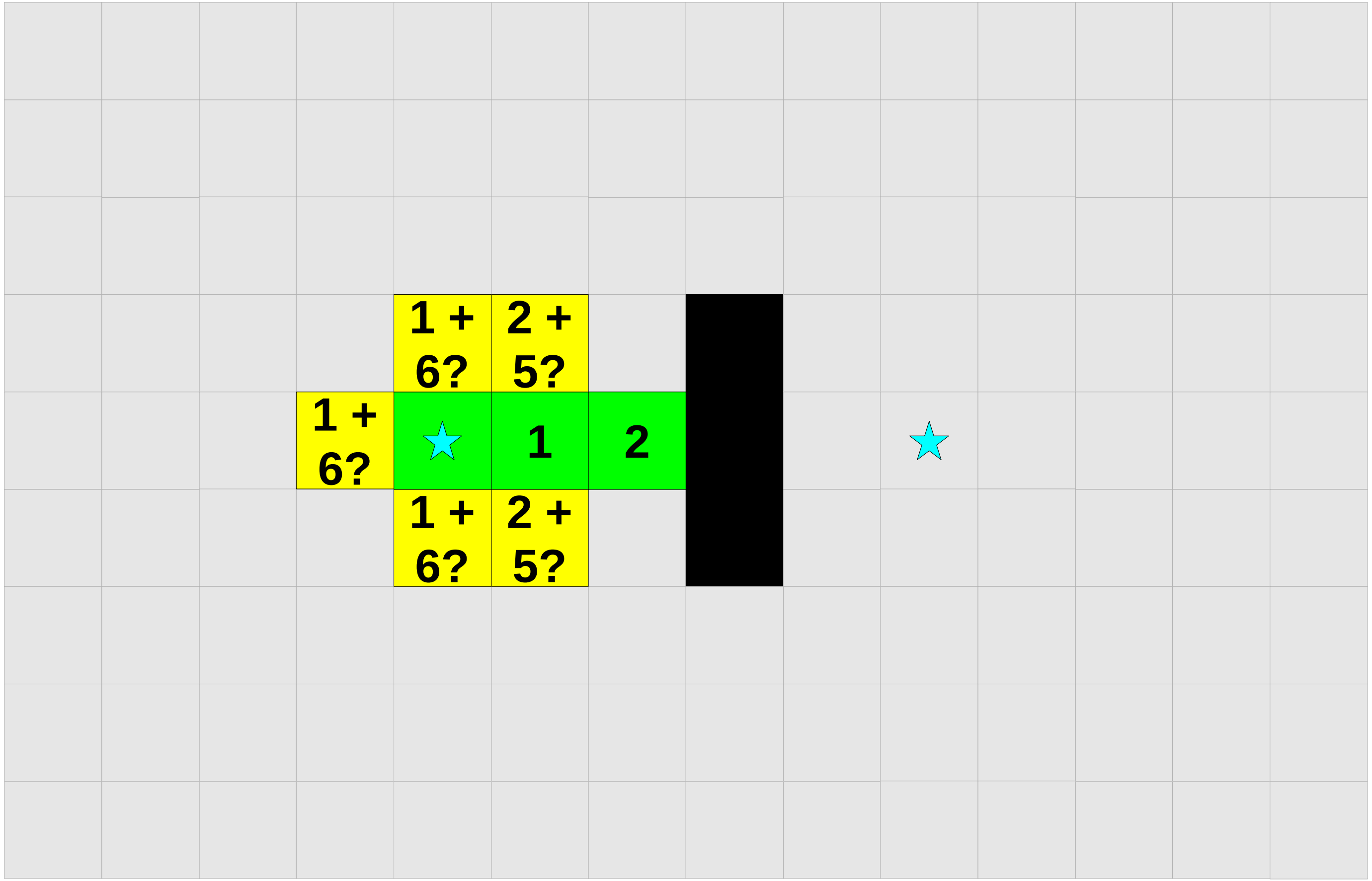




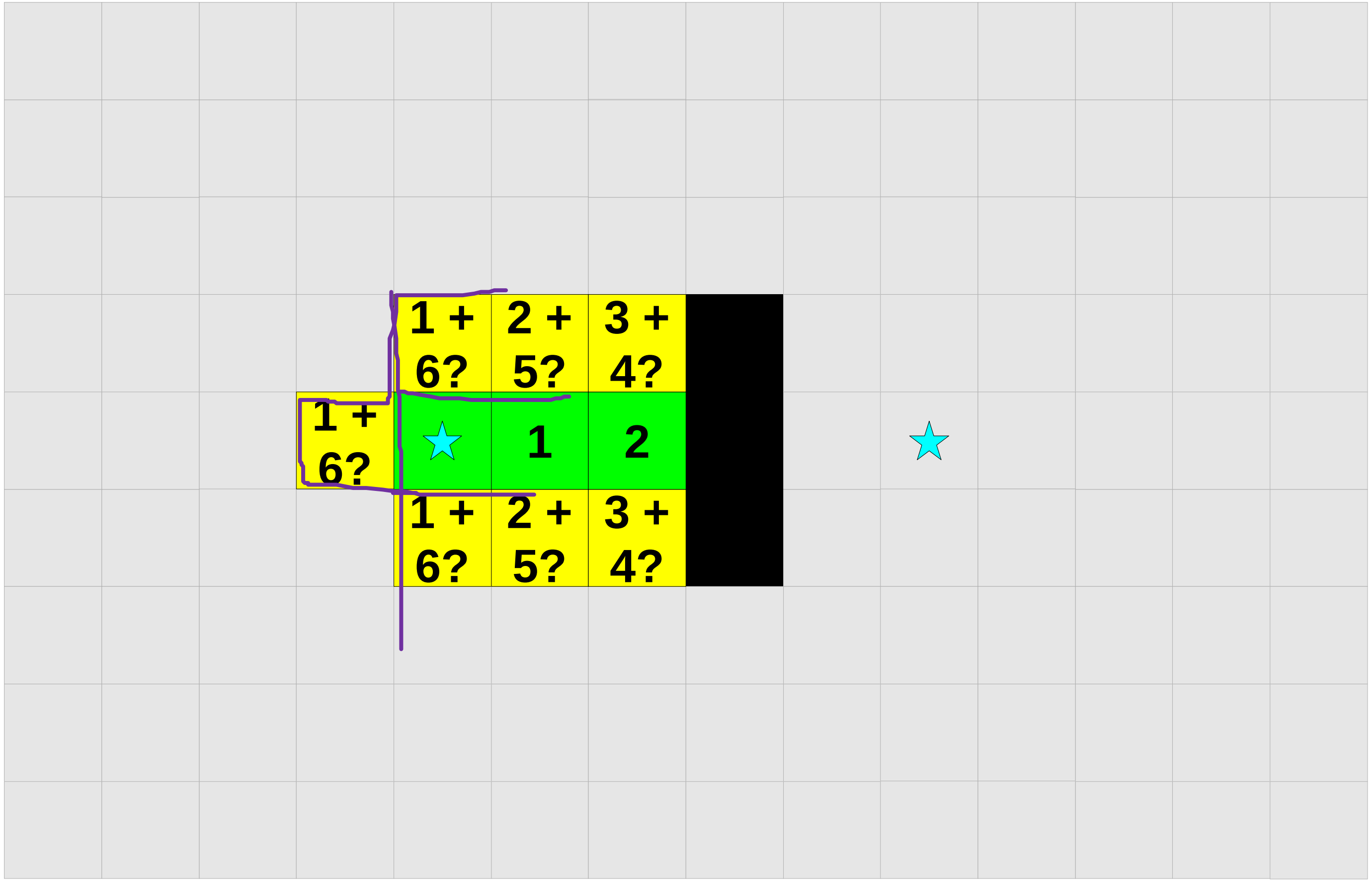
Now we're done with the green "1" node's turn.

What is the next node to turn green? (and what would it be if this were Dijkstra's?)

