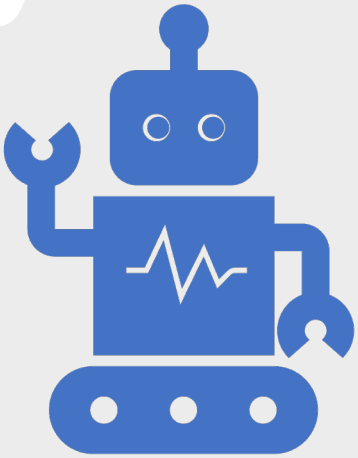
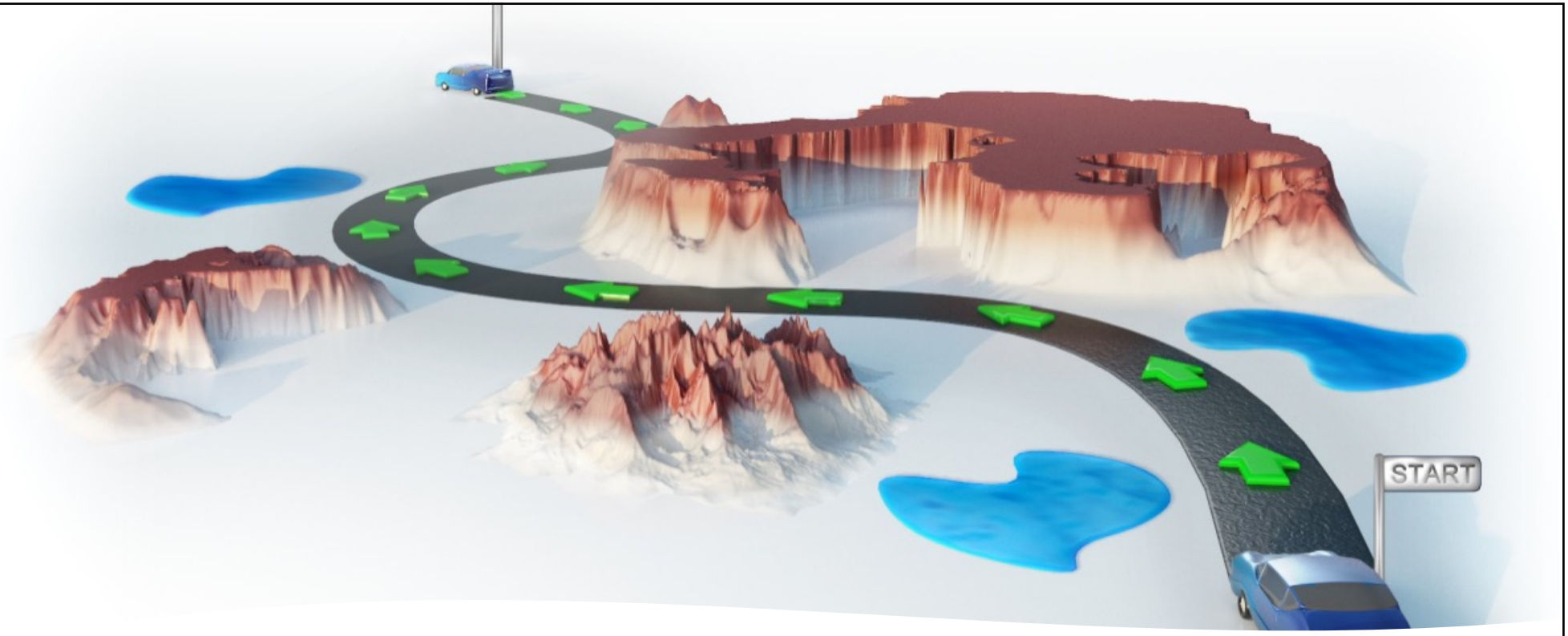
EE3305 / ME3243 Robotic System Design

Navigation and Path Planning for Mobile Robots with ROS Implementation



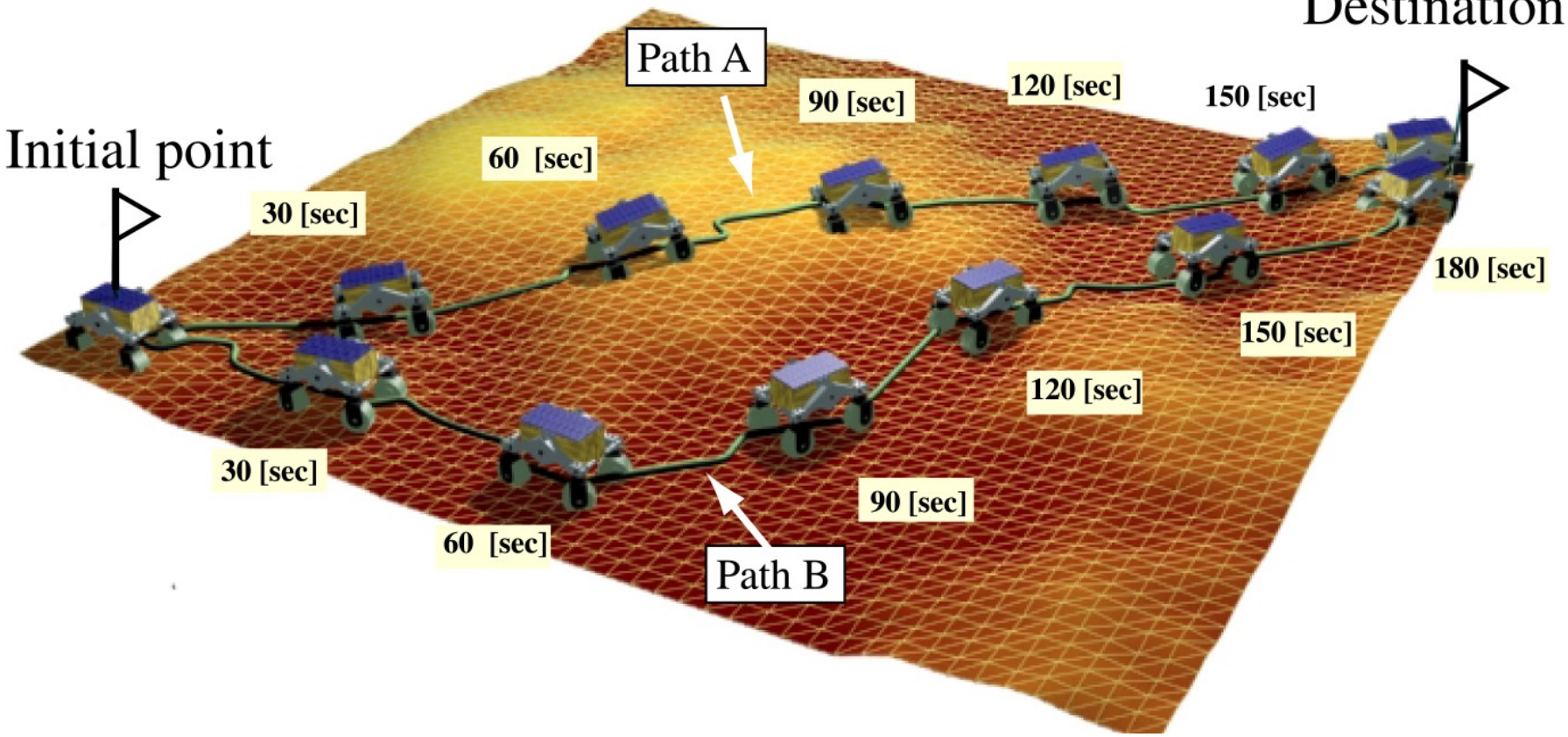
A/Prof. Prahlad Vadakkepat

Dr. Andi Sudjana Putra



Contents

- Graph
- Depth First Search
- Breadth First Search
- Dijkstra’s Algorithm
- A-Star Algorithm



What did you do this morning to come to class?

Being in a course about robot, you can imagine you are one.

You have a map.

You know where you are on the map.

You know your destination on the map.

What you need to do is to plan a path between where you are and the destination, and to follow that path.

This is a path planning problem.

Example?

You have a map.

You know where you are on the map.

You do not know your destination on the map.

What you need to do is go around searching for the destination and ensure you cover every point strategically.

This is a coverage problem.

Example?

Assumptions in path planning

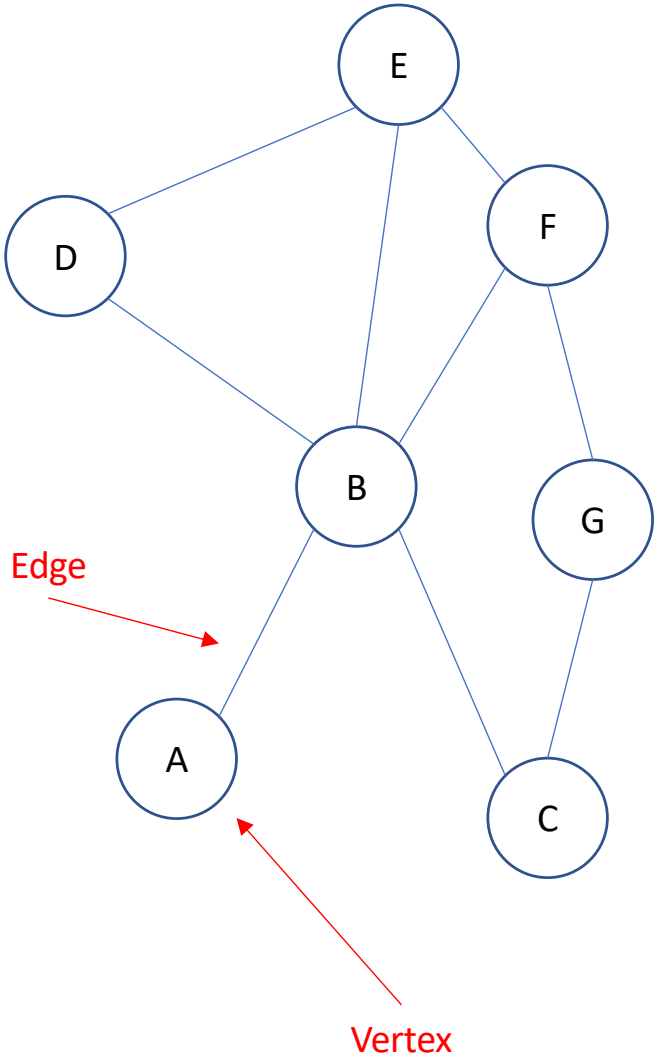
- There is an accurate map of the world.
- There is enough memory to hold the map.
- There is enough time to run the planner.
- The robot is controllable.

These assumptions are valid for structured applications, such as planning a sequence of assembly operations on the assembly line.

Graph

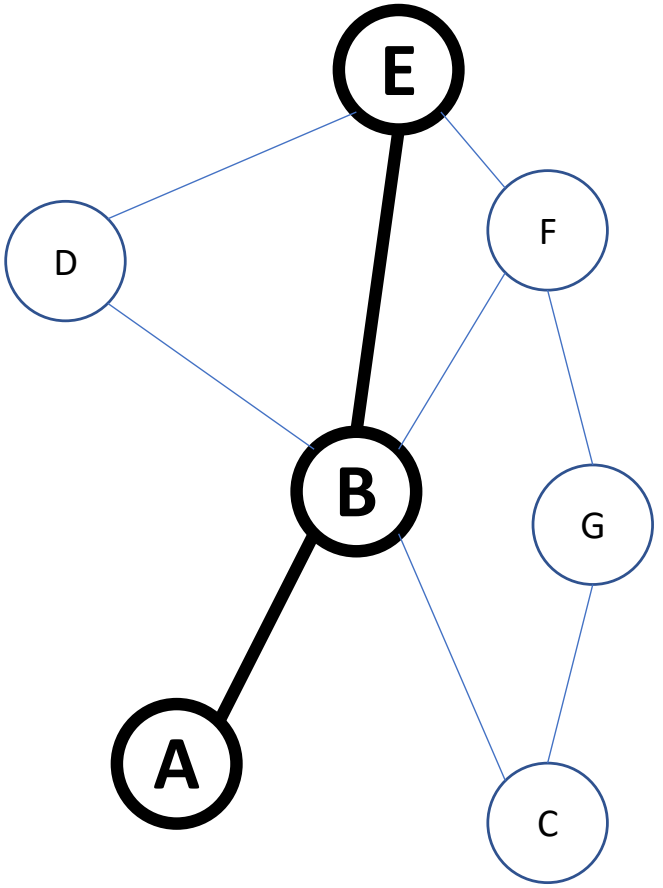
What is a Graph

- Graph: a set of vertices connected by edges
- Path: sequence of vertices and edges from one vertex to another vertex
- Directed graph: when the edges have direction
- Undirected graph: when the edges do not have direction (i.e., bidirectional)



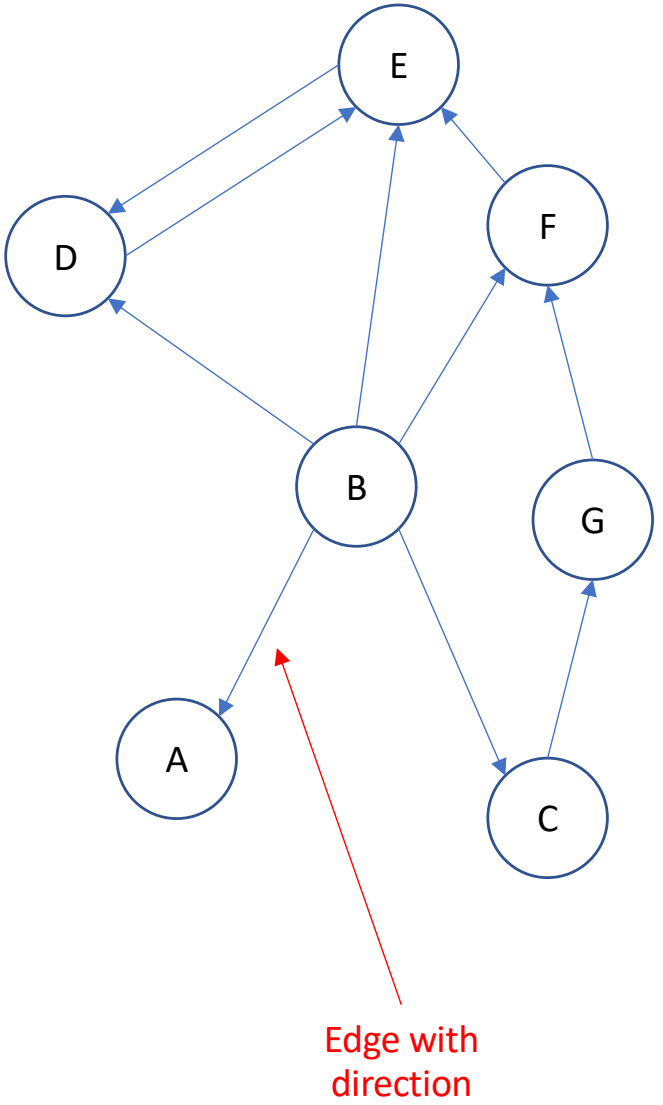
What is a Path

- Graph: a set of vertices connected by edges
- Path: sequence of vertices and edges from one vertex to another vertex
- Directed graph: when the edges have direction
- Undirected graph: when the edges do not have direction (i.e., bidirectional)



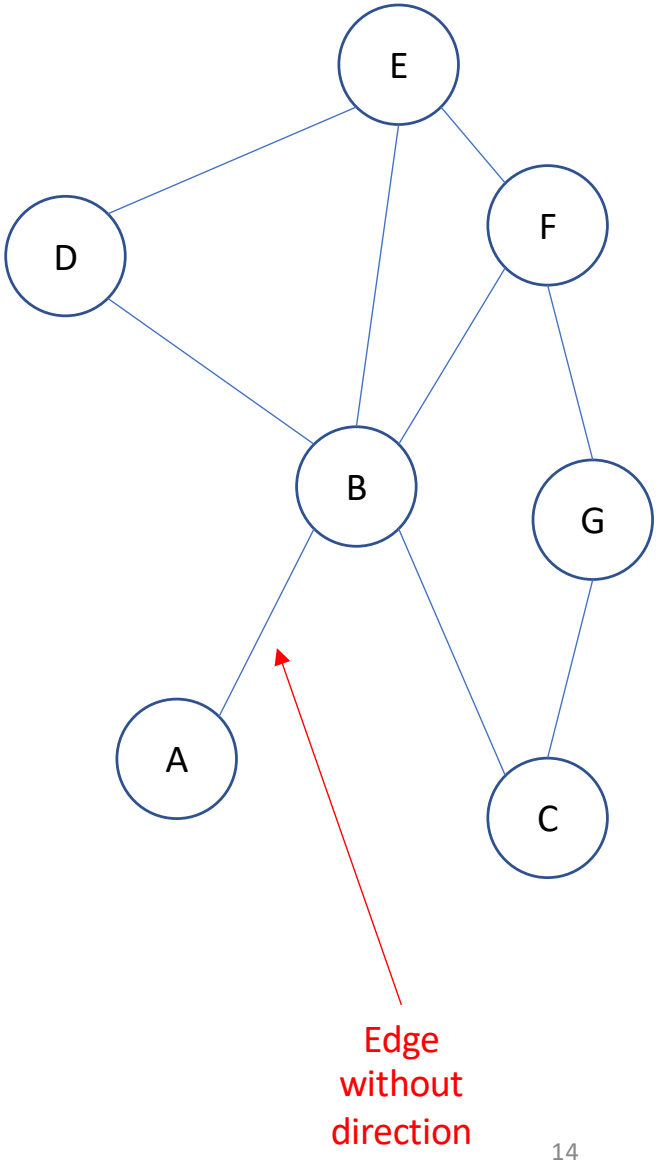
What is a Directed Graph

- Graph: a set of vertices connected by edges
- Path: sequence of vertices and edges from one vertex to another vertex
- Directed graph: when the edges have direction
- Undirected graph: when the edges do not have direction (i.e., bidirectional)



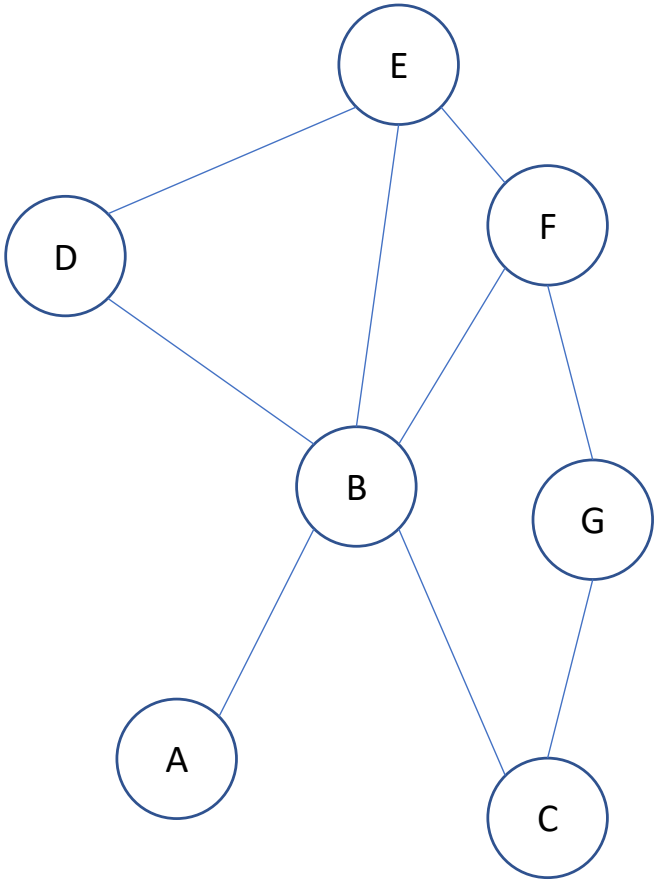
What is an Undirected Graph

- Graph: a set of vertices connected by edges
- Path: sequence of vertices and edges from one vertex to another vertex
- Directed graph: when the edges have direction
- **Undirected graph: when the edges do not have direction (i.e., bidirectional) → your project**

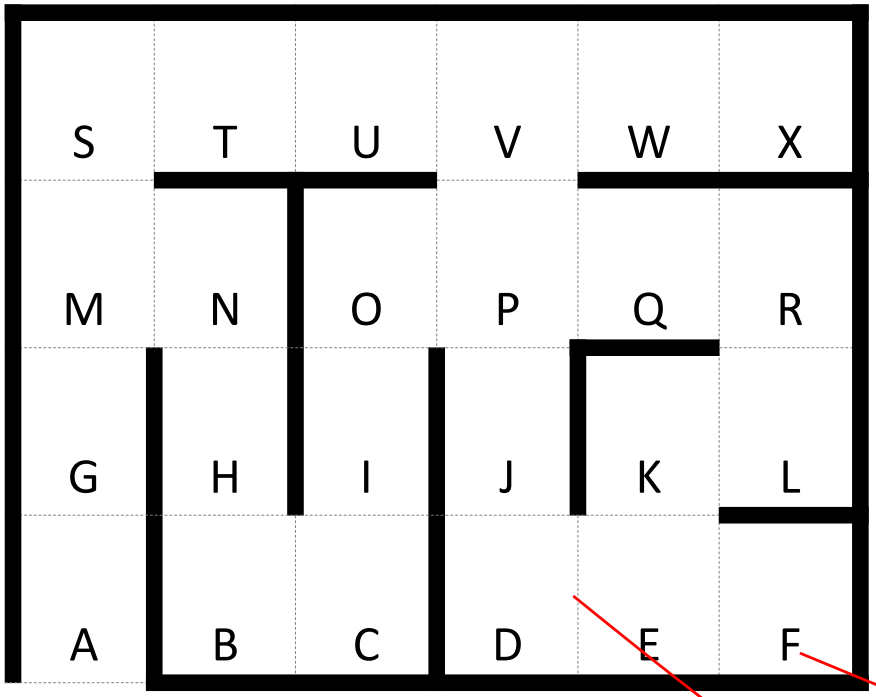


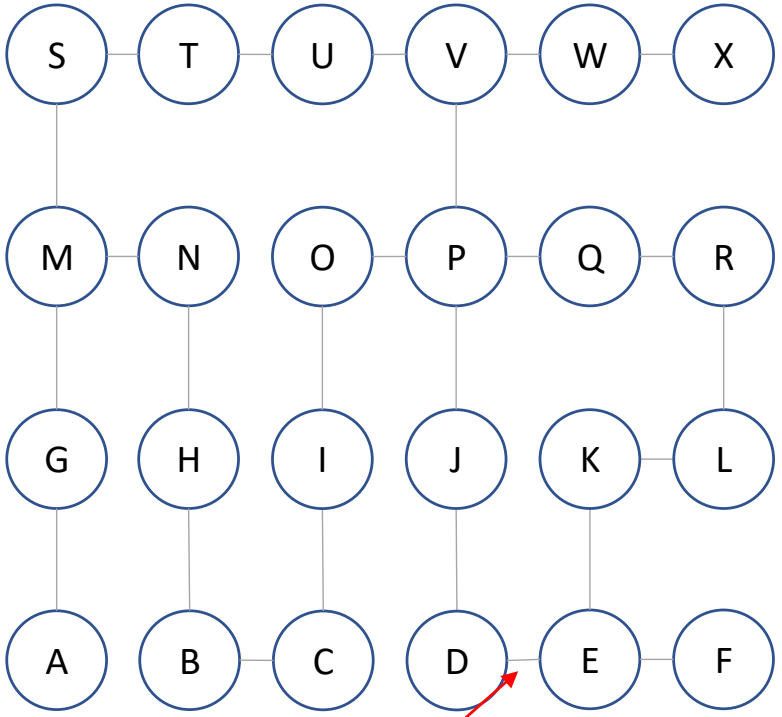
Traversing an Undirected Graph

- Graph: a set of vertices connected by edges
- Path: sequence of vertices and edges from one vertex to another vertex
- Undirected graph: when the edges do not have direction (i.e., bidirectional)
- Traversing (visiting each vertex) an undirected graph :
  - Visiting a vertex: going into a vertex
  - Examining a vertex: when in a particular vertex, visit all adjacent vertices



Graph representation of the space





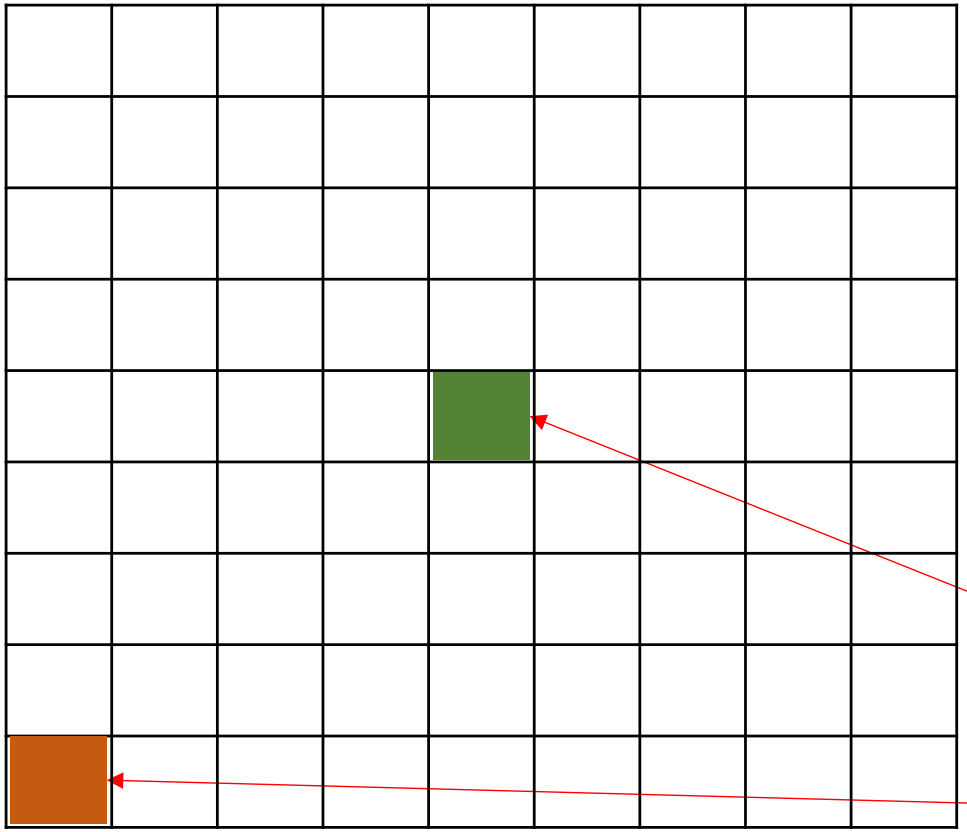
Edge represents a passage.

Vertex represents a cell.

Imagine the robot maneuvering this maze

Initial Knowledge of Robot

The map initially known by the robot (no obstacles)



Assume at first the robot knows the following information:

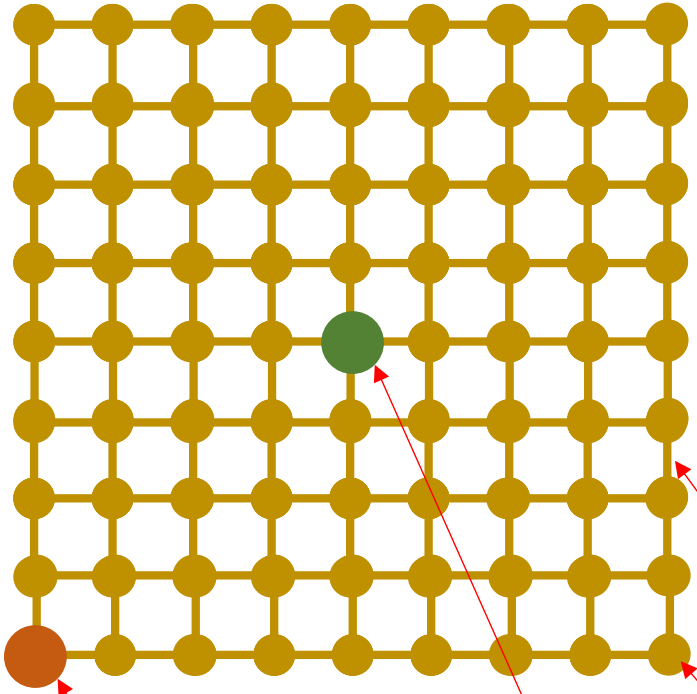
- The size of the maze
- The size of each cell
- The initial cell
- The destination cell

What does the robot **not** know?

- Obstacles (e.g., walls)

Initial Knowledge of Robot

The map initially known by the robot (no obstacles)



initial cell

destination cell

a vertex (cell)

an edge (link between cells)

Assume at first the robot knows the following information:

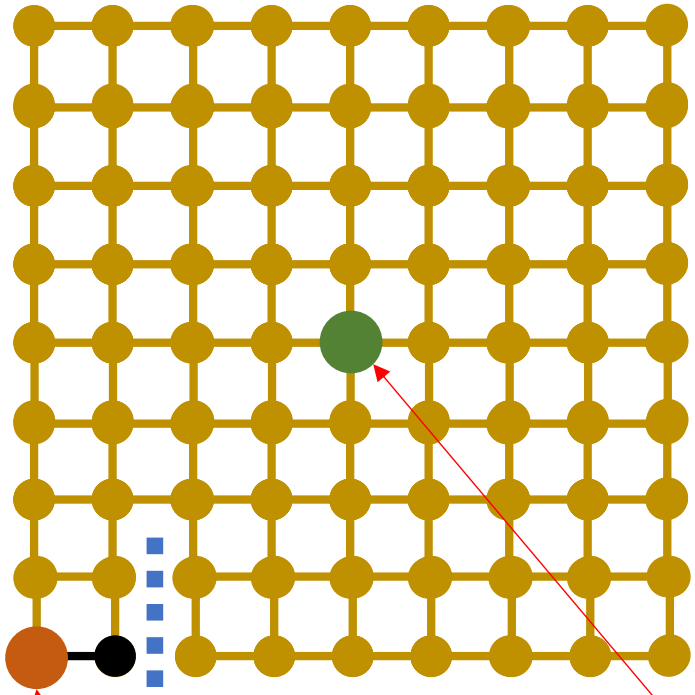
- The size of the maze
- The size of each cell
- The initial cell
- The destination cell

What does the robot **not** know?

- Obstacles (e.g., walls)

Assume the robot prioritise East, South, West, North (clockwise).

