# Shortest Paths and MSTs

## Exam Prep 09



# Announcements

| Sunday | Monday | Tuesday | Wednesday | Thursday | Friday | Saturday |
|---|---|---|---|---|---|---|
| | 3/18<br>Homework 3 Due |  | 3/20<br>Guest Lecture!!! | 3/21<br>**Midterm 2** | | |
| | | | | | | |



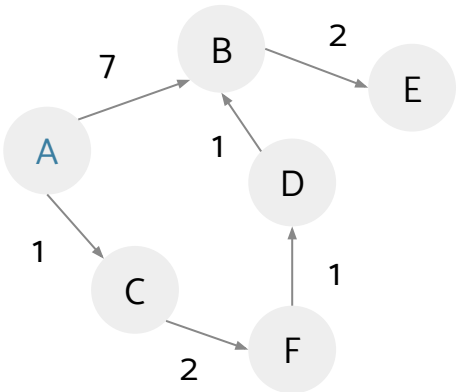
# Content Review



# Dijkstra's Algorithm

We've learned that BFS can help us find paths from the start to other nodes with a minimum number of edges. However, neither BFS or DFS account for finding shortest paths based off edge weight.

**Dijkstra's algorithm** is a method of finding the shortest path from one node to every other node in the graph. You use a priority queue that sorts vertices based off of their distance to the root node.

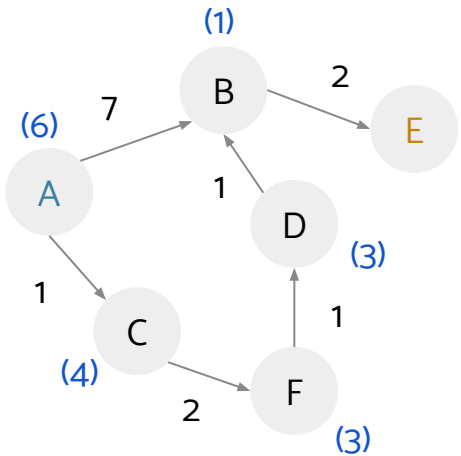



Steps:

1. Pop node from the front of the queue - this is the current node.
2. Add/update distances of all of the neighbors of the current node in the PQ.
3. Re-sort the priority queue (technically the PQ does this itself).
4. Finalize the distance to the current node from the root.
5. Repeat while the PQ is not empty.

# A*

A* is a method of finding the shortest path from one node to a specific other node in the graph. It operates similarly to Dijkstra's except for that we use a (given) heuristic to estimate a vertex's distance from the goal.



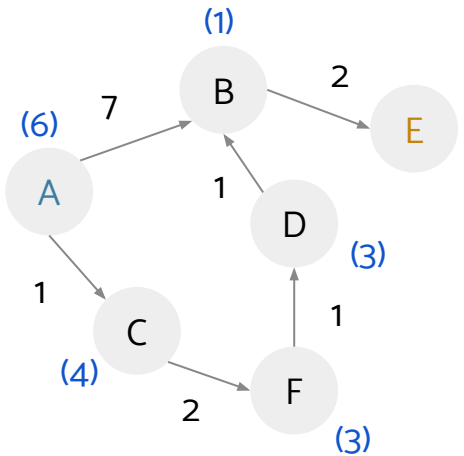

We're guaranteed to get the shortest path if our heuristic is admissible (never overestimates the true distance to the goal) and consistent (estimate always <= the estimated distance from any neighboring vertex to the goal + the cost of reaching that neighbor).

# A*

A* is a method of finding the shortest path from one node to a specific other node in the graph. It operates similarly to Dijkstra's except for that we use a (given) heuristic to estimate a vertex's distance from the goal.

Steps:

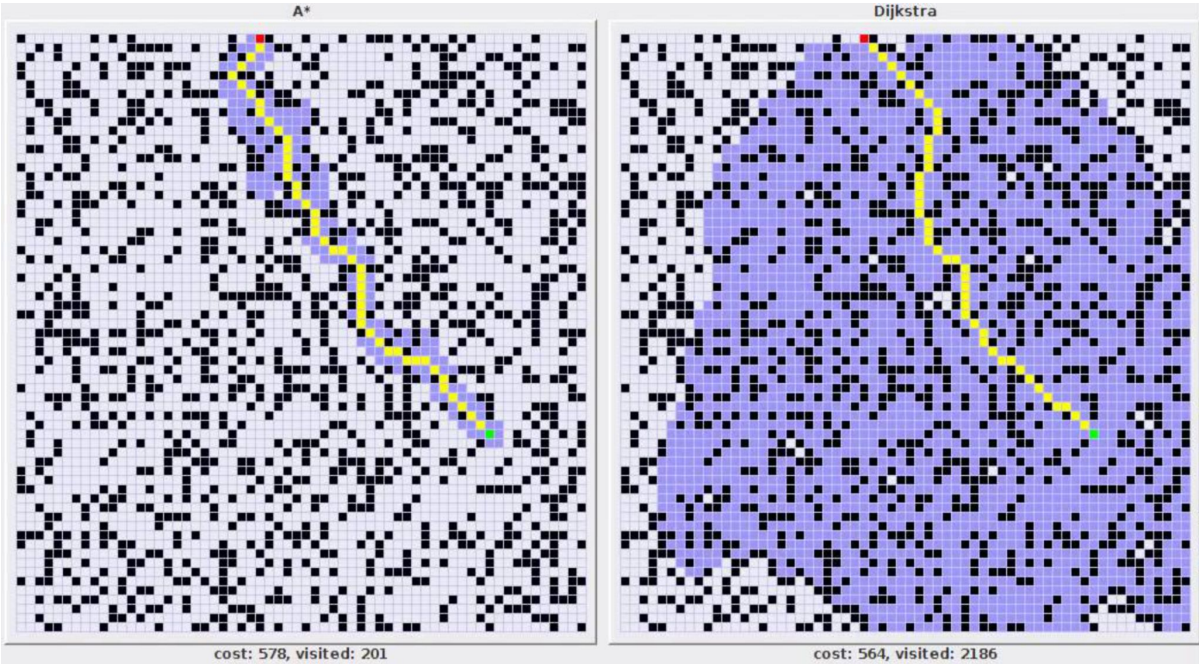
1. Pop node from the top of the queue - this is the current node.
2. Add/update distances of all of the children of the current node. This distance will be the sum of the distance up to that child node and our guess of how far away the goal node is (our heuristic).
3. Re-sort the priority queue.
4. Check if we've hit the goal node (if so we stop).
5. Repeat while the PQ is not empty.




Note: the heuristic may not always be very good and might lead us down a path that isn't the shortest!

# A*

Stolen from Austin



## Dijkstra's Pseudocode

```
PQ = new PriorityQueue()
PQ.add(A, 0)
PQ.add(v, infinity) # (all nodes except A).

distTo = {} # map
edgeTo = {} # map
distTo[A] = 0
distTo[v] = infinity # (all nodes except A).

while (not PQ.isEmpty()):
  poppedNode, poppedPriority = PQ.pop()

  for child in poppedNode.children:
    if PQ.contains(child):
      potentialDist = distTo[poppedNode] +
        edgeWeight(poppedNode, child)
      if potentialDist < distTo[child]:
        distTo.put(child, potentialDist)
        PQ.changePriority(child, potentialDist)
        edgeTo[child] = poppedNode
```

## A* Pseudocode

Stolen from Austin

```
PQ = new PriorityQueue()
PQ.add(A, h(A))
PQ.add(v, infinity) # (all nodes except A).

distTo = {} # map
edgeTo = {} # map
distTo[A] = 0
distTo[v] = infinity # (all nodes except A).

while (not PQ.isEmpty()):
  poppedNode, poppedPriority = PQ.pop()
  if (poppedNode == goal): terminate

    for child in poppedNode.children:
      if PQ.contains(child):
        potentialDist = distTo[poppedNode] +
          edgeWeight(poppedNode, child)

        if potentialDist < distTo[child]:
          distTo.put(child, potentialDist)
          PQ.changePriority(child, potentialDist + h(child))
          edgeTo[child] = poppedNode
```

## Dijkstra's Pseudocode

```
1  PQ = new PriorityQueue()
2  PQ.add(A, 0)
3  PQ.add(v, infinity) # (all nodes except A).
4
5  distTo = {} # map
6  edgeTo = {} # map
7  distTo[A] = 0
8  distTo[v] = infinity # (all nodes except A).
9
10 while (not PQ.isEmpty()):
11   poppedNode, poppedPriority = PQ.pop()
12
13   for child in poppedNode.children:
14     if PQ.contains(child):
15       potentialDist = distTo[poppedNode] +
16         edgeWeight(poppedNode, child)
17       if potentialDist < distTo[child]:
18         distTo.put(child, potentialDist)
19         PQ.changePriority(child, potentialDist)
20         edgeTo[child] = poppedNode
```

## A* Pseudocode

Stolen from Austin

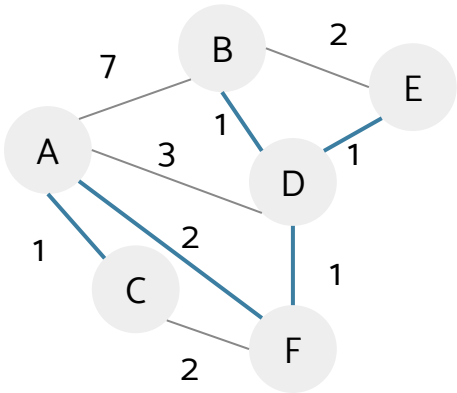
```
1  PQ = new PriorityQueue()
2  PQ.add(A, h(A))
3  PQ.add(v, infinity) # (all nodes except A).
4
5  distTo = {} # map
6  edgeTo = {} # map
7  distTo[A] = 0
8  distTo[v] = infinity # (all nodes except A).
9
9  while (not PQ.isEmpty()):
10   poppedNode, poppedPriority = PQ.pop()
11   if (poppedNode == goal): terminate
12
13     for child in poppedNode.children:
14       if PQ.contains(child):
15         potentialDist = distTo[poppedNode] +
16           edgeWeight(poppedNode, child)
17
18         if potentialDist < distTo[child]:
19           distTo.put(child, potentialDist)
20           PQ.changePriority(child, potentialDist + h(child))
21           edgeTo[child] = poppedNode
```

# Minimum Spanning Trees

**Minimum Spanning Trees** are set of edges that connect all the nodes in a graph while being of the smallest possible weight.

MSTs may not be unique if there are multiple edges of the same weight.

There are two main algorithms for finding MSTs in this class: Prim's and Kruskal's. Both are based on the **cut property**: if we "cut" across any edges and separate the graph into two groups, the minimum weight edge that falls along that cut will be in some MST.



# Worksheet

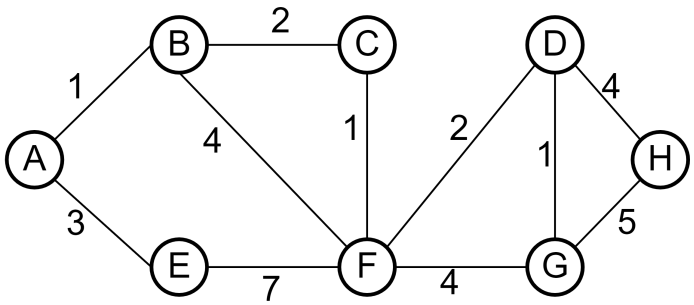


# CS 61B Spring 2024

# Shortest Paths, MSTs

Exam-Level 09: March 18, 2024

## 1 Dijkstra's, A* - Mechanical Problem, Sliders are best




(a) Run Dijkstra's Algorithm on the graph above starting from vertex $A$, breaking ties alphabetically. Fill in how the priority values change below. When you remove a node from the fringe, mark it with a check, and leave it blank for the subsequent rows. Stop when you remove G. Also sketch the resulting shortest paths tree in the end.

| Node | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| Iter 1 | ✓ | 1 | | | 3 | | | |
| Iter 2 | | ✓ | 3 | | 3 | 5 | | |
| Iter 3 | | | ✓ | | 3 | 4 | | |
| Iter 4 | | | | | ✓ | 4 | | |
| Iter 5 | | | | 6 | | ✓ | 8 | |
| Iter 6 | | | | ✓ | | | 7 | 10 |
| Iter 7 | | | | | | | ✓ | |

(b) The heuristic distance from all nodes to G is defined below. Run A*, starting from $A$ and with $G$ as a goal. For each entry in the table below, fill in the distance, followed by the priority value of each node, separated by a comma. Is the heuristic admissible?

| $u$ | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| $h(u, G)$ | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

| Node | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0, 9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| Iter 1 | ✓ | 1,8 | | | 3,13 | | | |
| Iter 2 | | ✓ | 3,7 | | | 5,8 | | |
| Iter 3 | | | ✓ | | | 4,7 | | |
| Iter 4 | | | | 6,7 | | ✓ | 8,8 | |
| Iter 5 | | | | ✓ | | | 7,7 | 10,15 |
| Iter 6 | | | | | | | ✓ | |

# 2 Conceptual Shortest Paths

Answer the following questions regarding shortest path algorithms for a **weighted, undirected graph**. If the statement is true, provide an explanation. If the statement is false, provide a counterexample.