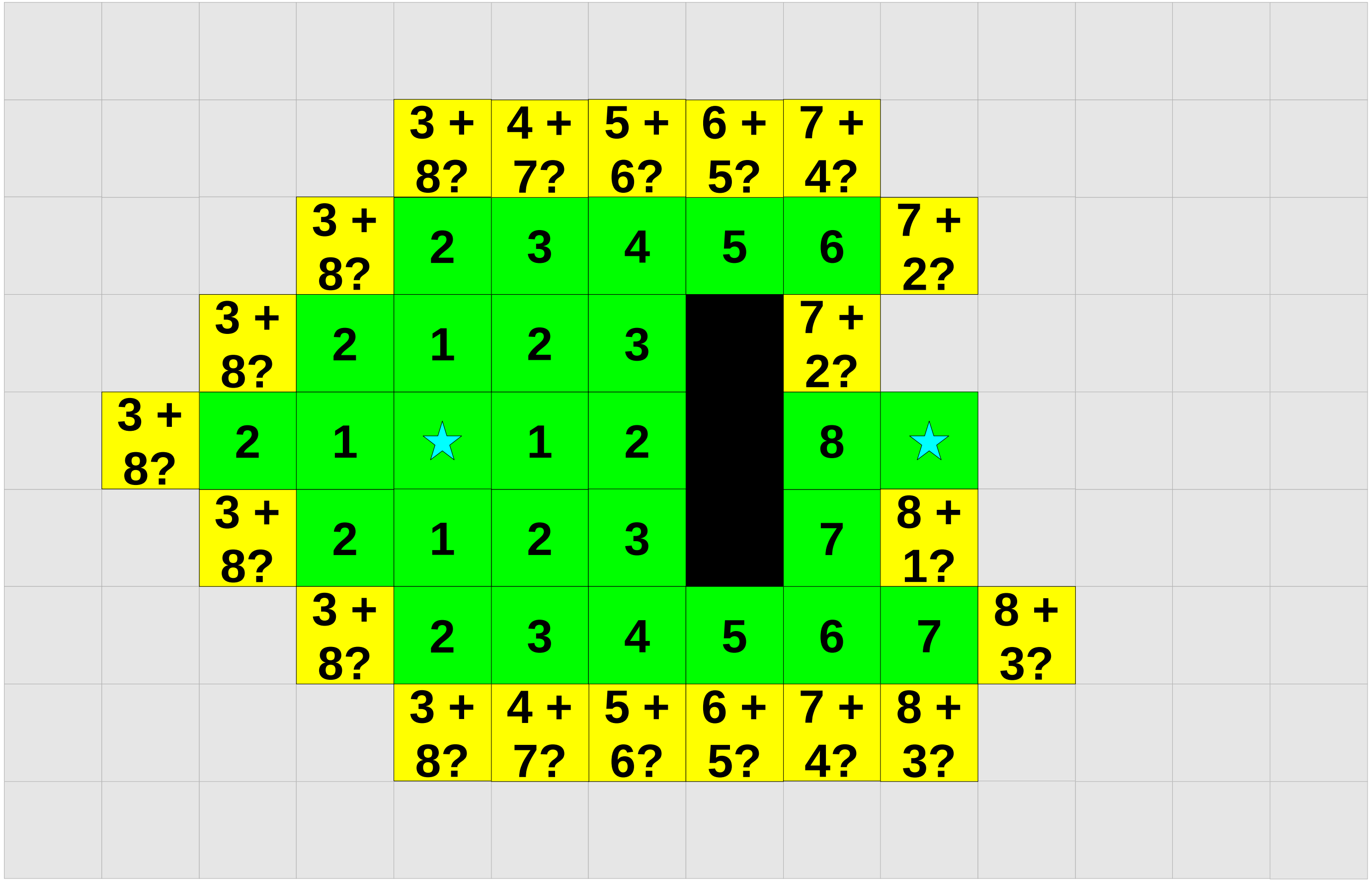
A*: dequeue next lowest priority value node. Notice we are making a straight line right for the end point, not wasting time with other directions.



A*: enqueue neighbors—uh-oh, wall blocks us from continuing forward.



A*: eventually figures out how to go around the wall, with some waste in each direction.



For Comparison: What Dijkstra's Algorithm Would Have Searched

8	7	6	5	4	5	6	7	8	9?				
7	6	5	4	3	4	5	6	7	8	9?			
6	5	4	3	2	3	4	5	6	7	8	9?		
5	4	3	2	1	2	3		7	8	9?			
4	3	2	1	★	1	2		8	★				
5	4	3	2	1	2	3		7	8	9?			
6	5	4	3	2	3	4	5	6	7	8	9?		
7	6	5	4	3	4	5	6	7	8	9?			
8	7	6	5	4	5	6	7	8	9?				