As the robot is moving …

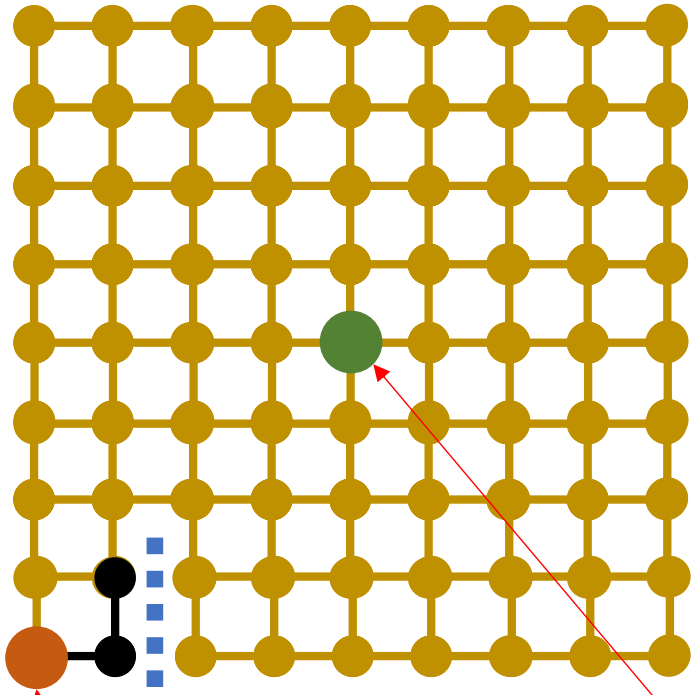


1

initial cell

destination cell

As the robot is moving …

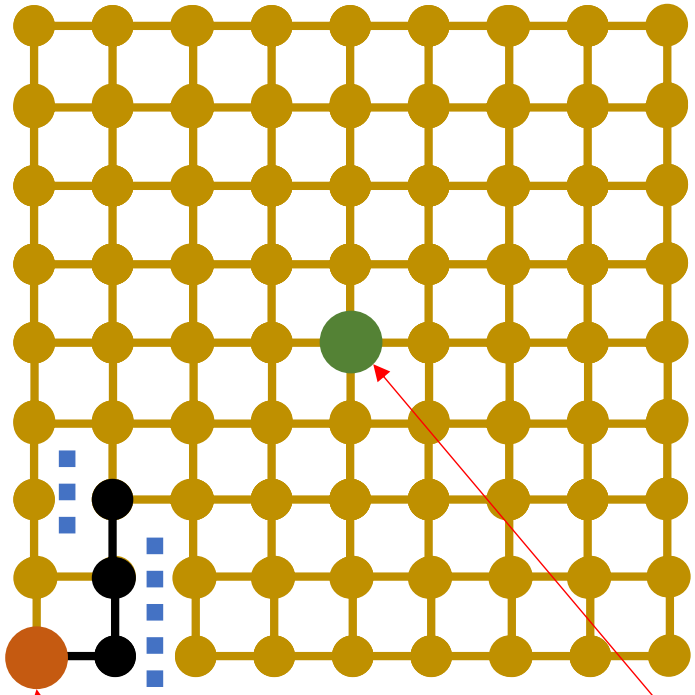


initial cell

destination cell

2

As the robot is moving …

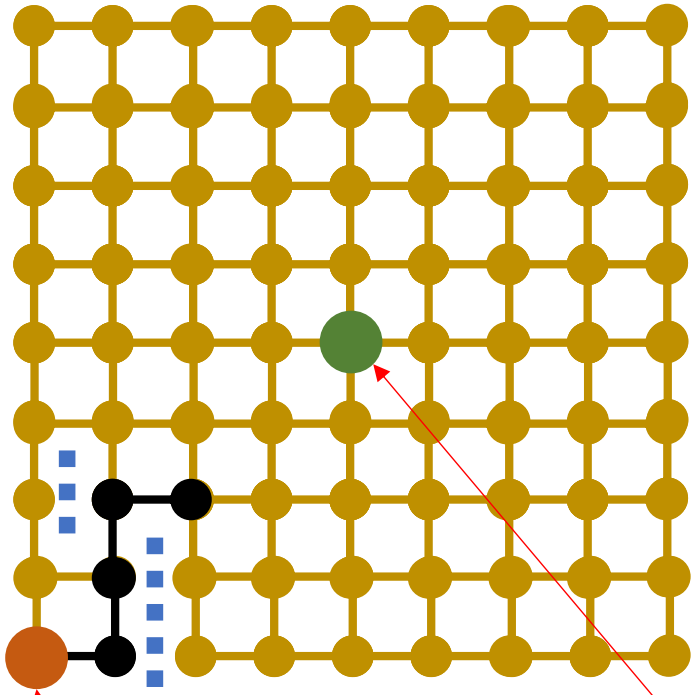


3

initial cell

destination cell

As the robot is moving …

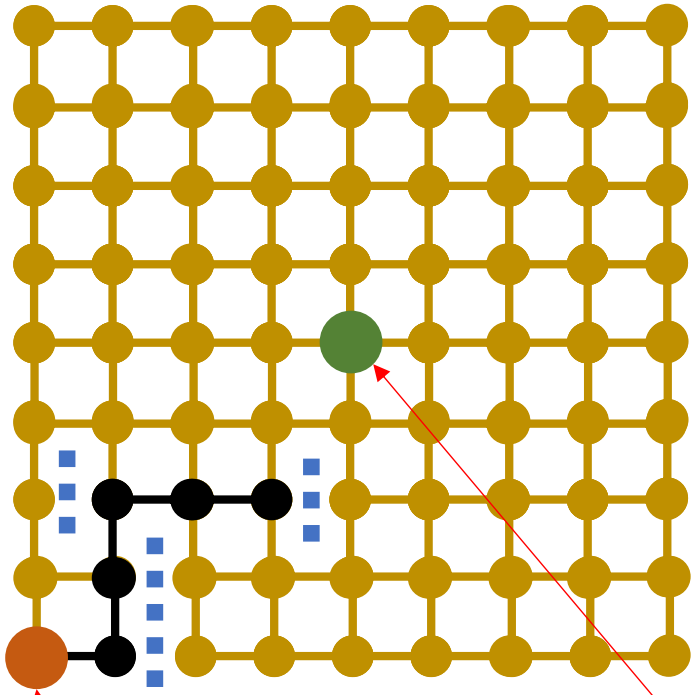


4

initial cell

destination cell

As the robot is moving …

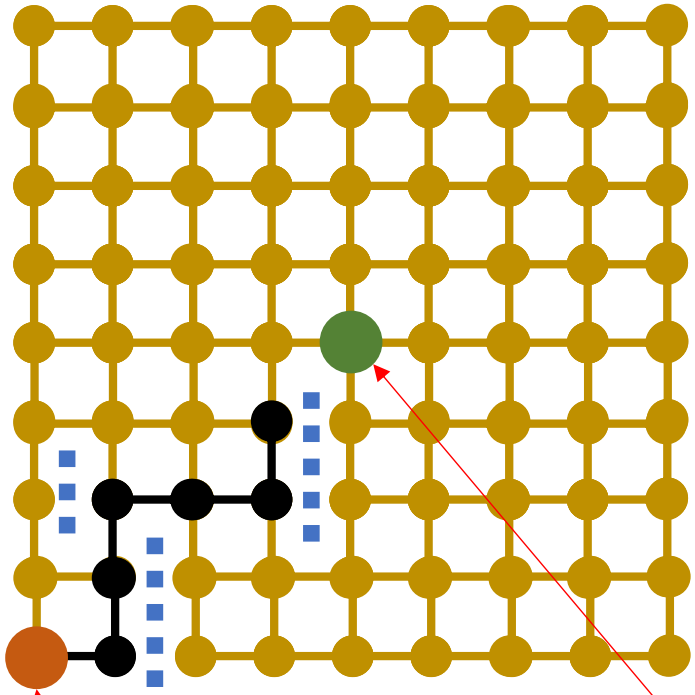


5

initial cell

destination cell

As the robot is moving …

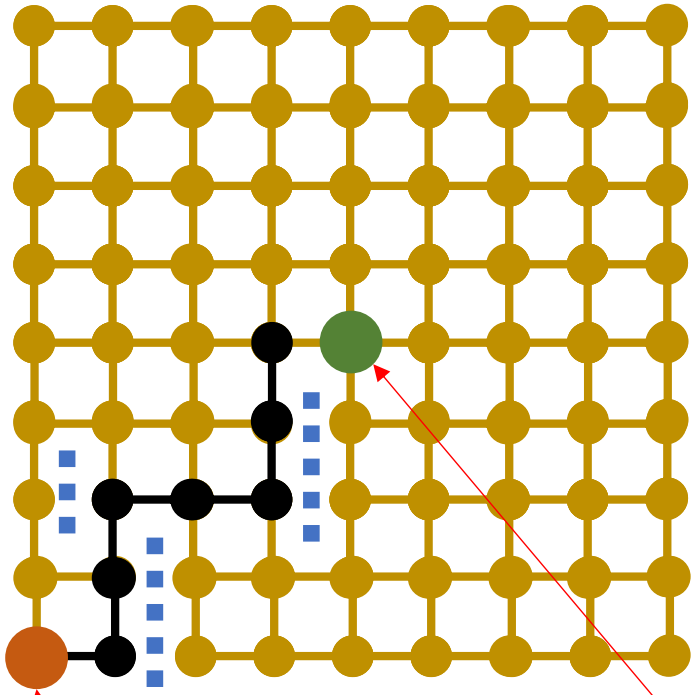


6

initial cell

destination cell

As the robot is moving …

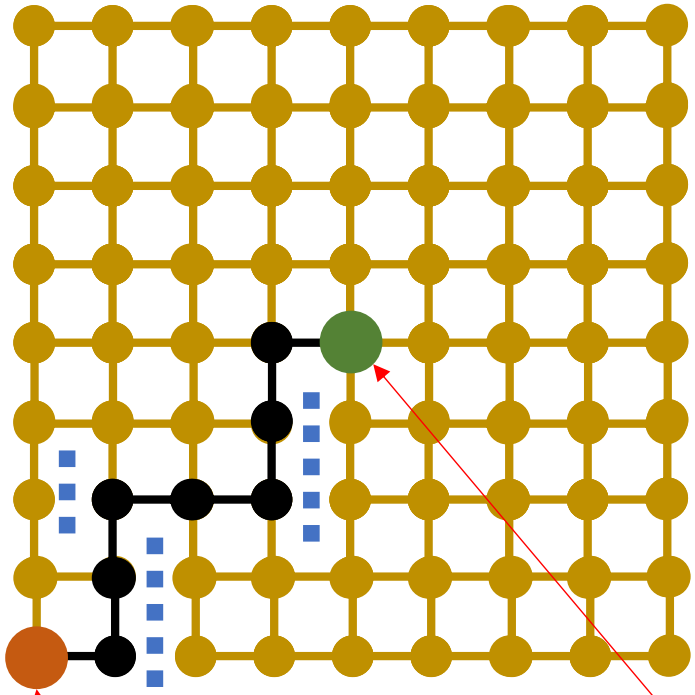


7

initial cell

destination cell

As the robot is moving …

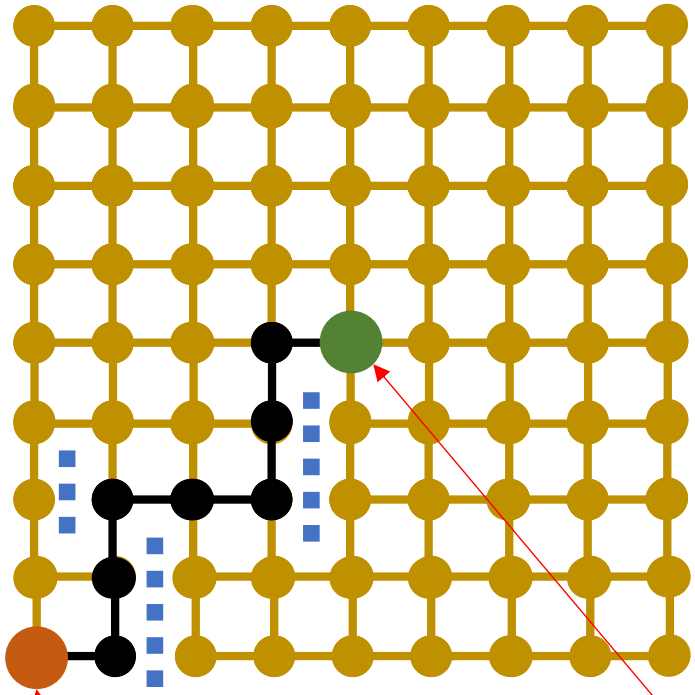


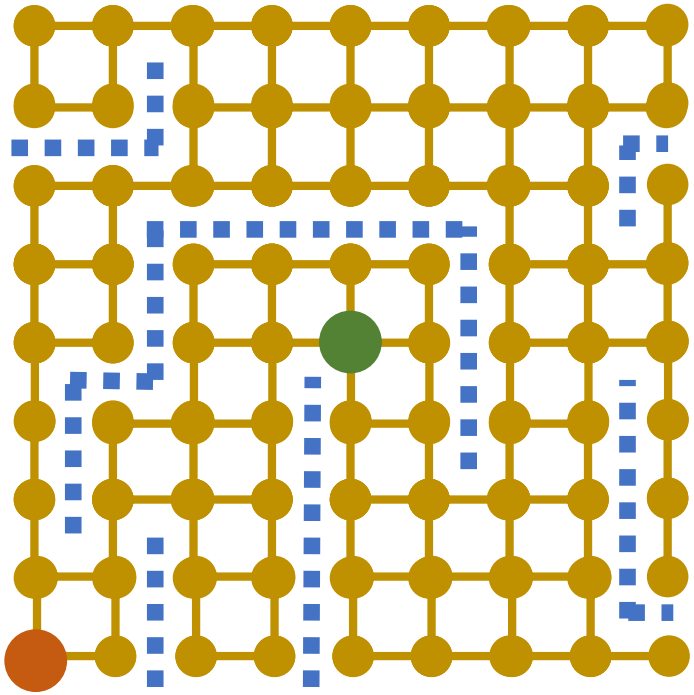
initial cell

destination cell

8

When reaching destination



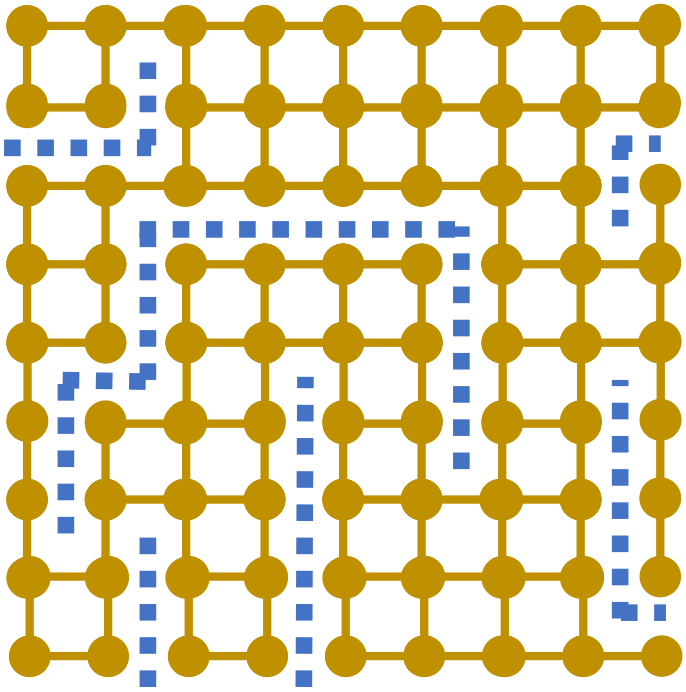


Map that the robot knows when reaching destination

Actual, full map

initial cell

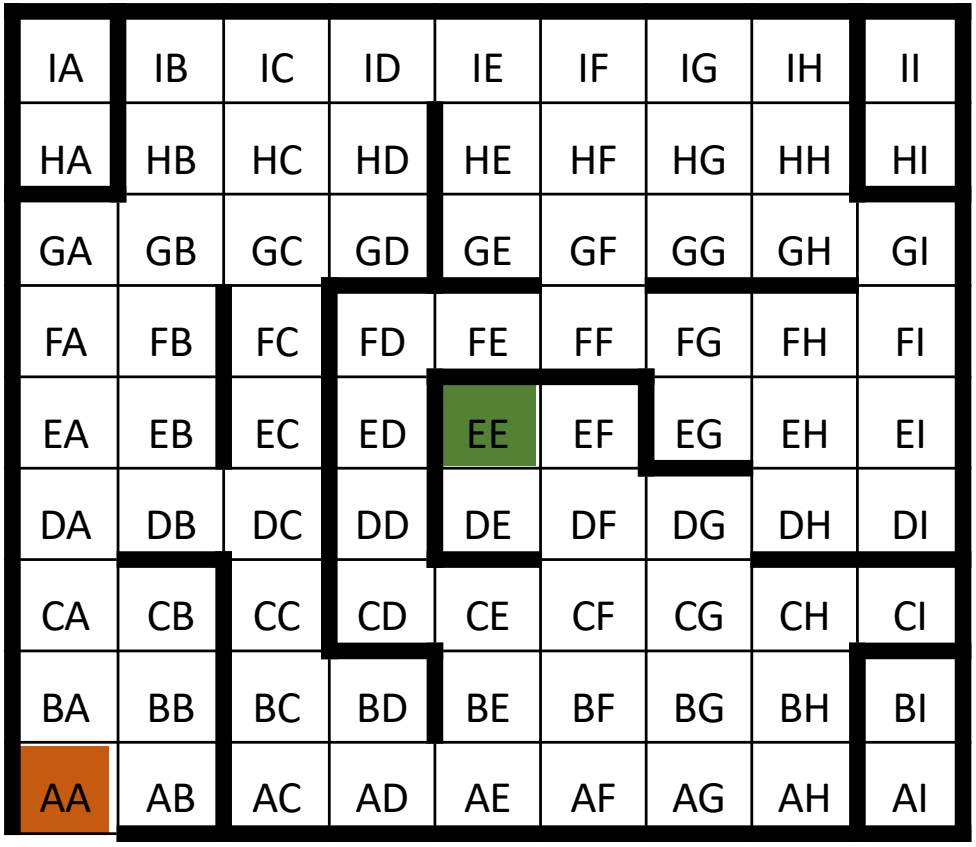
destination cell



The complete map is not necessarily known to the robot to finish the task.

At the end of the navigation, the robot may not know the whole maze, but still able to finish the task.

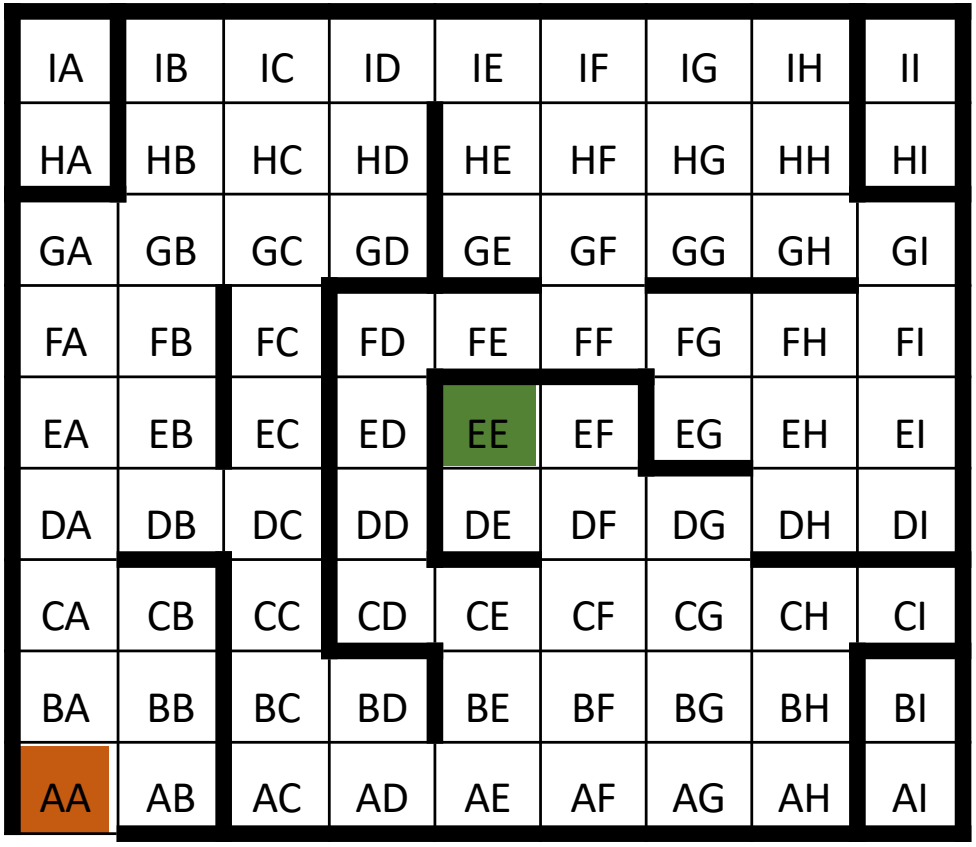
Consider another maze



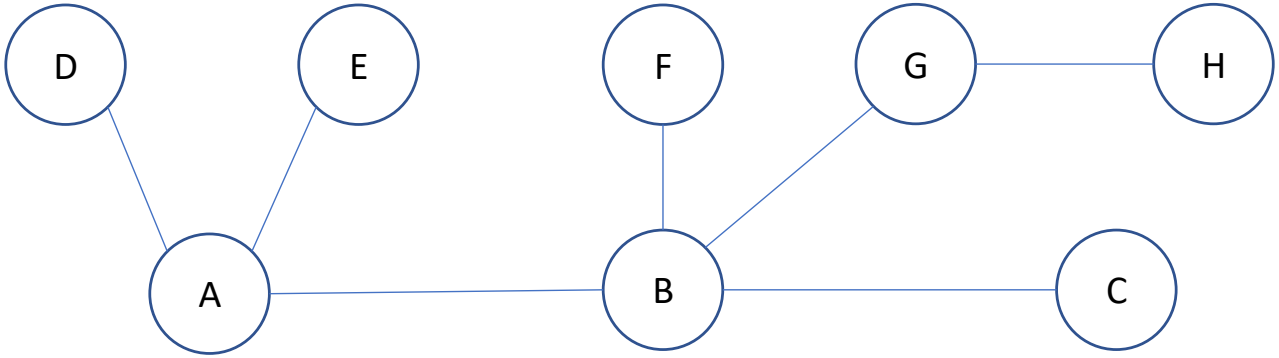
What is the path to destination?

Share your answer.

What path does the robot take?



Consider Traversing* This Graph



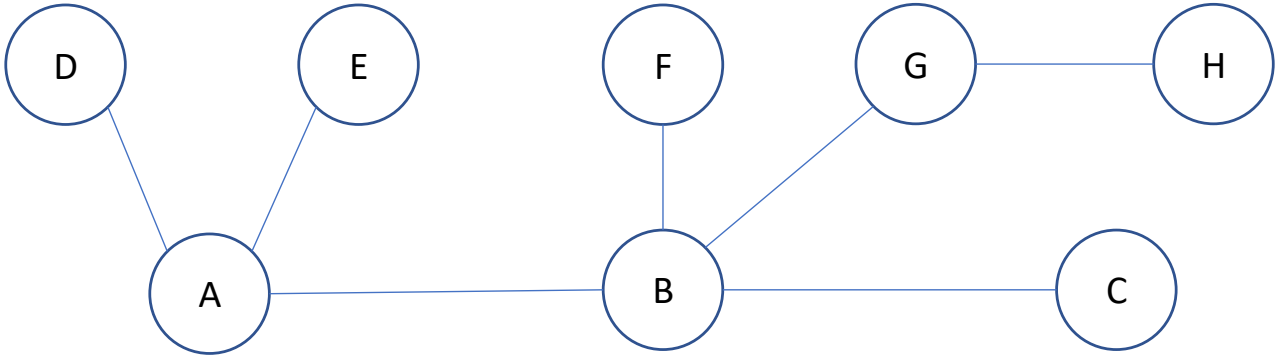
Start at vertex A.

How to traverse the graph? → **Graph Traversal**

*traverse: travel across or through

Depth First Search

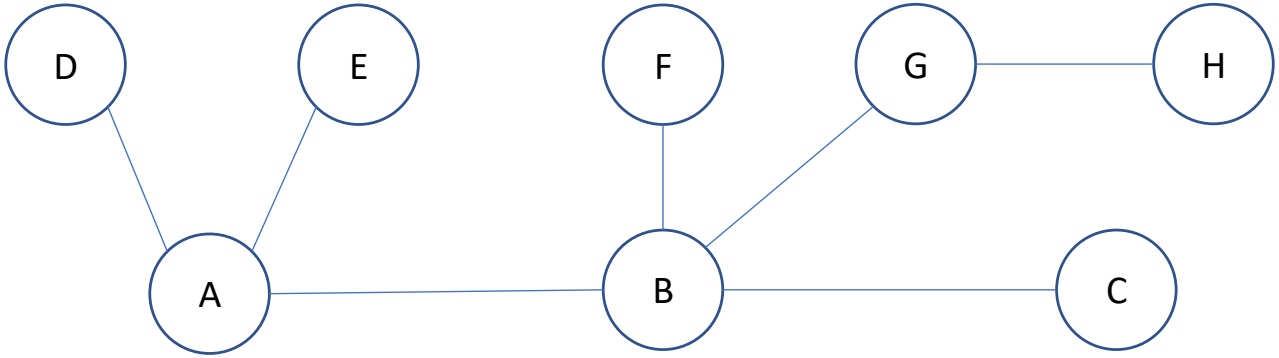
Depth First Search (DFS)



DFS Concept:

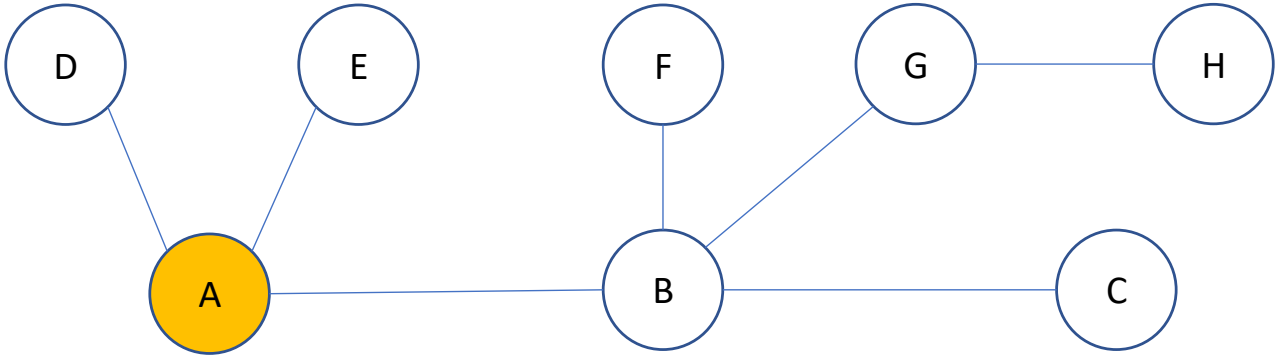
- Start at a vertex and examine as far as possible along a branch.
- After reaching the end of a branch, backtrack to examine the latest branch.
- The priority can be assigned. (In the upcoming example, assume that the priority is **eastward and then move clockwise**.)

Depth First Search (DFS)



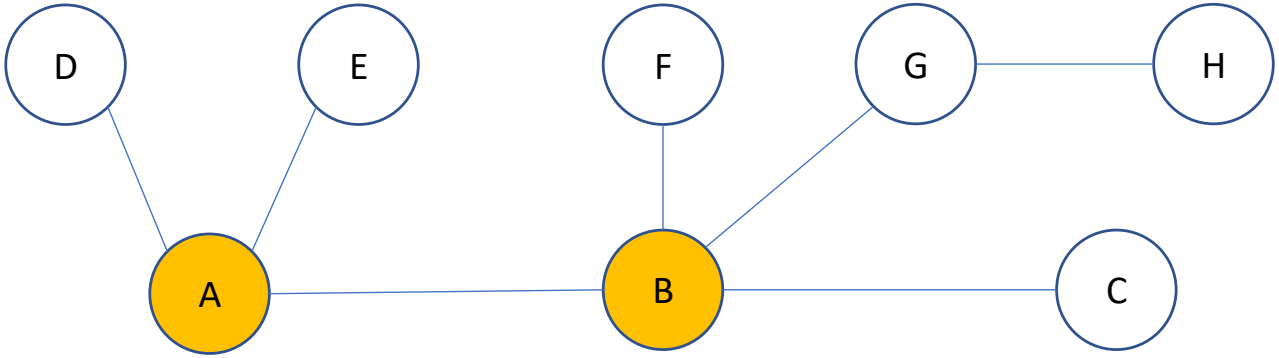
DFS:

Depth First Search (DFS)



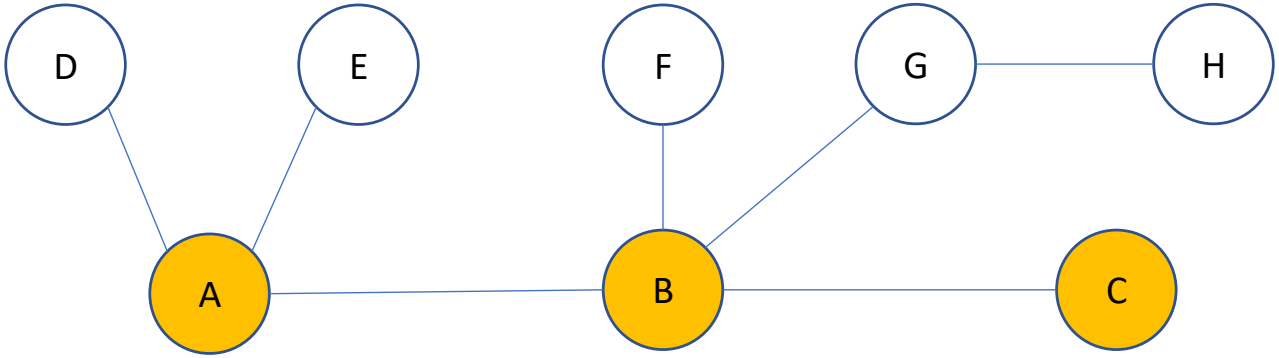
DFS: A

Depth First Search (DFS)



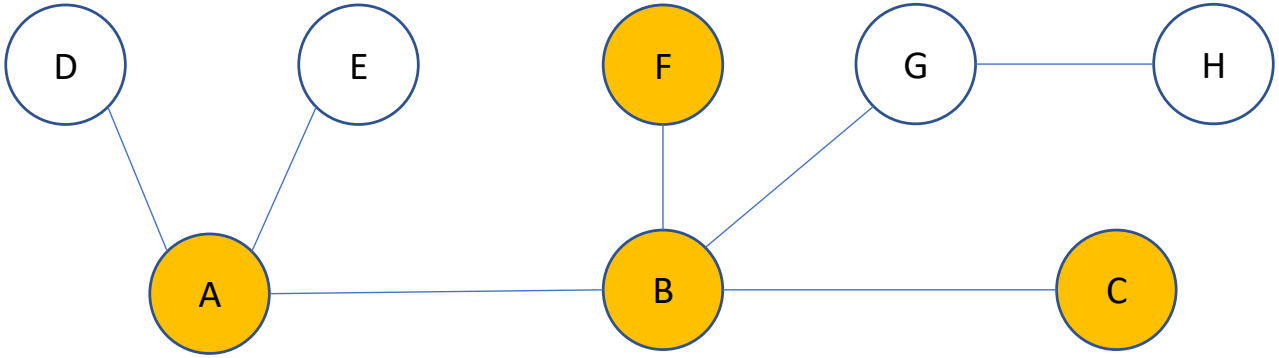
DFS: A, B

Depth First Search (DFS)



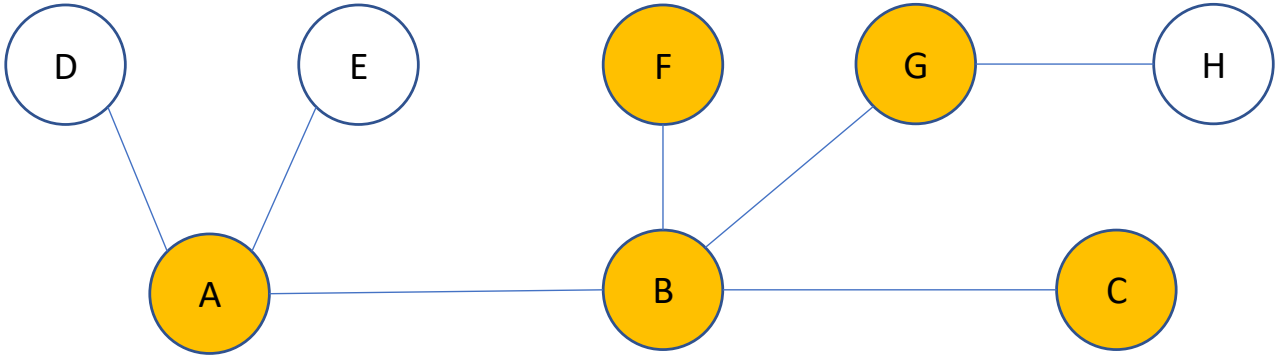
DFS: A, B, C

Depth First Search (DFS)



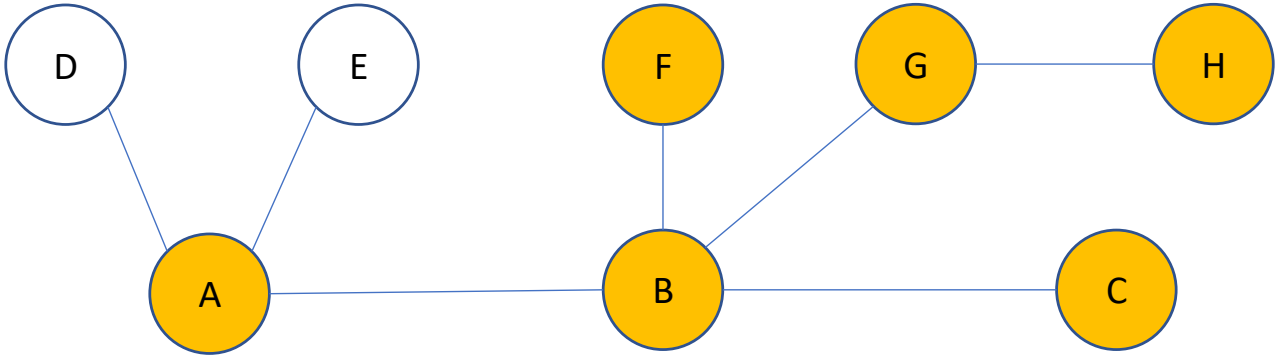
DFS: A, B, C, F

Depth First Search (DFS)



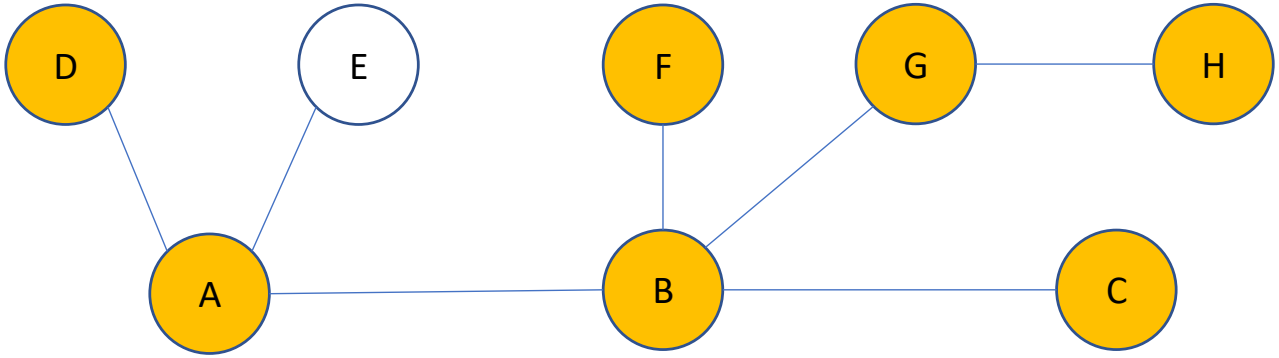
DFS: A, B, C, F, G

Depth First Search (DFS)



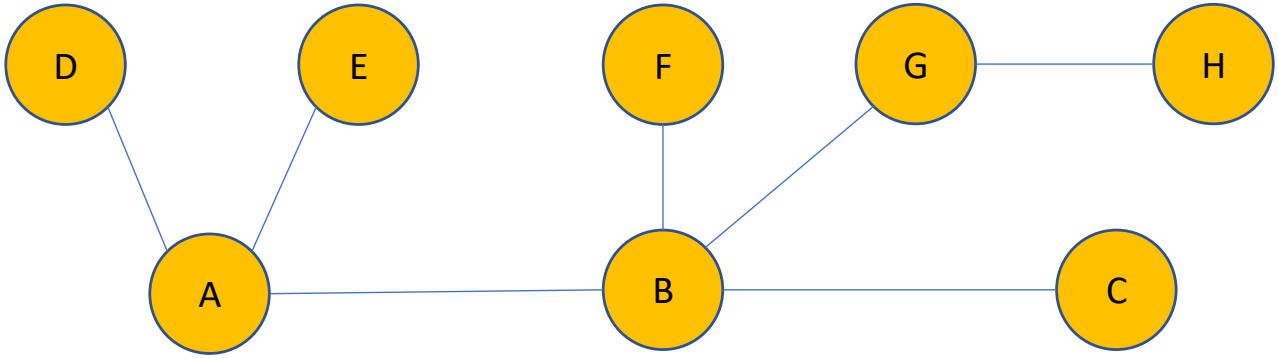
DFS: A, B, C, F, G, H

Depth First Search (DFS)



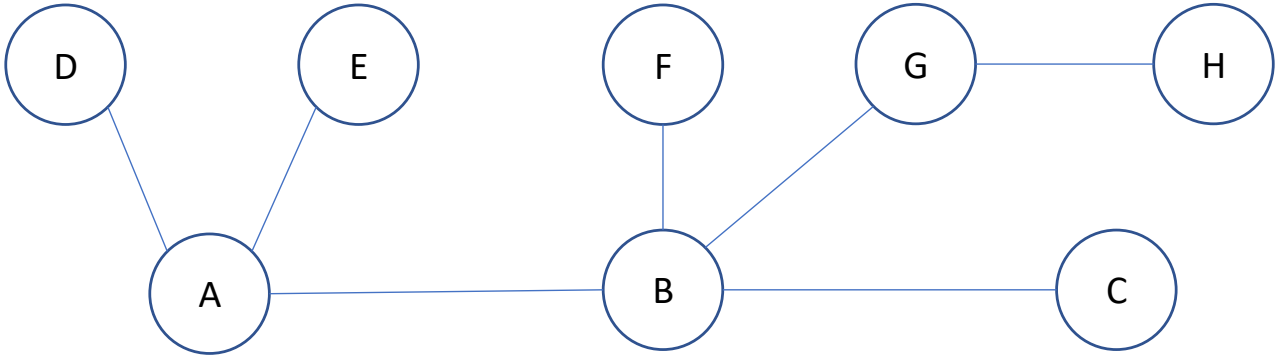
DFS: A, B, C, F, G, H, D

Depth First Search (DFS)



DFS: A, B, C, F, G, H, D, E

Depth First Search (DFS)

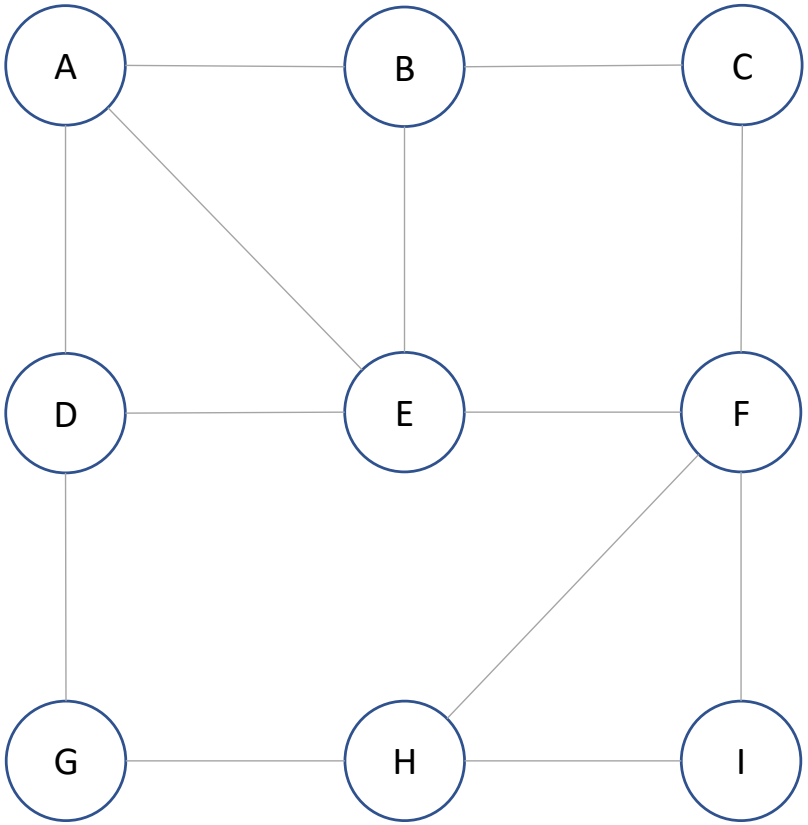


DFS: A, B, C, F, G, H, D, E

Another Example:
Depth First Search for Path Finding

Stop traversing when the destination has been visited.