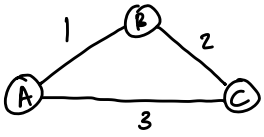
(a) (T/F) If all edge weights are equal and positive, the breadth-first search starting from node A will return the shortest path from a node A to a target node B.

True - becomes equivalent to Dijkstra's if you divide every weight by itself

(b) (T/F) If all edges have distinct weights, the shortest path between any two vertices is unique.

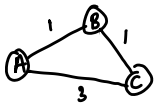
false

A → C has 2 paths of length 3

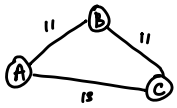



(c) (T/F) **Adding** a constant positive integer $k$ to all edge weights will not affect any shortest path between two vertices.

false - based on number of edges present




Say k is 10 →




(d) (T/F) **Multiplying** a constant positive integer $k$ to all edge weights will not affect any shortest path between two vertices.

True - can factor out from the path

# 3 Shortest Paths Algorithm Design

Two countries, Mondstadt and Fontaine, are located in the fictional world of Teyvat. The railroad system of Teyvat can be modeled as a **weighted directed graph**, with $V$ vertices, $E$ edges, and weights being the length of the railway. Circle the traveler wishes to take railway from Mondstadt to Fontaine, and needs to determine the shortest railway distance between them. Define the set $M$ to be all cities in Mondstadt, and $F$ to be all cities in Fontaine. The shortest distance between the two countries is the shortest distance between any city $c_M$ in Mondstadt and $c_F$ in Fontaine.

For each of the subparts below, describe an algorithm that compute the minimum railway distance from Mondstadt to Fontaine, in $O((V+E)\log V)$ time. You are able to call all graph algorithms you learned in class as a black box. *Hint: For some parts, consider modifying the graph so that running a graph algorithm yields an equivalent answer to solving the original problem.*

(a) Mondstadt only contains 1 city ($|M| = 1$), but Fontaine contains many cities ($|F| > 1$).

Run Dijkstra's to generate SPT from m to every city in F

(b) Mondstadt contains many cities ($|M| > 1$), but Fontaine only contains one city ($|F| = 1$).

Either make a dummy start vertex (see (c))

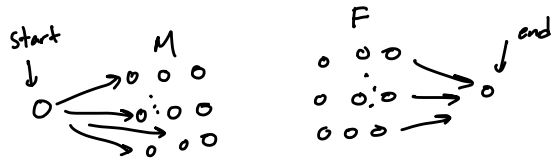
OR

Reverse the graph and do Dijkstra's from f → M, use that SPT

(c) Both countries contain many cities ($|M|, |F| > 1$).

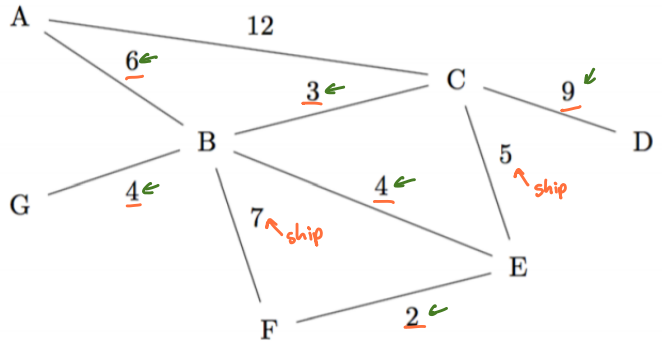
Create a dummy start and end vertex with edge weights 0

↑ connected to M ↑ to F




start → end with Dijkstra's

# 4 Introduction to MSTs




(a) For the graph above, list the edges in the order they're added to the MST by Kruskal's and Prim's algorithm. Assume Prim's algorithm starts at vertex A. Assume ties are broken in alphabetical order. Denote each edge as a pair of vertices (e.g. AB is the edge from A to B).

Prim's algorithm order: AB, BC, BE, EF, BG, CD

Kruskal's algorithm order: EF, BC, BE, BG, AB, CD

(b) True/False: Adding 1 to the smallest edge of a graph G with unique edge weights must change the total weight of its MST.

True - that edge must be in the MST
Either it's still in or a new one is selected, both cases is higher

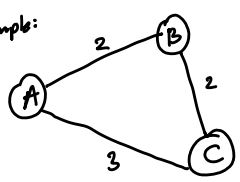
(c) True/False: If all the weights in an MST are unique, there is only one possible MST.

True - Kruskal's would select one unique option
Can also think about unique edges selected via the cut property

(d) True/False: The shortest path from vertex $u$ to vertex $v$ in a graph G is the same as the shortest path from $u$ to $v$ using only edges in T, where T is the MST of G.

False - two different things

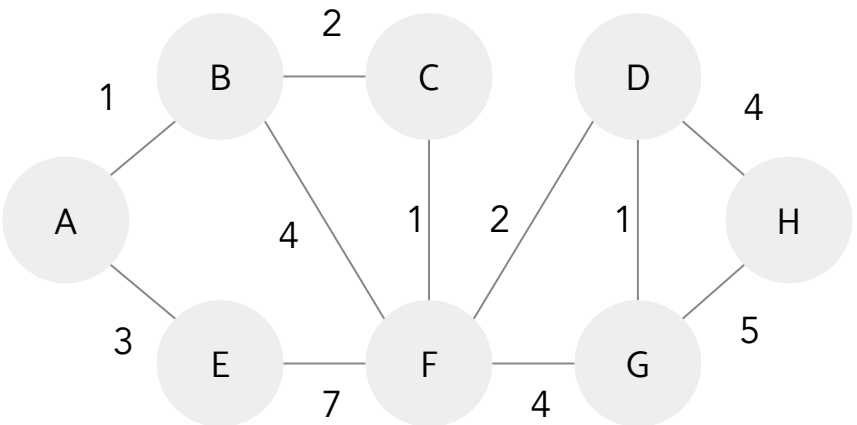
Counter example:




MST is AB, BC
SPT from A is AB, AC
↓
A→C via B in MST is longer

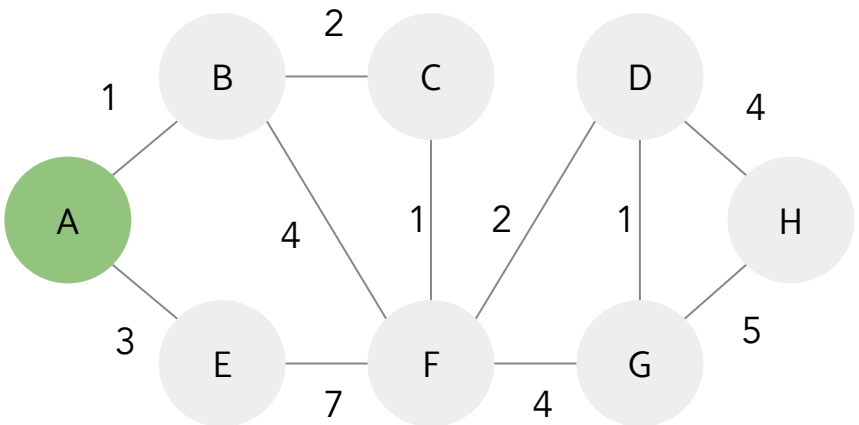
# 1A Dijkstra's, A*

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |



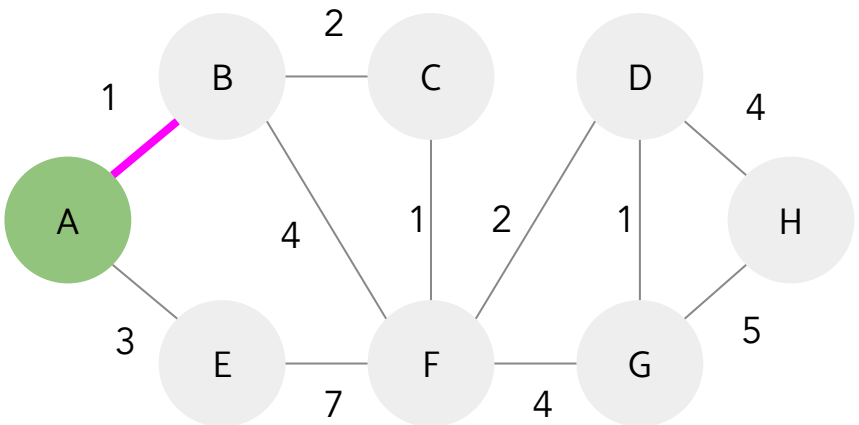
# 1A Dijkstra's, A*

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |



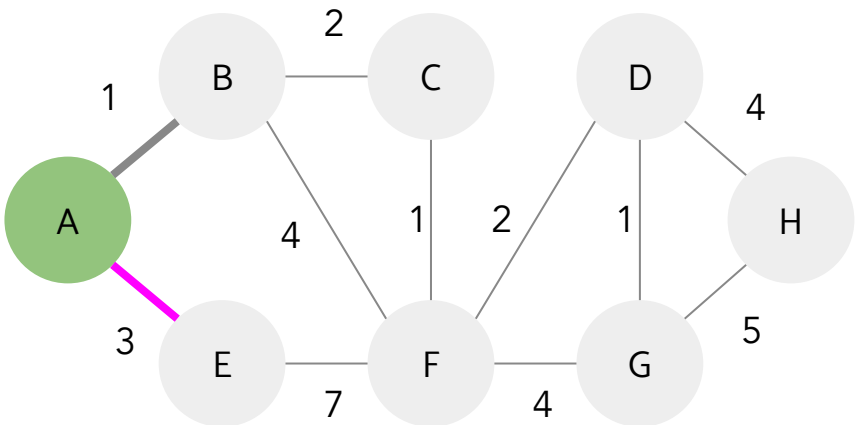
# 1A Dijkstra's, A*

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |

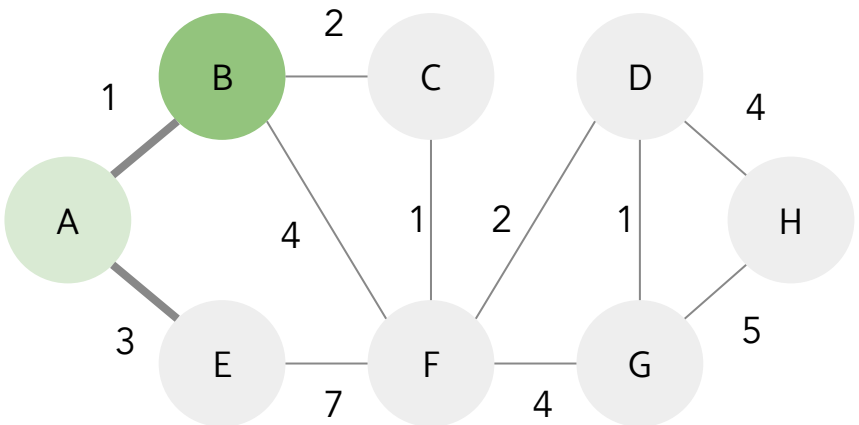


# 1A Dijkstra's, A*

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1 | ∞ | ∞ | 3 | ∞ | ∞ | ∞ |



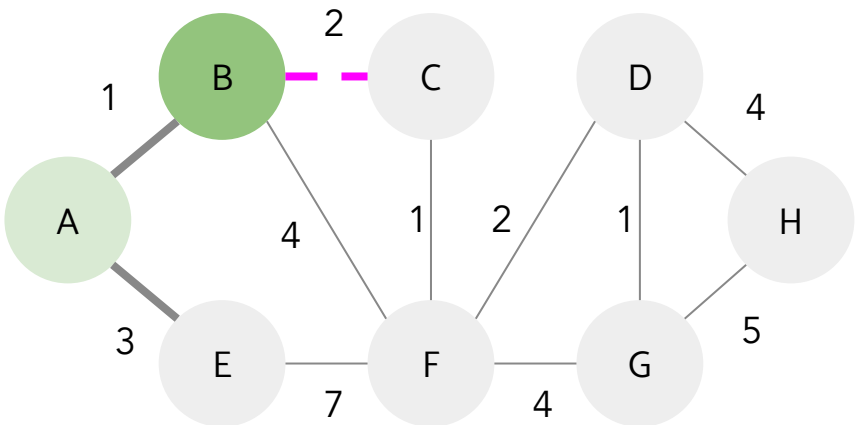
# 1A Dijkstra's, A*




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1 | ∞ | ∞ | 3 | ∞ | ∞ | ∞ |
| 2 | | ✓ | ∞ | ∞ | 3 | ∞ | ∞ | ∞ |