Dijkstra's Algorithm

- Mark all nodes as gray.
- Mark the initial node ***s*** as yellow and at candidate distance **0**.
- Enqueue ***s*** into the priority queue with priority **0**.
- While not all nodes have been visited:
- Dequeue the lowest-cost node ***u*** from the priority queue.
- Color ***u*** green. The candidate distance ***d*** that is currently stored for node ***u*** is the length of the shortest path from ***s*** to ***u***.
- If ***u*** is the destination node ***t***, you have found the shortest path from ***s*** to ***t*** and are done.
- For each node ***v*** connected to ***u*** by an edge of length ***L***:
  - If ***v*** is gray:
    - Color ***v*** yellow.
    - Mark ***v***'s distance as ***d* + *L***.
    - Set ***v***'s parent to be ***u***.
    - Enqueue ***v*** into the priority queue with priority ***d* + *L***.
  - If ***v*** is yellow and the candidate distance to ***v*** is greater than ***d* + *L***:
    - Update ***v***'s candidate distance to be ***d* + *L***.
    - Update ***v***'s parent to be ***u***.
    - Update ***v***'s priority in the priority queue to ***d* + *L***.

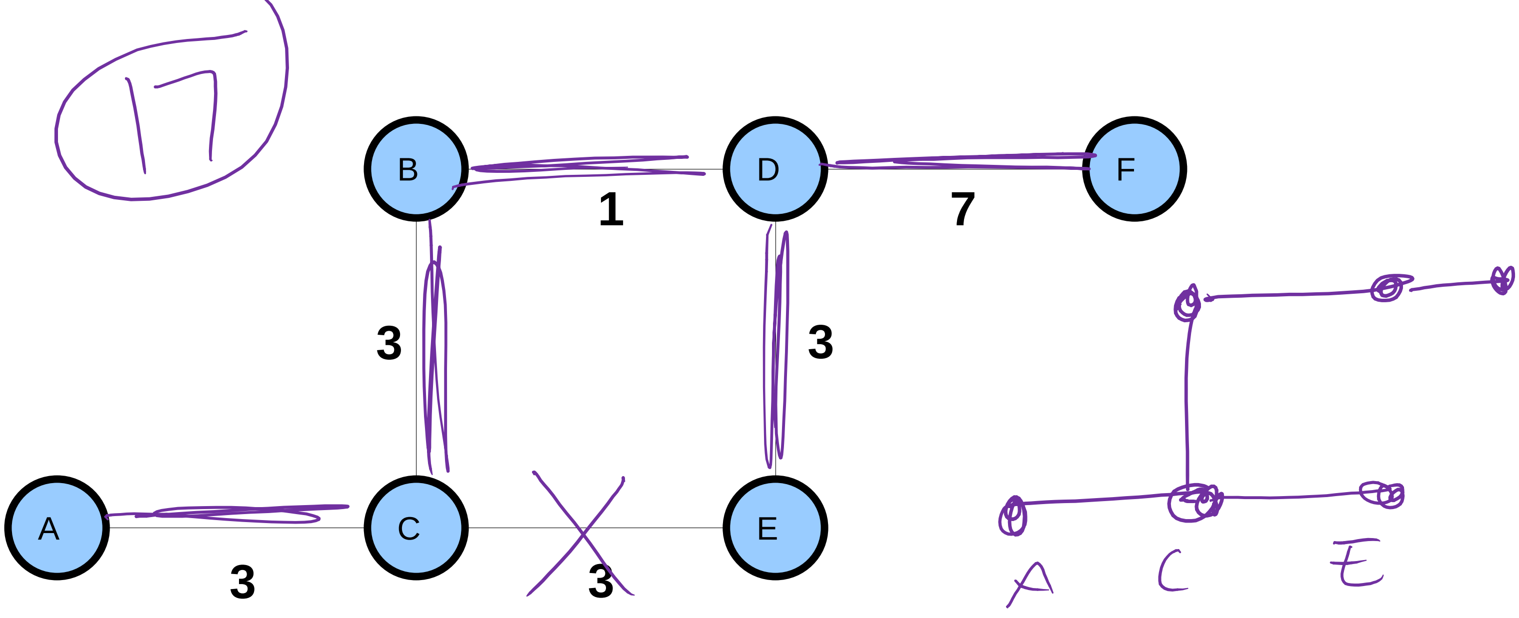
A* Search

- Mark all nodes as gray.
- Mark the initial node ***s*** as yellow and at candidate distance **0**.
- Enqueue ***s*** into the priority queue with priority **h(s,t)**.
- While not all nodes have been visited:
- Dequeue the lowest-cost node ***u*** from the priority queue.
- Color ***u*** green. The candidate distance ***d*** that is currently stored for node ***u*** is the length of the shortest path from ***s*** to ***u***.
- If ***u*** is the destination node ***t***, you have found the shortest path from ***s*** to ***t*** and are done.
- For each node ***v*** connected to ***u*** by an edge of length ***L***:
  - If ***v*** is gray:
    - Color ***v*** yellow.
    - Mark ***v***'s distance as ***d* + *L***.
    - Set ***v***'s parent to be ***u***.
    - Enqueue ***v*** into the priority queue with priority ***d* + *L* + h(v,t)**.
  - If ***v*** is yellow and the candidate distance to ***v*** is greater than ***d* + *L***:
    - Update ***v***'s candidate distance to be ***d* + *L***.
    - Update ***v***'s parent to be ***u***.
    - Update ***v***'s priority in the priority queue to ***d* + *L* + h(v,t)**.