DFS Algorithm



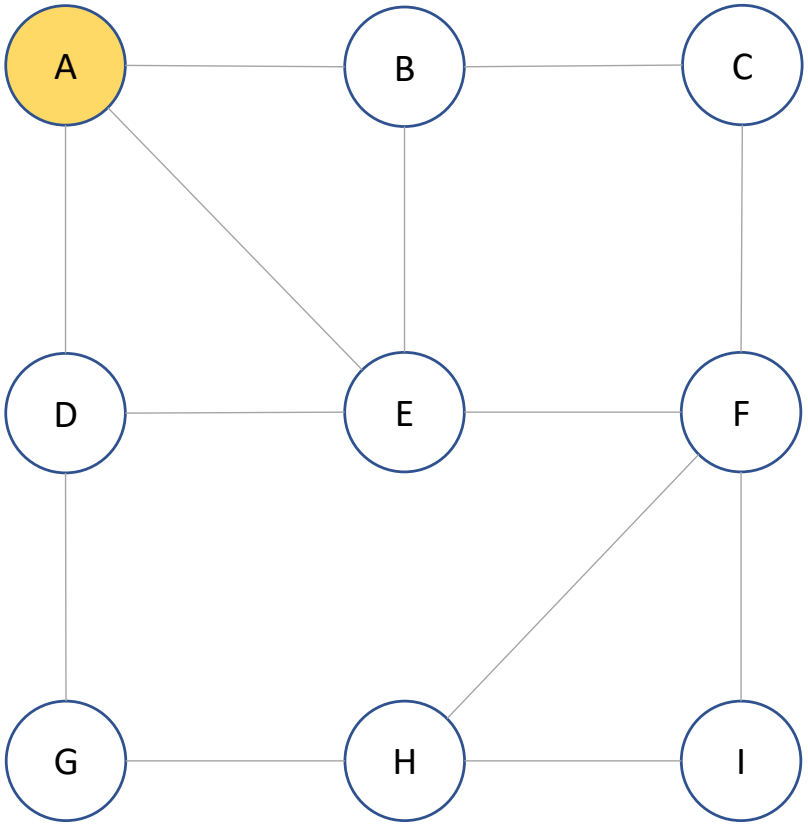
A → F

(suppose we prioritise eastward, and then move clockwise)

DFS path:

_____ (?)

DFS Algorithm



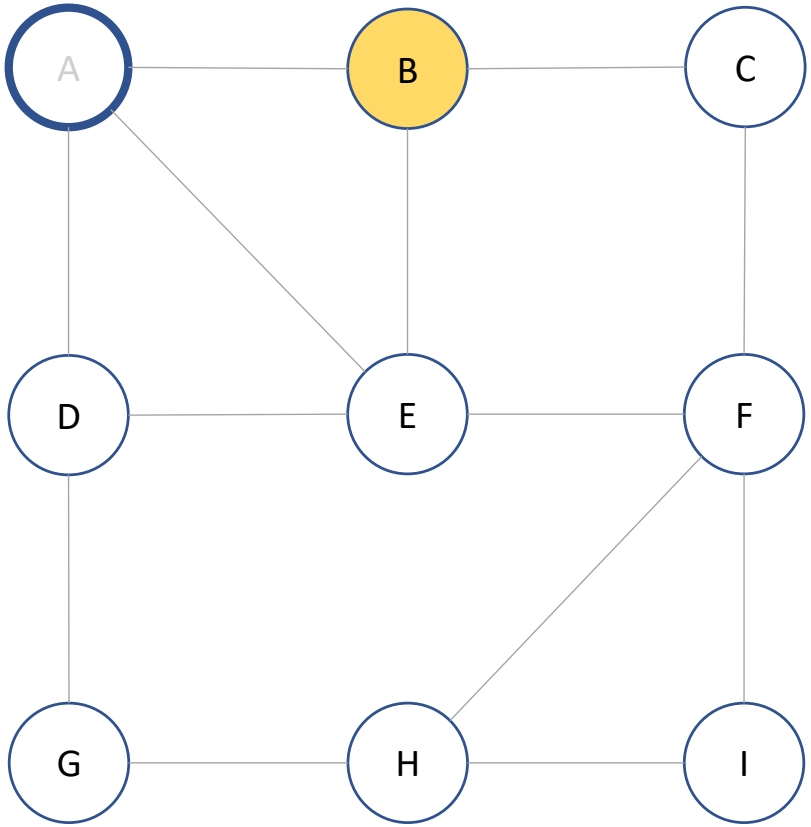
A → F

(suppose we prioritise eastward, and then move clockwise)

DFS path:

_____ (?)

DFS Algorithm



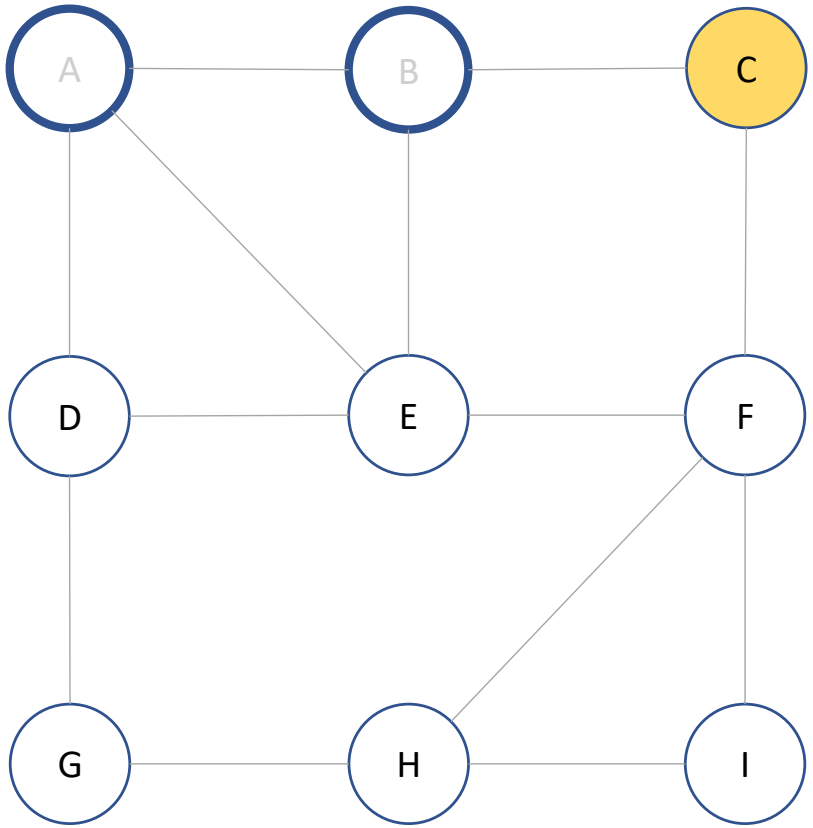
A → F

(suppose we prioritise eastward, and then move clockwise)

DFS path:

_____ (?)

DFS Algorithm



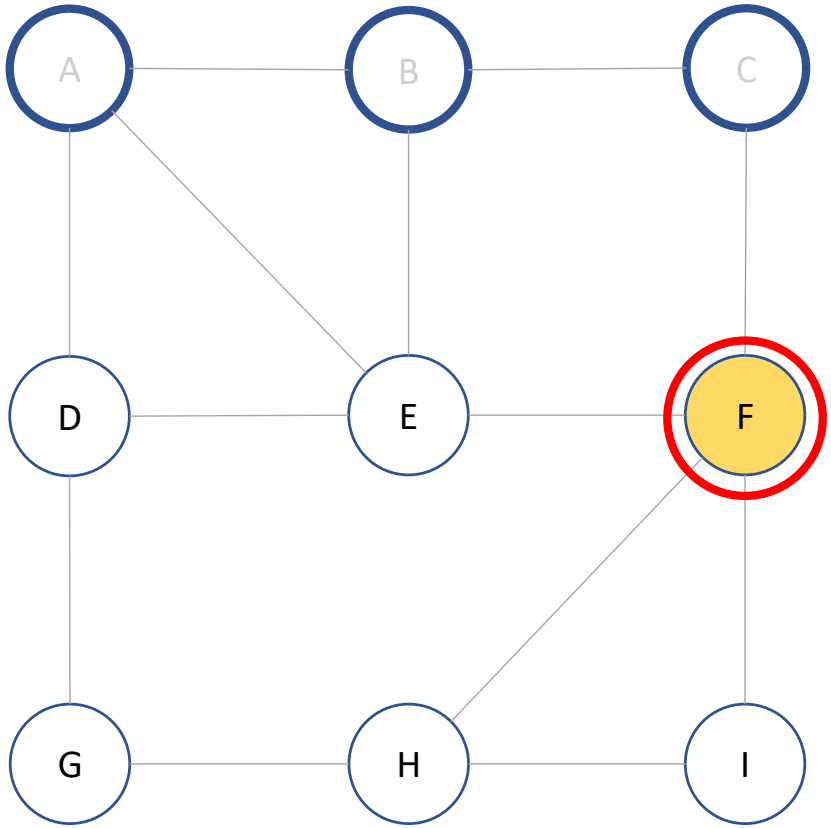
A → F

(suppose we prioritise eastward, and then move clockwise)

DFS path:

_____ (?)

DFS Algorithm



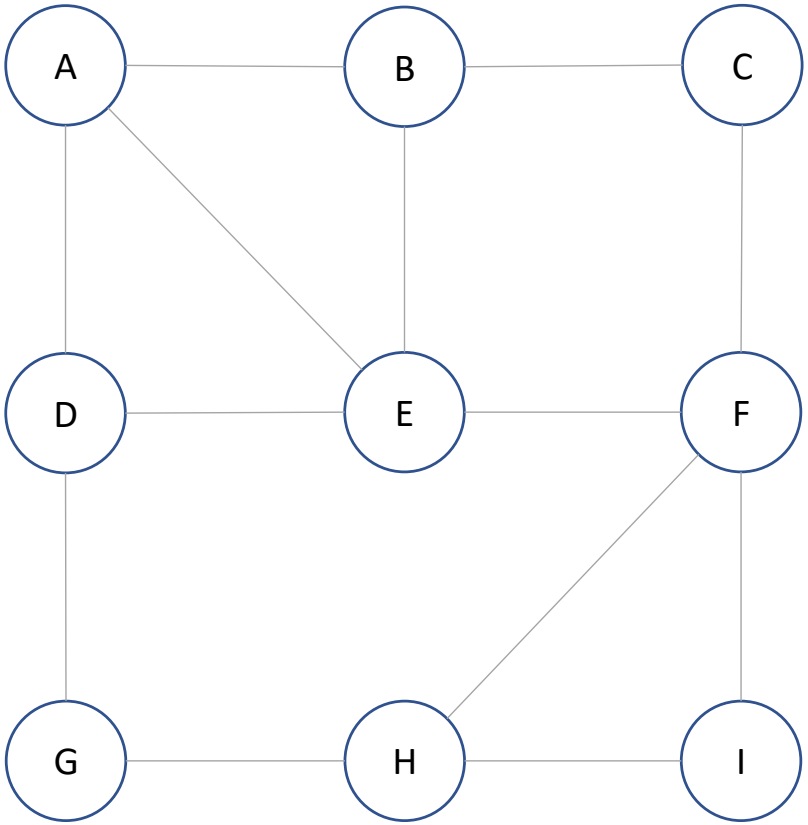
A → F

(suppose we prioritise eastward, and then move clockwise)

DFS path:

_____ (?)

DFS Algorithm



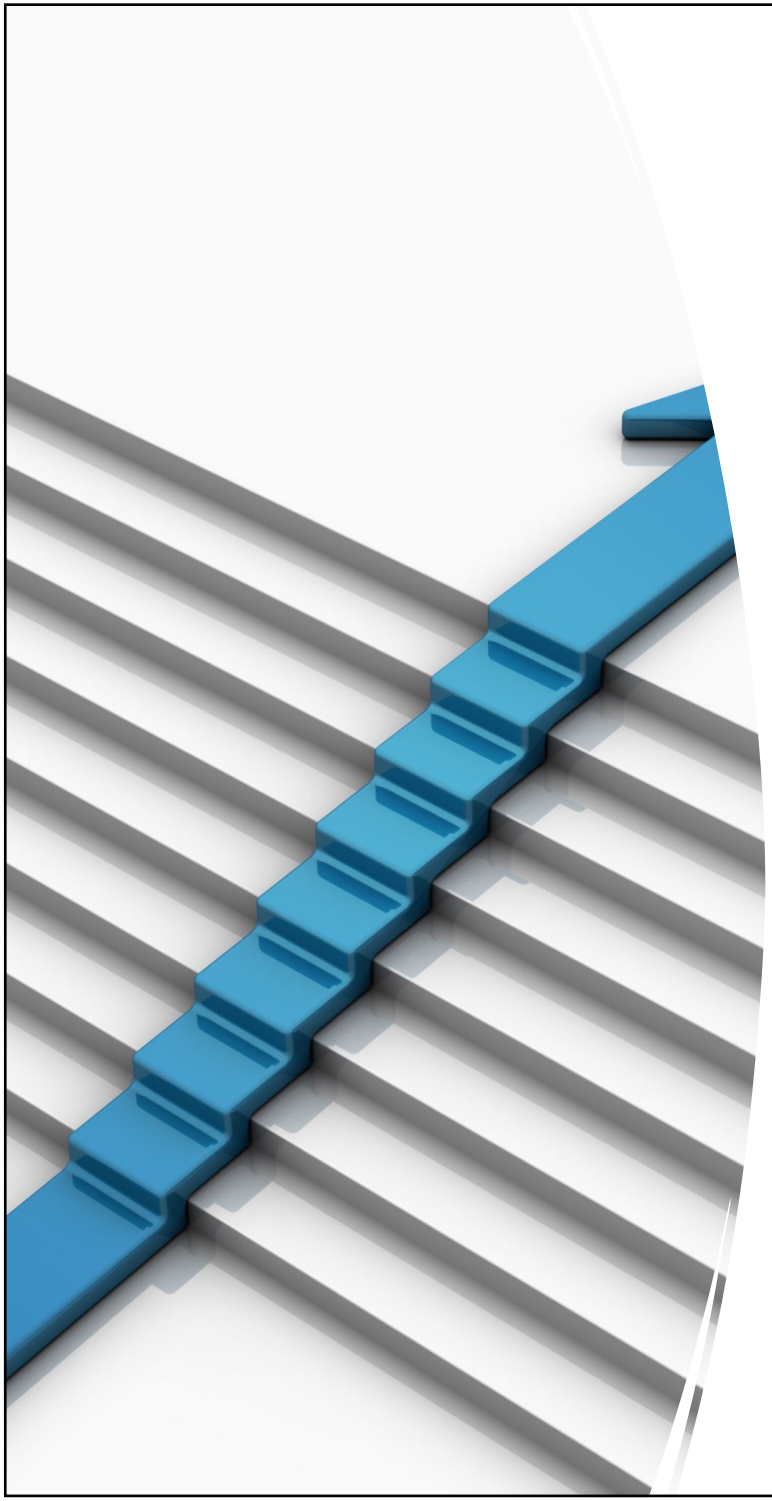
A → F

(suppose we prioritise eastward, and then move clockwise)

DFS path:

A → B → C → F

Steps: 4

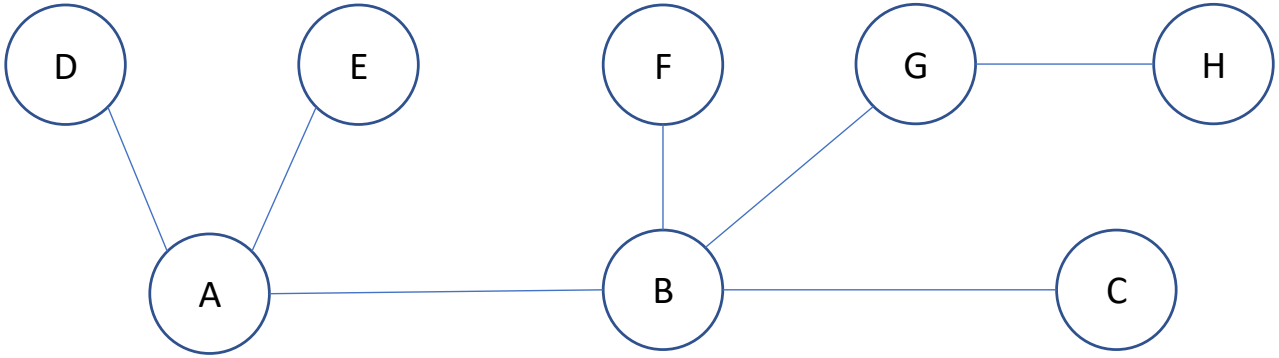


Characteristics

- **Discovery**: DFS is guaranteed to find a path if one exists.
- **Retrieval**: It is easy to retrieve exactly what the path is (the sequence of edges taken) if we find it.
- **Optimality**: DFS does not necessarily find the best path in any sense.

Breadth First Search

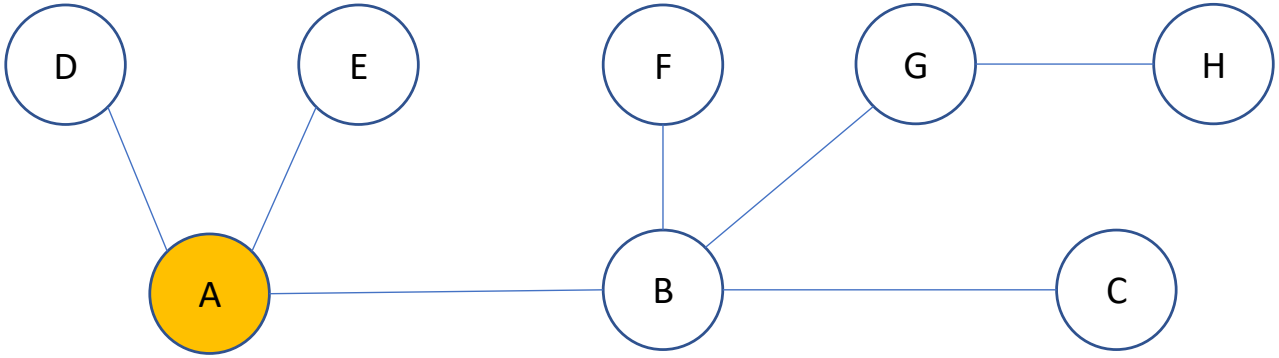
Breadth First Search (BFS)



BFS Concept:

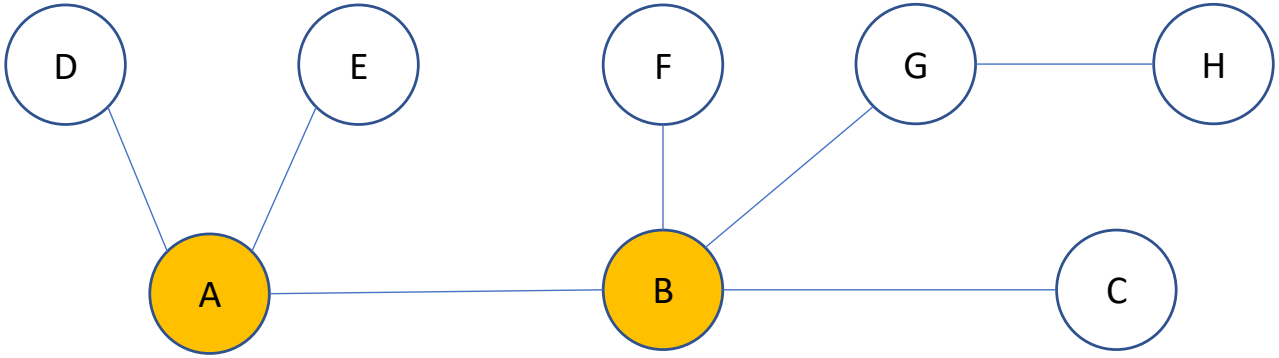
- Start at a vertex and examine all adjacent vertices.
- After all vertices have been examined, visit the next vertex and repeat examination.
- The priority can be assigned. (In the upcoming example, assume that the priority is eastward and then move clockwise.)

Breadth First Search (DFS)



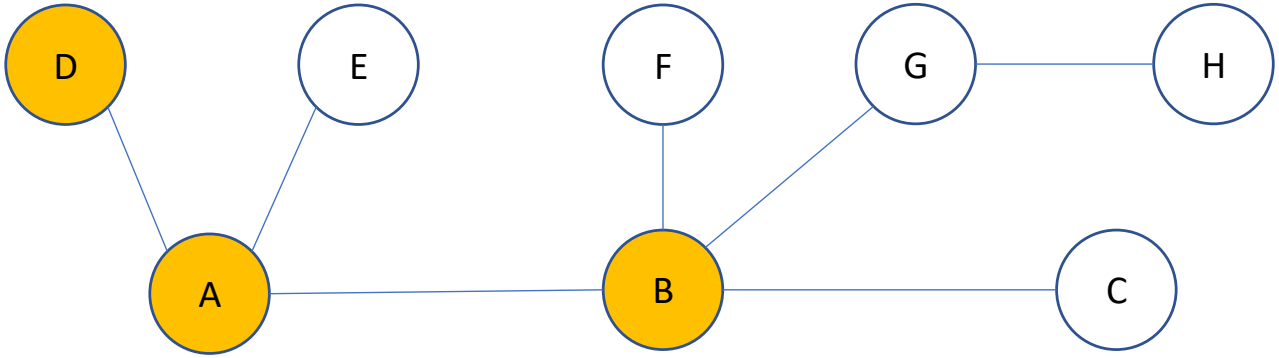
BFS: A

Breadth First Search (DFS)



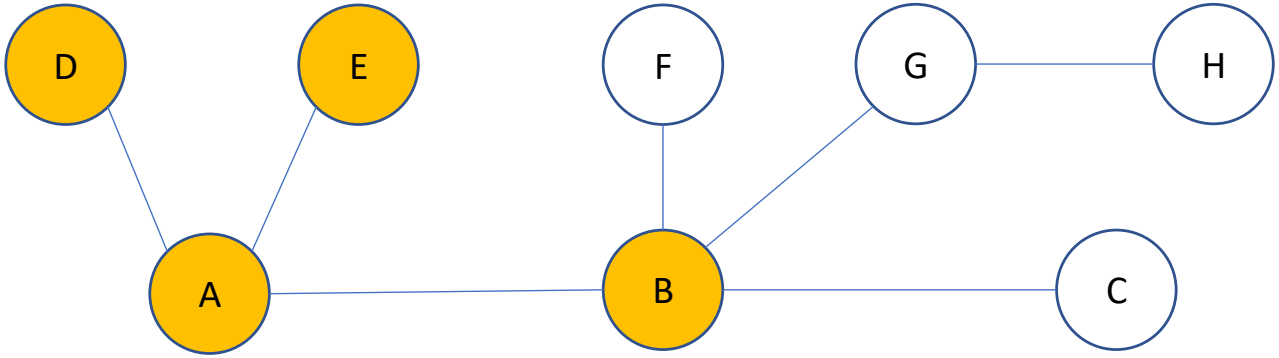
BFS: A, B

Breadth First Search (DFS)



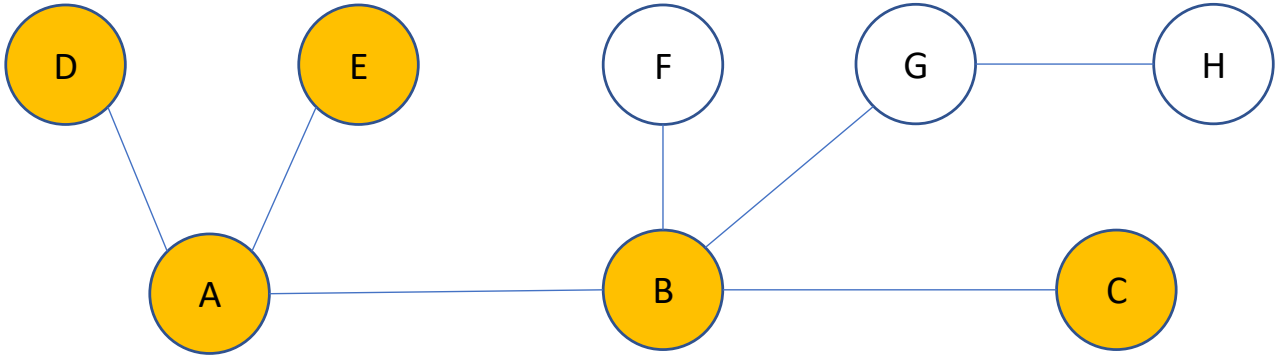
BFS: A, B, D

Breadth First Search (DFS)



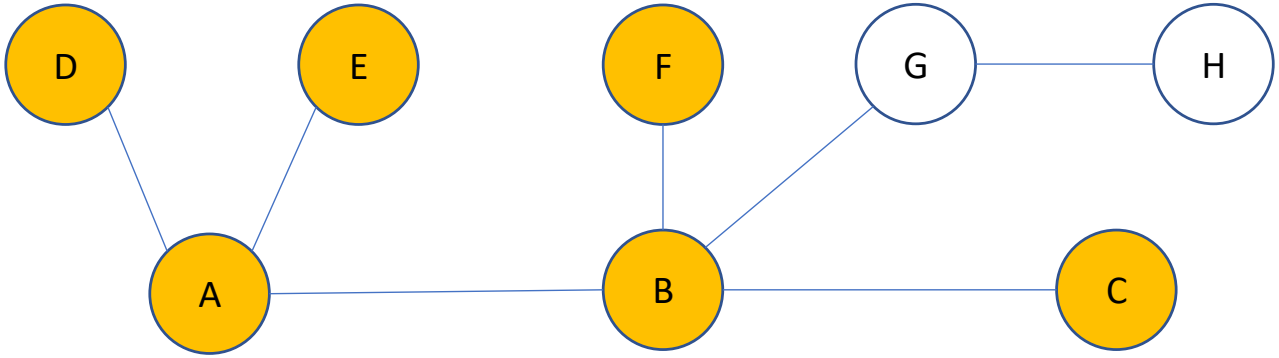
BFS: A, B, D, E

Breadth First Search (DFS)



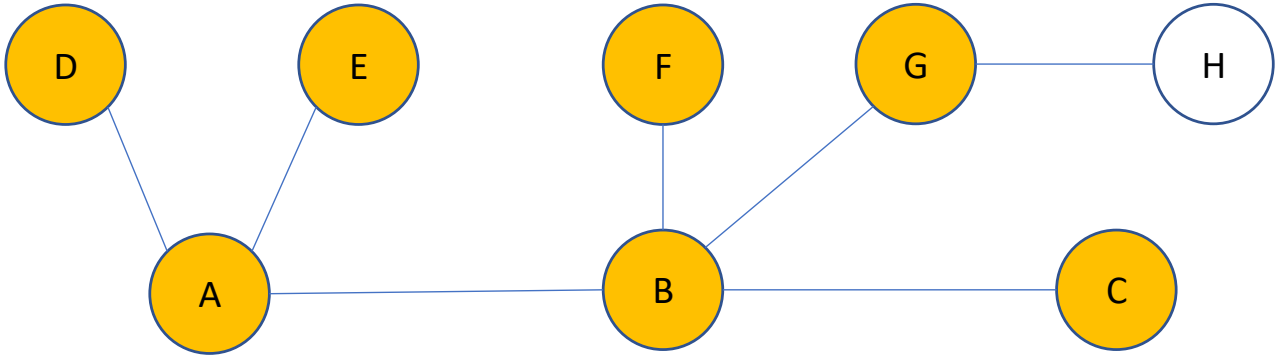
BFS: A, B, D, E, C

Breadth First Search (DFS)



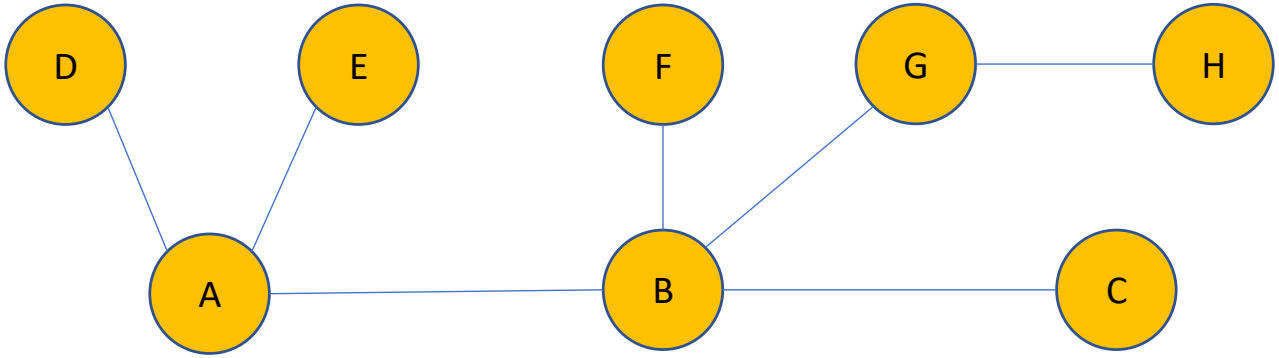
BFS: A, B, D, E, C, F

Breadth First Search (DFS)



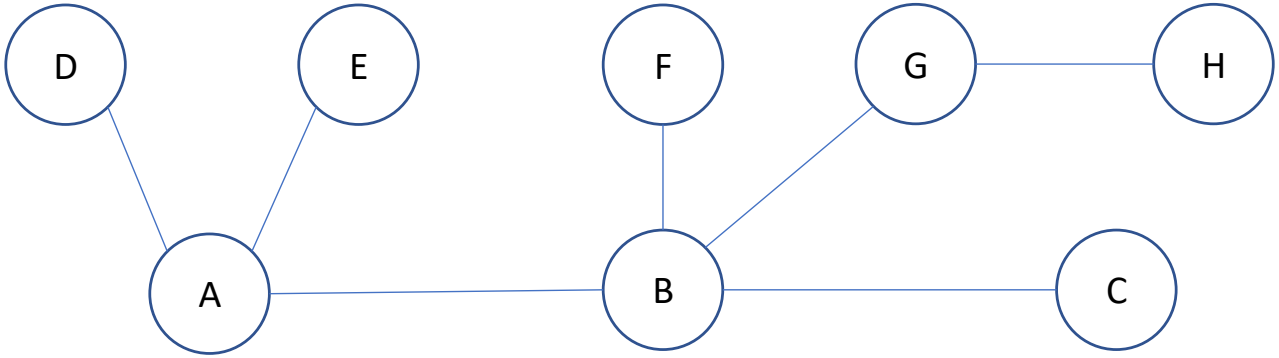
BFS: A, B, D, E, C, F, G

Breadth First Search (DFS)



BFS: A, B, D, E, C, F, G, H

Breadth First Search (DFS)

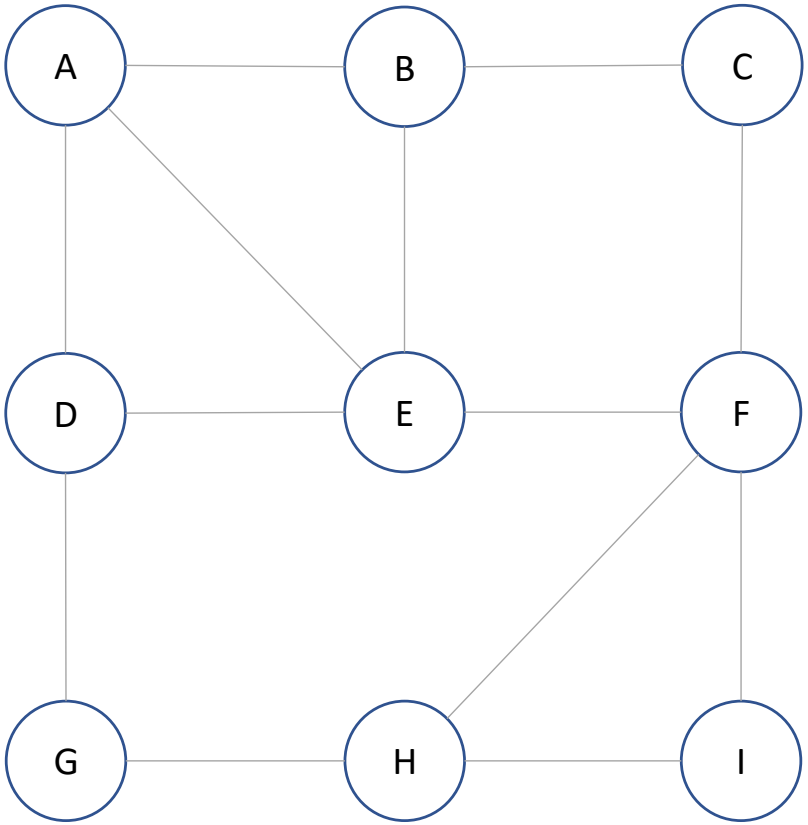


BFS: A, B, D, E, C, F, G, H

Another Example:
Breadth First Search for Path Finding

Stop traversing when the destination has been visited.

BFS Algorithm



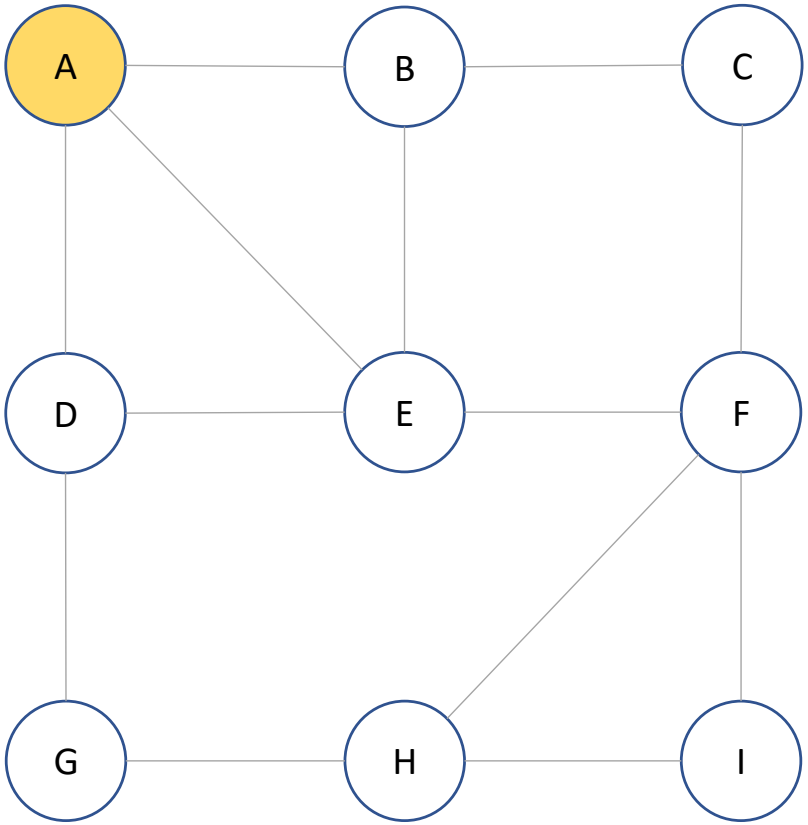
A → F

(suppose we prioritise eastward, and then move clockwise)

BFS path:

_____ (?)

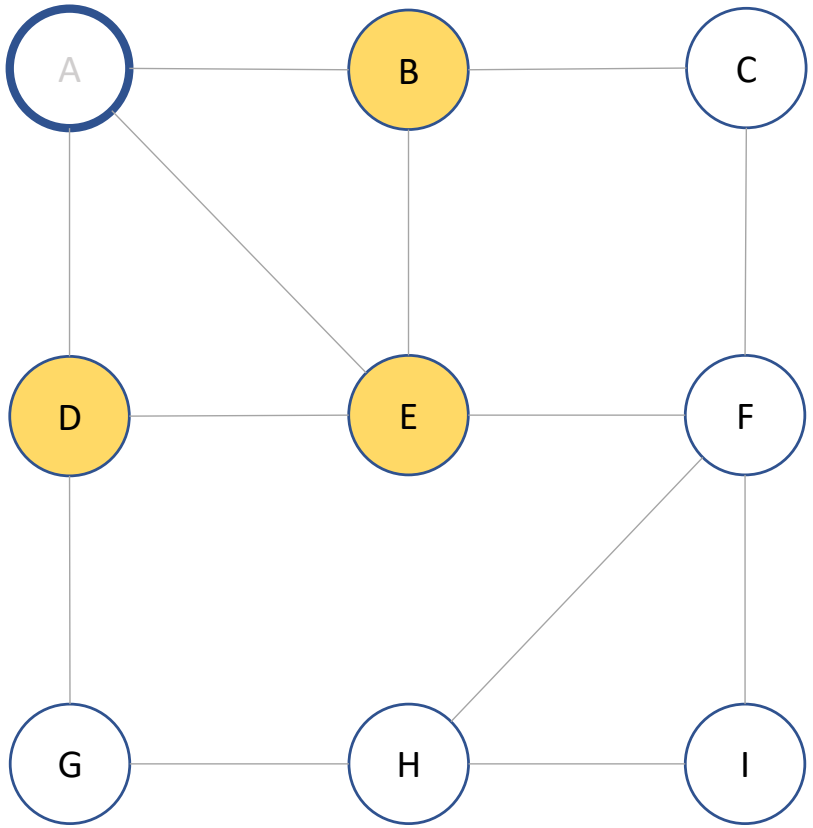
BFS Algorithm



A → F

(suppose we prioritise eastward, and then move clockwise)

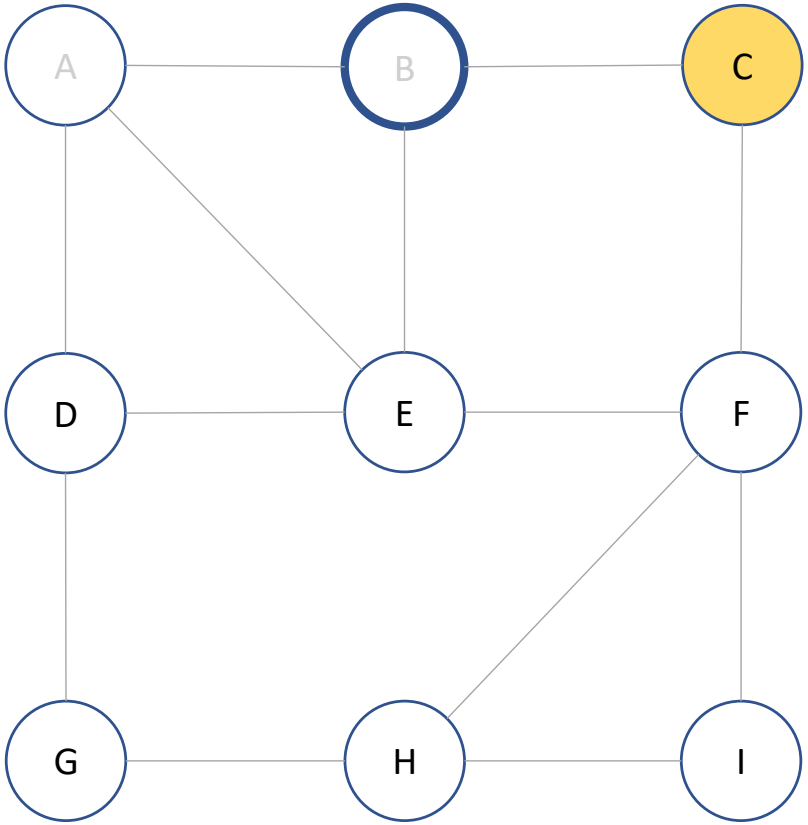
BFS Algorithm



A → F

(suppose we prioritise eastward, and then move clockwise)

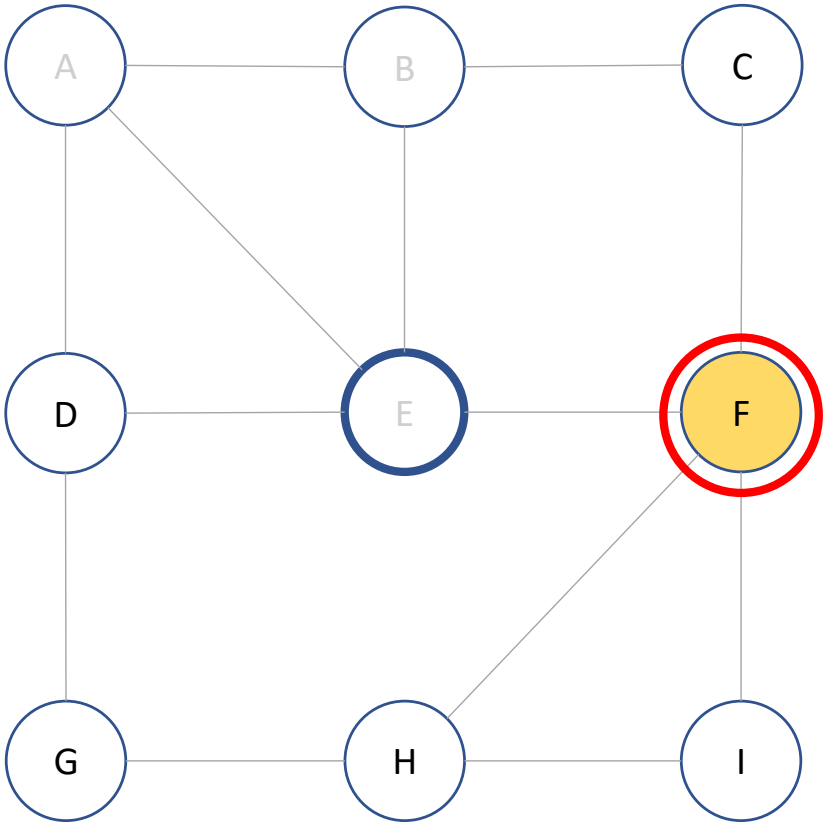
BFS Algorithm



A → F

(suppose we prioritise eastward, and then move clockwise)

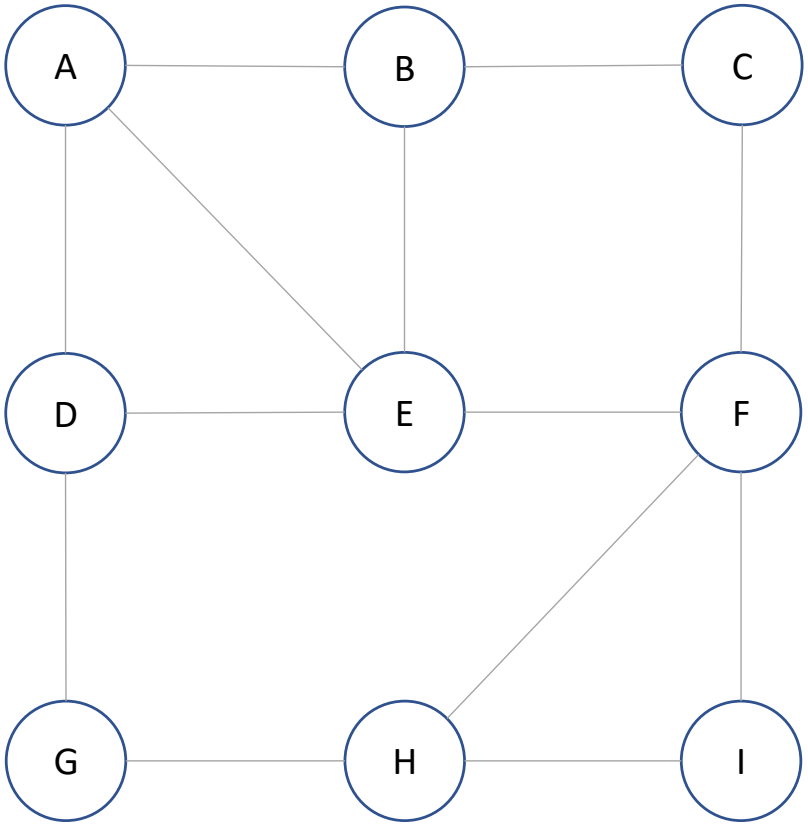
BFS Algorithm



A → F

(suppose we prioritise eastward, and then move clockwise)