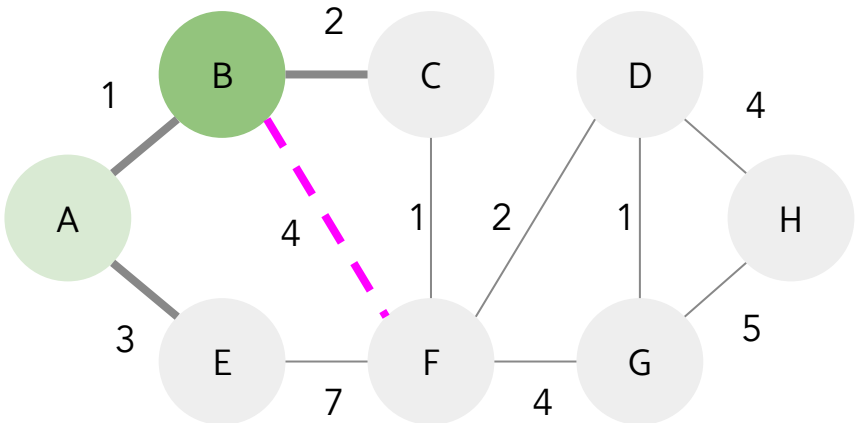
# 1A Dijkstra's, A*




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1 | ∞ | ∞ | 3 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3 | ∞ | 3 | ∞ | ∞ | ∞ |

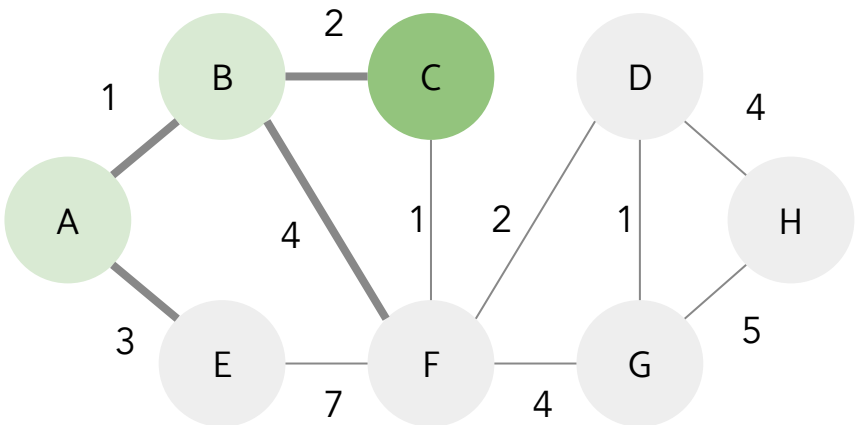


# 1A Dijkstra's, A*



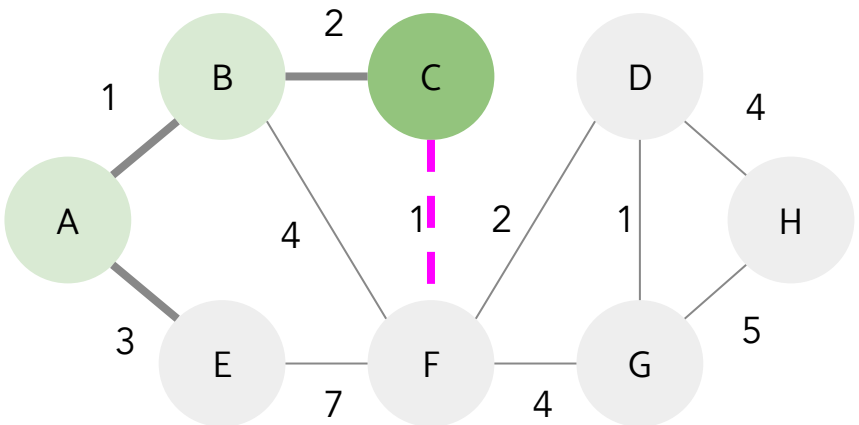

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1 | ∞ | ∞ | 3 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3 | ∞ | 3 | 5 | ∞ | ∞ |

# 1A Dijkstra's, A*




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1 | ∞ | ∞ | 3 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3 | ∞ | 3 | 5 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3 | 5 | ∞ | ∞ |

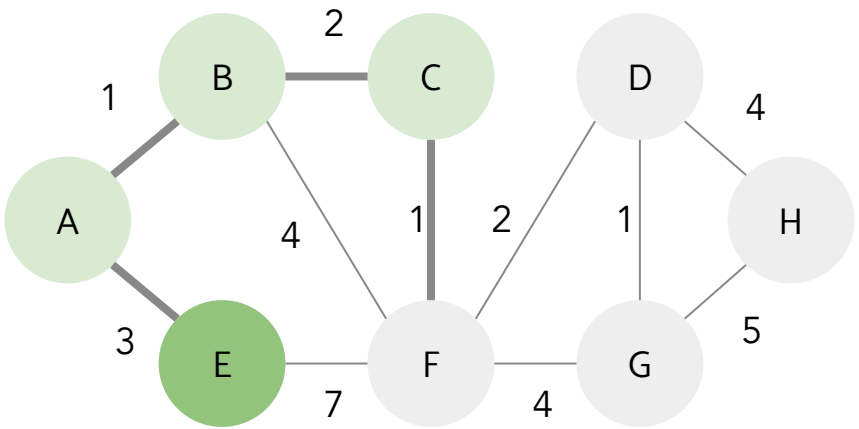
# 1A Dijkstra's, A*




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1 | ∞ | ∞ | 3 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3 | ∞ | 3 | 5 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3 | 4 | ∞ | ∞ |

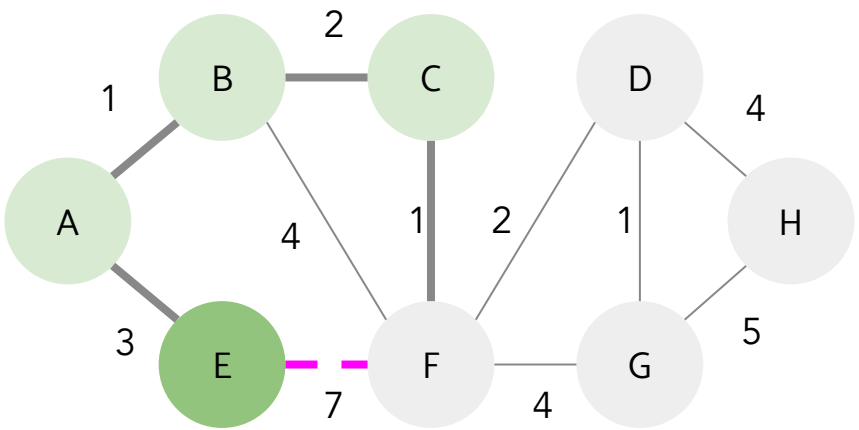
We found a better path to F, so we update distTo[F]

# 1A Dijkstra's, A*




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1 | ∞ | ∞ | 3 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3 | ∞ | 3 | 5 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3 | 4 | ∞ | ∞ |
| 4 | | | | ∞ | ✓ | 4 | ∞ | ∞ |

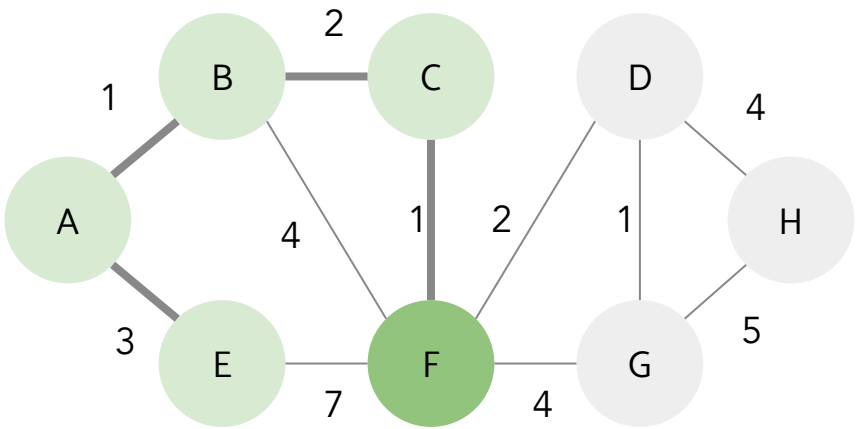
# 1A Dijkstra's, A*




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1 | ∞ | ∞ | 3 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3 | ∞ | 3 | 5 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3 | 4 | ∞ | ∞ |
| 4 | | | | ∞ | ✓ | 4 | ∞ | ∞ |

This new path to F is NOT better than our current one

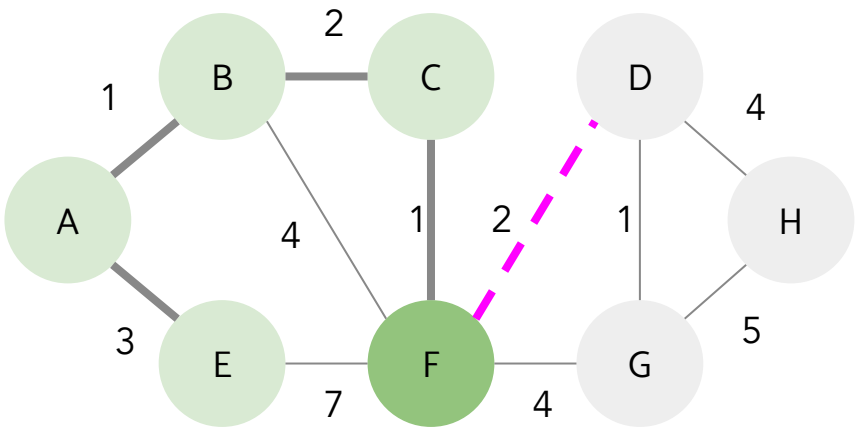
# 1A Dijkstra's, A*




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1 | ∞ | ∞ | 3 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3 | ∞ | 3 | 5 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3 | 4 | ∞ | ∞ |
| 4 | | | | ∞ | ✓ | 4 | ∞ | ∞ |
| 5 | | | | ∞ | | ✓ | ∞ | ∞ |



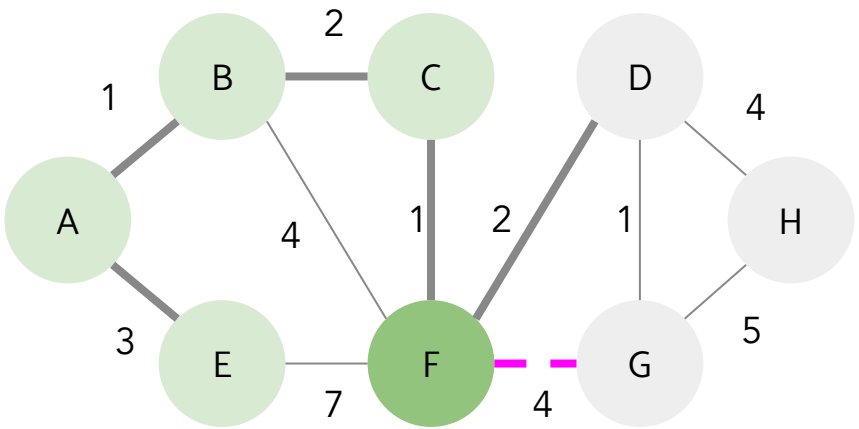
# 1A Dijkstra's, A*




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1 | ∞ | ∞ | 3 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3 | ∞ | 3 | 5 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3 | 4 | ∞ | ∞ |
| 4 | | | | ∞ | ✓ | 4 | ∞ | ∞ |
| 5 | | | | 6 | | ✓ | ∞ | ∞ |



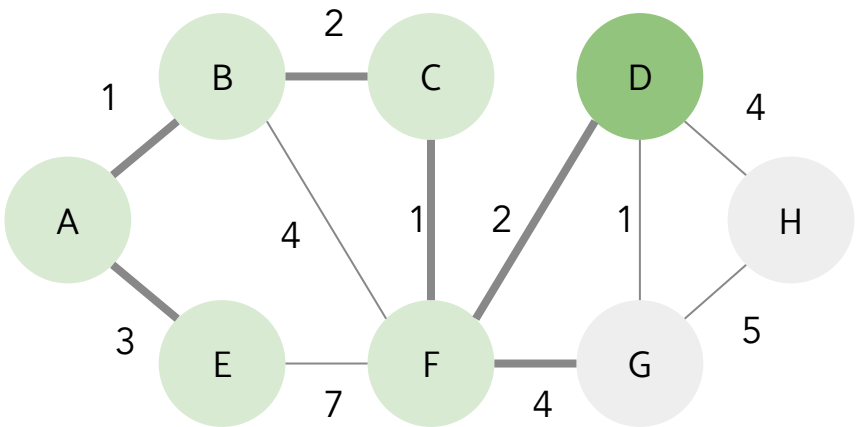
# 1A Dijkstra's, A*




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1 | ∞ | ∞ | 3 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3 | ∞ | 3 | 5 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3 | 4 | ∞ | ∞ |
| 4 | | | | ∞ | ✓ | 4 | ∞ | ∞ |
| 5 | | | | 6 | | ✓ | 8 | ∞ |