Minimum Spanning Tree

A **spanning tree** in an undirected graph is a set of edges with no cycles that connects all nodes.

A **minimum spanning tree** (or **MST**) is a spanning tree with the least total cost.



How many distinct minimum spanning trees are in this graph?

A. 0-1
B. 2-3
C. 4-5
D. 6-7
E. >7

Kruskal's algorithm

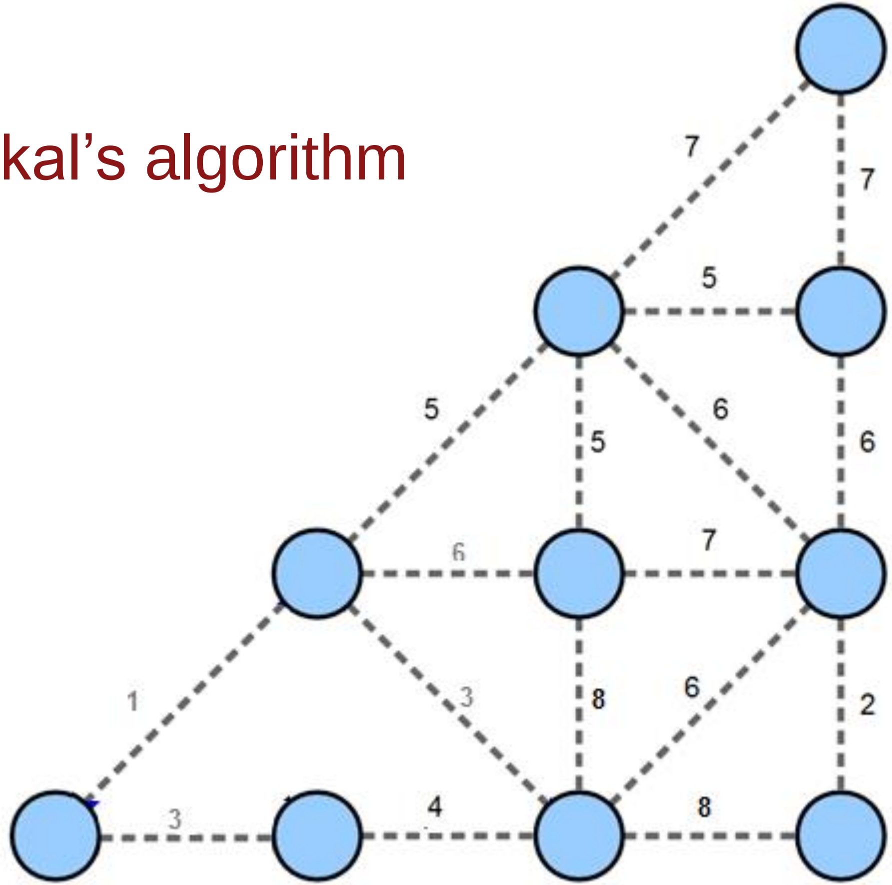
Remove all edges from graph

Place all edges in a PQ based on length/weight

While !PQ.isEmpty():

- Dequeue edge
- If the edge connects previous disconnected nodes or groups of nodes, keep the edge
- Otherwise discard the edge

Kruskal’s algorithm



The Good Will Hunting Problem

Video Clip

https://www.youtube.com/watch?v=N7b0cLn-wHU

“Draw all the homeomorphically irreducible trees with n=10.”

“Draw all the homeomorphically irreducible trees with n=10.”

In this case **“trees”** simply means **graphs with no cycles**

“with n = 10” (i.e., has **10 nodes**)

“homeomorphically irreducible”

- **No nodes of degree 2 allowed in your solutions**
  - For this problem, nodes of degree 2 are useless in terms of tree structure—they just act as a blip on an edge—and are therefore banned
- Have to be actually different
  - Ignore superficial changes in rotation or angles of drawing