BFS Algorithm



A → F

A → B

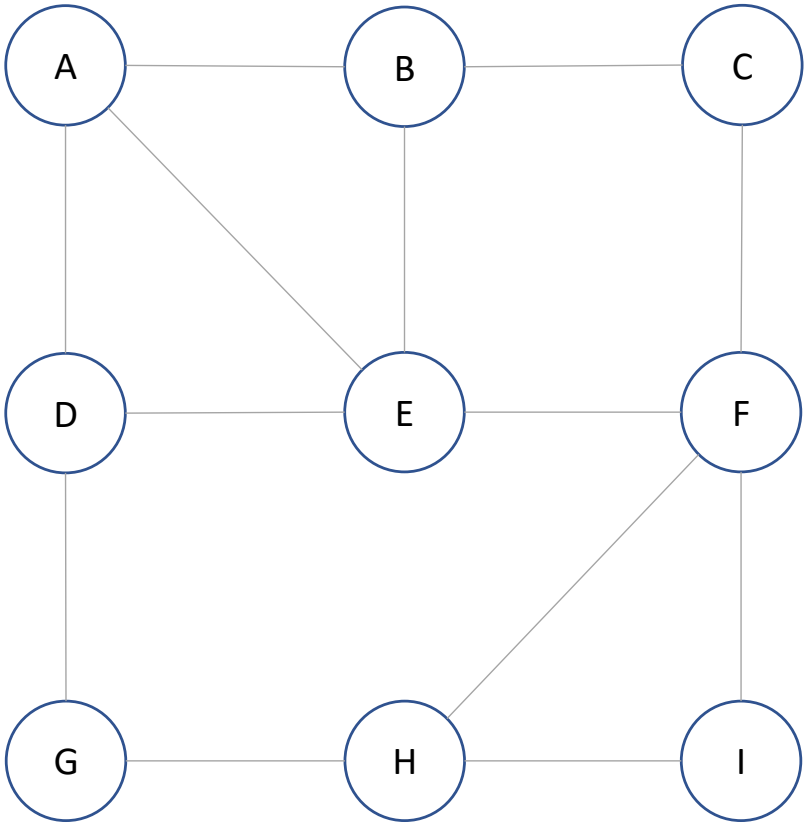
A → E

A → D

A → B → C

A → E → F (end)

BFS Algorithm



A → F

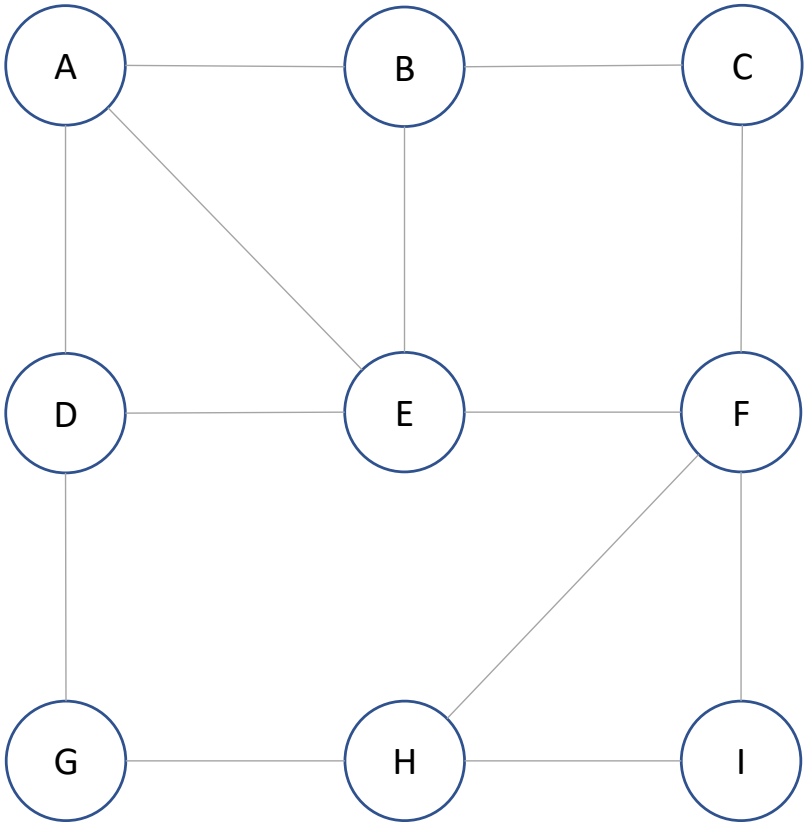
(A – Z sequence)

BFS path:

A → E → F

Steps: 3

BFS Algorithm



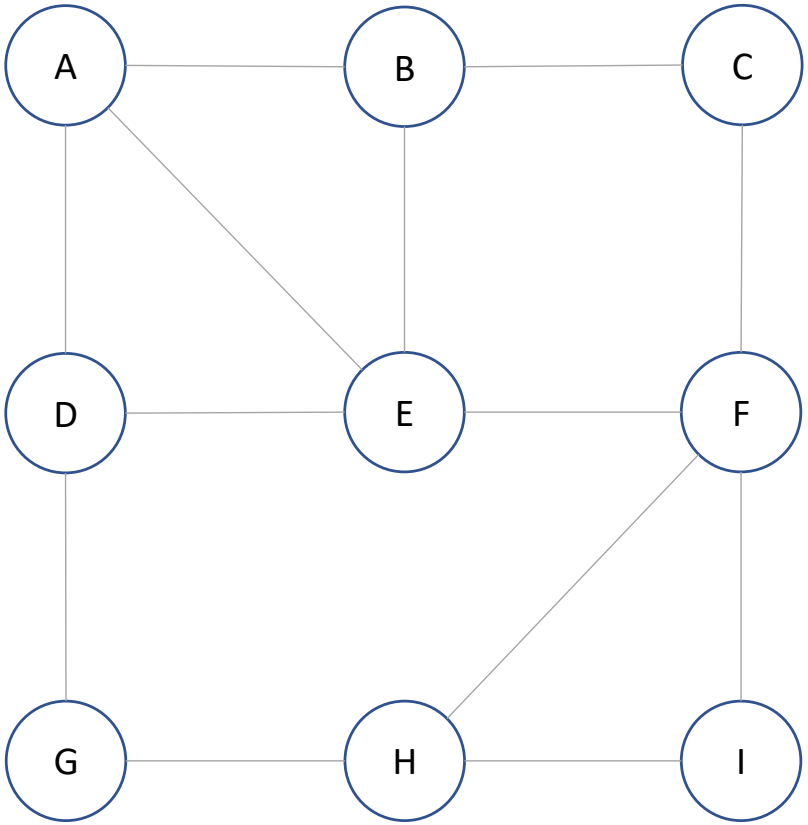
Find the shortest path from A to F

Vertex	Current vertex	Previous vertex

Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

BFS Algorithm



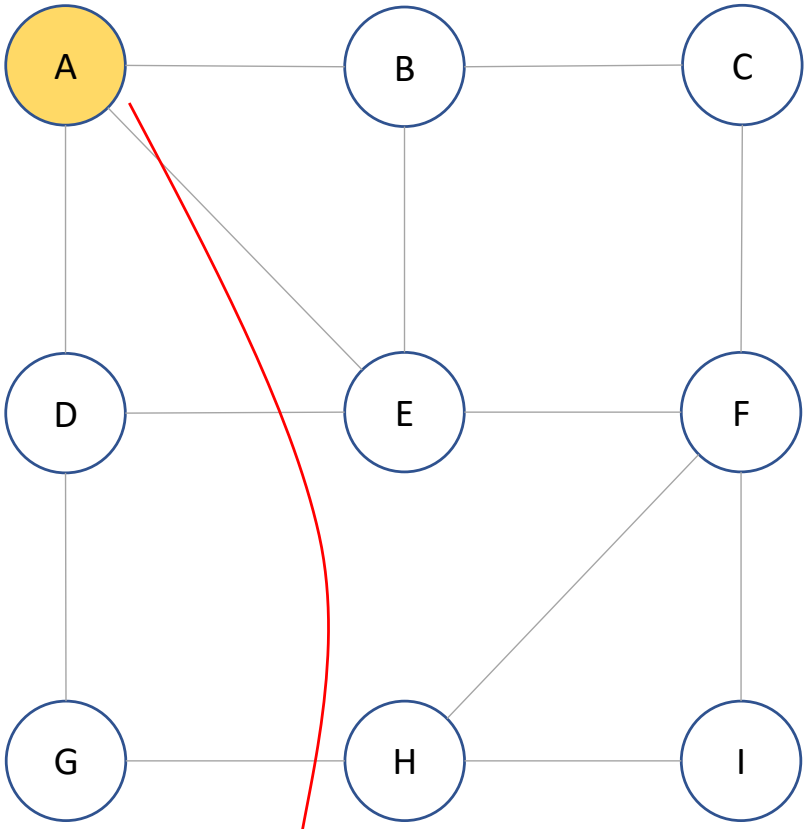
1 Go to the next current vertex at the queue

Vertex	Current vertex	Previous vertex
A		

Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

BFS Algorithm



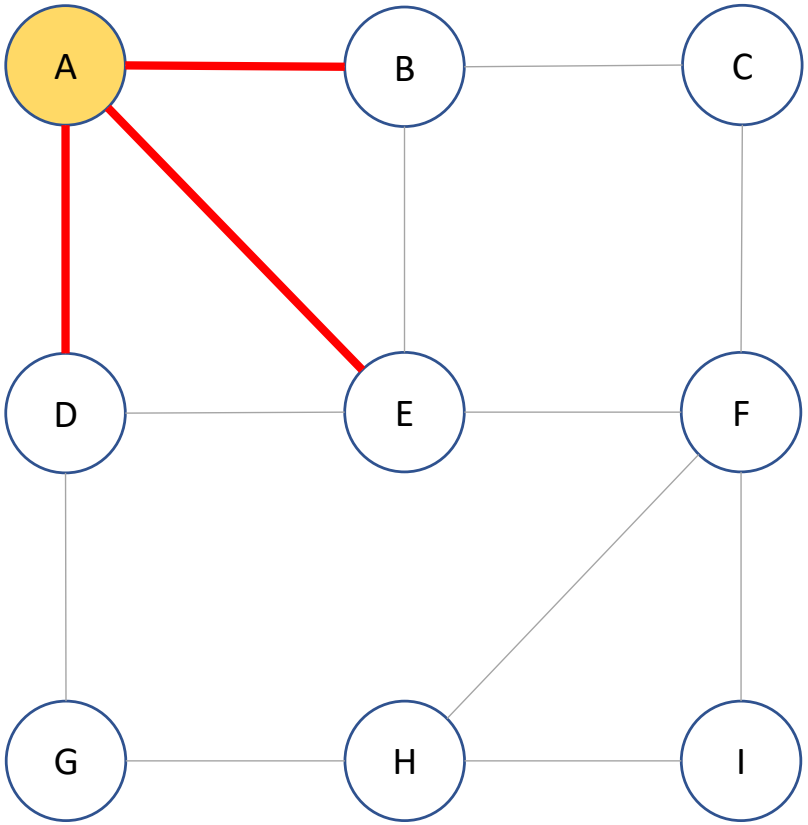
Visited = [A

Unvisited = [B, C, D, E, F, G, H, I]

2 Mark the current vertex as visited

Vertex	Current vertex	Previous vertex
A	◀	

BFS Algorithm



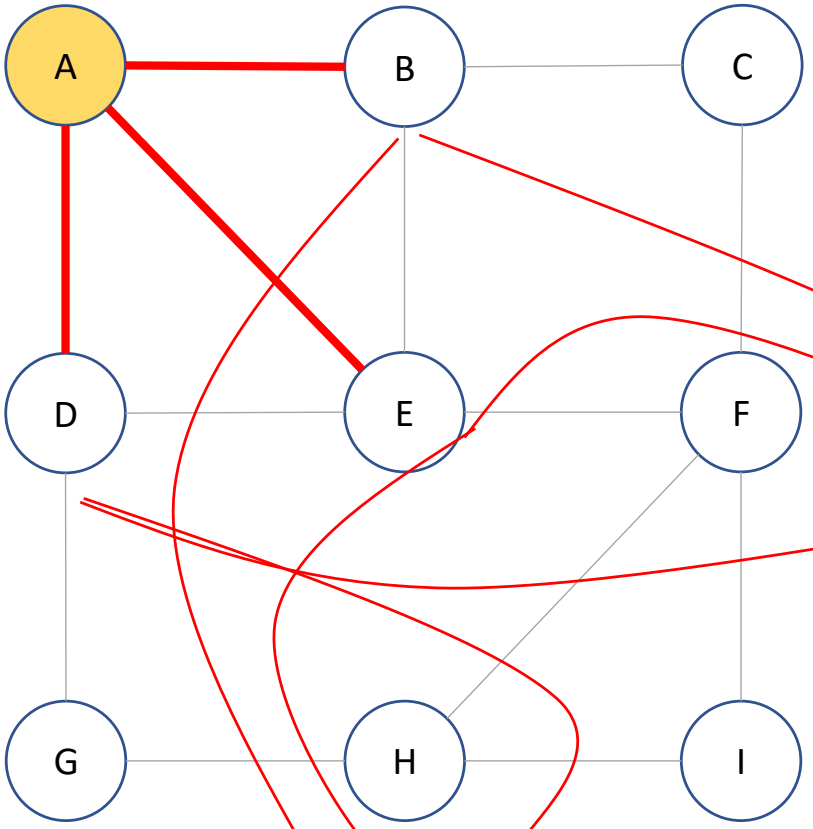
Visited = [A]

Unvisited = [B, C, D, E, F, G, H, I]

3 For the current vertex, if it is not the destination, examine the unvisited neighbours

Vertex	Current vertex	Previous vertex
A	◀	

BFS Algorithm



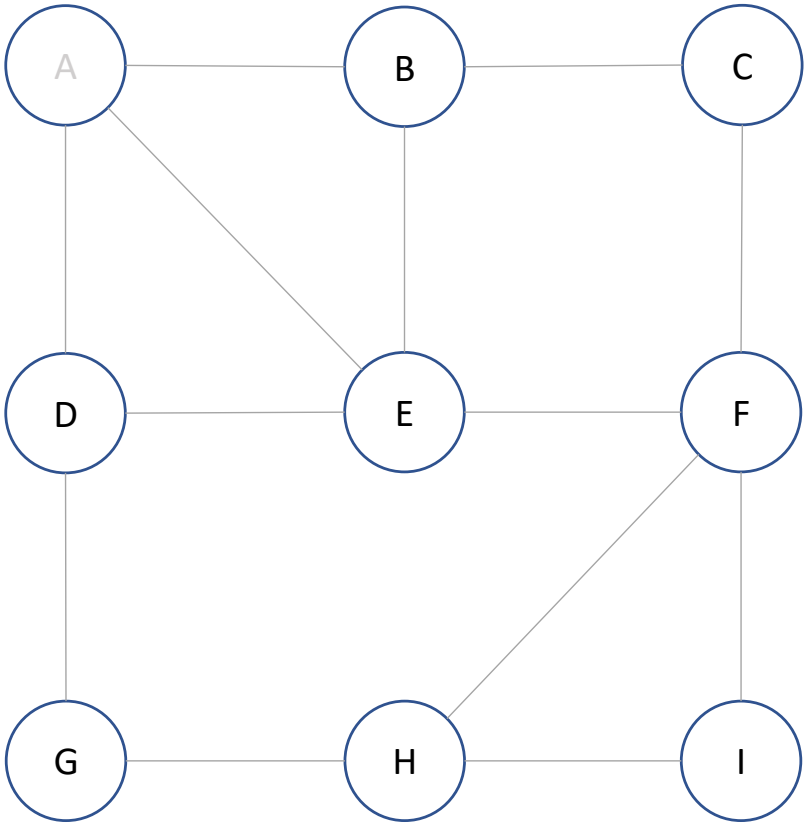
Visited = [A B E D

Unvisited = [C, F, G, H, I]

4 For each vertex being examined, mark as visited and enqueue

Vertex	Current vertex	Previous vertex
A	◀	
B		A
E		A
D		A

BFS Algorithm

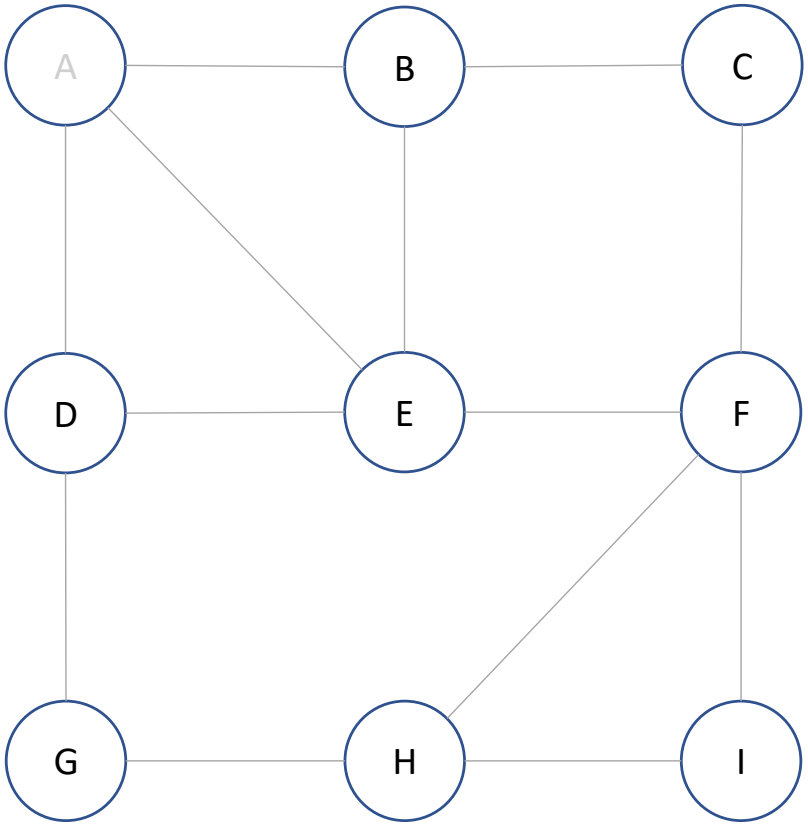


Visited = [A, B, E, D]

Unvisited = [C, F, G, H, I]

Vertex	Current vertex	Previous vertex
A		
B		A
E		A
D		A

BFS Algorithm



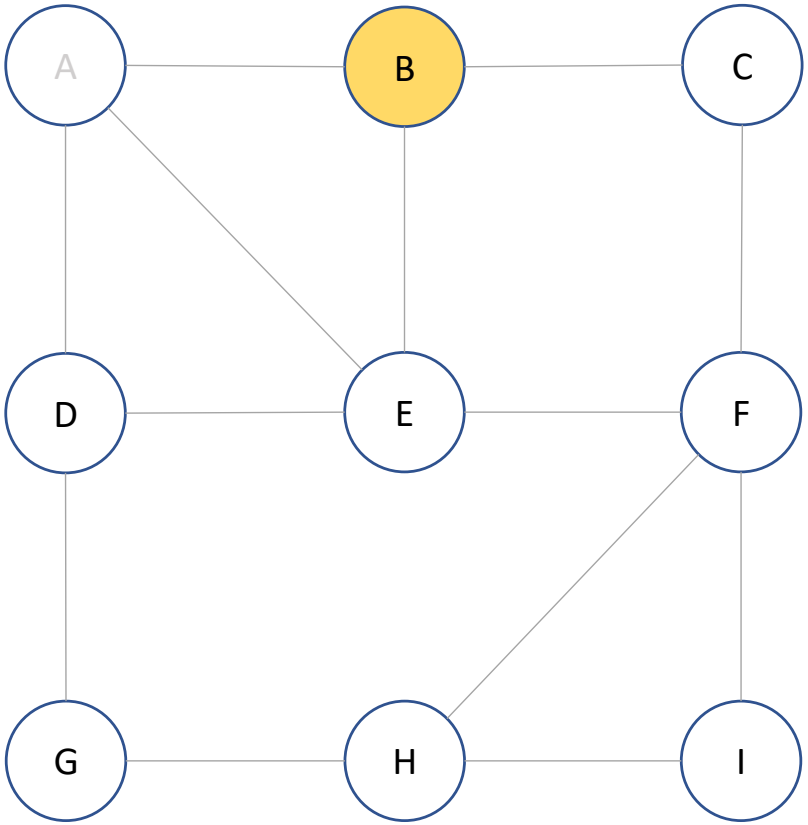
1 Go to the next current vertex at the queue

Vertex	Current vertex	Previous vertex
A		
B		A
E		A
D		A

Visited = [A, B, E, D]

Unvisited = [C, F, G, H, I]

BFS Algorithm



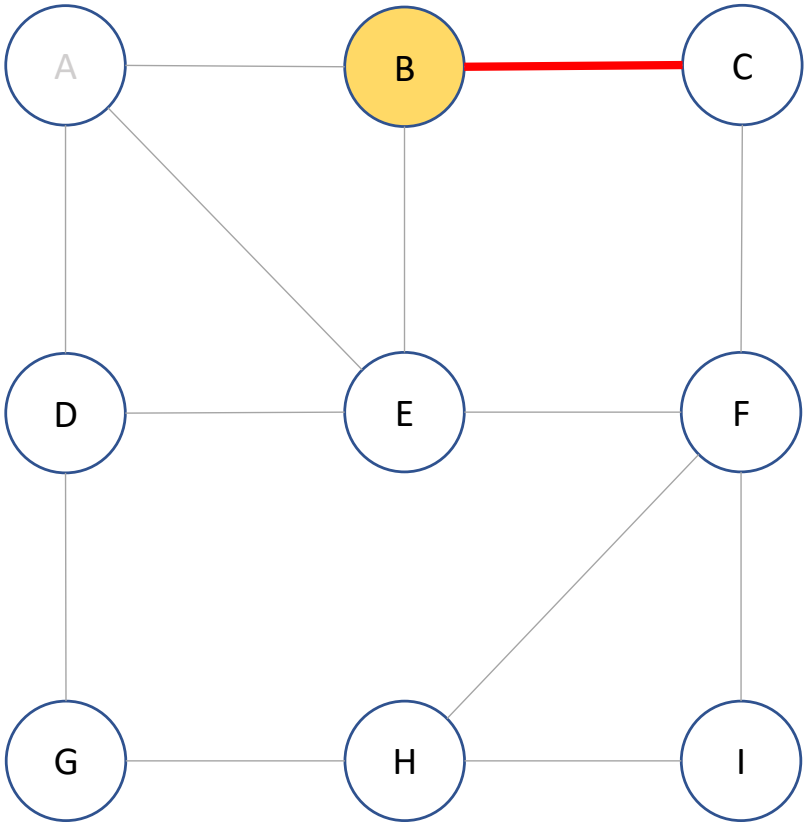
2 Mark the current vertex as visited

Vertex	Current vertex	Previous vertex
A		
B	◀	A
E		A
D		A

Visited = [A, B, E, D]

Unvisited = [C, F, G, H, I]

BFS Algorithm



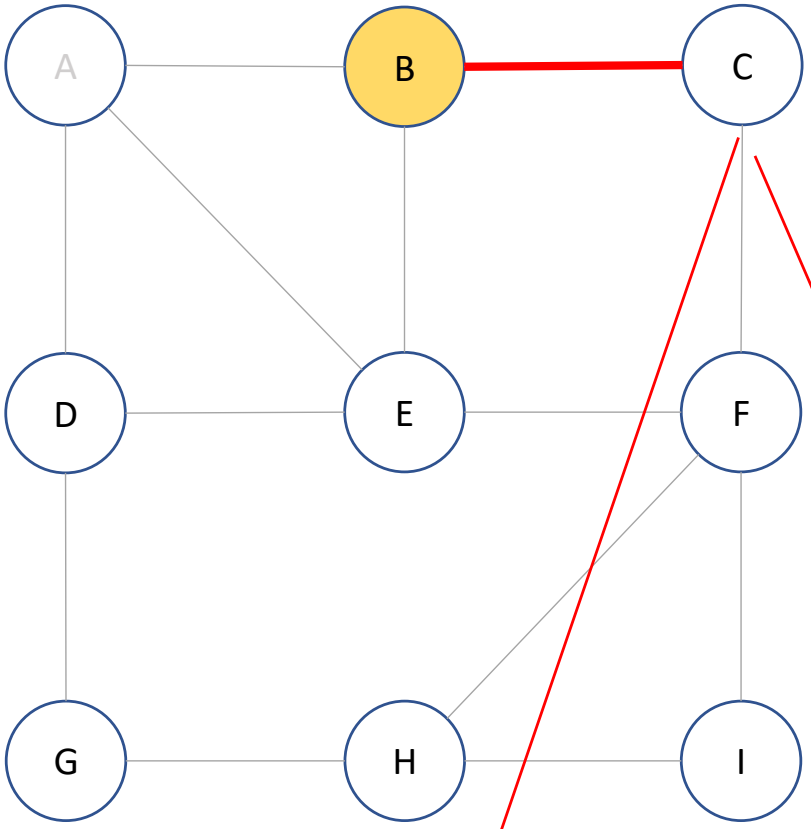
Visited = [A, B, E, D]

Unvisited = [C, F, G, H, I]

3 For the current vertex, if it is not the destination, examine the unvisited neighbours

Vertex	Current vertex	Previous vertex
A		
B	◀	A
E		A
D		A

BFS Algorithm



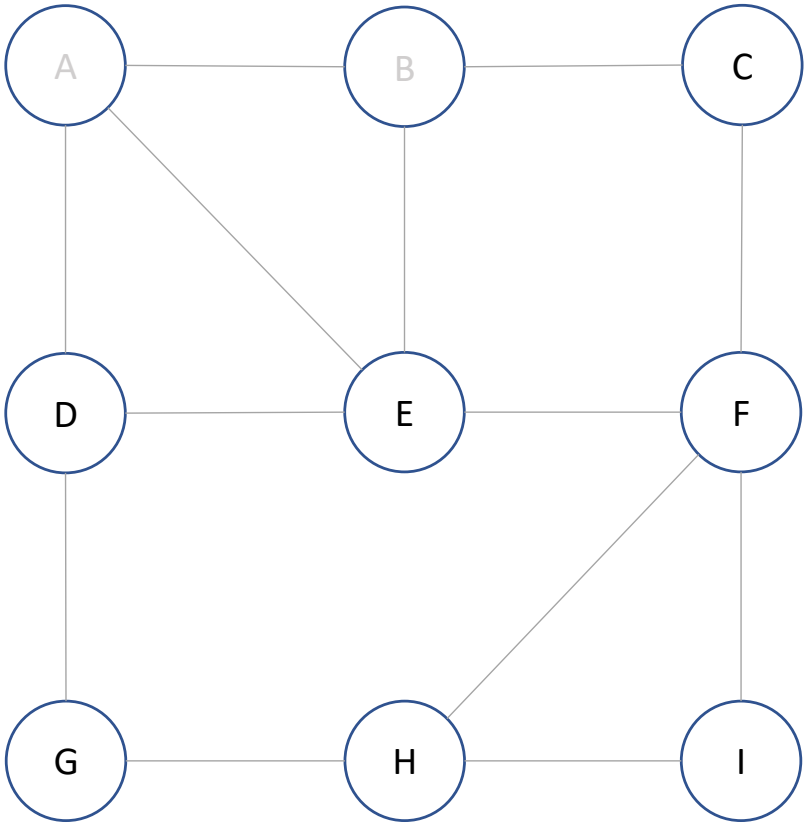
Visited = [A, B, E, D, C]

Unvisited = [F, G, H, I]

4 For each vertex being examined, mark as visited and enqueue

Vertex	Current vertex	Previous vertex
A		
B	◀	A
E		A
D		A
C		B

BFS Algorithm

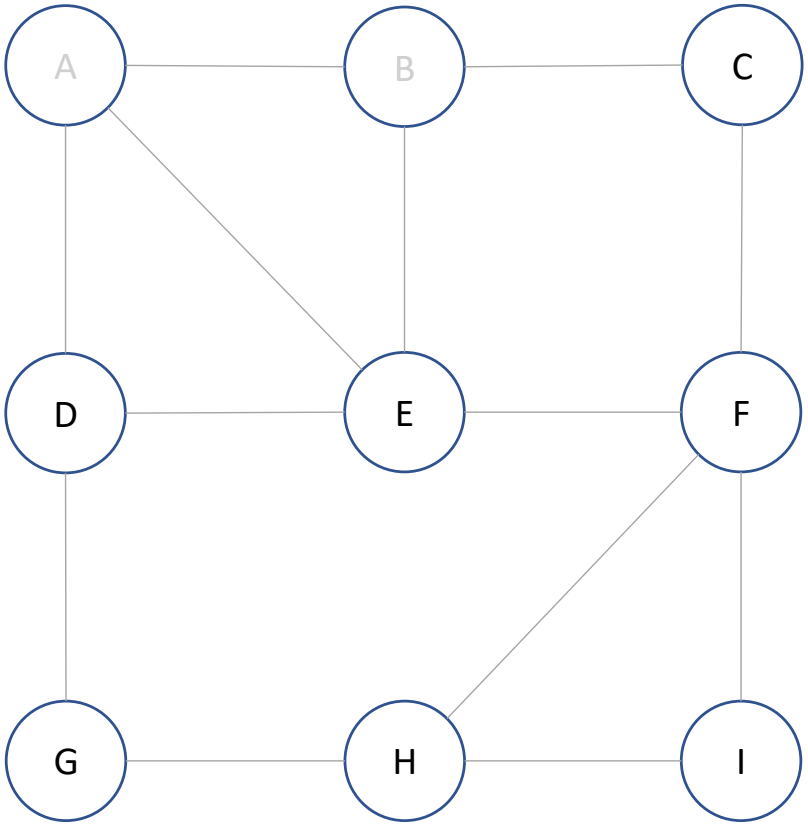


Visited = [A, B, E, D, C]

Unvisited = [F, G, H, I]

Vertex	Current vertex	Previous vertex
A		
B		A
E		A
D		A
C		B

BFS Algorithm



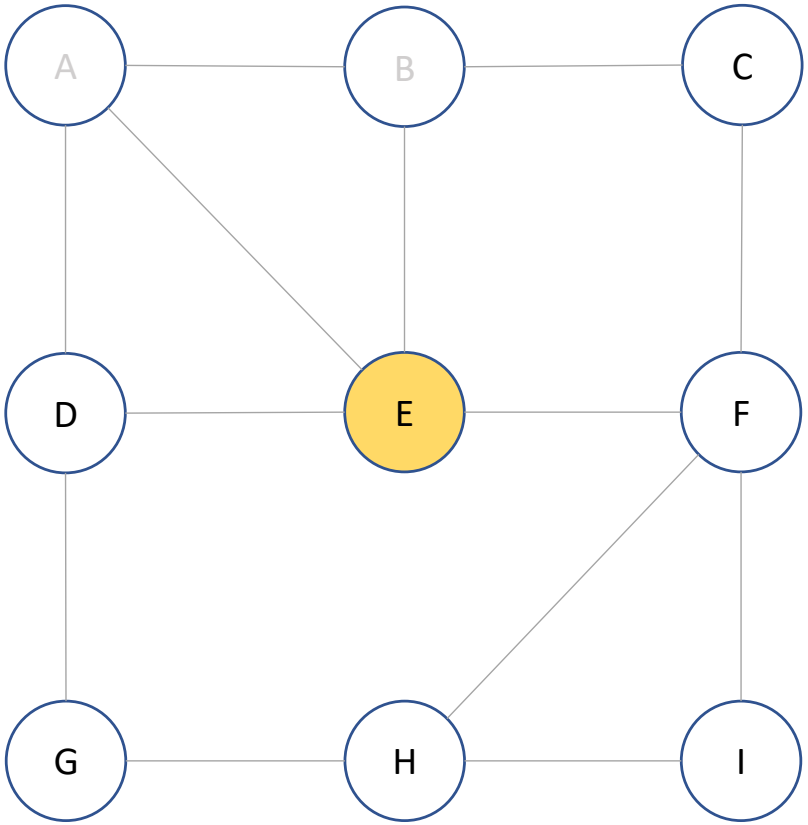
1 Go to the next current vertex at the queue

Vertex	Current vertex	Previous vertex
A		
B		A
E		A
D		A
C		B

Visited = [A, B, E, D, C]

Unvisited = [F, G, H, I]

BFS Algorithm



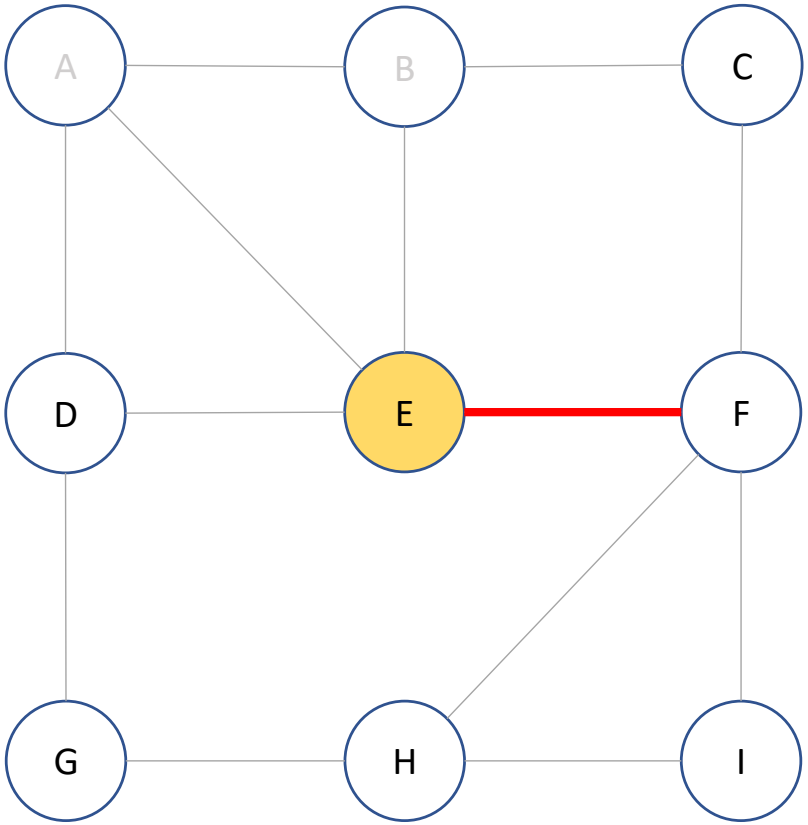
2 Mark the current vertex as visited

Vertex	Current vertex	Previous vertex
A		
B		A
E	◀	A
D		A
C		B

Visited = [A, B, E, D, C]

Unvisited = [F, G, H, I]

BFS Algorithm



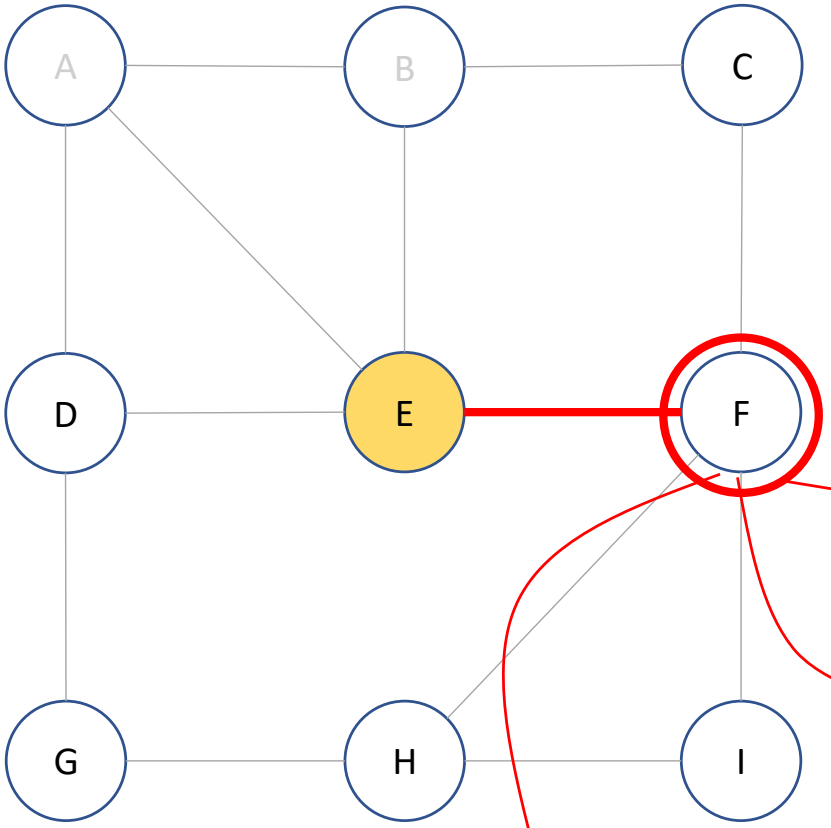
Visited = [A, B, E, D, C]

Unvisited = [F, G, H, I]