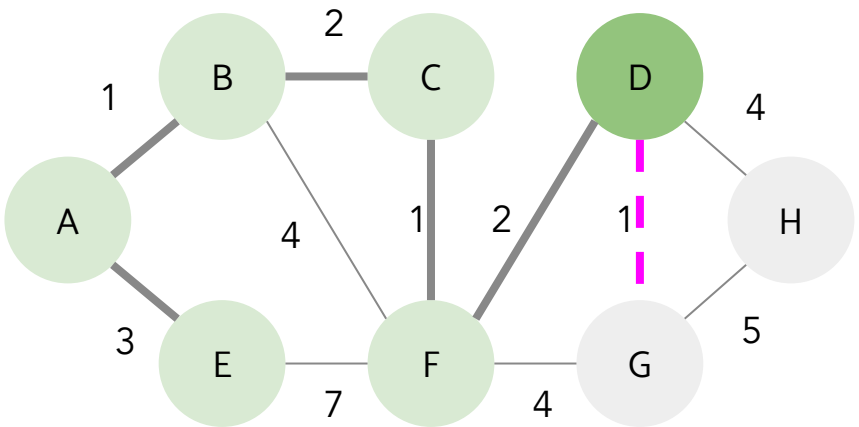
# 1A Dijkstra's, A*




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1 | ∞ | ∞ | 3 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3 | ∞ | 3 | 5 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3 | 4 | ∞ | ∞ |
| 4 | | | | ∞ | ✓ | 4 | ∞ | ∞ |
| 5 | | | | 6 | | ✓ | 8 | ∞ |
| 6 | | | | ✓ | | | 8 | ∞ |



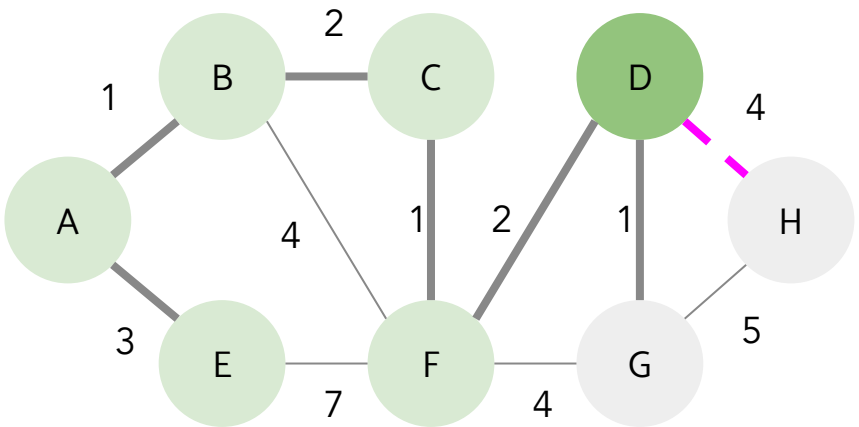
# 1A Dijkstra's, A*




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1 | ∞ | ∞ | 3 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3 | ∞ | 3 | 5 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3 | 4 | ∞ | ∞ |
| 4 | | | | ∞ | ✓ | 4 | ∞ | ∞ |
| 5 | | | | 6 | | ✓ | 8 | ∞ |
| 6 | | | | ✓ | | | 7 | ∞ |

We found a better path to G, so we update distTo[G]

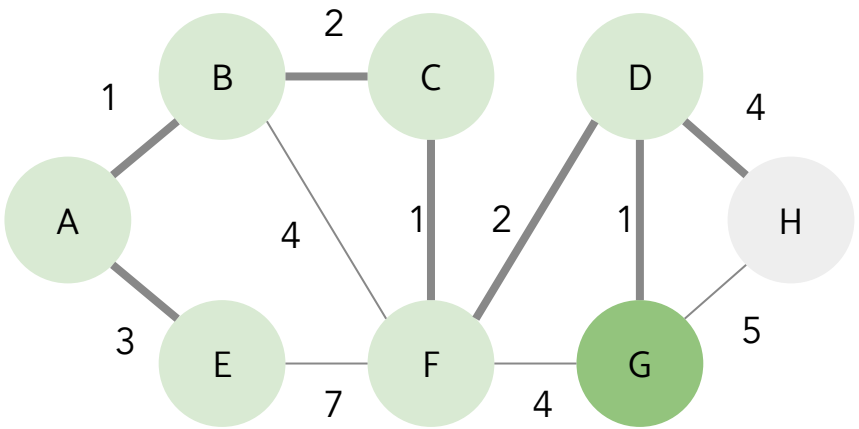
# 1A Dijkstra's, A*




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1 | ∞ | ∞ | 3 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3 | ∞ | 3 | 5 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3 | 4 | ∞ | ∞ |
| 4 | | | | ∞ | ✓ | 4 | ∞ | ∞ |
| 5 | | | | 6 | | ✓ | 8 | ∞ |
| 6 | | | | ✓ | | | 7 | 10 |



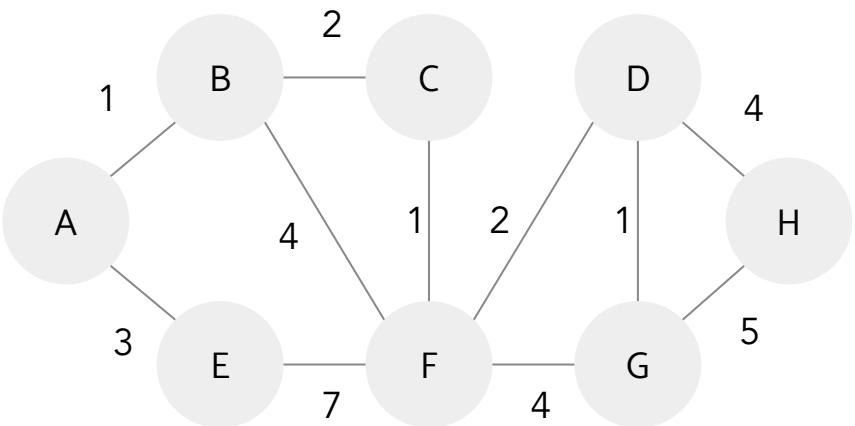
# 1A Dijkstra's, A*




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| DistTo | 0 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1 | ∞ | ∞ | 3 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3 | ∞ | 3 | 5 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3 | 4 | ∞ | ∞ |
| 4 | | | | ∞ | ✓ | 4 | ∞ | ∞ |
| 5 | | | | 6 | | ✓ | 8 | ∞ |
| 6 | | | | ✓ | | | 7 | 10 |
| 7 | | | | | | | ✓ | |



# 1B Dijkstra's, A*

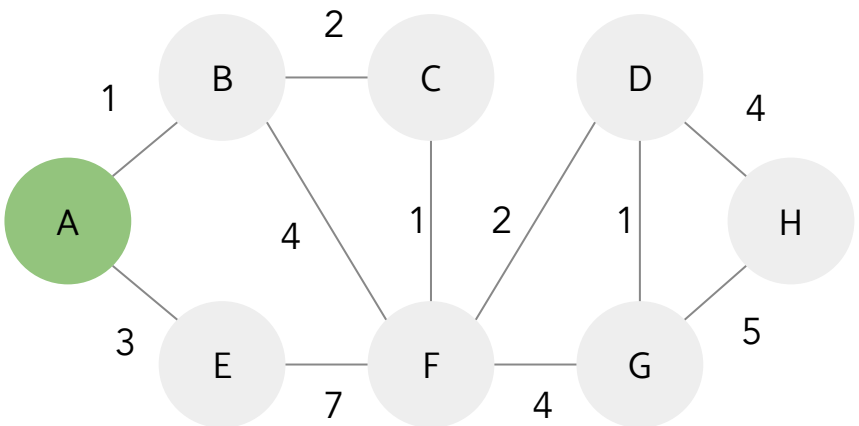



| u | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| h(u,G) | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0,9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |



# 1B Dijkstra's, A*

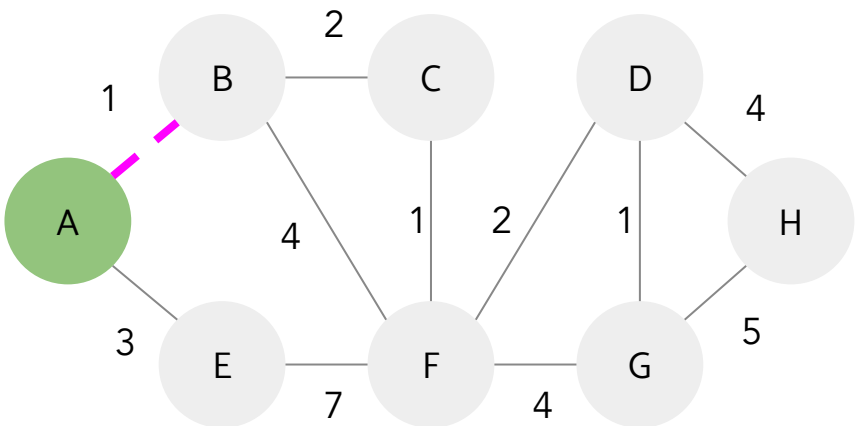



| u | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| h(u,G) | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0,9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |



# 1B Dijkstra's, A*

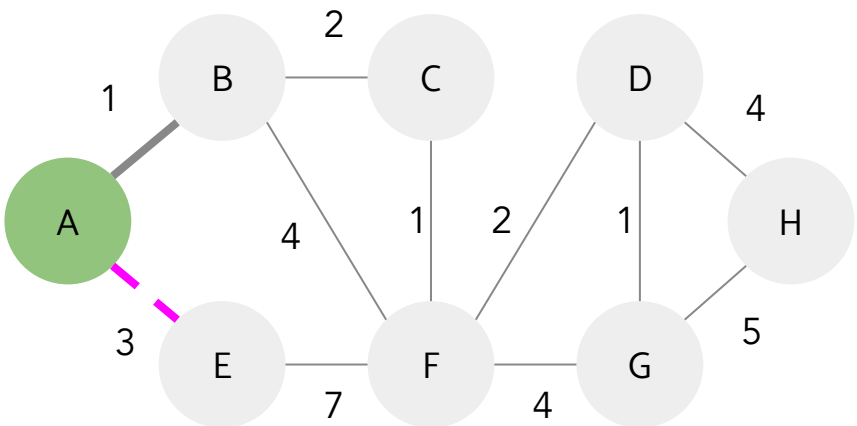



| u | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| h(u,G) | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0,9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1,8 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |



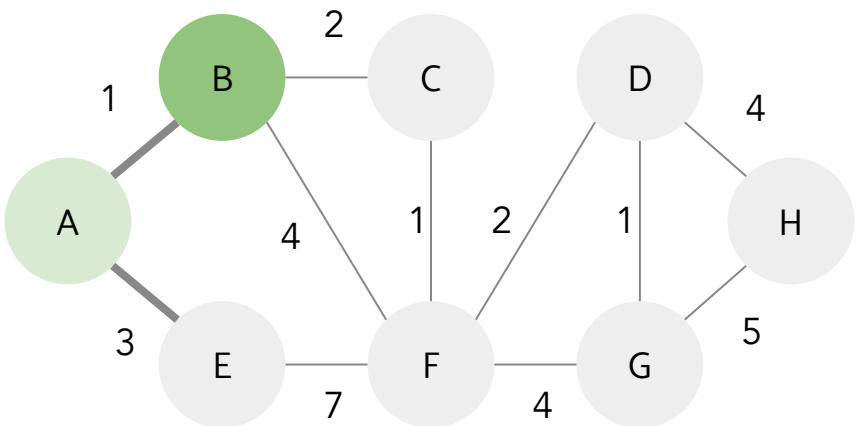
# 1B Dijkstra's, A*




| u | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| h(u,G) | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0,9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1,8 | ∞ | ∞ | 3,13 | ∞ | ∞ | ∞ |

# 1B Dijkstra's, A*

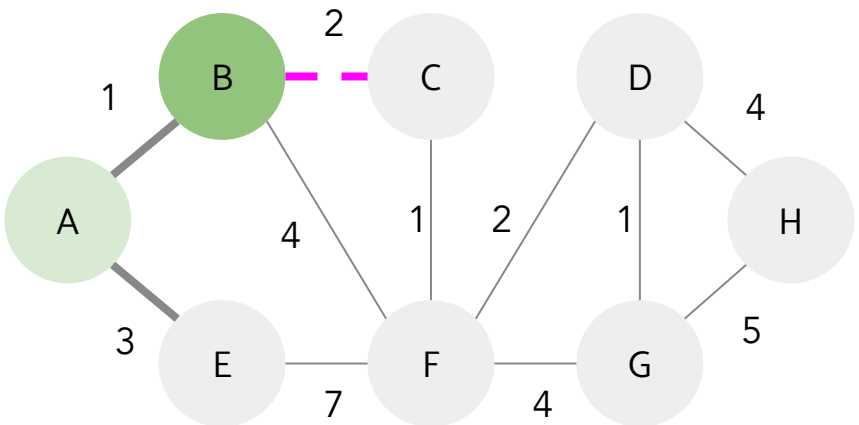



| u | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| h(u,G) | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0,9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1,8 | ∞ | ∞ | 3,13 | ∞ | ∞ | ∞ |
| 2 | | ✓ | ∞ | ∞ | 3,13 | ∞ | ∞ | ∞ |



# 1B Dijkstra's, A*

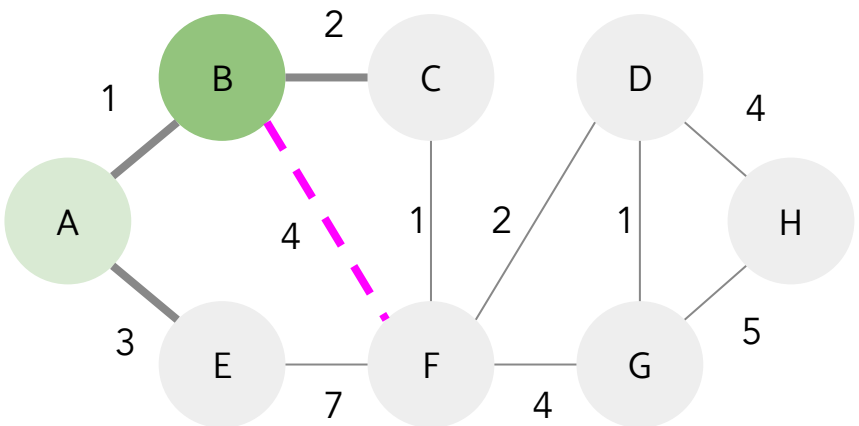



| u | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| h(u,G) | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0,9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1,8 | ∞ | ∞ | 3,13 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3,7 | ∞ | 3,13 | ∞ | ∞ | ∞ |



# 1B Dijkstra's, A*

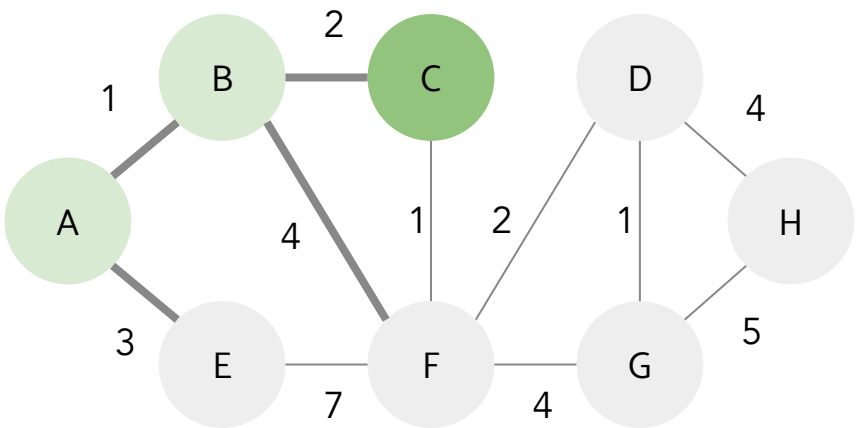



| u | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| h(u,G) | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0,9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1,8 | ∞ | ∞ | 3,13 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3,7 | ∞ | 3,13 | 5,8 | ∞ | ∞ |



# 1B Dijkstra's, A*

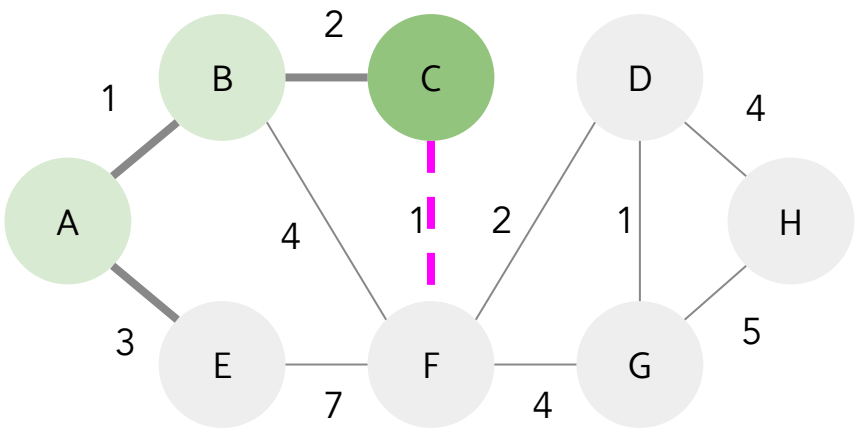



| u | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| h(u,G) | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0,9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1,8 | ∞ | ∞ | 3,13 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3,7 | ∞ | 3,13 | 5,8 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3,13 | 5,8 | ∞ | ∞ |



# 1B Dijkstra's, A*

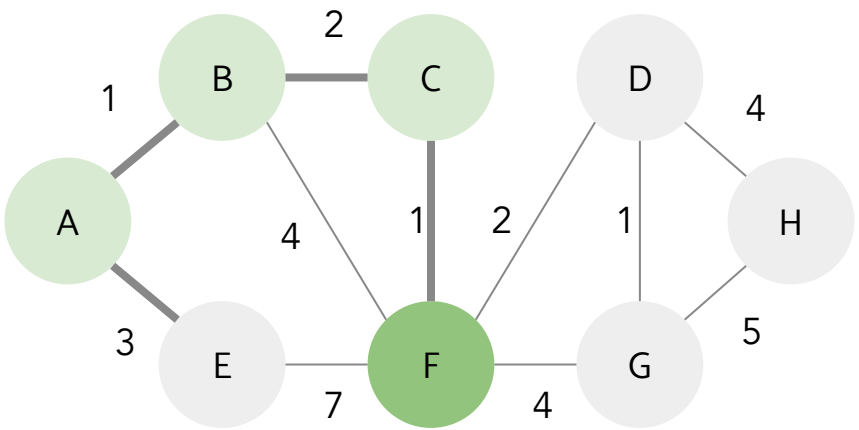



| u | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| h(u,G) | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0,9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1,8 | ∞ | ∞ | 3,13 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3,7 | ∞ | 3,13 | 5,8 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3,13 | 4,7 | ∞ | ∞ |



# 1B Dijkstra's, A*

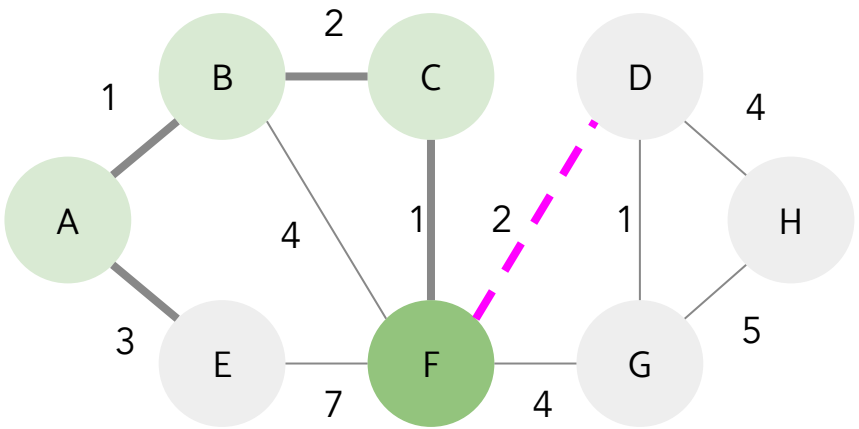



| u | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| h(u,G) | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0,9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1,8 | ∞ | ∞ | 3,13 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3,7 | ∞ | 3,13 | 5,8 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3,13 | 4,7 | ∞ | ∞ |
| 4 | | | | ∞ | 3,13 | ✓ | ∞ | ∞ |

Even though E has a closer distance, it has lower priority than F because it has a high heuristic!

# 1B Dijkstra's, A*

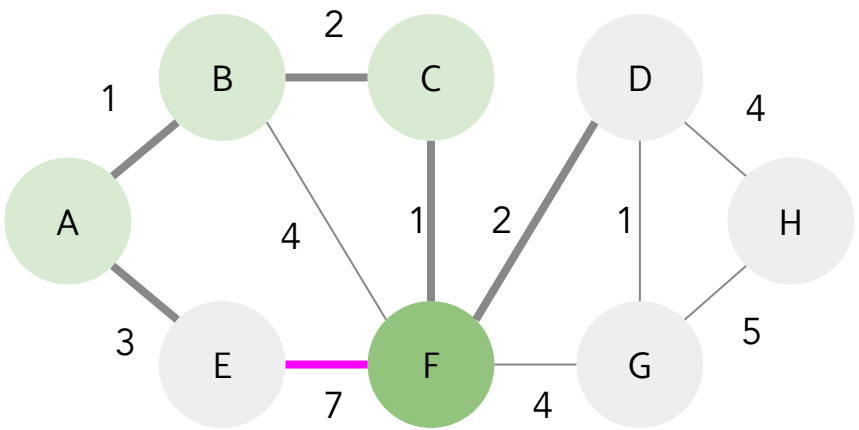



| u | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| h(u,G) | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0,9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1,8 | ∞ | ∞ | 3,13 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3,7 | ∞ | 3,13 | 5,8 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3,13 | 4,7 | ∞ | ∞ |
| 4 | | | | 6,7 | 3,13 | ✓ | ∞ | ∞ |



# 1B Dijkstra's, A*

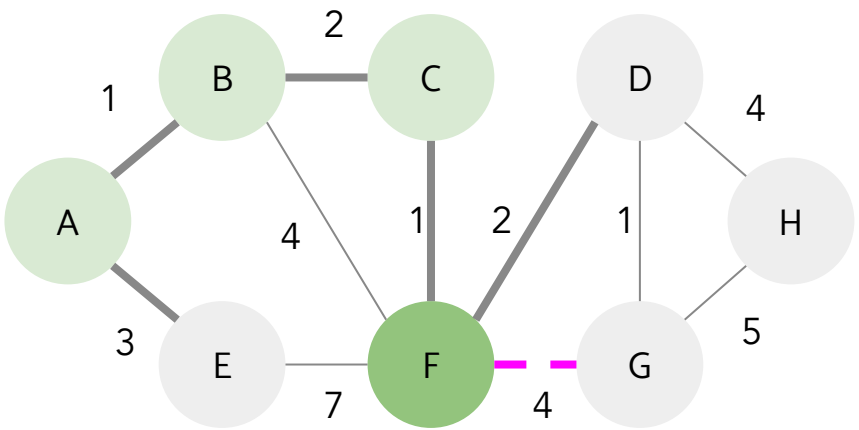



| u | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| h(u,G) | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0,9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1,8 | ∞ | ∞ | 3,13 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3,7 | ∞ | 3,13 | 5,8 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3,13 | 4,7 | ∞ | ∞ |
| 4 | | | | 6,7 | 3,13 | ✓ | ∞ | ∞ |

The path to E from F is not better than our current path

# 1B Dijkstra's, A*

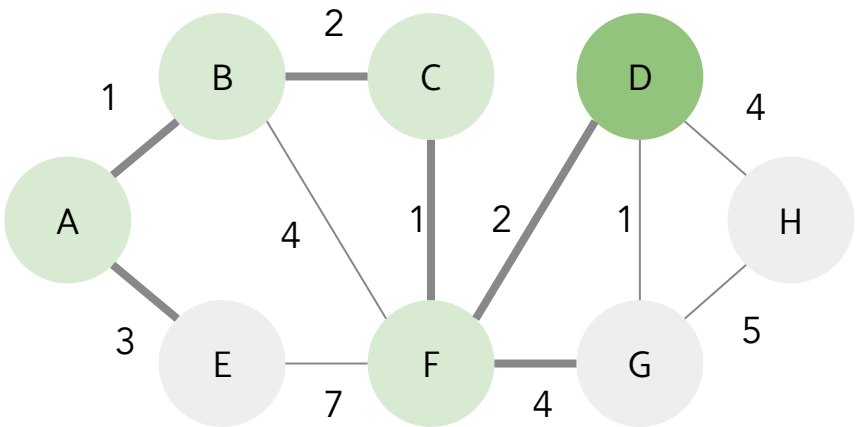



| u | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| h(u,G) | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0,9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1,8 | ∞ | ∞ | 3,13 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3,7 | ∞ | 3,13 | 5,8 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3,13 | 4,7 | ∞ | ∞ |
| 4 | | | | 6,7 | 3,13 | ✓ | 8,8 | ∞ |



# 1B Dijkstra's, A*

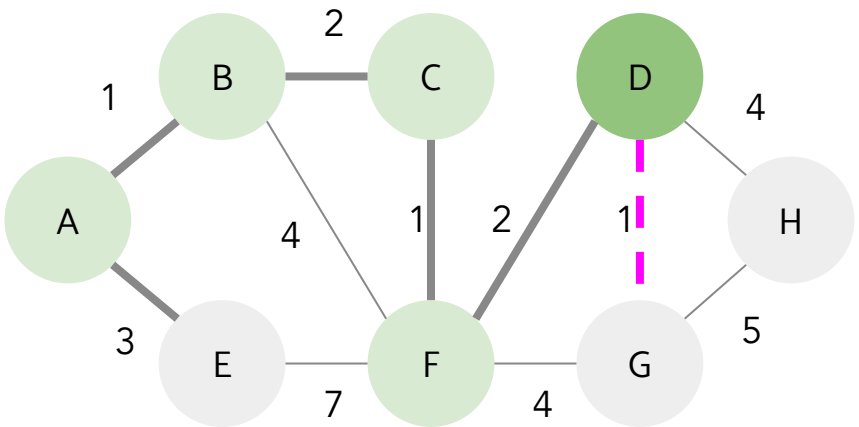



| u | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| h(u,G) | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0,9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1,8 | ∞ | ∞ | 3,13 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3,7 | ∞ | 3,13 | 5,8 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3,13 | 4,7 | ∞ | ∞ |
| 4 | | | | 6,7 | 3,13 | ✓ | 8,8 | ∞ |
| 5 | | | | ✓ | 3,13 | | 8,8 | ∞ |