3 For the current vertex, if it is not the destination, examine the unvisited neighbours

Vertex	Current vertex	Previous vertex
A		
B		A
E	◀	A
D		A
C		B

BFS Algorithm



4 For each vertex being examined, mark as visited and enqueue

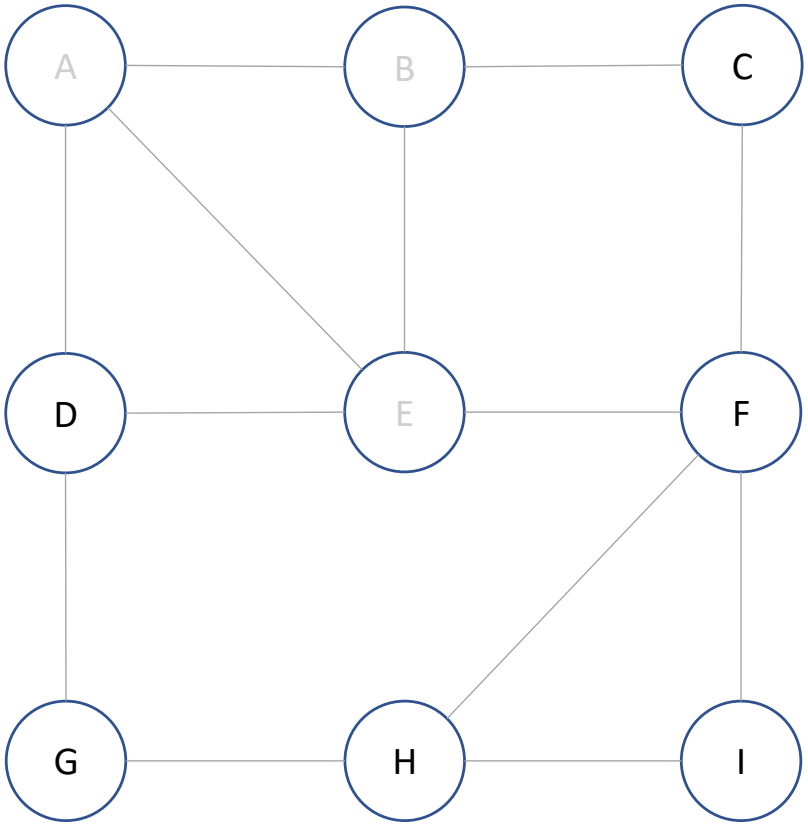
Vertex	Current vertex	Previous vertex
A		
B		A
E	◀	A
D		A
C		B
F		E

F

Visited = [A, B, E, D, C,

Unvisited = [G, H, I]

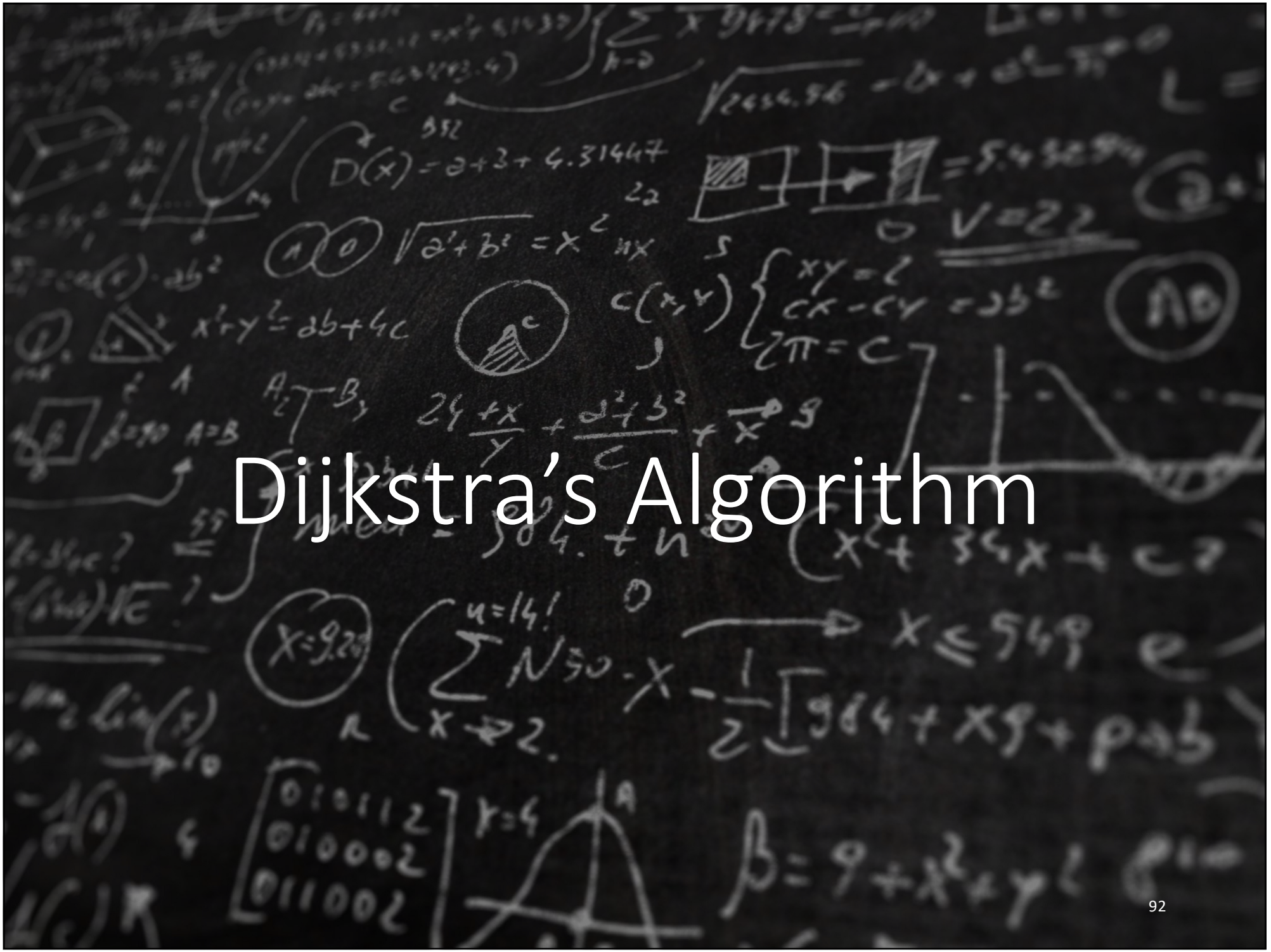
BFS Algorithm



Vertex	Current vertex	Previous vertex
A		
B		A
E		A
D		A
C		B
F		E

Visited = [A, B, E, D, C, F]

Unvisited = [G, H, I]



Dijkstra's Algorithm

37249 Thomas J. Misa, Editor

y

iew with
. Dijkstra

nary, in one of his last interviews
flects on a programmer's life.

NSTITUTE
world's
of re-
history
to the
re, and
inter-
in the
rojects,
er's ex-
en sug-
s. Tran-
es are a
the his-
ditional
ntly in-
th pro-
Dijkstra
by CBI
jkstra's
st 2001
lding a

Turing
known
mputer
assess-
butions
enrich
g scien-

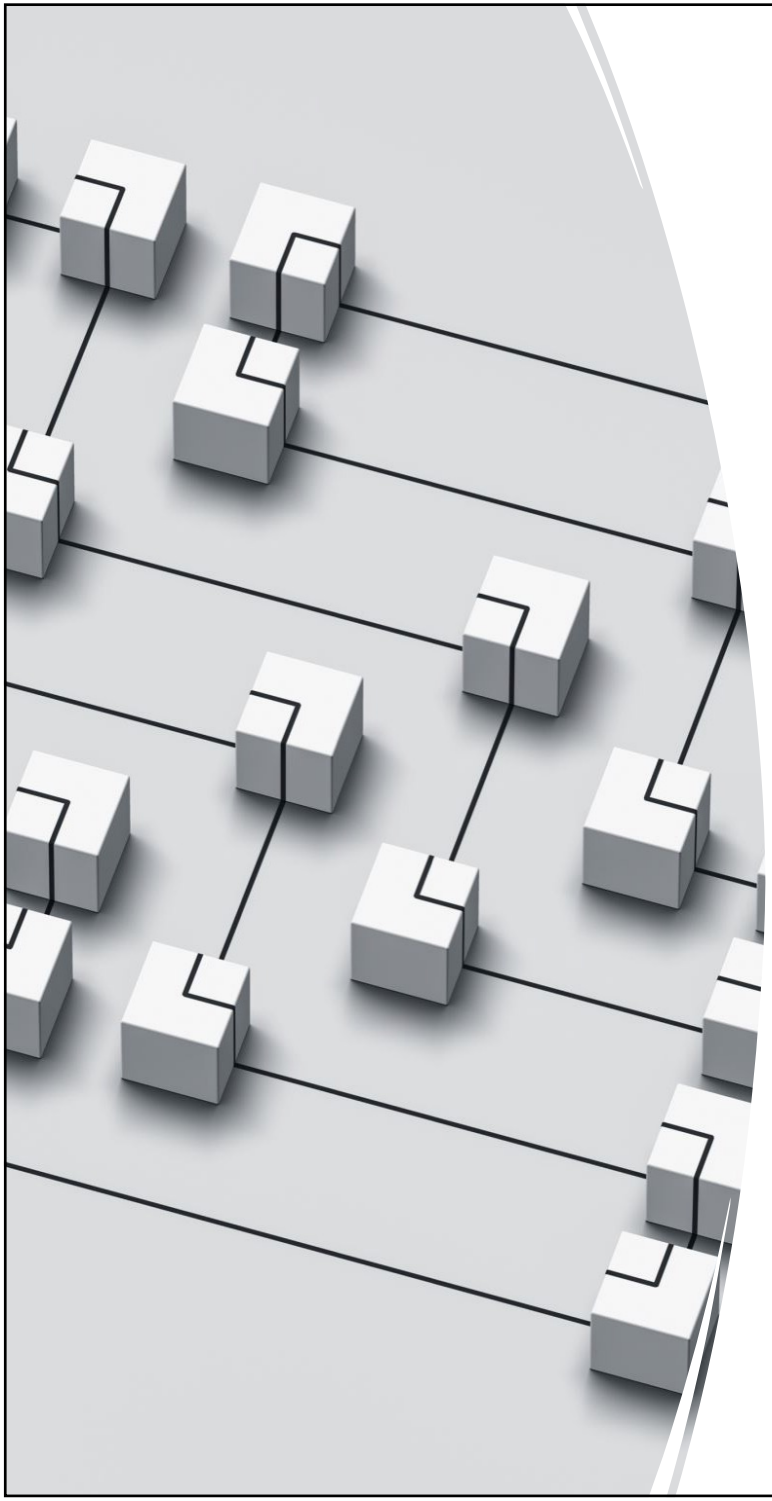
v post-
nniver-
e 72 in
as been
e tran-
bi.umn.

s J. Misa

AUGUST 2010 | VOL. 53 | NO. 8 | COMMUNICATIONS OF THE .C

Edsger Dijkstra (1930 – 2002)

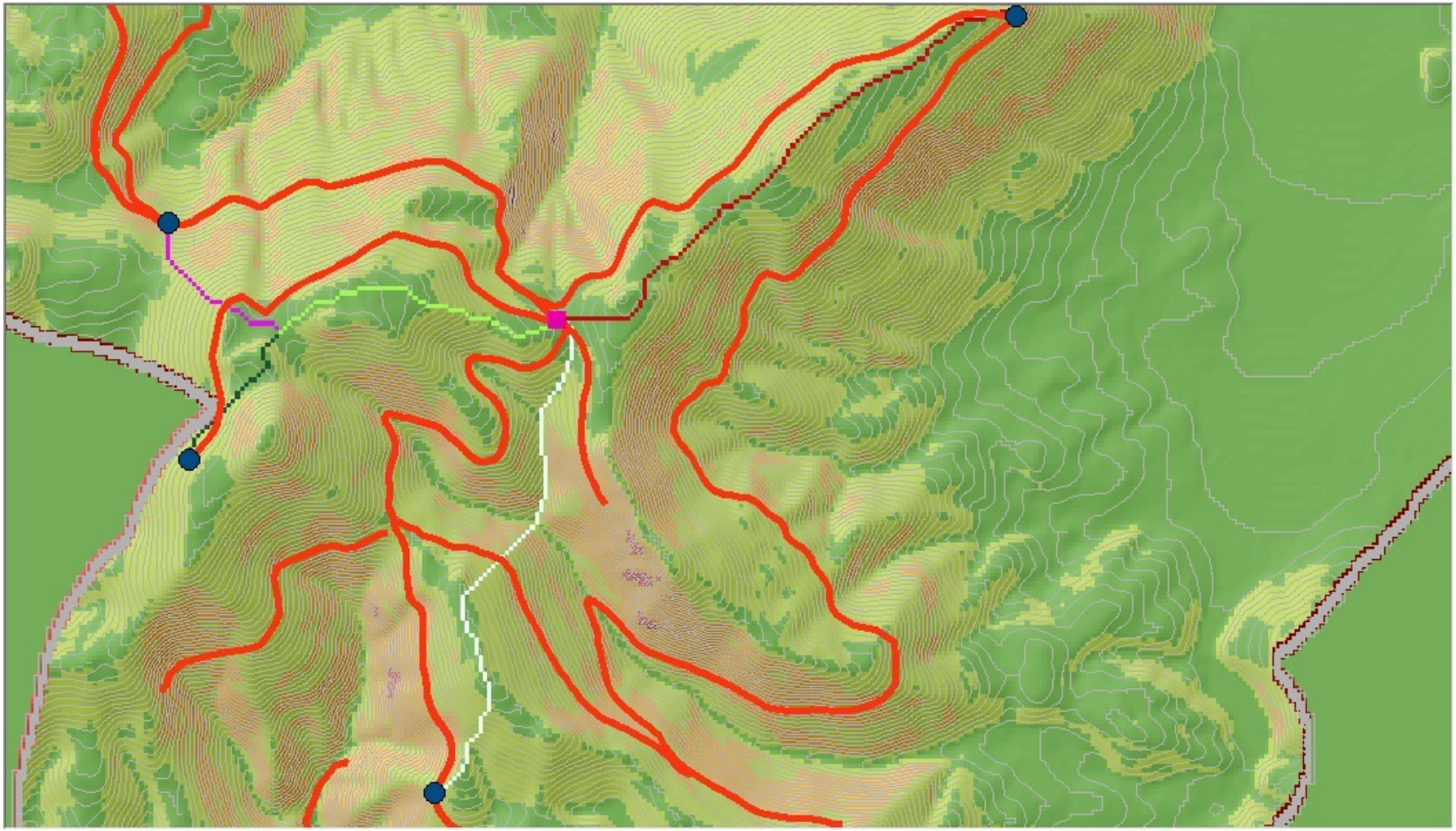
- Dutch computer scientist and professor at UT Austin
- Winner of Turing Award 1972
- Noteworthy algorithms:
  - Dijkstra's algorithm
  - Dining philosophers' problem
  - THE multiprogramming system
- Famous papers:
  - Go To Statement Considered Harmful
  - On the Cruelty of Really Teaching Computing Science



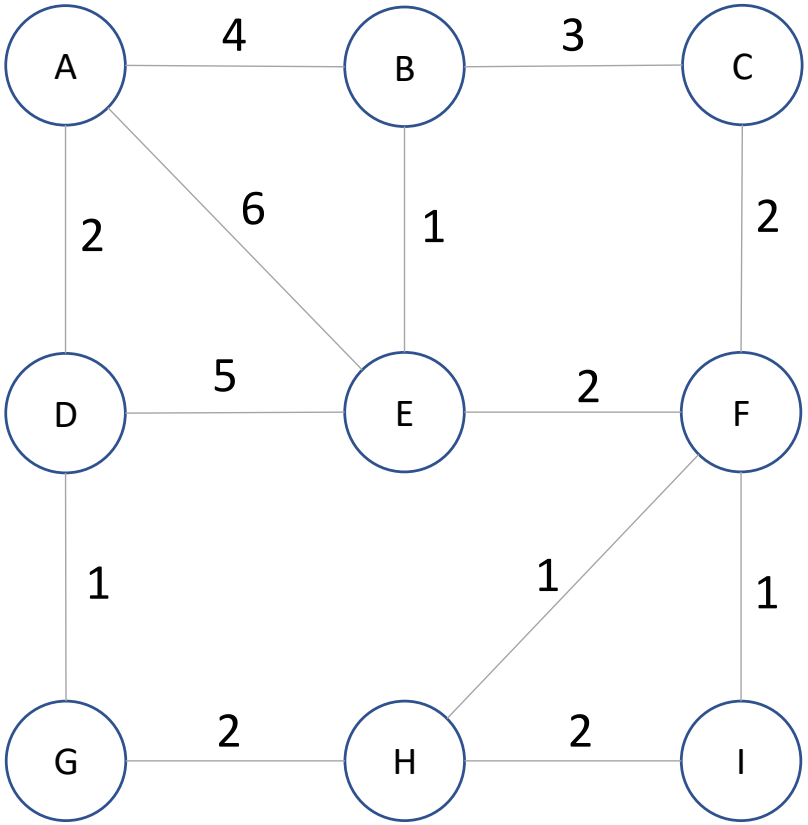
Concept of Dijkstra's Algorithm

- Create a table of information about the currently known best way to reach each vertex in term of cost.
- Update the table when an improved path is identified.
- Continue doing so until it reaches the best solution in term of cost.

How does “best in term of cost” look like in a real application?



Map with Cost Information



A → F

BFS:

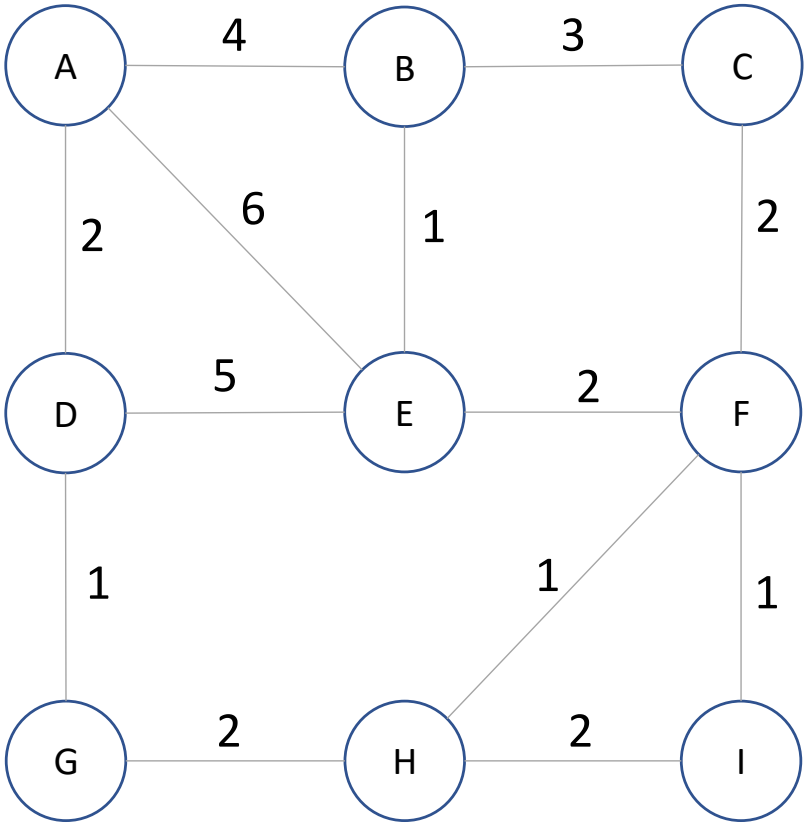
A → E → F

Steps: 2, Cost: 8

Dijkstra's:

_____ (?)

Dijkstra's Algorithm



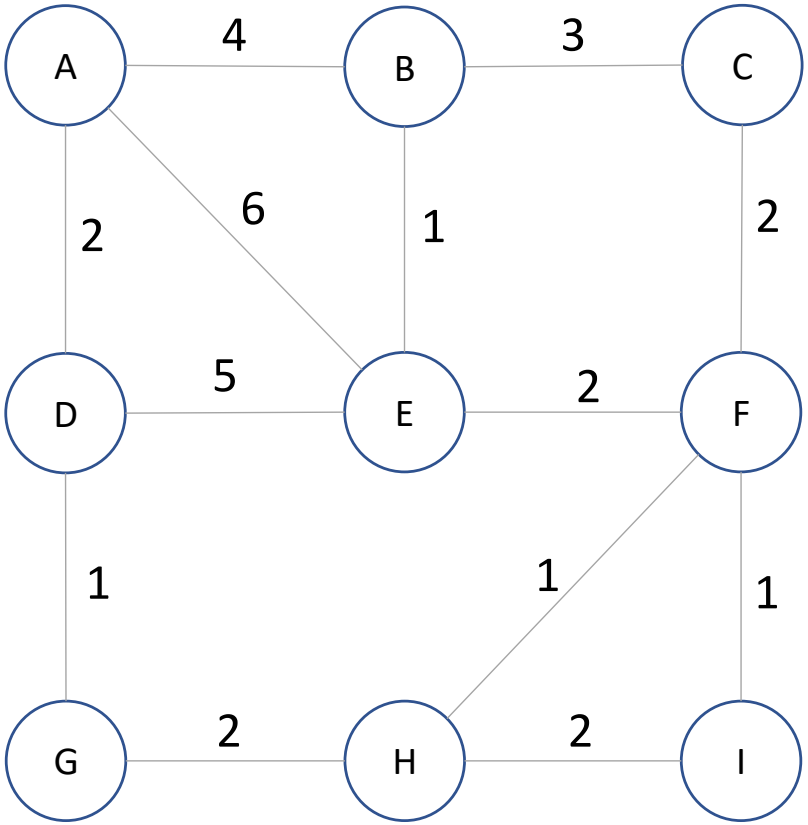
Find the cheapest path from A to F

Vertex	Lowest cost from origin	Previous vertex

Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

Dijkstra's Algorithm (from A to F)



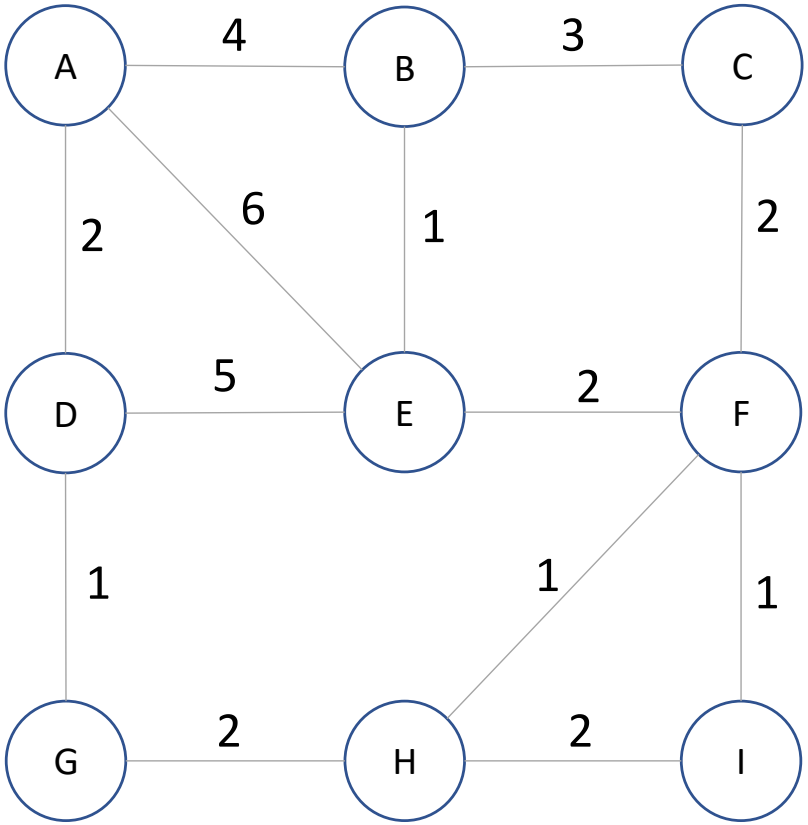
Cost to A from A = 0

Vertex	Lowest cost from origin	Previous vertex
A	0	

Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

Dijkstra's Algorithm (from A to F)



Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

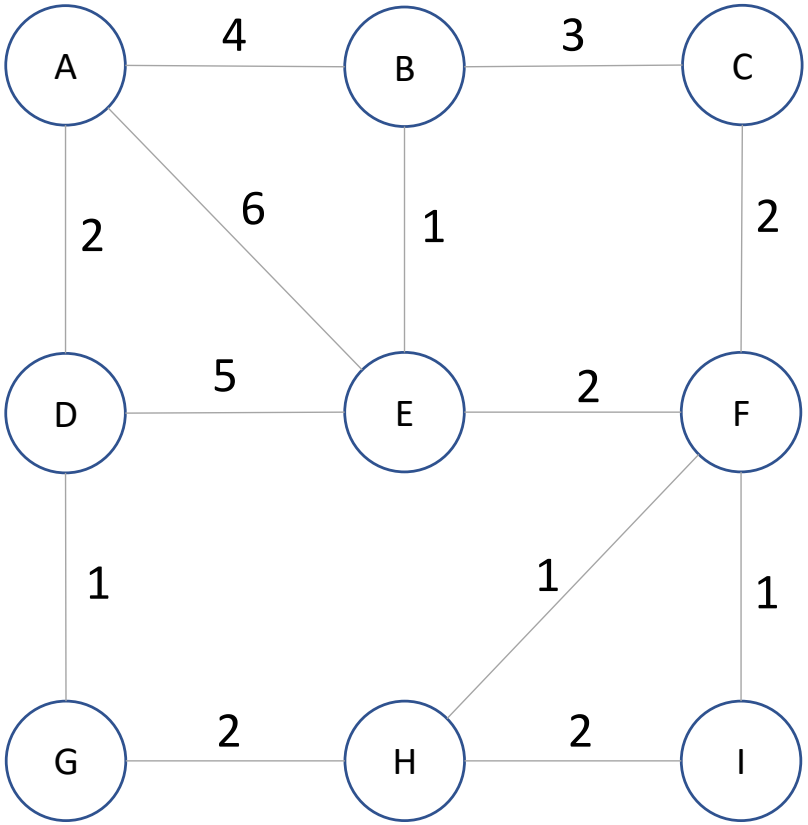
because the robot does not yet know

Cost to A from A = 0

Cost from other vertex to A = ∞

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	
G	∞	
H	∞	
I	∞	

Dijkstra's Algorithm (from A to F)



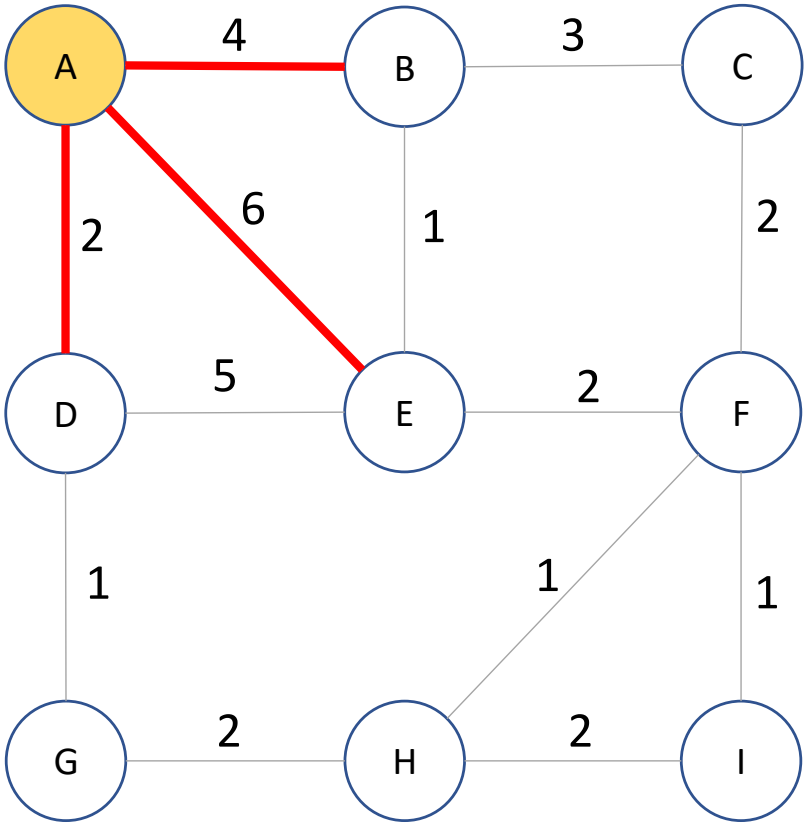
Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

1 Visit the unvisited vertex with the smallest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	
G	∞	
H	∞	
I	∞	

Dijkstra's Algorithm (from A to F)



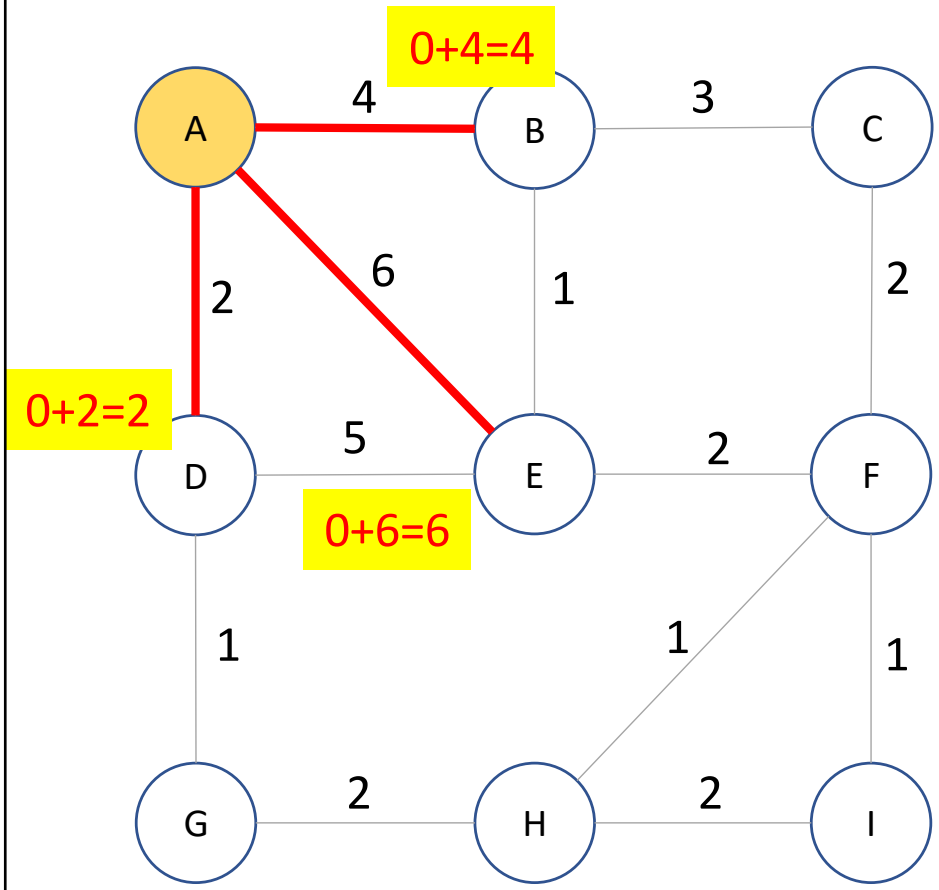
Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

2 For the current vertex, if it is not the destination, examine the unvisited neighbours

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	
G	∞	
H	∞	
I	∞	

Dijkstra's Algorithm (from A to F)



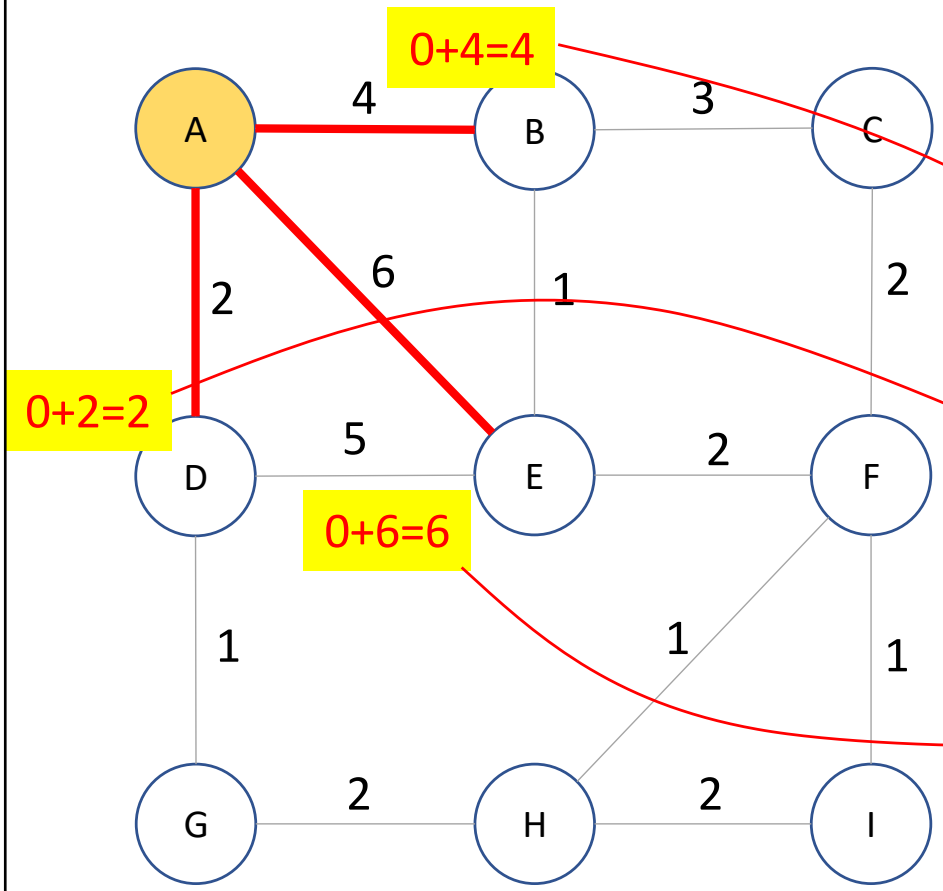
Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

3 Calculate the cost of each neighbour from the start vertex

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	∞	
C	∞	
D	∞	
E	∞	
F	∞	
G	∞	
H	∞	
I	∞	

Dijkstra's Algorithm (from A to F)



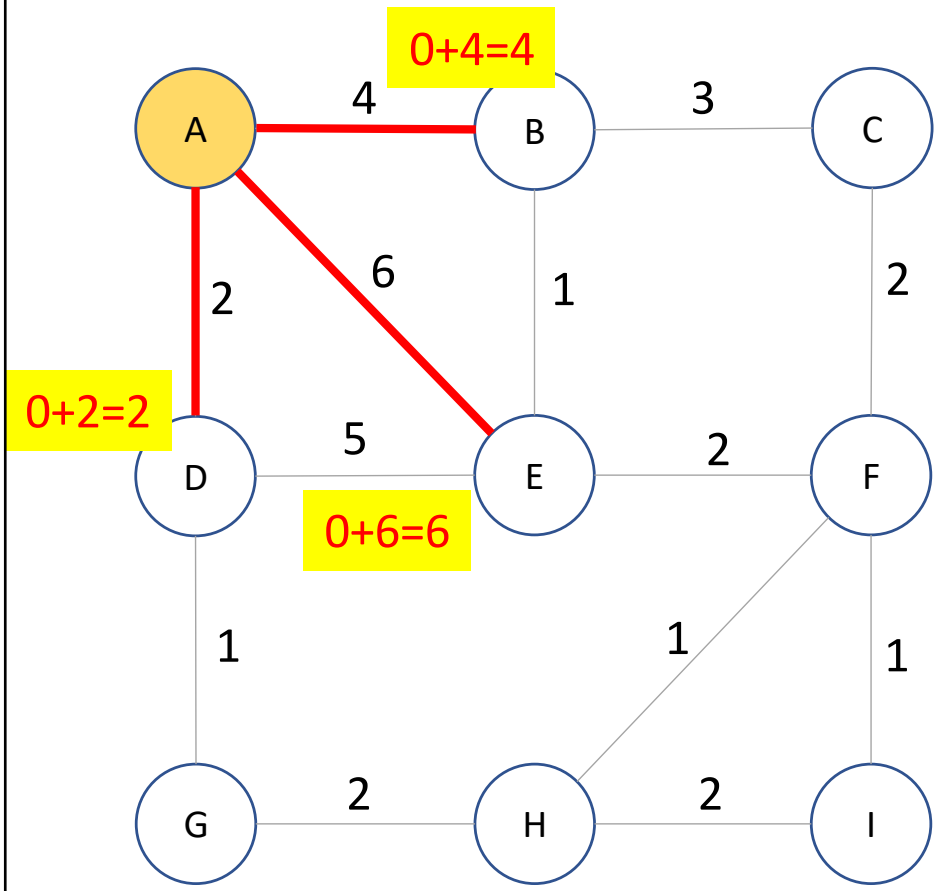
4 If the calculated cost is less than the known cost, update the lowest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	
C	∞	
D	2	
E	6	
F	∞	
G	∞	
H	∞	
I	∞	

Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

Dijkstra's Algorithm (from A to F)



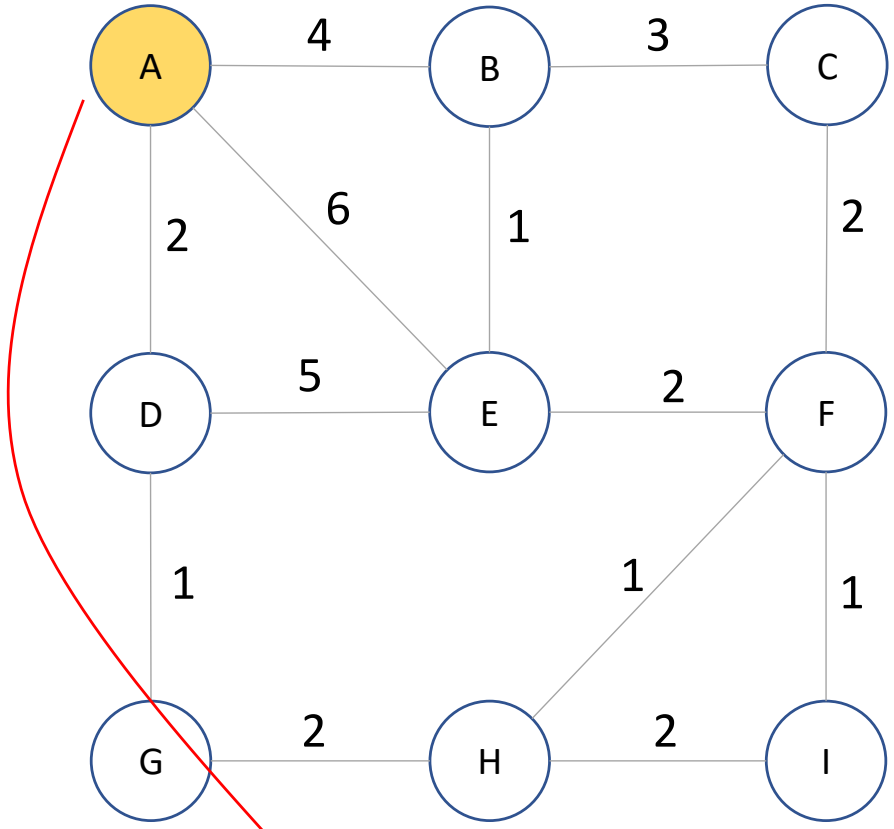
Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