# 1B Dijkstra's, A*

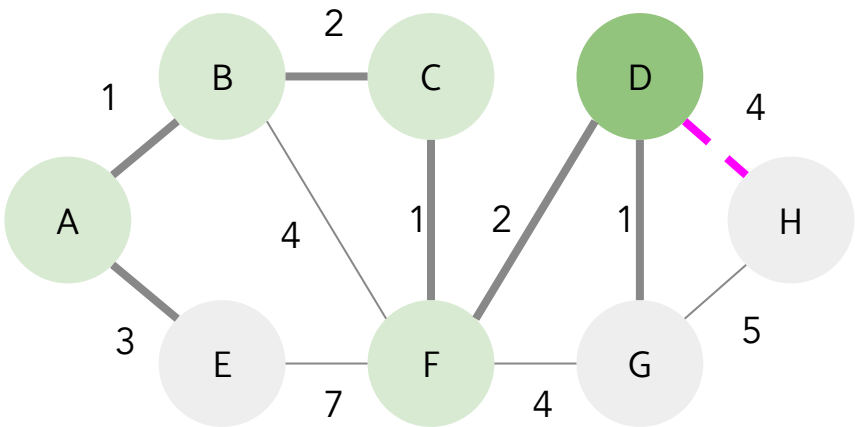



| u | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| h(u,G) | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0,9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1,8 | ∞ | ∞ | 3,13 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3,7 | ∞ | 3,13 | 5,8 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3,13 | 4,7 | ∞ | ∞ |
| 4 | | | | 6,7 | 3,13 | ✓ | 8,8 | ∞ |
| 5 | | | | ✓ | 3,13 | | 7,7 | ∞ |



# 1B Dijkstra's, A*

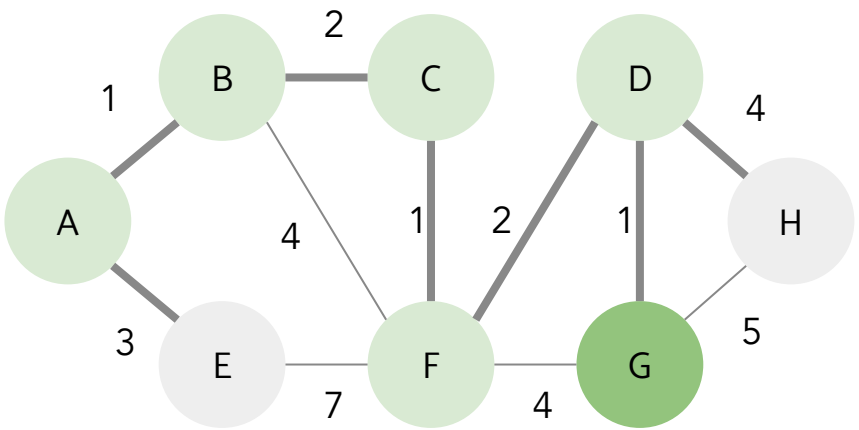



| u | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| h(u,G) | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0,9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1,8 | ∞ | ∞ | 3,13 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3,7 | ∞ | 3,13 | 5,8 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3,13 | 4,7 | ∞ | ∞ |
| 4 | | | | 6,7 | 3,13 | ✓ | 8,8 | ∞ |
| 5 | | | | ✓ | 3,13 | | 7,7 | 10,15 |



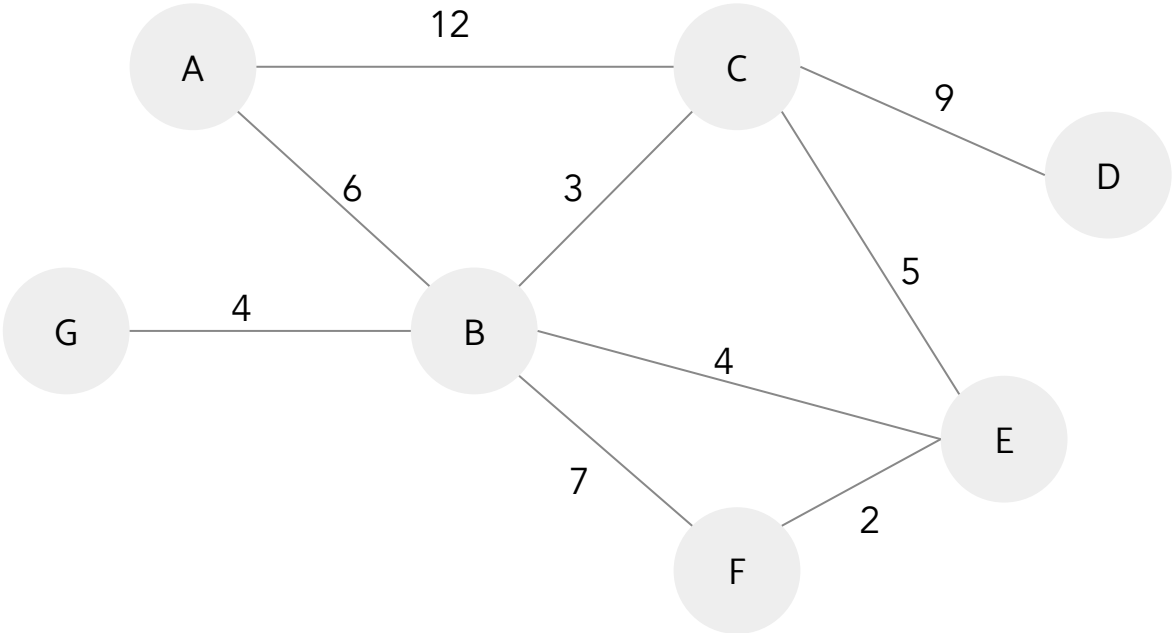
# 1B Dijkstra's, A*




| u | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| h(u,G) | 9 | 7 | 4 | 1 | 10 | 3 | 0 | 5 |

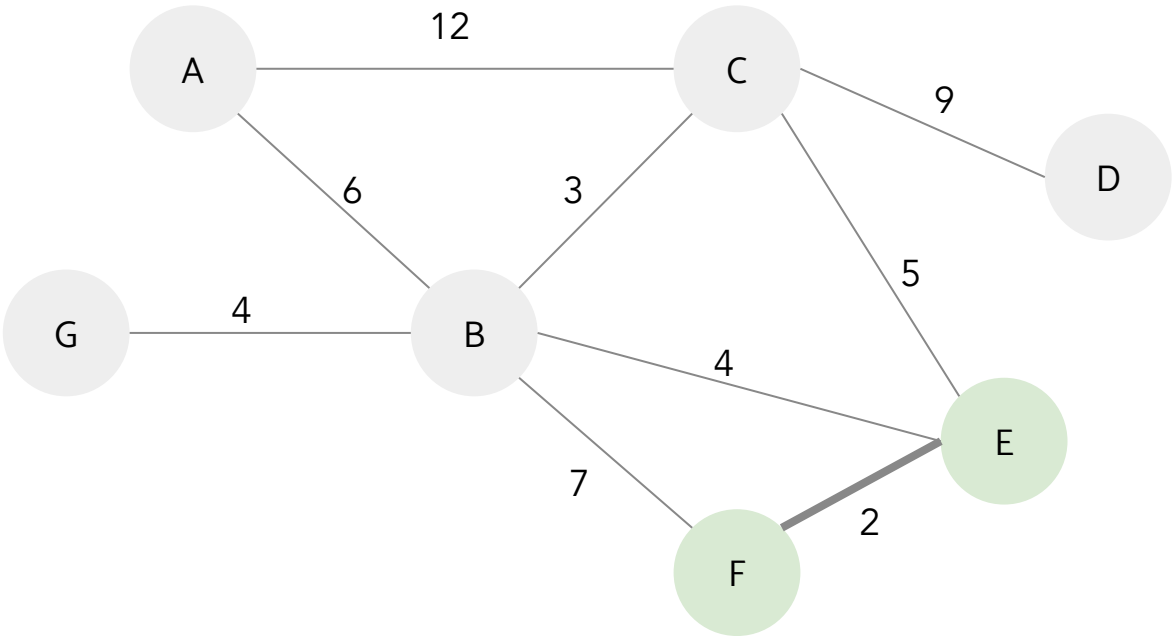
| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| Start | 0,9 | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ | ∞ |
| 1 | ✓ | 1,8 | ∞ | ∞ | 3,13 | ∞ | ∞ | ∞ |
| 2 | | ✓ | 3,7 | ∞ | 3,13 | 5,8 | ∞ | ∞ |
| 3 | | | ✓ | ∞ | 3,13 | 4,7 | ∞ | ∞ |
| 4 | | | | 6,7 | 3,13 | ✓ | 8,8 | ∞ |
| 5 | | | | ✓ | 3,13 | | 7,7 | 10,15 |
| 6 | | | | | | | ✓ | |

# 4A Introduction to MSTs - Kruskal's



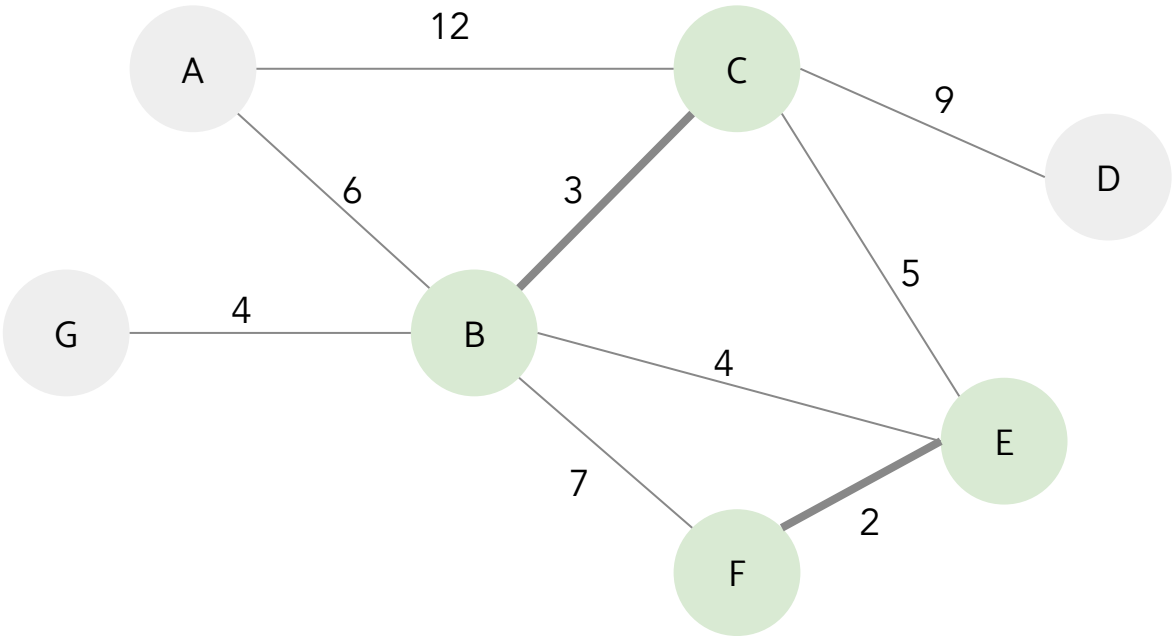

Connected Nodes:

# 4A Introduction to MSTs - Kruskal's



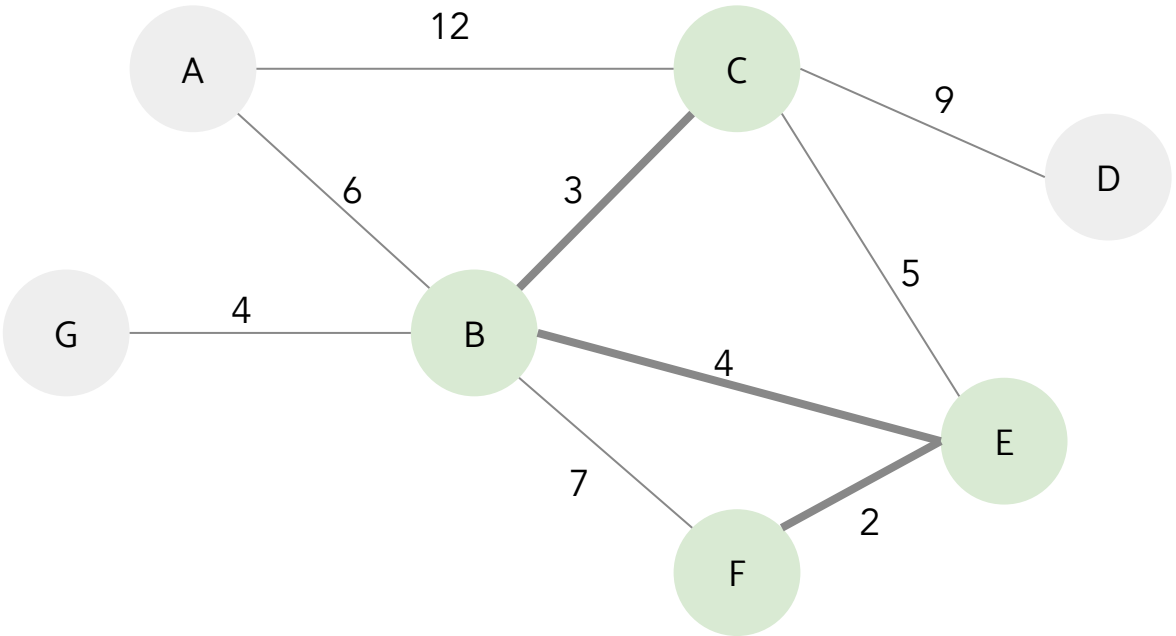

Connected Nodes:
F
E

# 4A Introduction to MSTs - Kruskal's



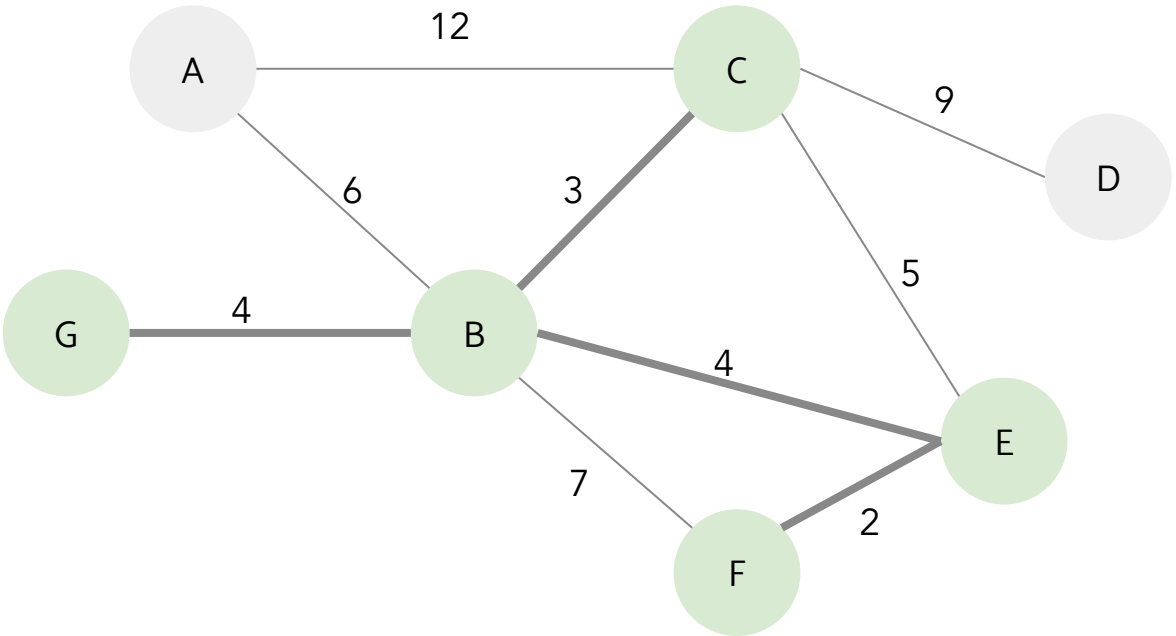

Connected Nodes:
F
E
B
C

# 4A Introduction to MSTs - Kruskal's



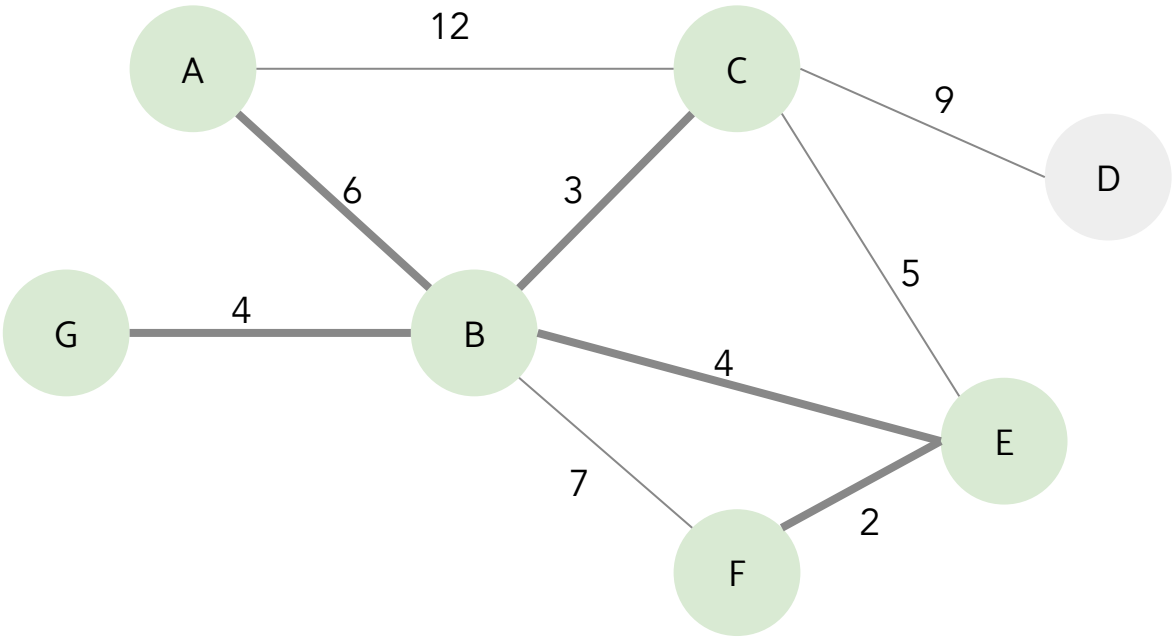

Connected Nodes:
F
E
B
C

# 4A Introduction to MSTs - Kruskal's




Connected Nodes:
F
E
B
C
G

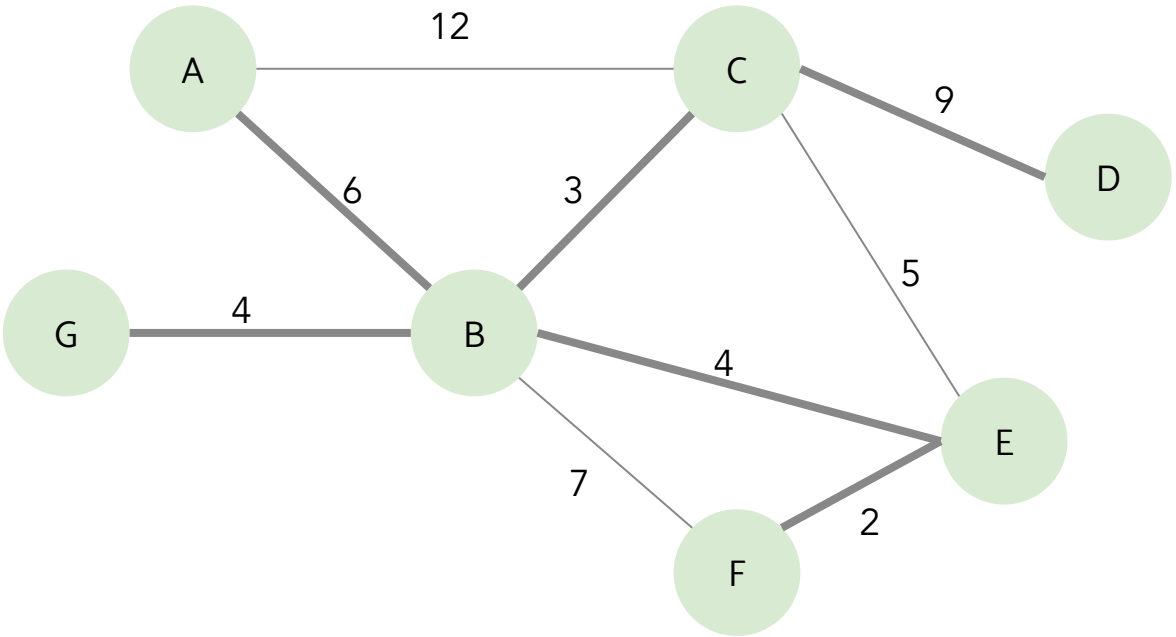
# 4A Introduction to MSTs - Kruskal's




Connected Nodes:
F
E
B
C
G
A

We don't take edge CE; it creates a cycle!

# 4A Introduction to MSTs - Kruskal's

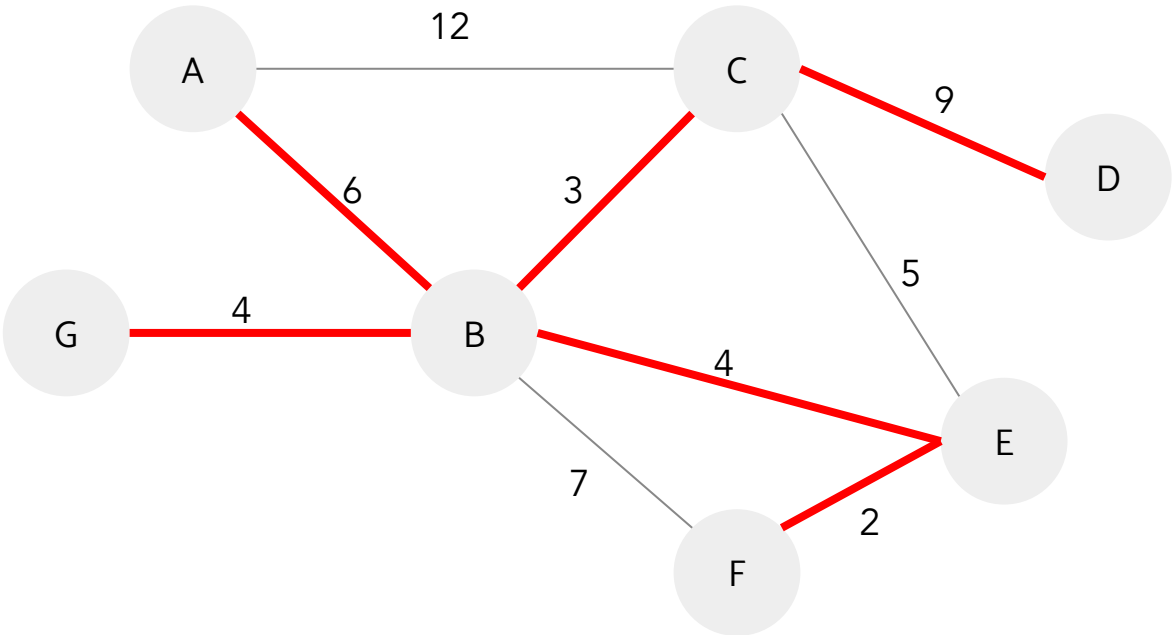



Connected Nodes:
F
E
B
C
G
A
D

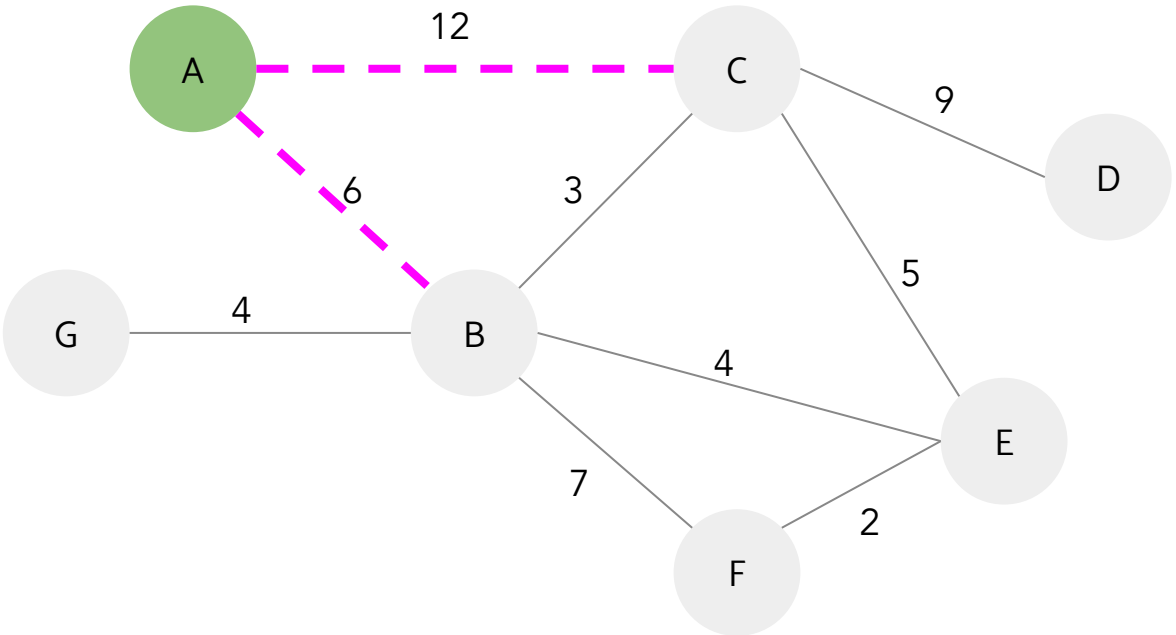
We don't take edge BF; it creates a cycle!