5 Update the previous vertex for each updated lowest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	∞	
H	∞	
I	∞	

Dijkstra's Algorithm (from A to F)



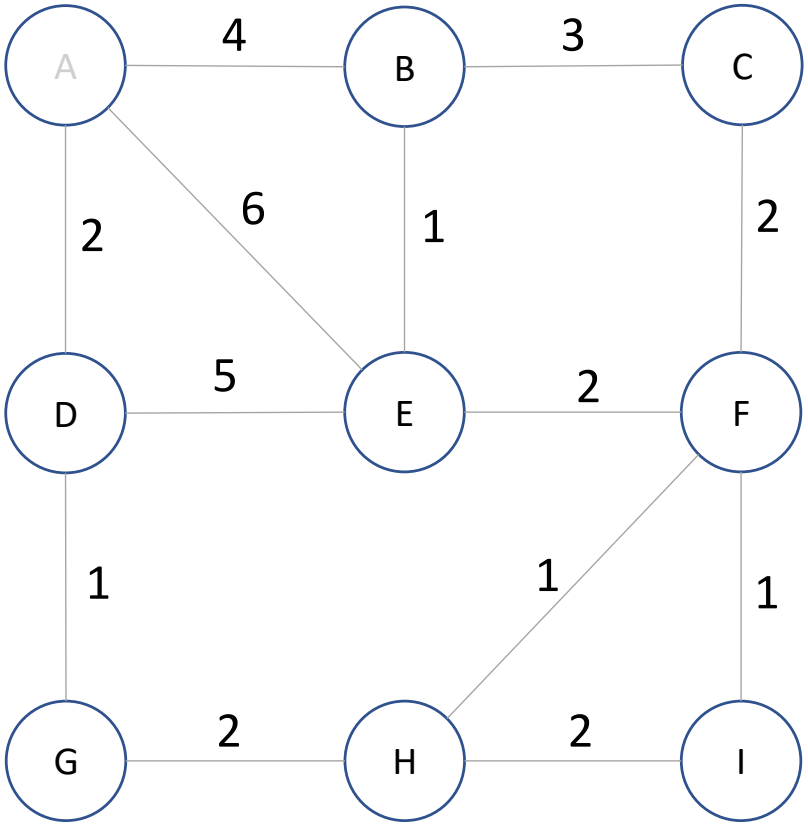
Visited = [A

Unvisited = [B, C, D, E, F, G, H, I]

6 Add the current vertex to the list of visited vertices

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	∞	
H	∞	
I	∞	

Dijkstra's Algorithm (from A to F)

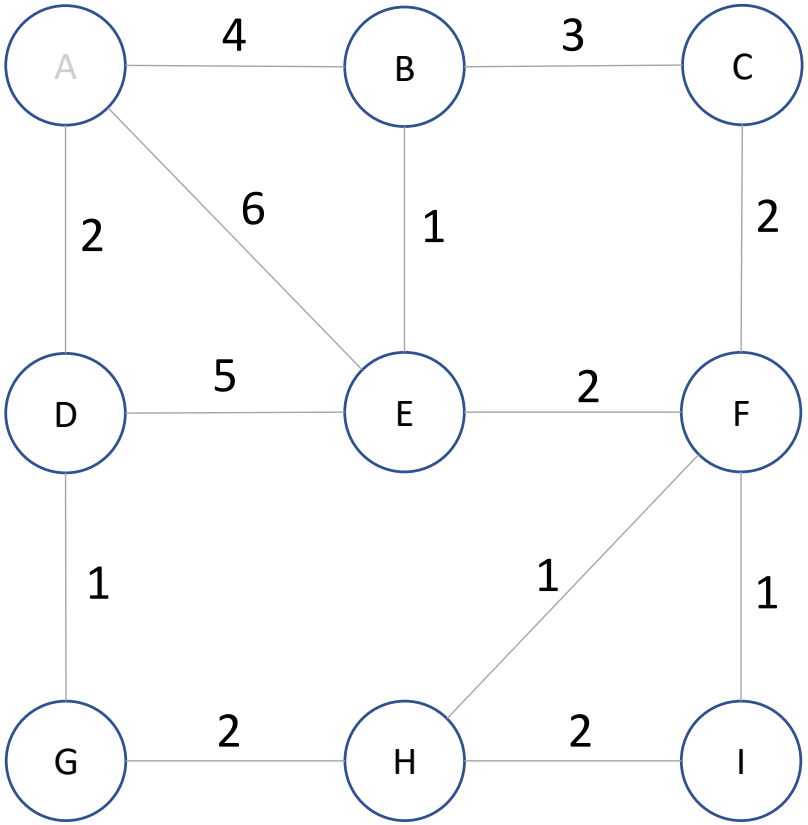


Visited = [A]

Unvisited = [B, C, D, E, F, G, H, I]

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	∞	
H	∞	
I	∞	

Dijkstra's Algorithm (from A to F)



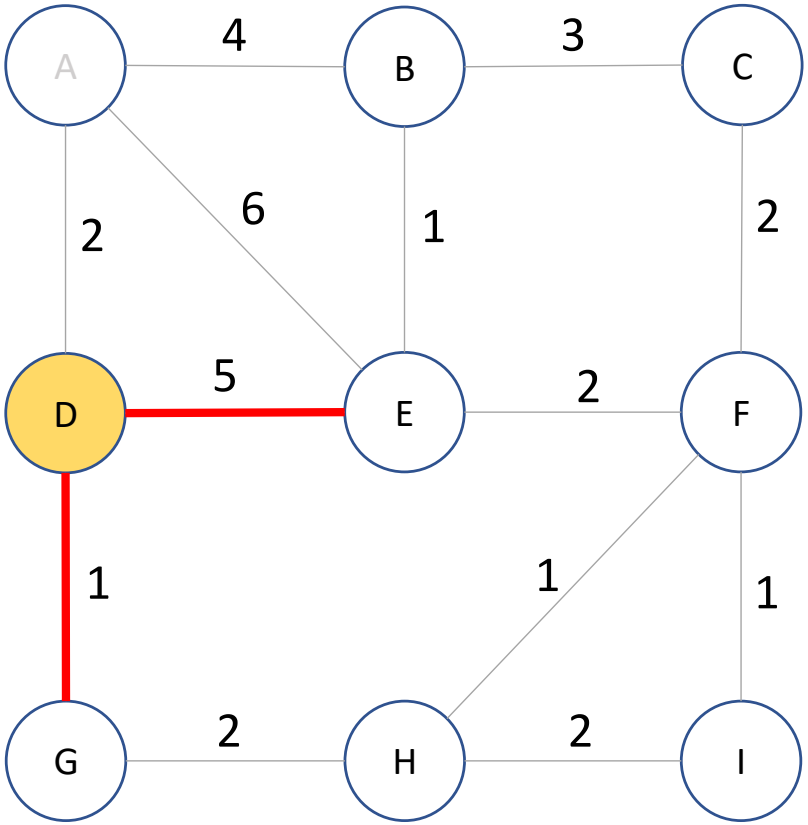
Visited = [A]

Unvisited = [B, C, D, E, F, G, H, I]

1 Visit the unvisited vertex with the smallest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	∞	
H	∞	
I	∞	

Dijkstra's Algorithm (from A to F)



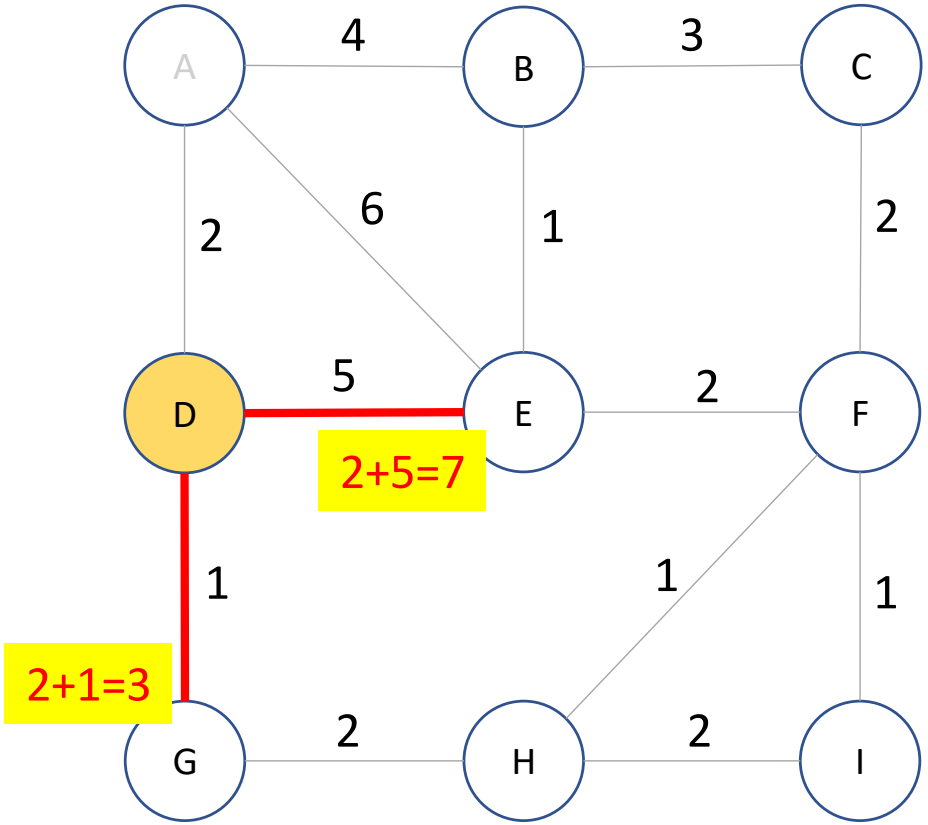
Visited = [A]

Unvisited = [B, C, D, E, F, G, H, I]

2 For the current vertex, if it is not the destination, examine the unvisited neighbours

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	∞	
H	∞	
I	∞	

Dijkstra's Algorithm (from A to F)



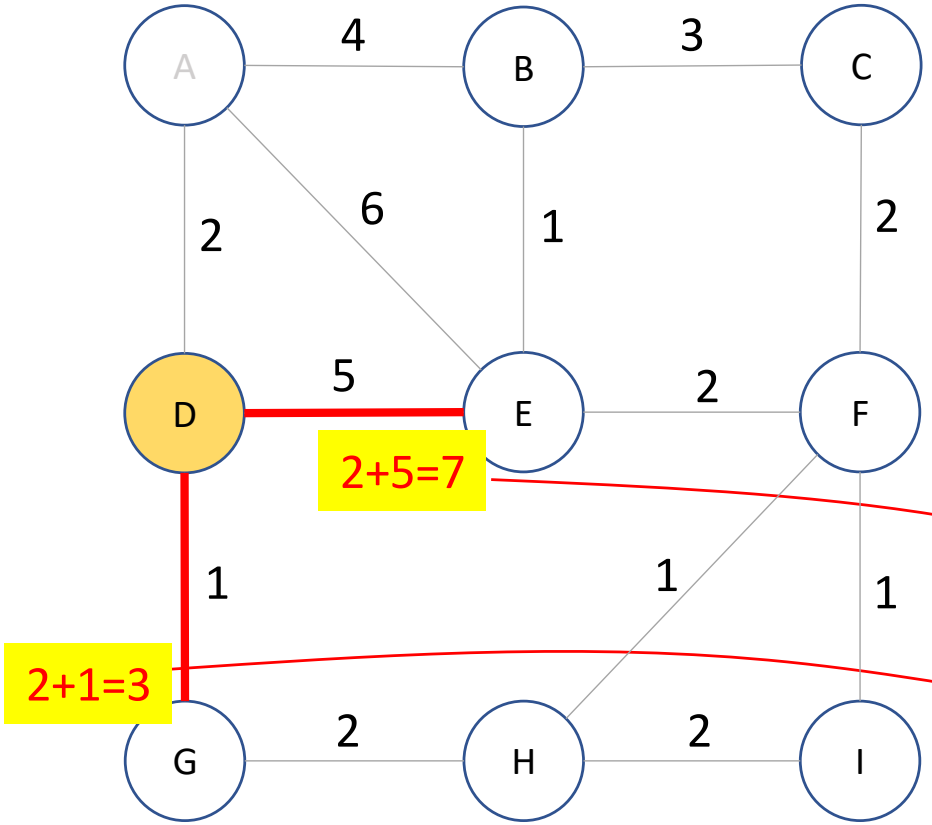
Visited = [A]

Unvisited = [B, C, D, E, F, G, H, I]

3 Calculate the cost of each neighbour from the start vertex

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	∞	
H	∞	
I	∞	

Dijkstra's Algorithm (from A to F)



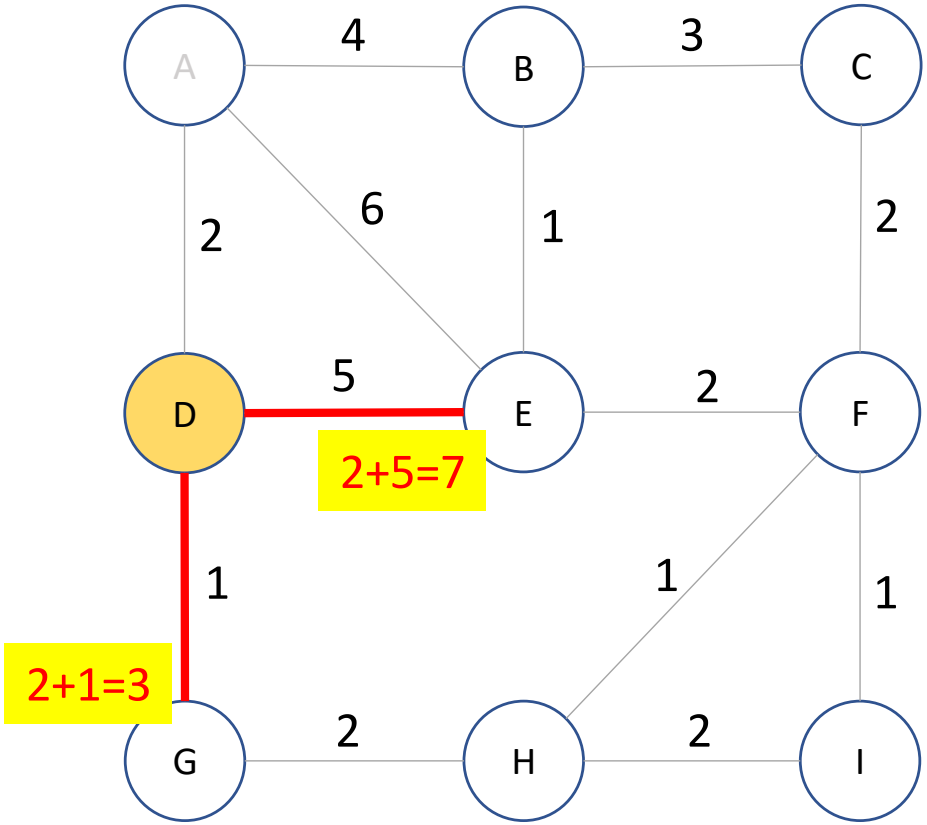
4 If the calculated cost is less than the known cost, update the lowest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	3	
H	∞	
I	∞	

Visited = [A]

Unvisited = [B, C, D, E, F, G, H, I]

Dijkstra's Algorithm (from A to F)



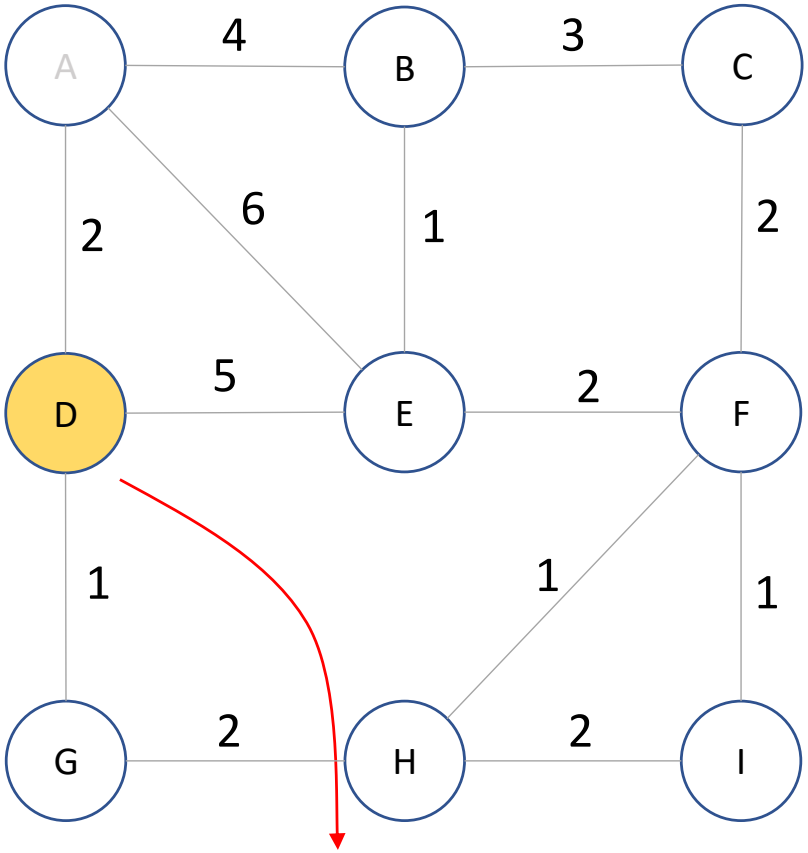
Visited = [A]

Unvisited = [B, C, D, E, F, G, H, I]

5 Update the previous vertex for each updated lowest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	3	D
H	∞	
I	∞	

Dijkstra's Algorithm (from A to F)



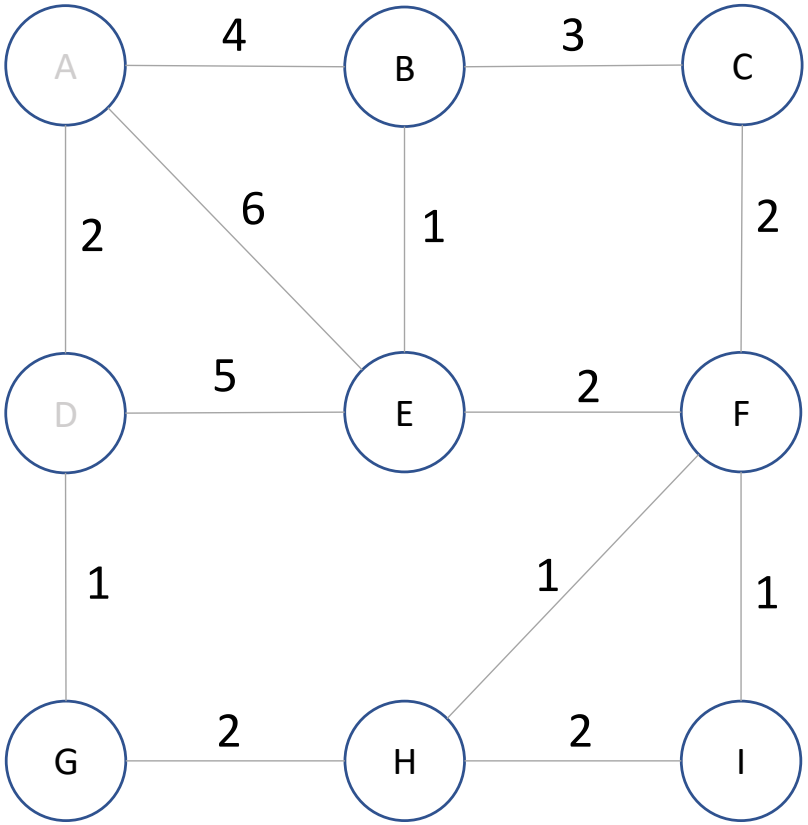
Visited = [A, D]

Unvisited = [B, C, E, F, G, H, I]

6 Add the current vertex to the list of visited vertices

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	3	D
H	∞	
I	∞	

Dijkstra's Algorithm (from A to F)

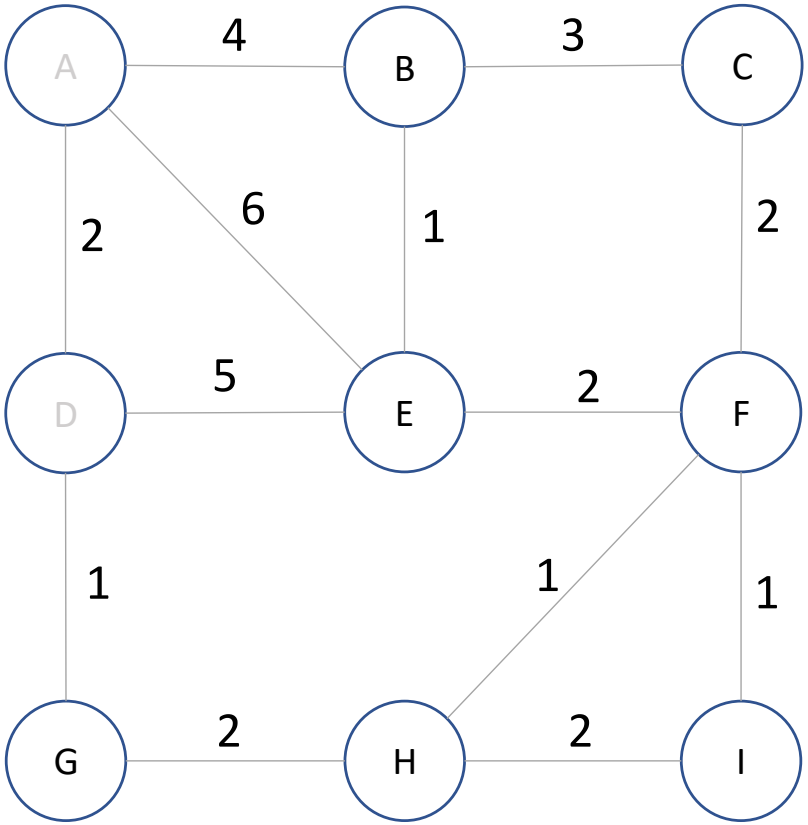


Visited = [A, D]

Unvisited = [B, C, E, F, G, H, I]

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	3	D
H	∞	
I	∞	

Dijkstra's Algorithm (from A to F)



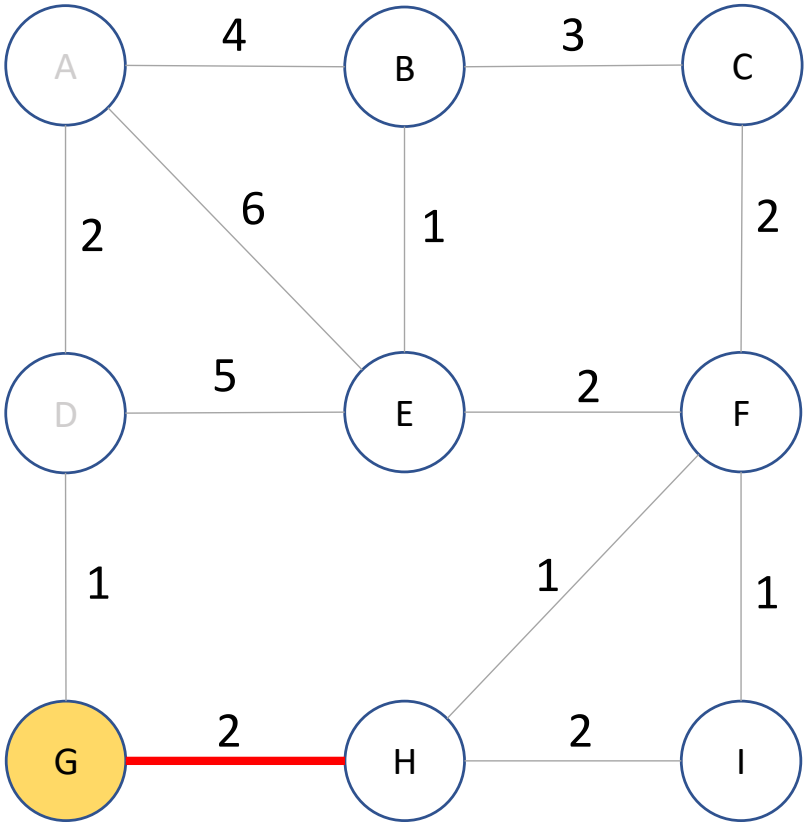
1 Visit the unvisited vertex with the smallest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	3	D
H	∞	
I	∞	

Visited = [A, D]

Unvisited = [B, C, E, F, G, H, I]

Dijkstra's Algorithm (from A to F)



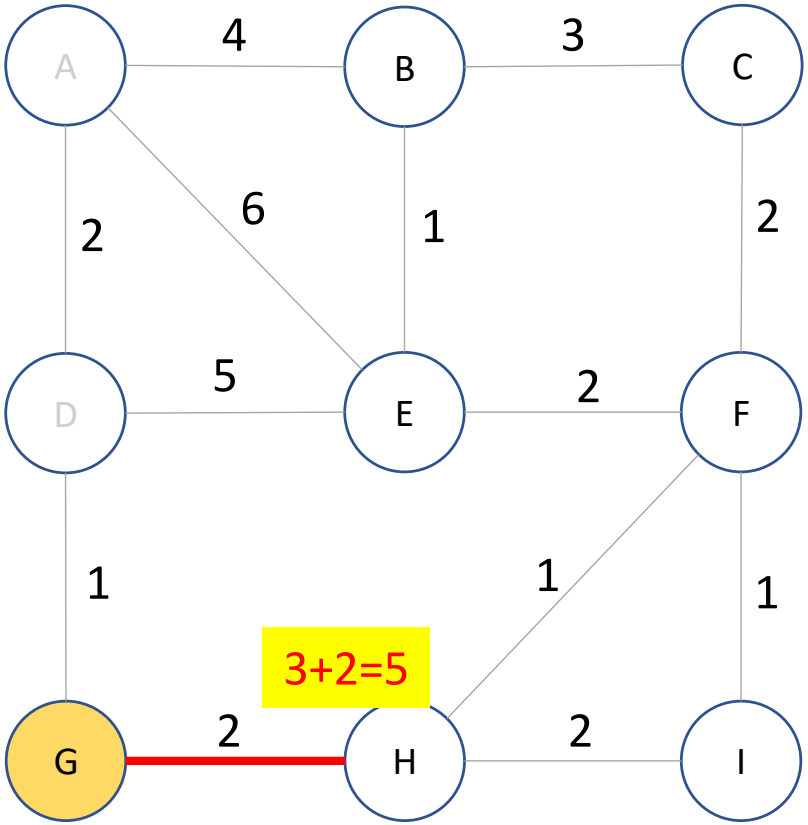
Visited = [A, D]

Unvisited = [B, C, E, F, G, H, I]

2 For the current vertex, if it is not the destination, examine the unvisited neighbours

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	3	D
H	∞	
I	∞	

Dijkstra's Algorithm (from A to F)



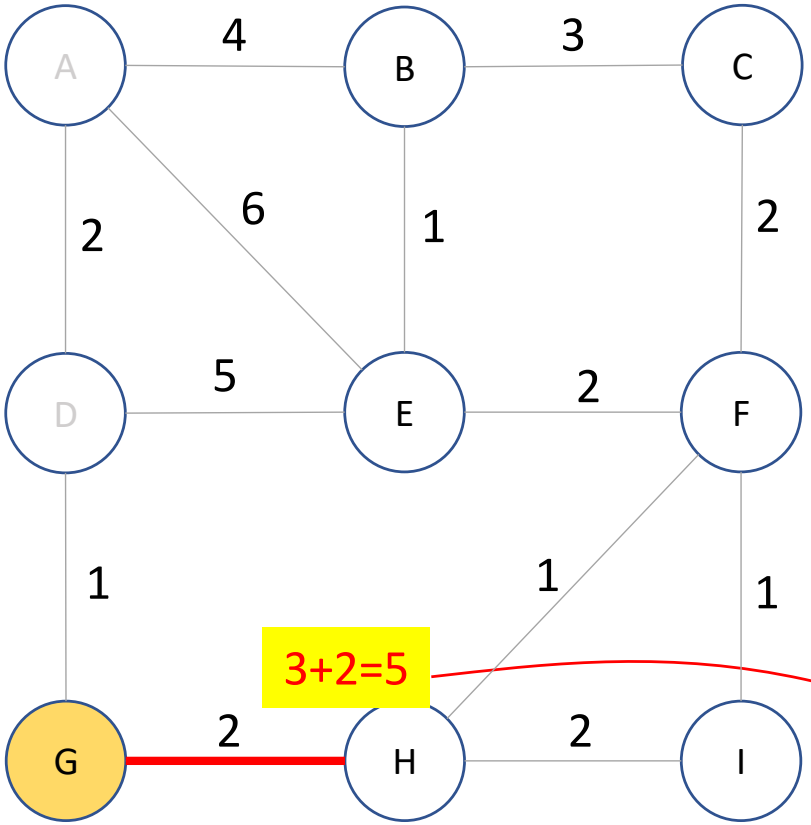
Visited = [A, D]

Unvisited = [B, C, E, F, G, H, I]

3 Calculate the cost of each neighbour from the start vertex

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	3	D
H	∞	
I	∞	

Dijkstra's Algorithm (from A to F)



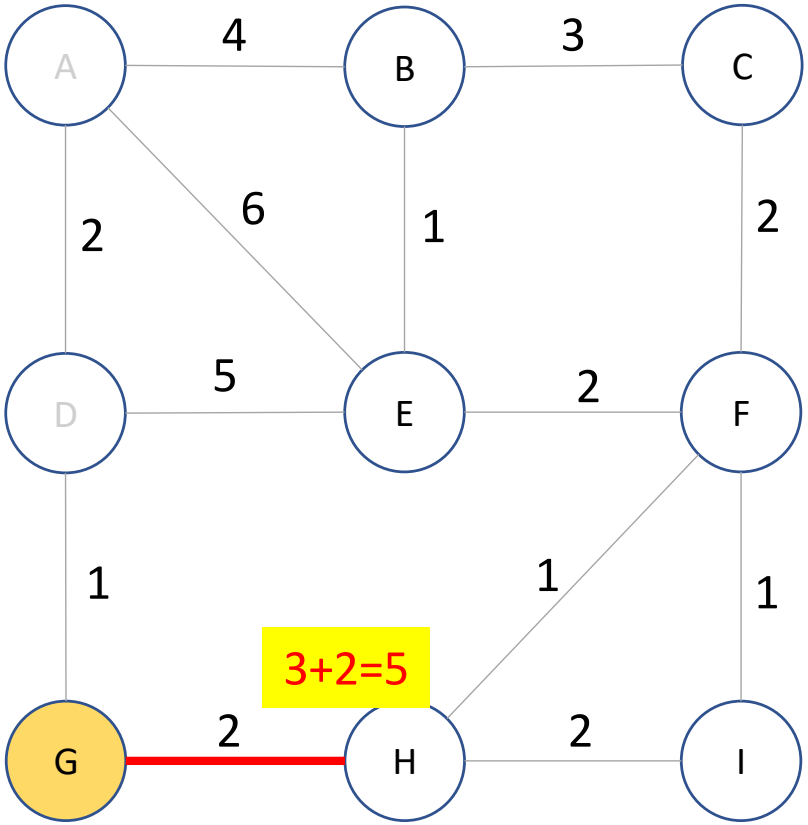
4 If the calculated cost is less than the known cost, update the lowest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	3	D
H	5	
I	∞	

Visited = [A, D]

Unvisited = [B, C, E, F, G, H, I]

Dijkstra's Algorithm (from A to F)



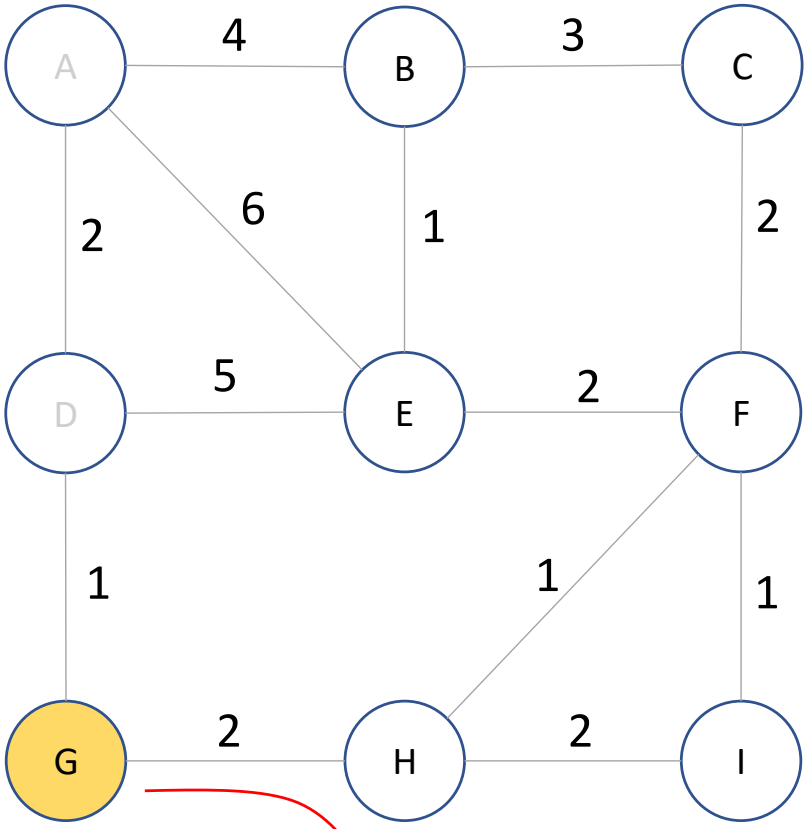
Visited = [A, D]

Unvisited = [B, C, E, F, G, H, I]

5 Update the previous vertex for each updated lowest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	3	D
H	5	G
I	∞	

Dijkstra's Algorithm (from A to F)



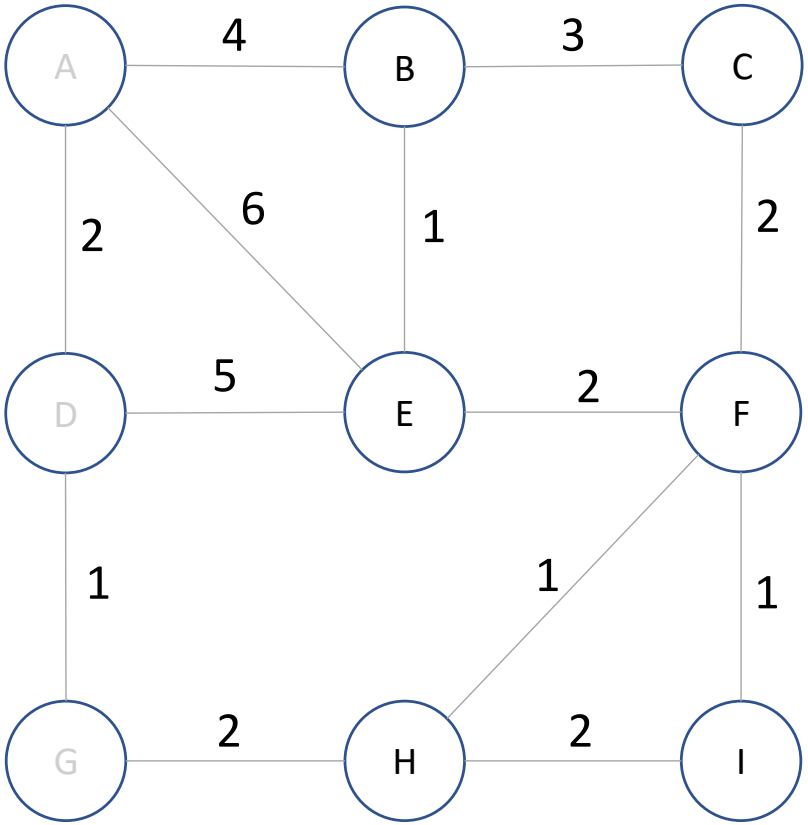
Visited = [A, D, G]

Unvisited = [B, C, E, F, H, I]

6 Add the current vertex to the list of visited vertices

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	3	D
H	5	G
I	∞	

Dijkstra's Algorithm (from A to F)

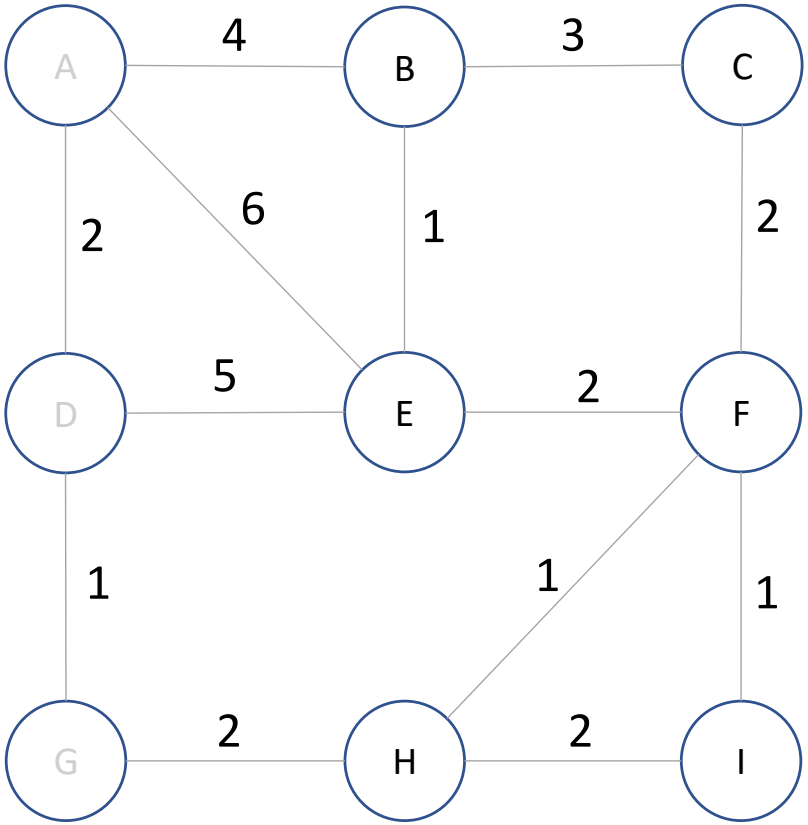


Visited = [A, D, G]

Unvisited = [B, C, E, F, H, I]

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	3	D
H	5	G
I	∞	

Dijkstra's Algorithm (from A to F)



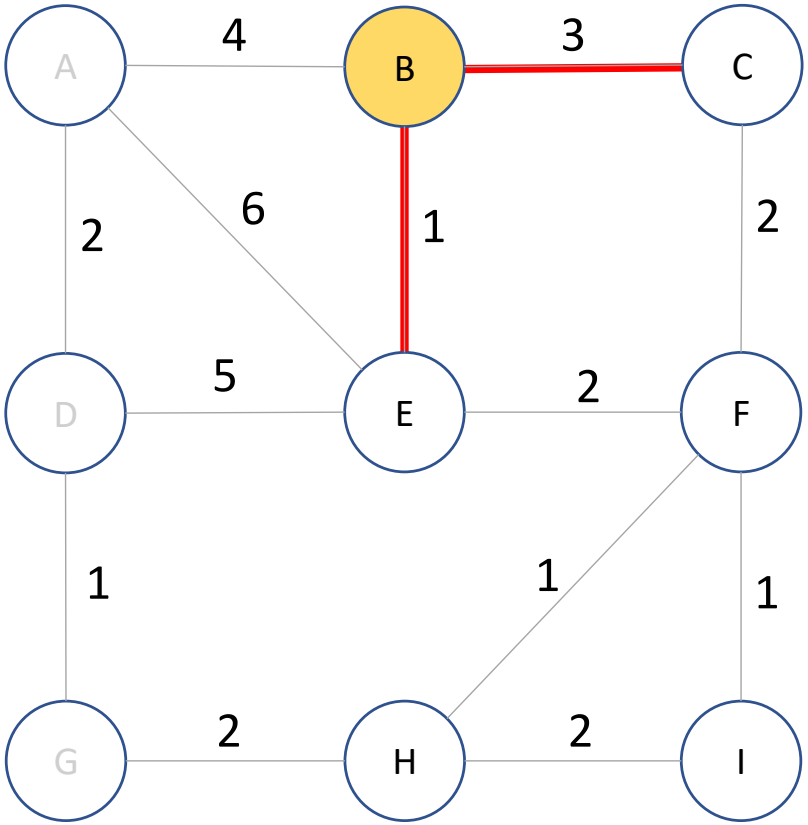
Visited = [A, D, G]

Unvisited = [B, C, E, F, H, I]

1 Visit the unvisited vertex with the smallest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	3	D
H	5	G
I	∞	

Dijkstra's Algorithm (from A to F)



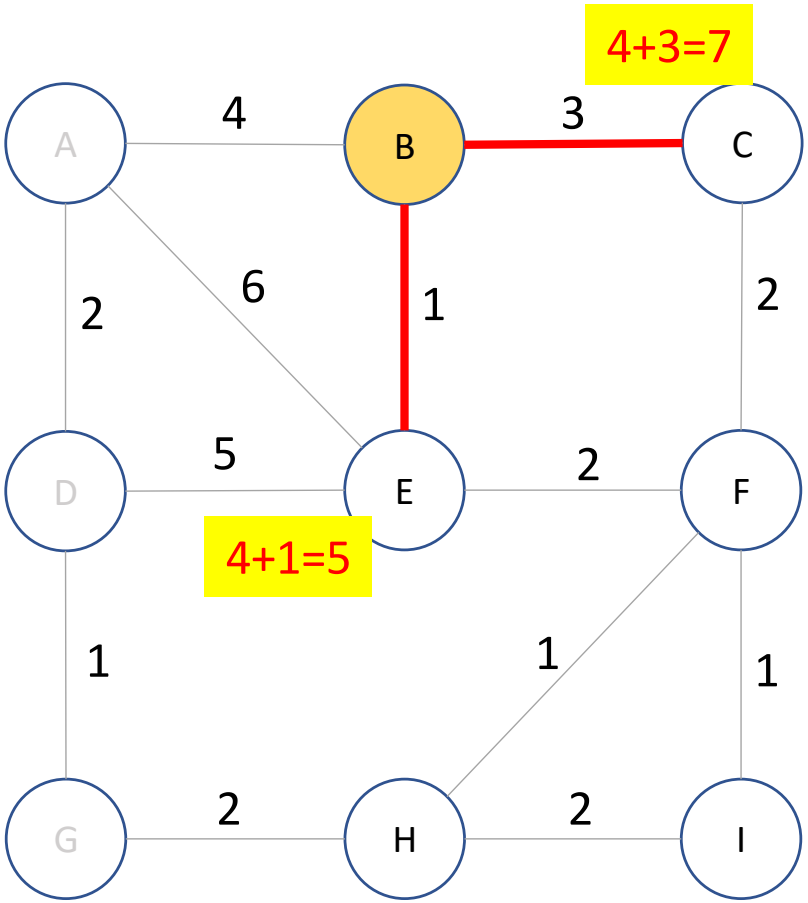
Visited = [A, D, G]

Unvisited = [B, C, E, F, H, I]

2 For the current vertex, if it is not the destination, examine the unvisited neighbours

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	3	D
H	5	G
I	∞	

Dijkstra's Algorithm (from A to F)



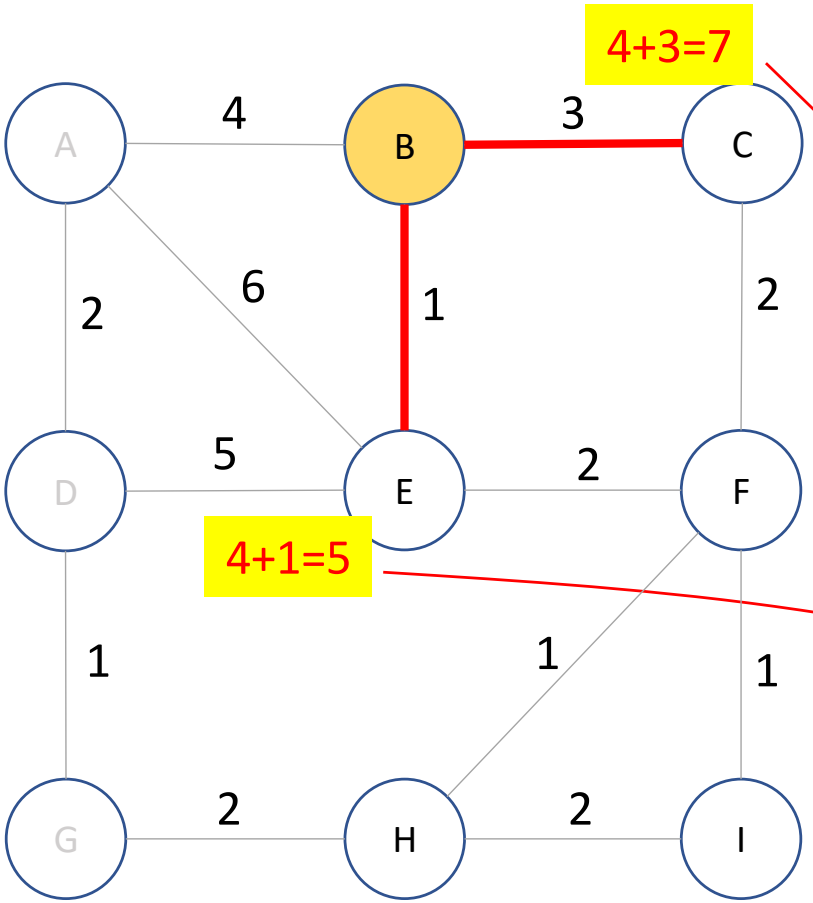
Visited = [A, D, G]

Unvisited = [B, C, E, F, H, I]

3 Calculate the cost of each neighbour from the start vertex

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	∞	
D	2	A
E	6	A
F	∞	
G	3	D
H	5	G
I	∞	

Dijkstra's Algorithm (from A to F)



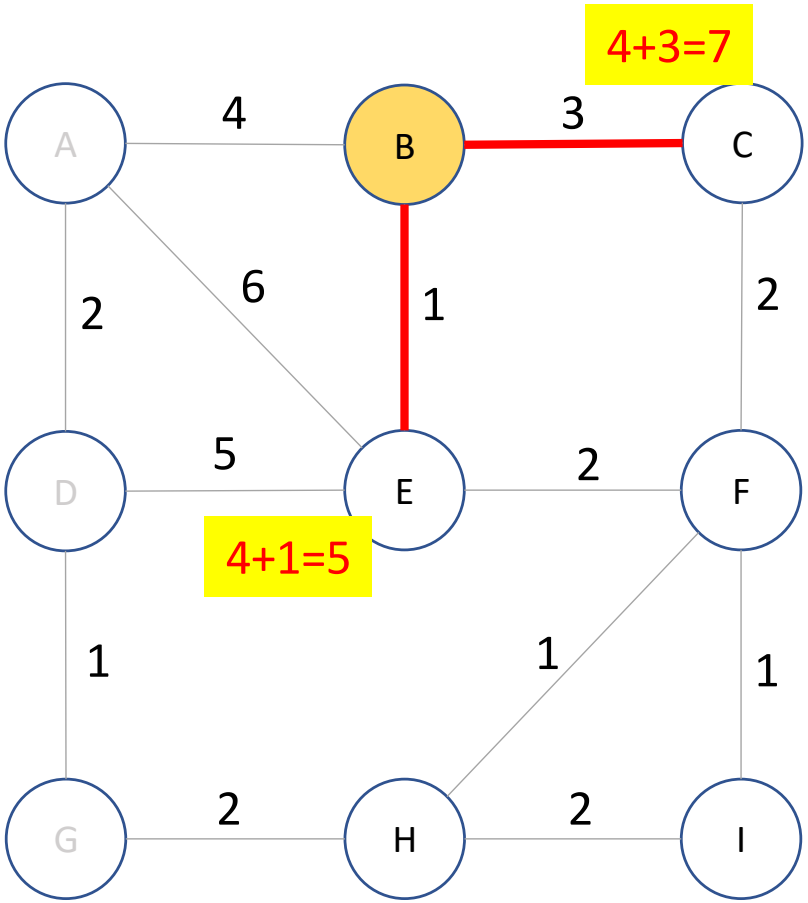
4 If the calculated cost is less than the known cost, update the lowest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	
D	2	A
E	5	A
F	∞	
G	3	D
H	5	G
I	∞	

Visited = [A, D, G]

Unvisited = [B, C, E, F, H, I]

Dijkstra's Algorithm (from A to F)



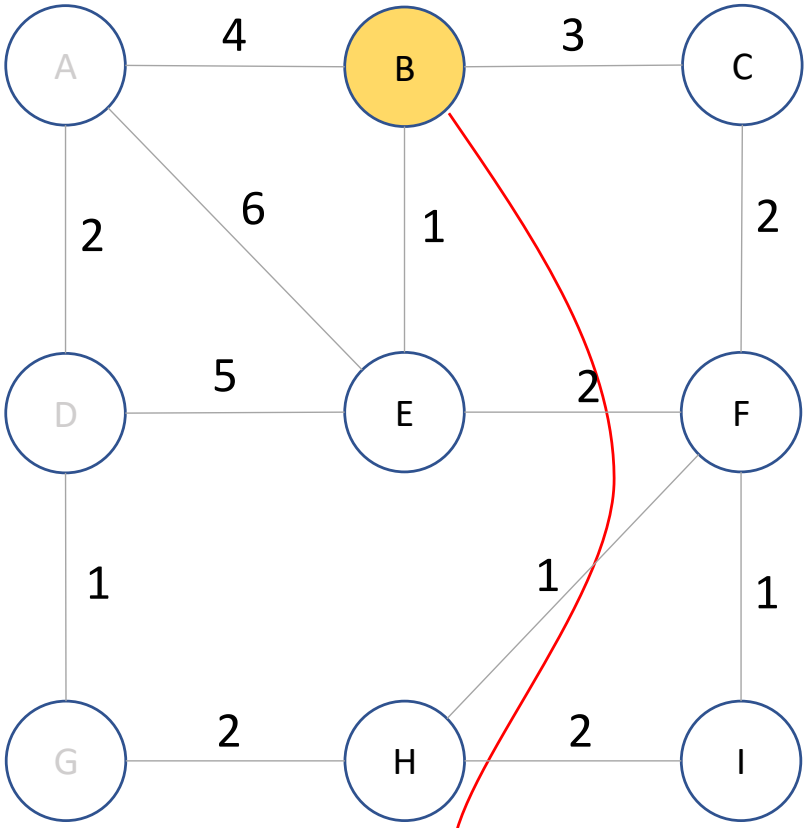
Visited = [A, D, G]

Unvisited = [B, C, E, F, H, I]

5 Update the previous vertex for each updated lowest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	∞	
G	3	D
H	5	G
I	∞	

Dijkstra's Algorithm (from A to F)



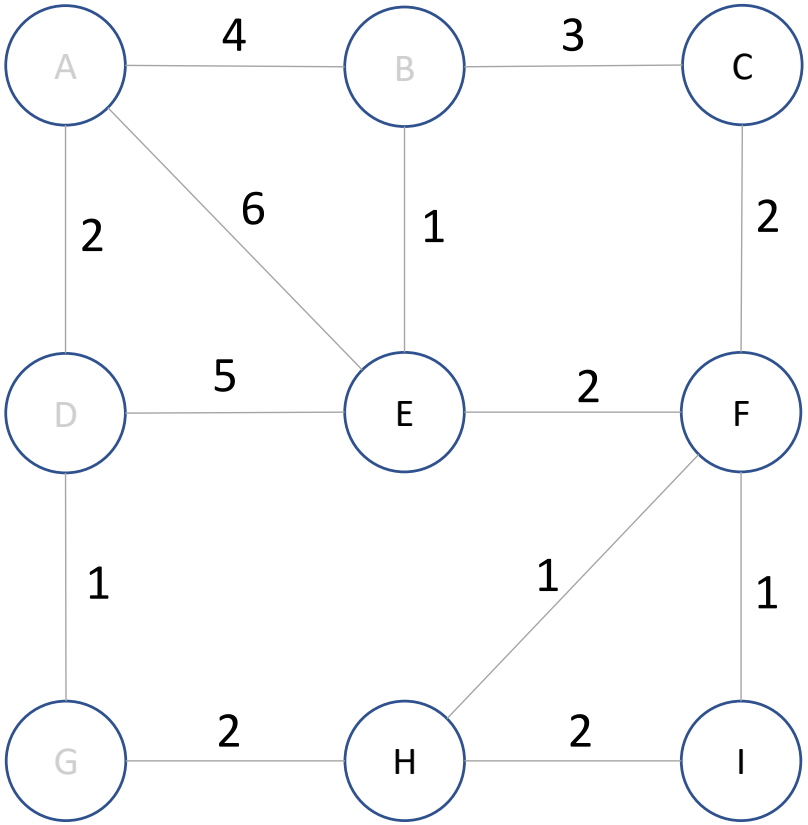
Visited = [A, D, G, B]

Unvisited = [C, E, F, H, I]

6 Add the current vertex to the list of visited vertices

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	∞	
G	3	D
H	5	G
I	∞	

Dijkstra's Algorithm (from A to F)

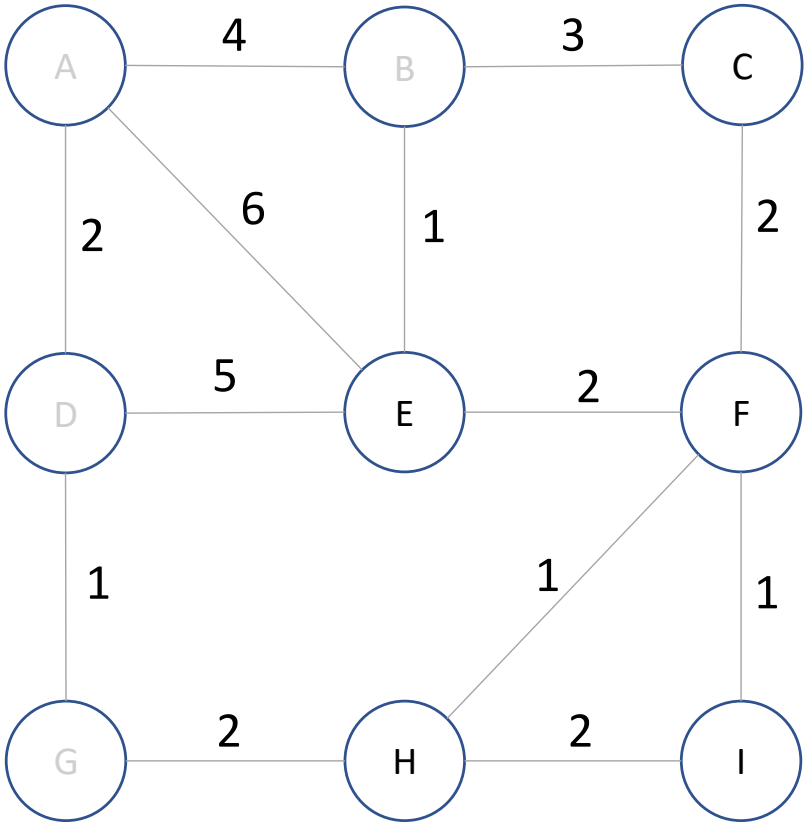


Visited = [A, D, G, B]

Unvisited = [C, E, F, H, I]

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	∞	
G	3	D
H	5	G
I	∞	

Dijkstra's Algorithm (from A to F)



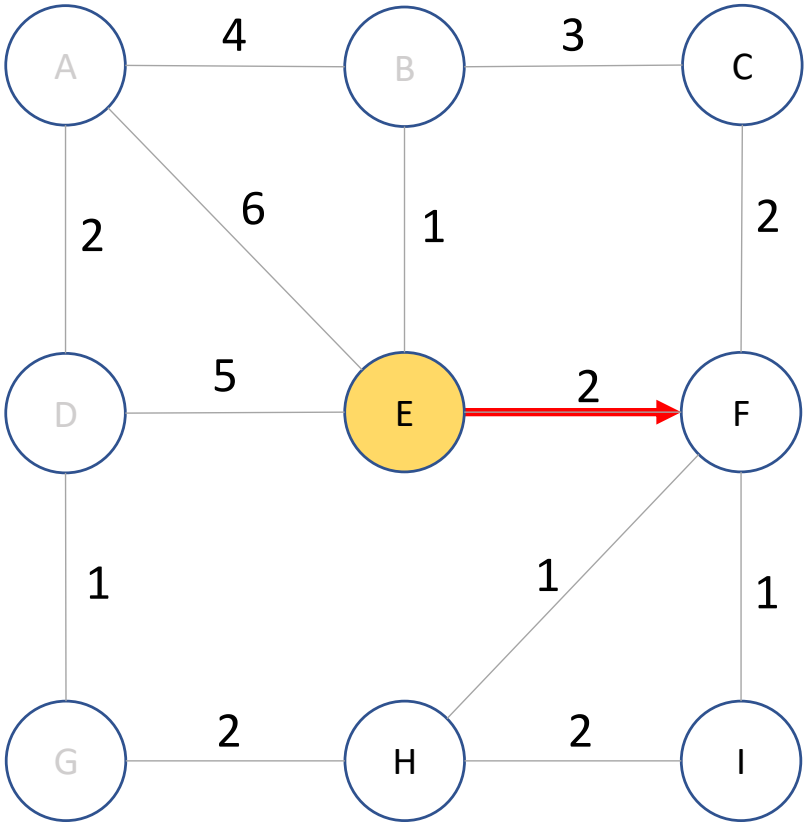
1 Visit the unvisited vertex with the smallest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	∞	
G	3	D
H	5	G
I	∞	

Visited = [A, D, G, B]

Unvisited = [C, E, F, H, I]

Dijkstra's Algorithm (from A to F)



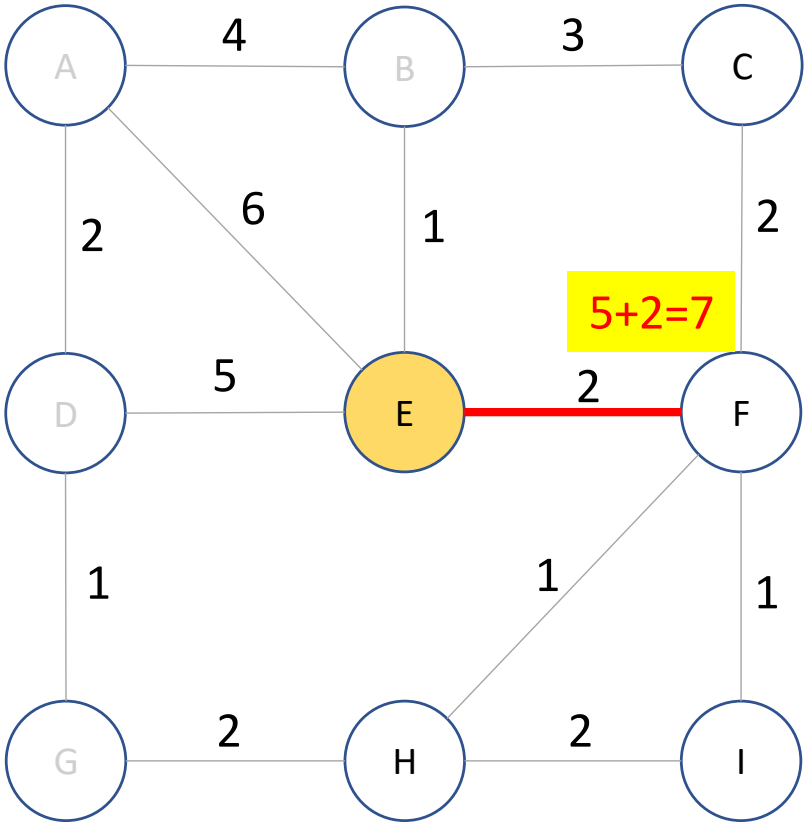
Visited = [A, D, G, B]

Unvisited = [C, E, F, H, I]

2 For the current vertex, if it is not the destination, examine the unvisited neighbours

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	∞	
G	3	D
H	5	G
I	∞	

Dijkstra's Algorithm (from A to F)



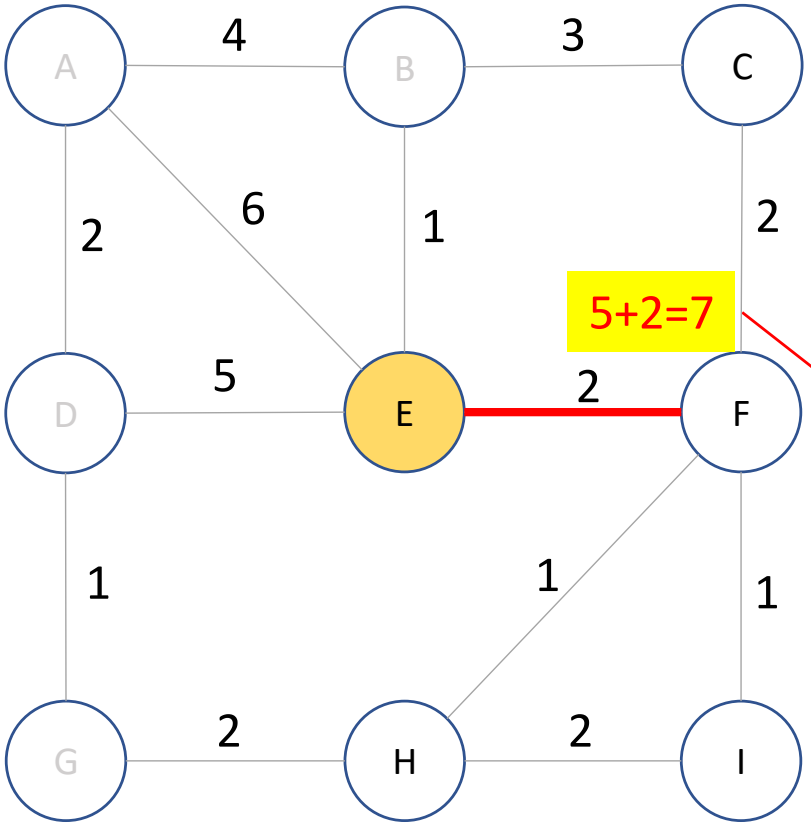
Visited = [A, D, G, B]

Unvisited = [C, E, F, H, I]

3 Calculate the cost of each neighbour from the start vertex

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	∞	
G	3	D
H	5	G
I	∞	

Dijkstra's Algorithm (from A to F)



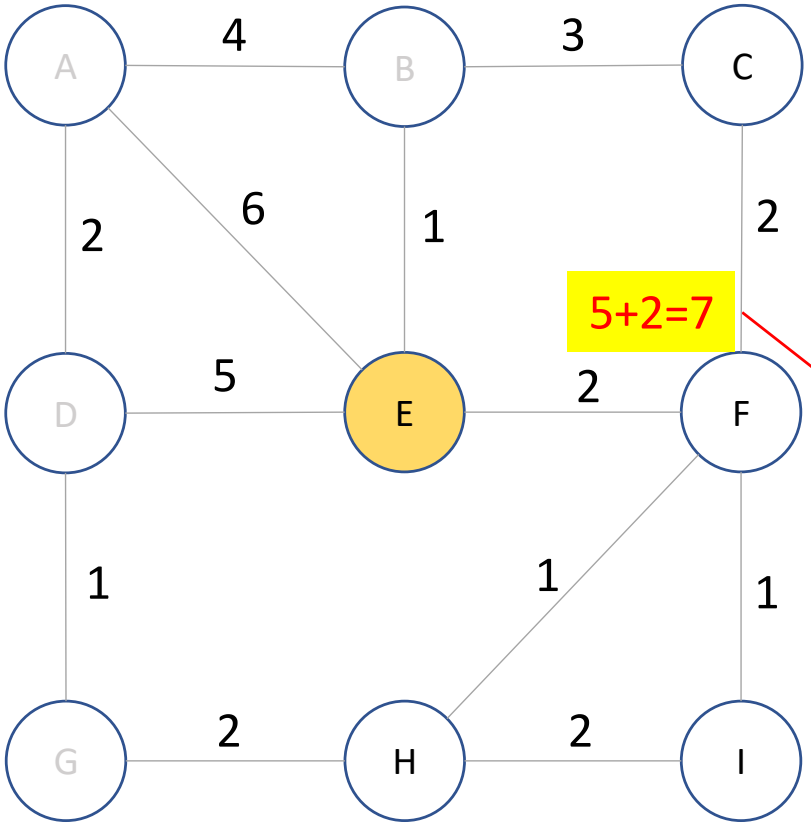
4 If the calculated cost is less than the known cost, update the lowest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	7	
G	3	D
H	5	G
I	∞	

Visited = [A, D, G, B]

Unvisited = [C, E, F, H, I]

Dijkstra's Algorithm (from A to F)



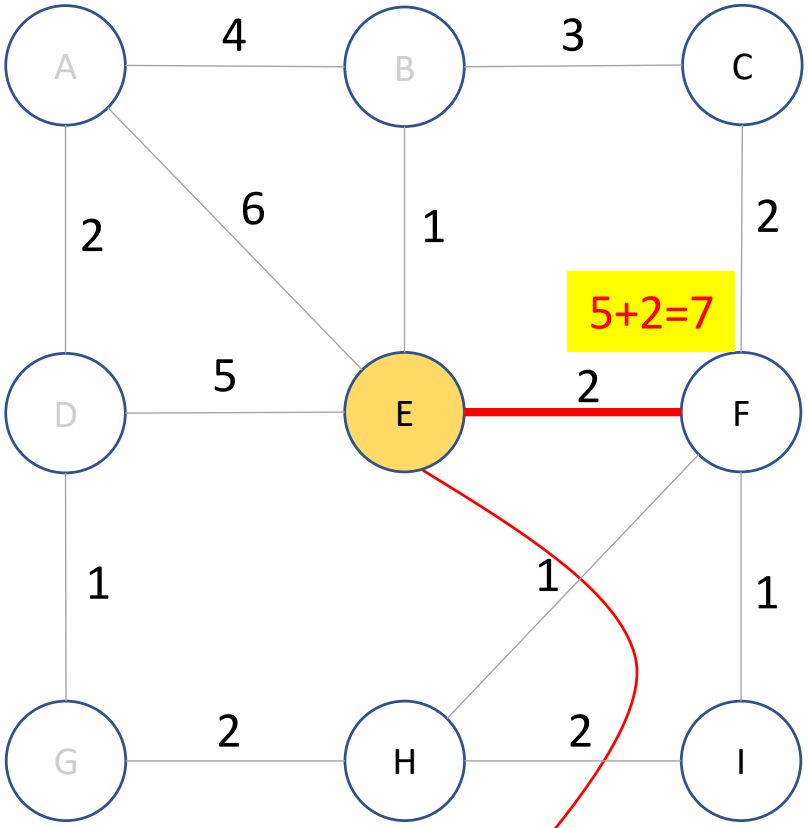
5 Update the previous vertex for each updated lowest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	7	E
G	3	D
H	5	G
I	∞	

Visited = [A, D, G, B]

Unvisited = [C, E, F, H, I]

Dijkstra's Algorithm (from A to F)



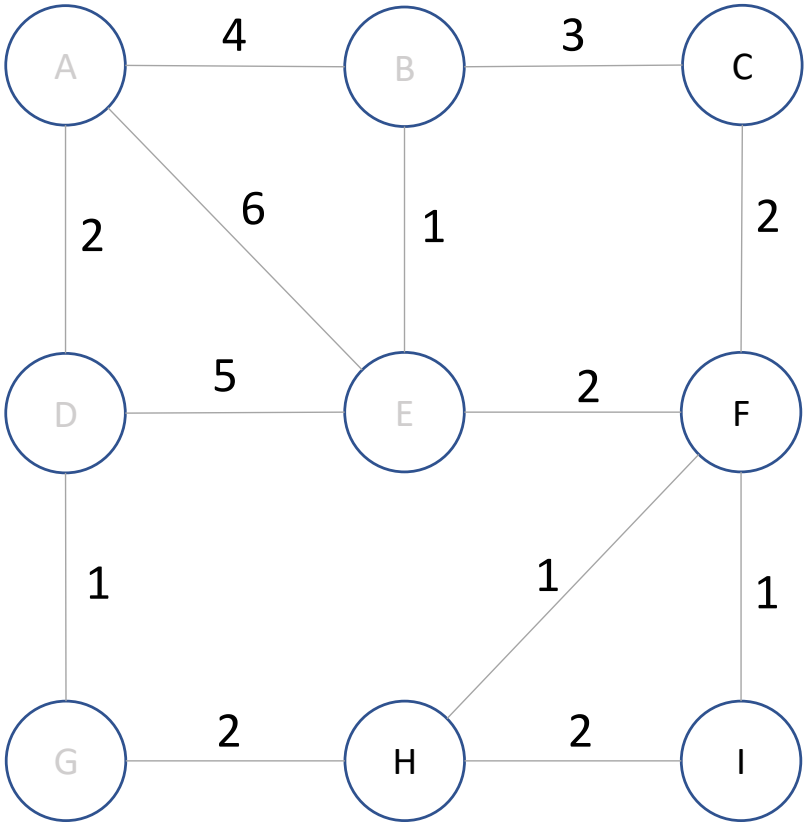
Visited = [A, D, G, B, E]

Unvisited = [C, F, H, I]

6 Add the current vertex to the list of visited vertices

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	7	E
G	3	D
H	5	G
I	∞	

Dijkstra's Algorithm (from A to F)

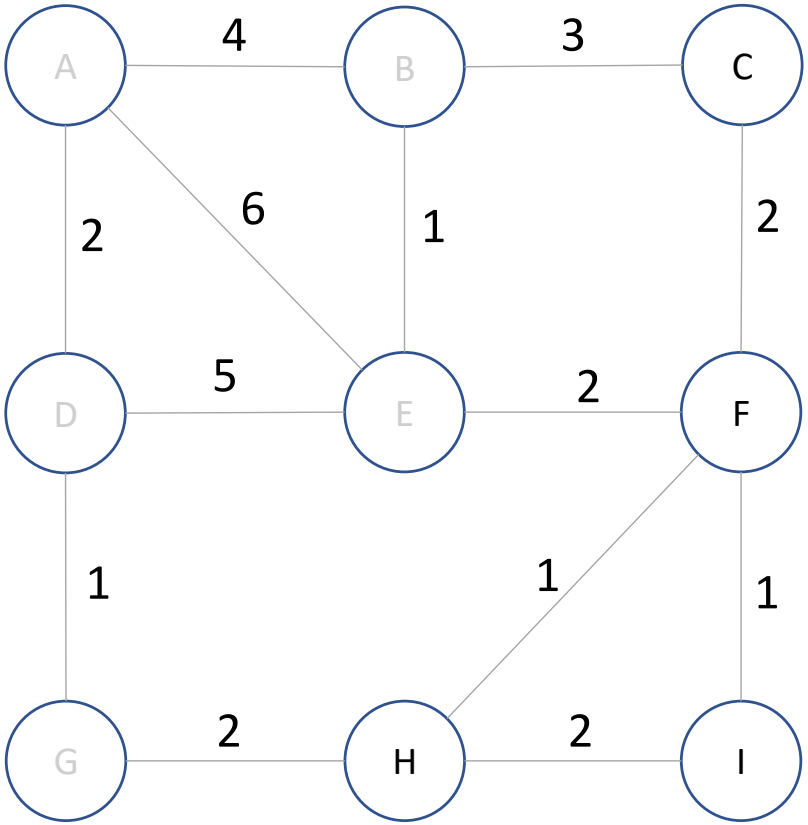


Visited = [A, D, G, B, E]

Unvisited = [C, F, H, I]

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	7	E
G	3	D
H	5	G
I	∞	

Dijkstra's Algorithm (from A to F)



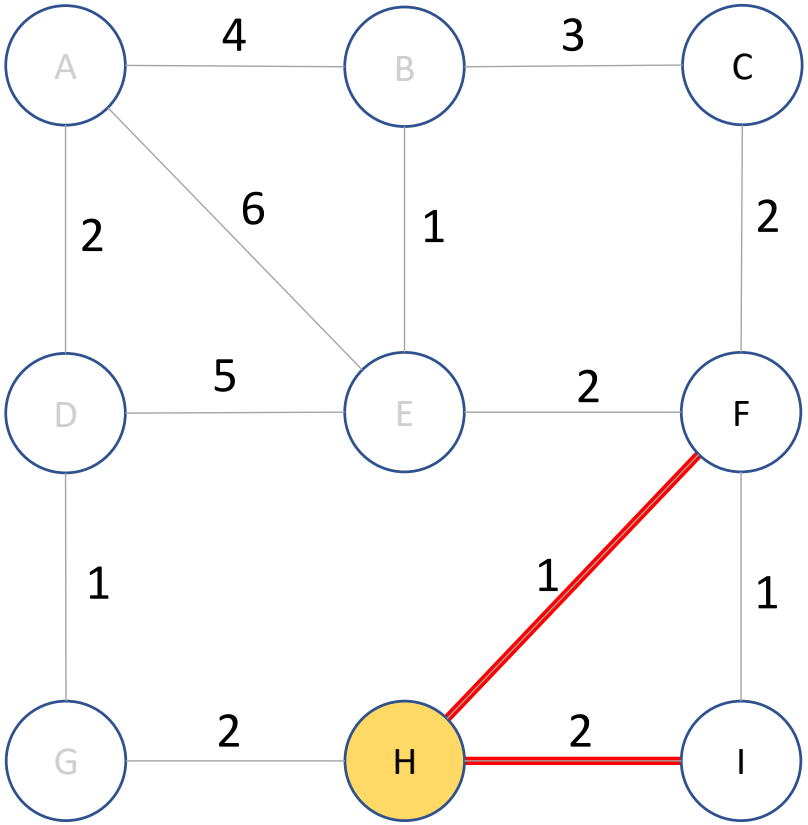
Visited = [A, D, G, B, E]

Unvisited = [C, F, H, I]

1 Visit the unvisited vertex with the smallest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	7	E
G	3	D
H	5	G
I	∞	

Dijkstra's Algorithm (from A to F)



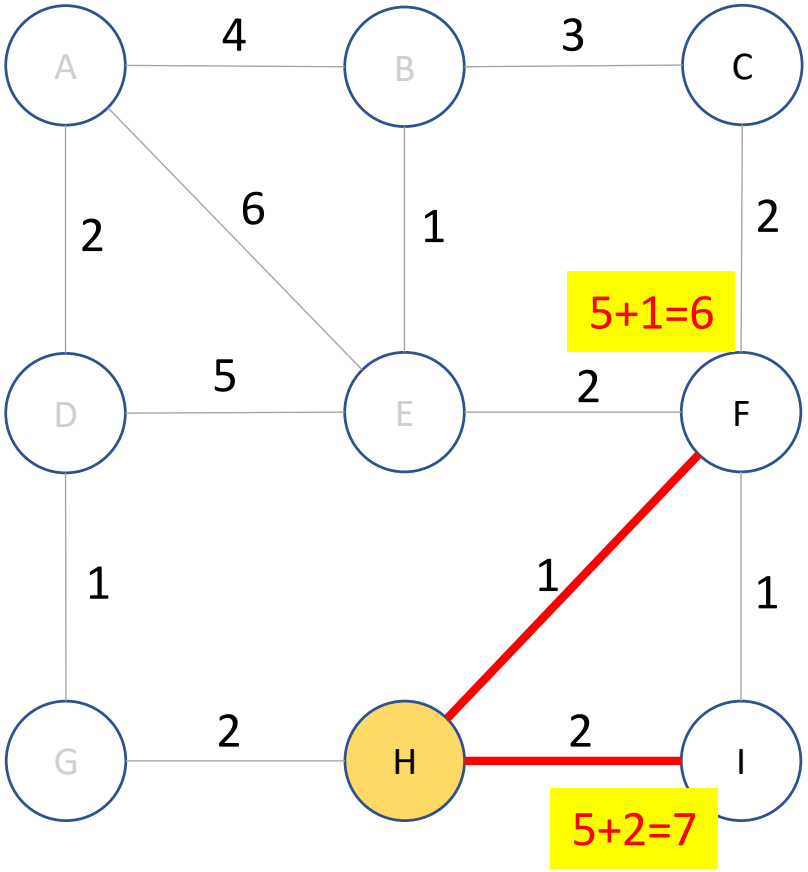
Visited = [A, D, G, B, E]