# 4A Introduction to MSTs - Kruskal's

Final Result

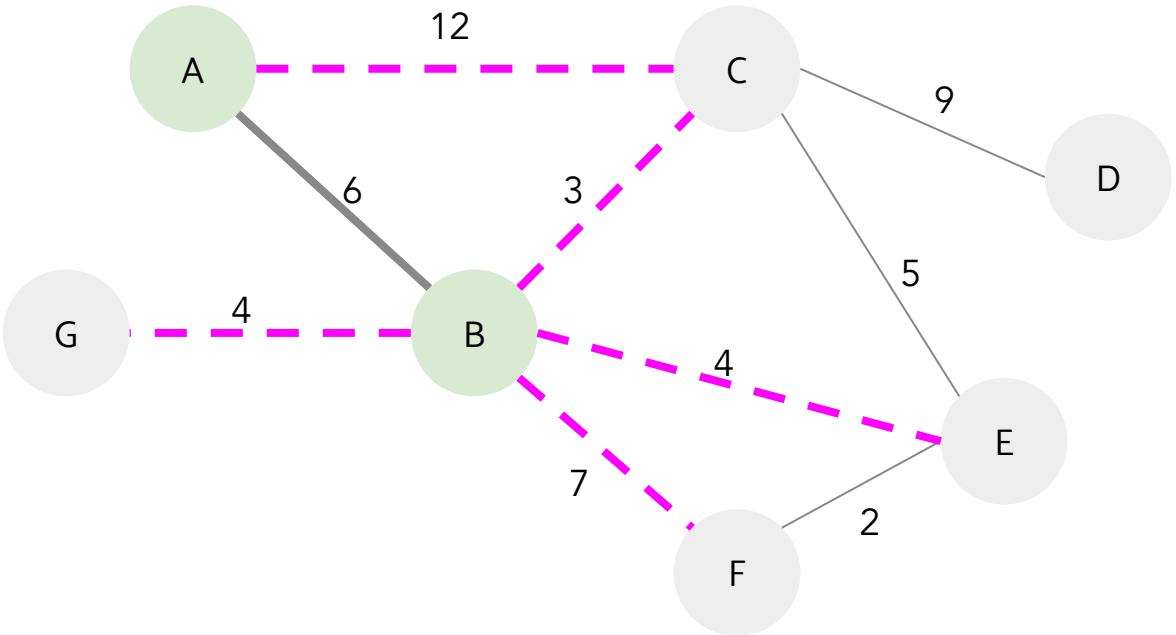


# 4A Introduction to MSTs - Prim's



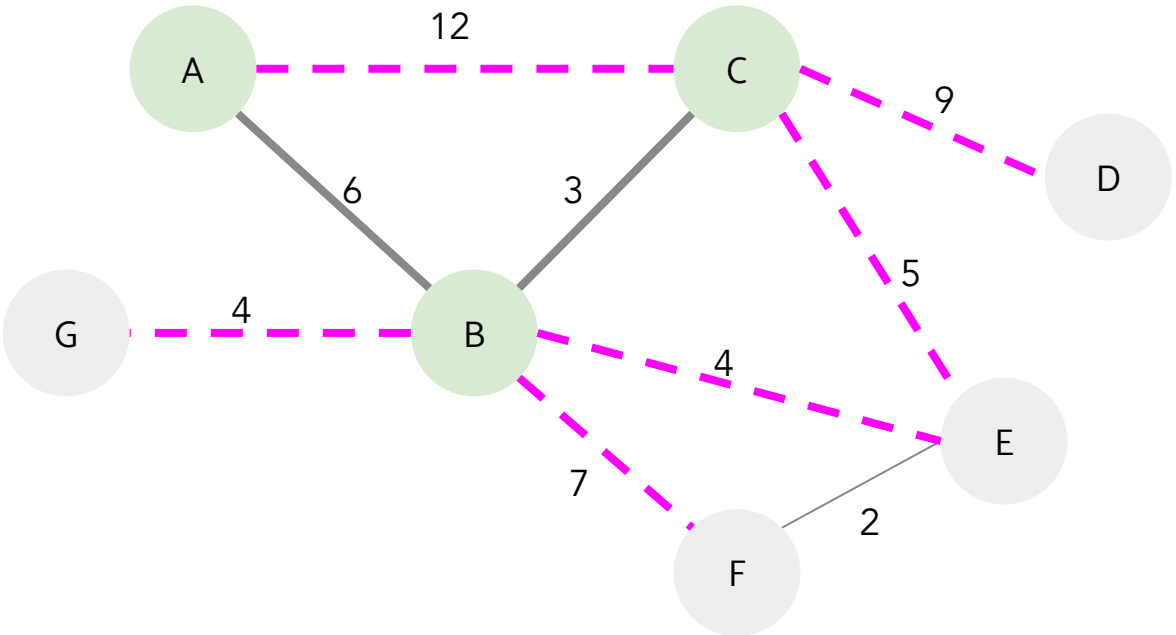

Connected Nodes:
A

# 4A Introduction to MSTs - Prim's



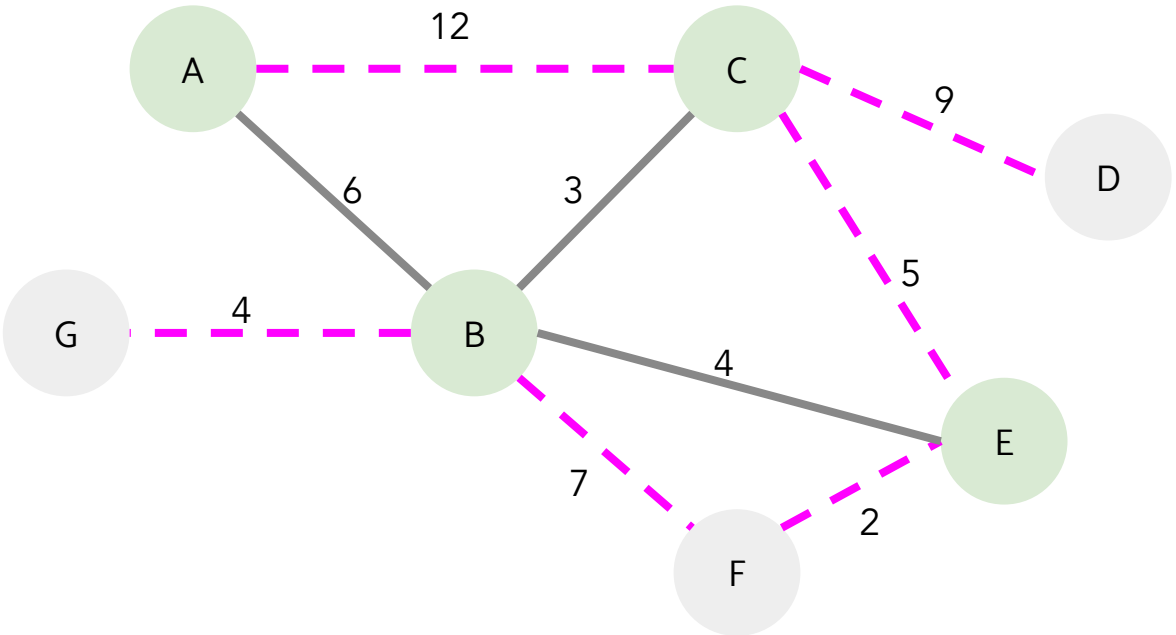

Connected Nodes:
A
B

# 4A Introduction to MSTs - Prim's



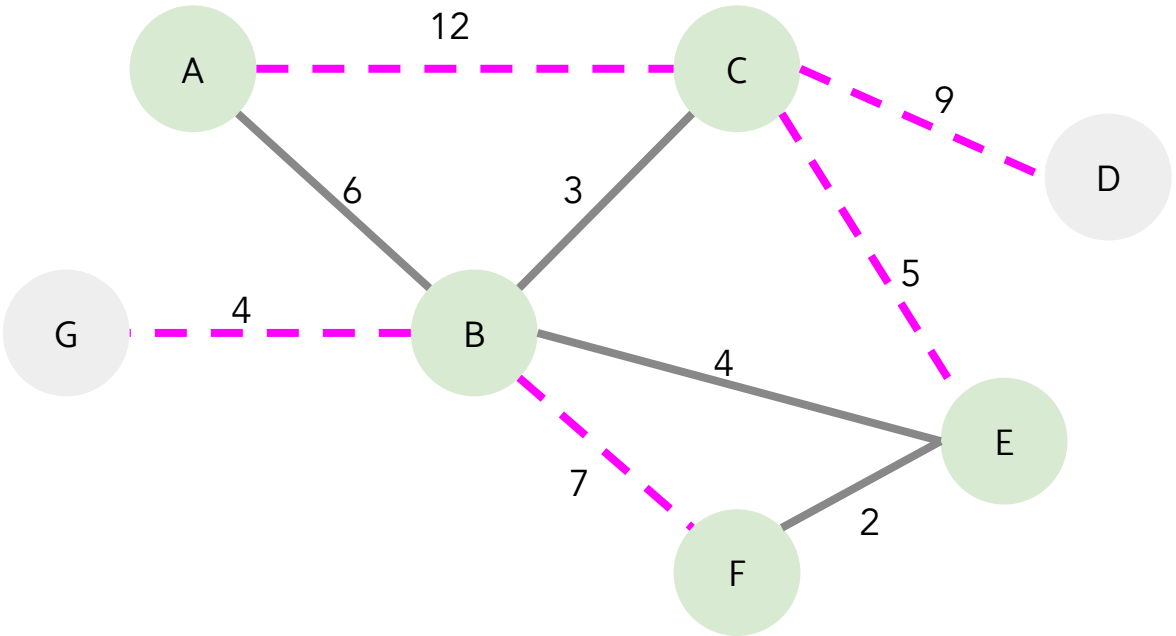

Connected Nodes:
A
B
C

# 4A Introduction to MSTs - Prim's



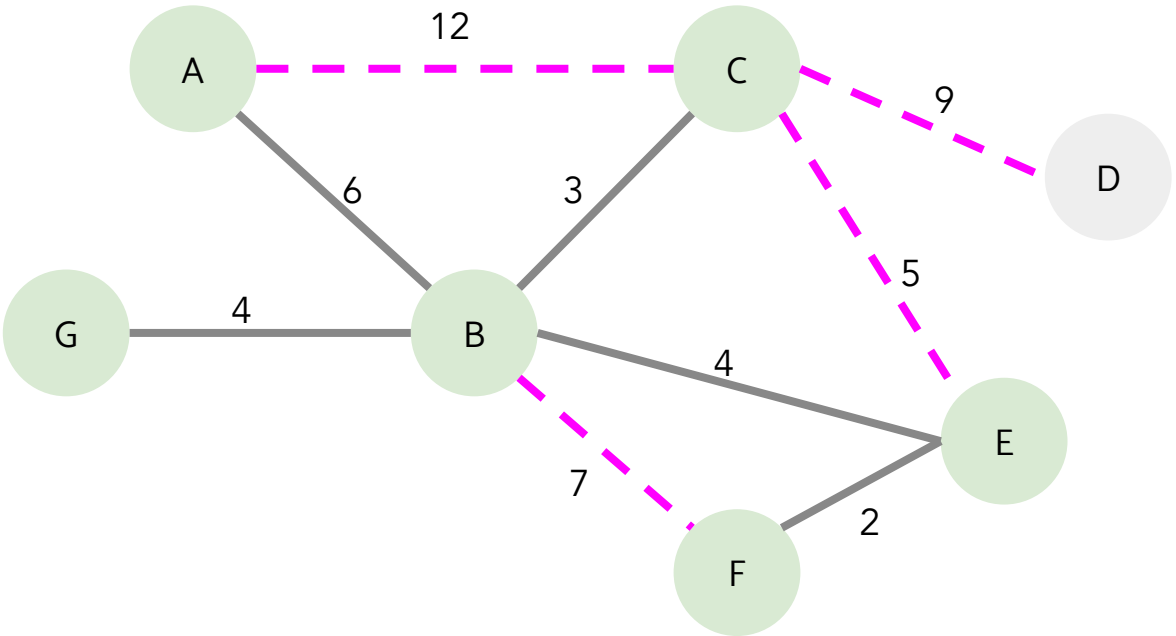

Connected Nodes:
A
B
C
E

# 4A Introduction to MSTs - Prim's




Connected Nodes:
A
B
C
E
F

# 4A Introduction to MSTs - Prim's




Connected Nodes:
A
B
C
E
F
G

# 4A Introduction to MSTs - Prim's

Final Result