Unvisited = [C, F, H, I]

2 For the current vertex, if it is not the destination, examine the unvisited neighbours

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	7	E
G	3	D
H	5	G
I	∞	

Dijkstra's Algorithm (from A to F)



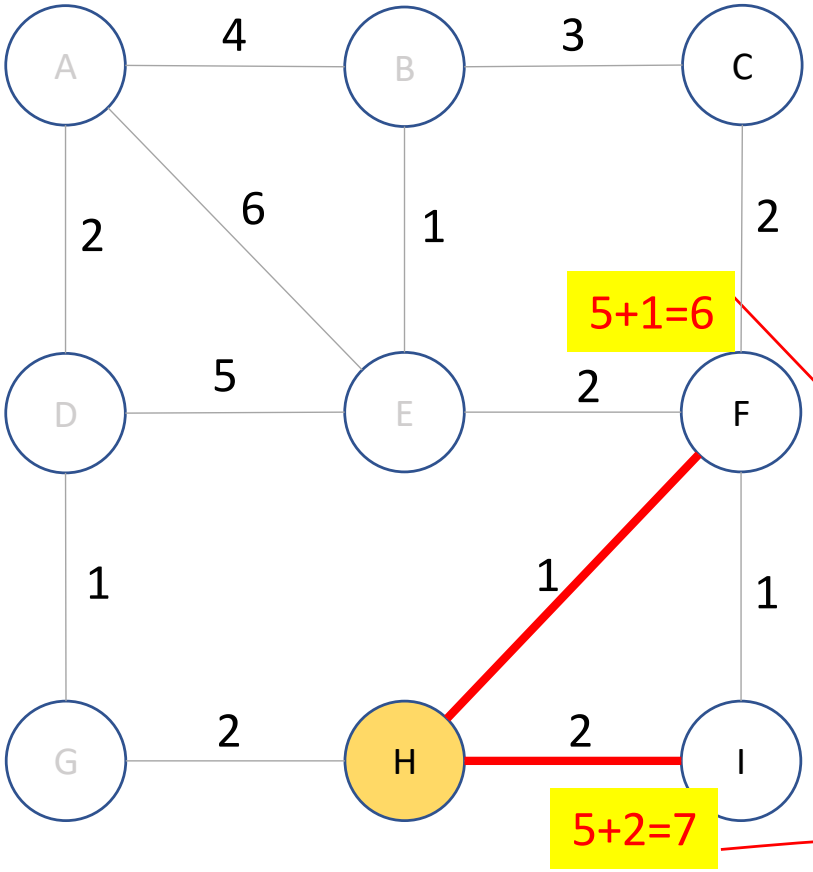
Visited = [A, D, G, B, E]

Unvisited = [C, F, H, I]

3 Calculate the cost of each neighbour from the start vertex

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	7	E
G	3	D
H	5	G
I	∞	

Dijkstra's Algorithm (from A to F)



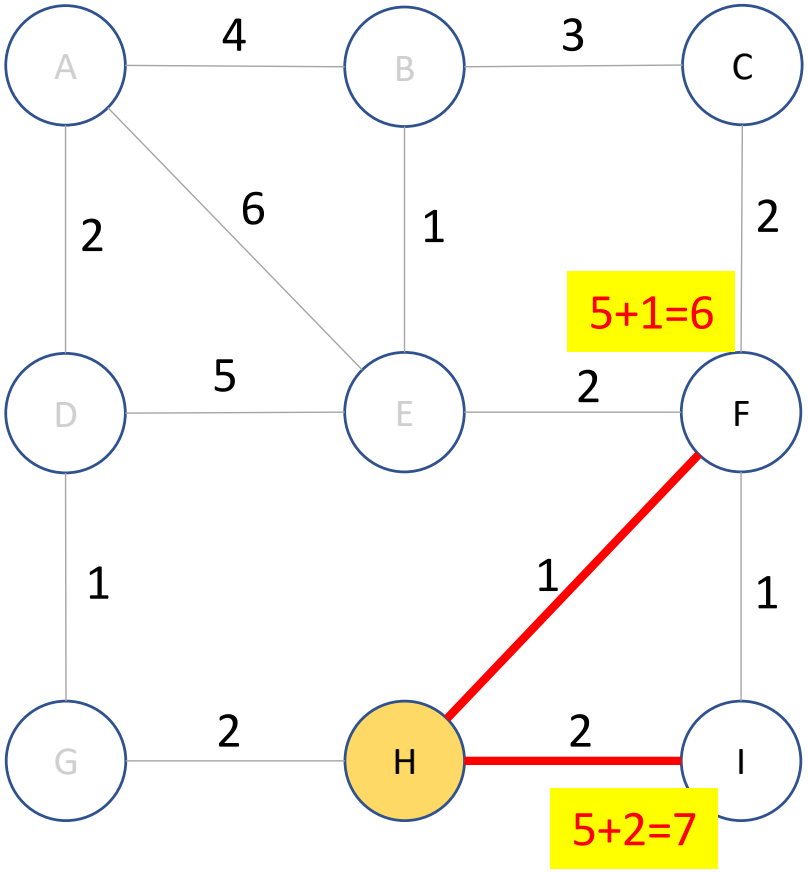
4 If the calculated cost is less than the known cost, update the lowest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	6	E
G	3	D
H	5	G
I	7	

Visited = [A, D, G, B, E]

Unvisited = [C, F, H, I]

Dijkstra's Algorithm (from A to F)



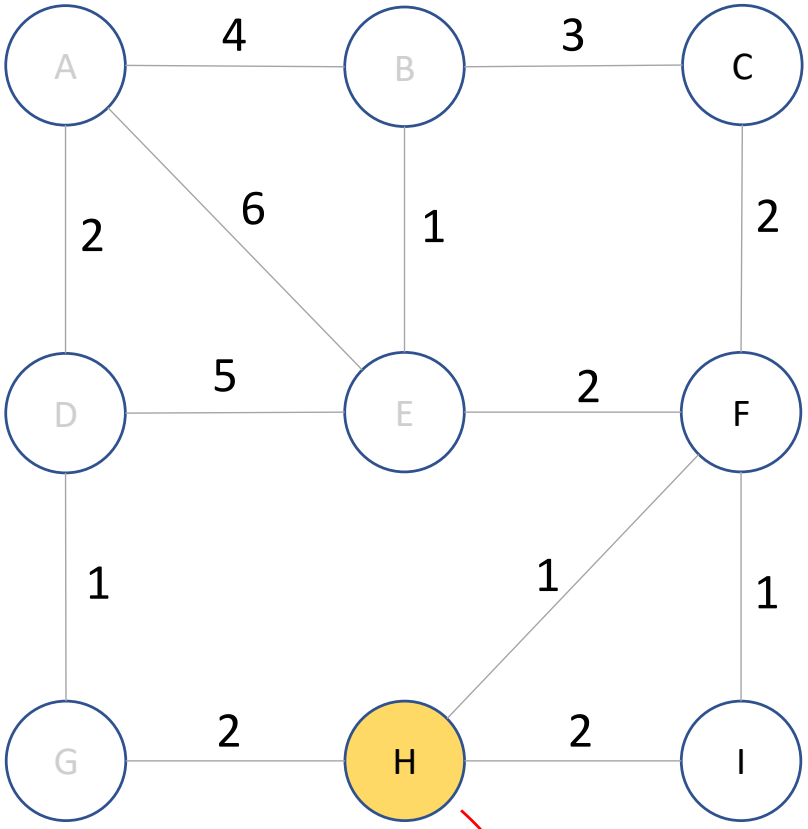
Visited = [A, D, G, B, E]

Unvisited = [C, F, H, I]

5 Update the previous vertex for each updated lowest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	6	H
G	3	D
H	5	G
I	7	H

Dijkstra's Algorithm (from A to F)



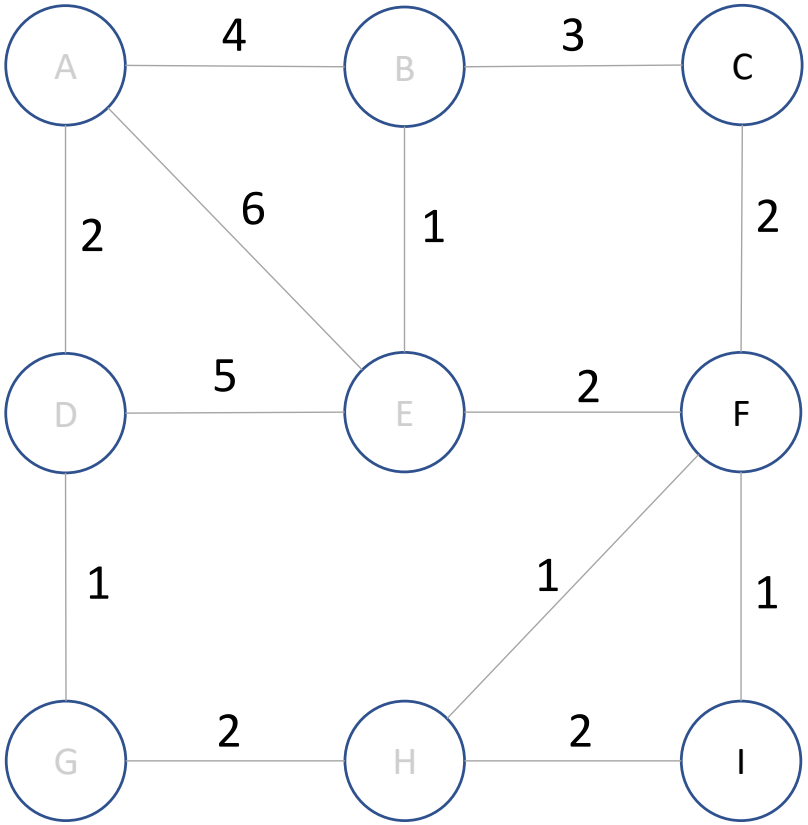
Visited = [A, D, G, B, E, H]

Unvisited = [C, F, I]

6 Add the current vertex to the list of visited vertices

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	6	H
G	3	D
H	5	G
I	7	H

Dijkstra's Algorithm (from A to F)

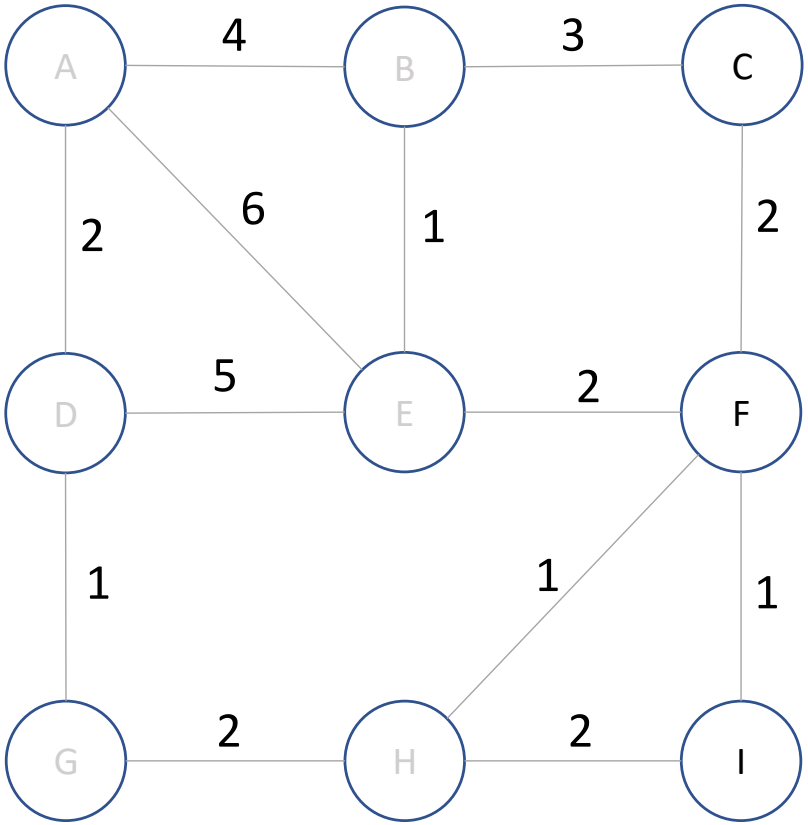


Visited = [A, D, G, B, E, H]

Unvisited = [C, F, I]

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	6	H
G	3	D
H	5	G
I	7	H

Dijkstra's Algorithm (from A to F)



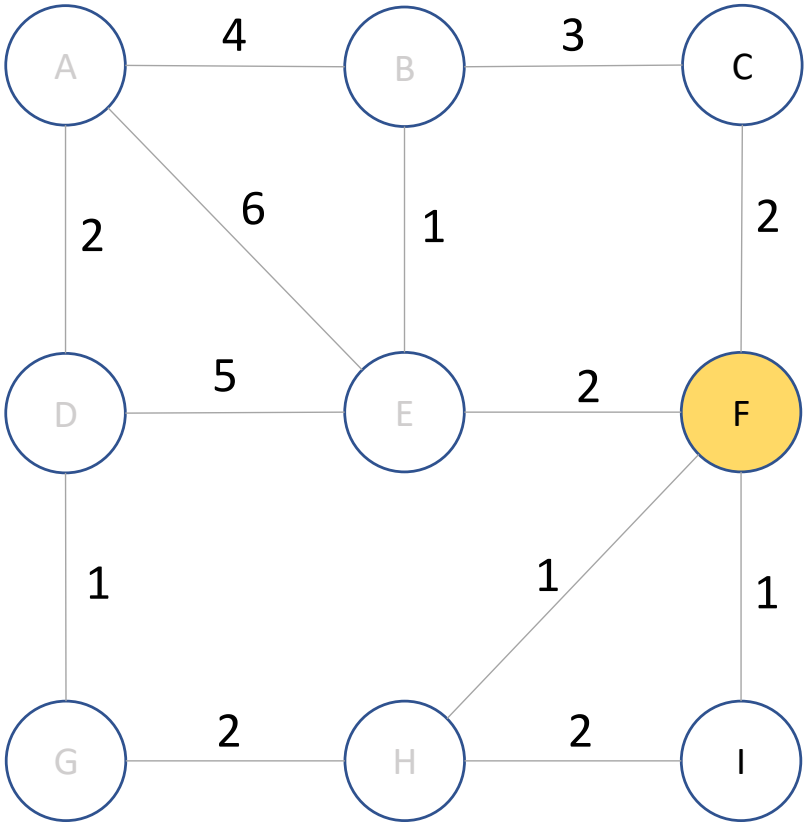
1 Visit the unvisited vertex with the smallest cost

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	6	H
G	3	D
H	5	G
I	7	H

Visited = [A, D, G, B, E, H]

Unvisited = [C, F, I]

Dijkstra's Algorithm (from A to F)



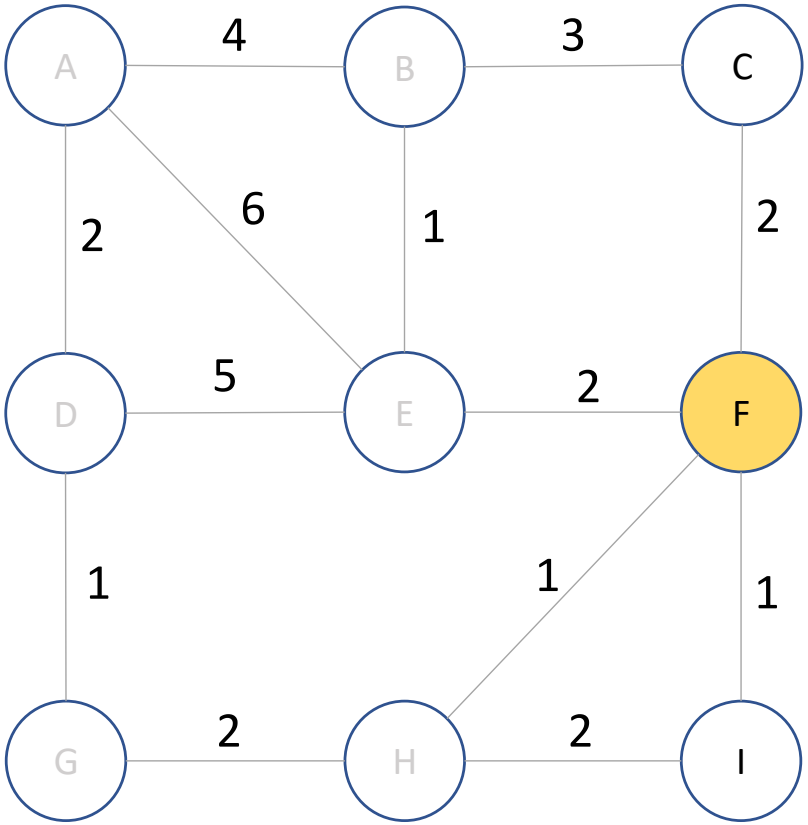
Visited = [A, D, G, B, E, H]

Unvisited = [C, F, I]

2 For the current vertex, if it is not the destination, examine the unvisited neighbours

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	6	H
G	3	D
H	5	G
I	7	H

Dijkstra's Algorithm (from A to F)

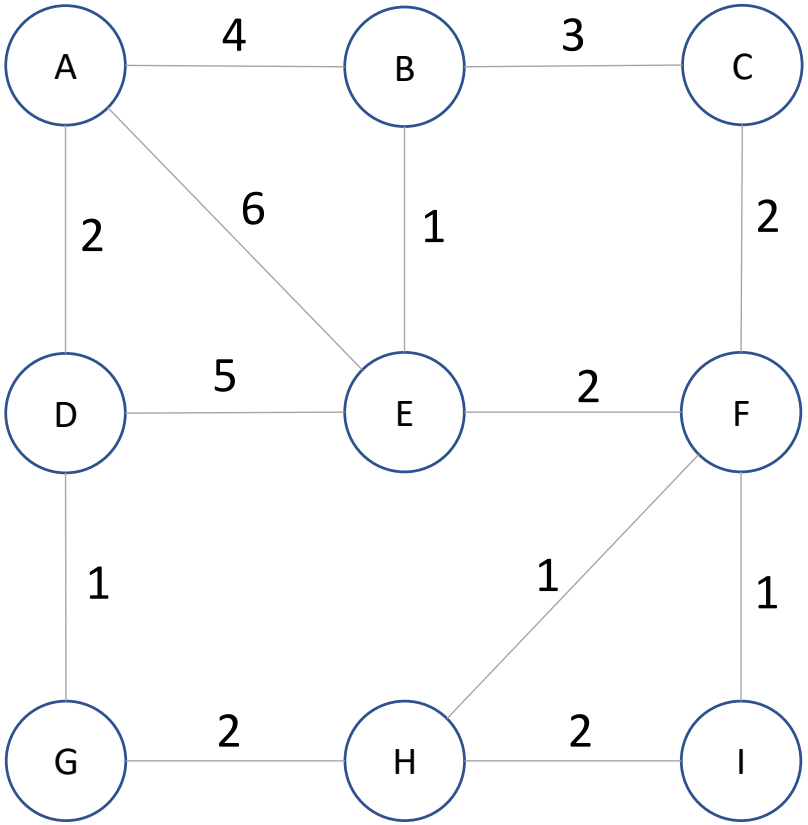


Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	6	H
G	3	D
H	5	G
I	7	H

Visited = [A, D, G, B, E, H]

Unvisited = [C, F, I]

DFS, BFS and Dijkstra's



A → F

DFS: A → B → C → F

Vertices: 4, Cost: 9

BFS: A → E → F

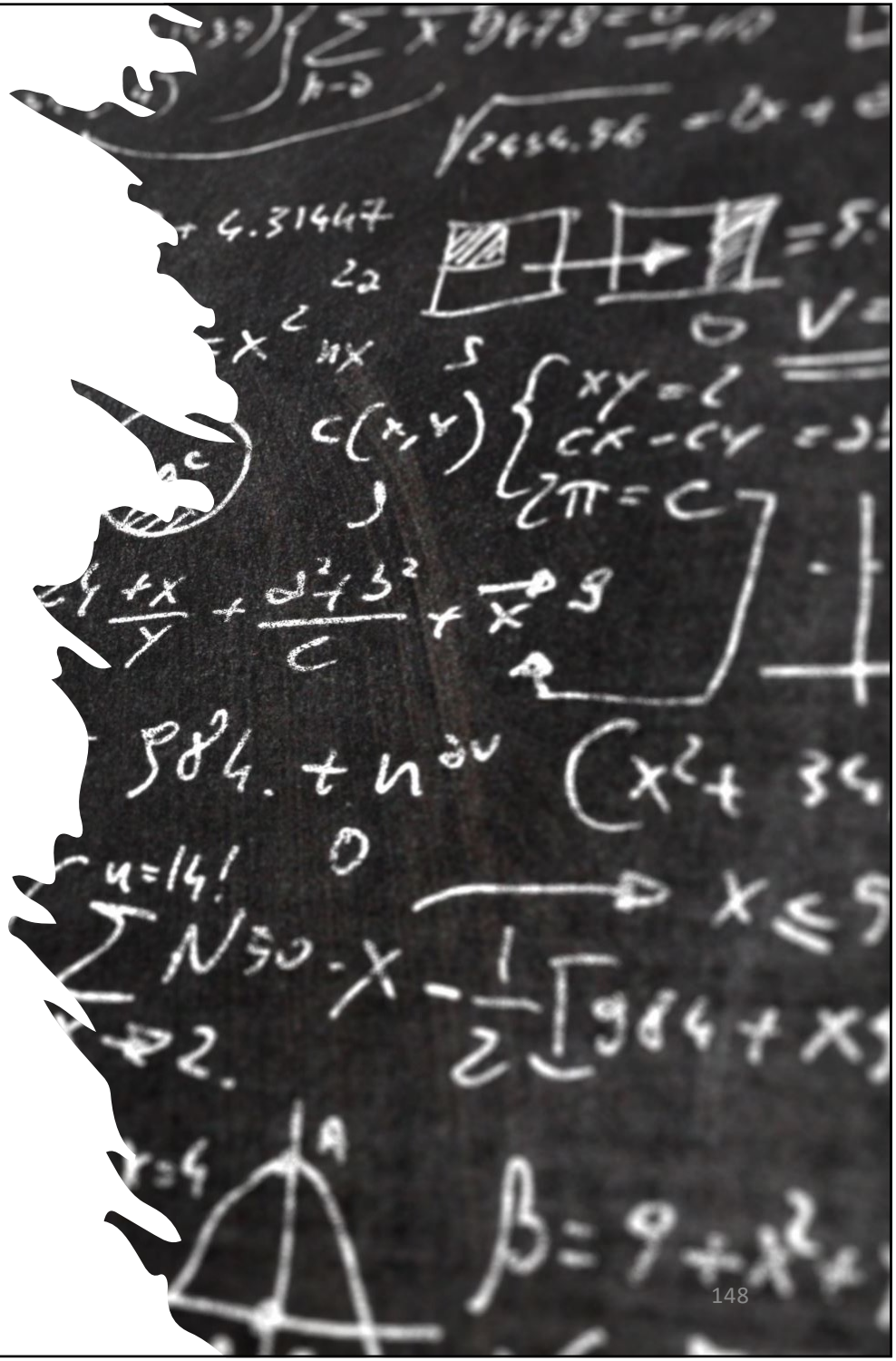
Vertices: 3, Cost: 8

Dijkstra's: A → D → G → H → F

Vertices: 5, Cost: 6

How Dijkstra's Algorithm Works

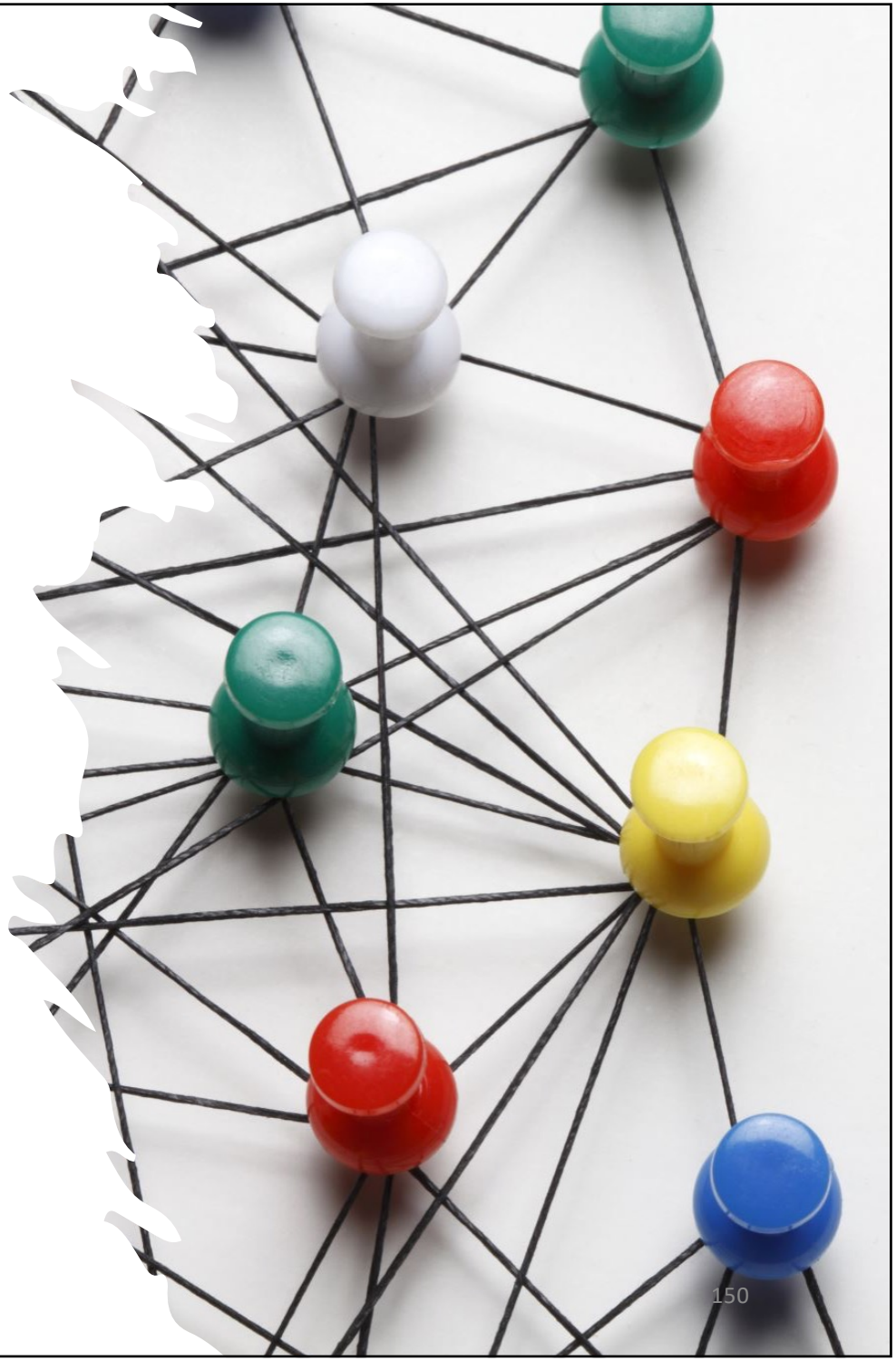
- It follows the path the is currently cheapest.
- It does not pay attention to the distance that it is going.



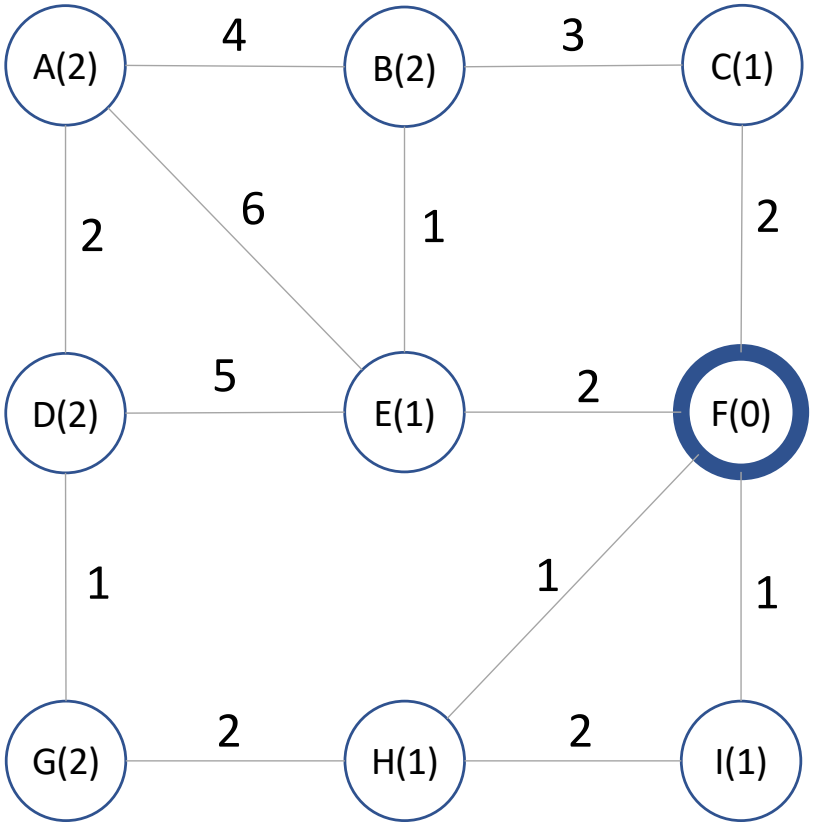

A-Star Algorithm

Concept of A-Star Algorithm

- Create a table of information about the currently known best way to reach each vertex in term of cost and distance.
- Update the table when an improved path is identified.
- Continue doing so until it reaches the best solution.

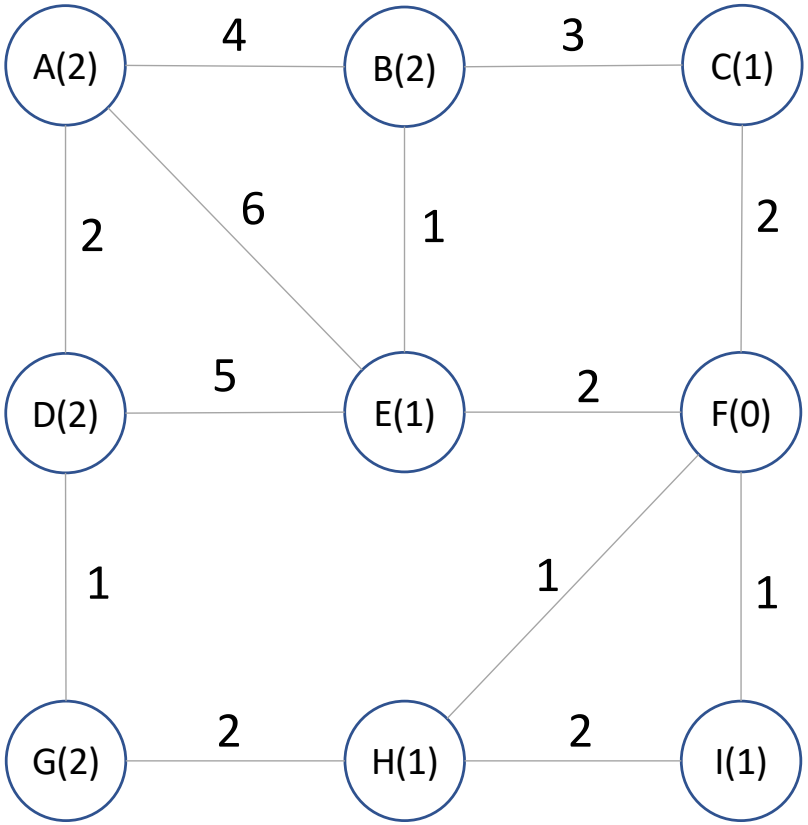


Map with Cost and Primitive (Distance) Measure Information



Vertex	Primitive (distance) measure
A	2
B	2
C	1
D	2
E	1
F	0
G	2
H	1
I	1

A-Star Algorithm

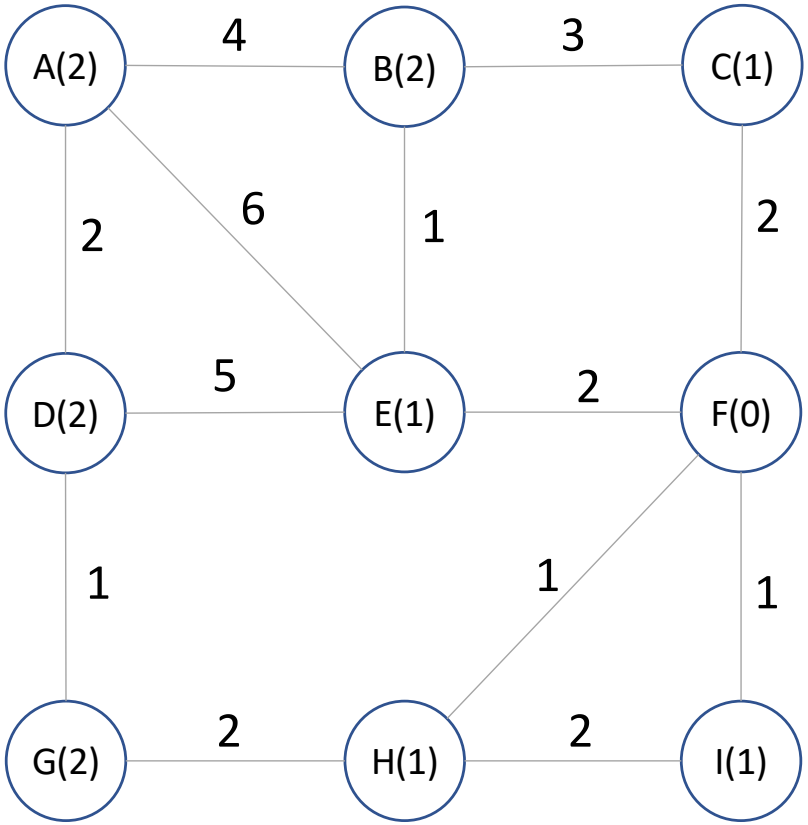


Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex

Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

A-Star Algorithm (from A to F)

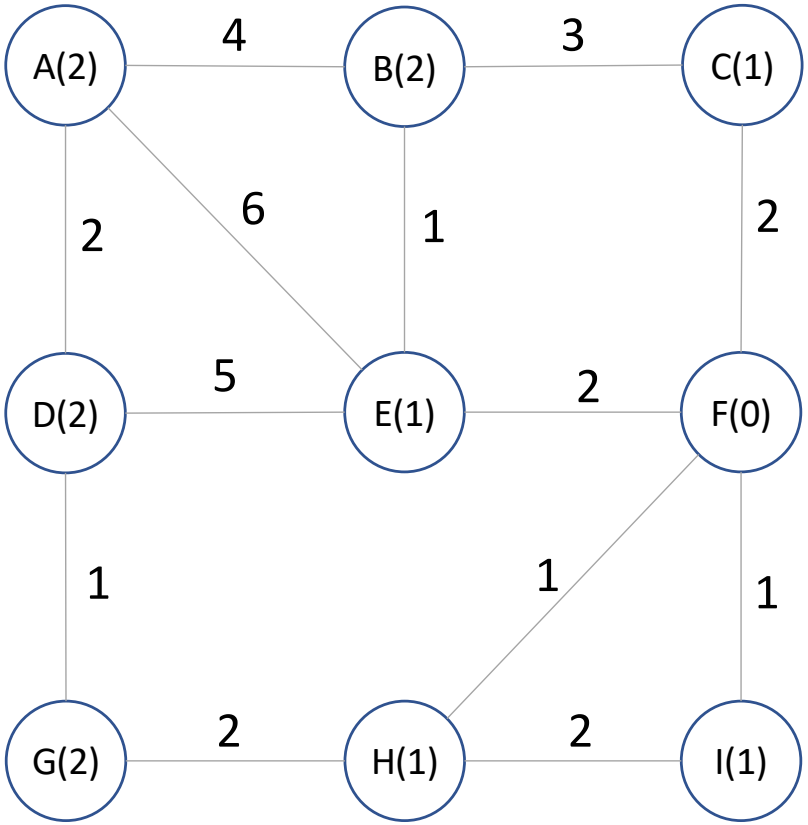


Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	

Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

A-Star Algorithm (from A to F)

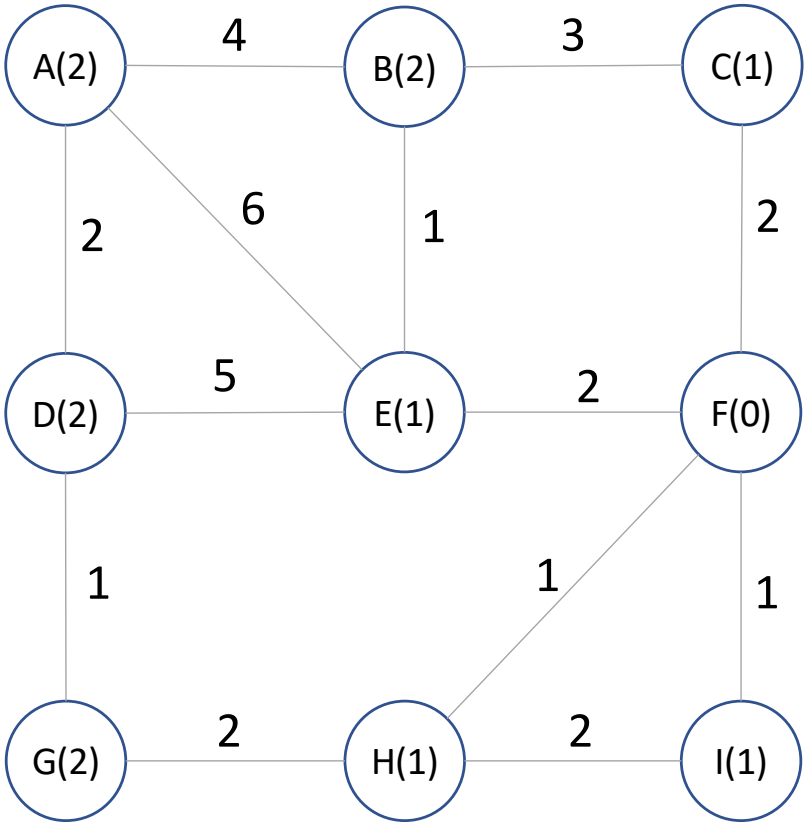


Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	∞	∞	
C	∞	∞	
D	∞	∞	
E	∞	∞	
F	∞	∞	
G	∞	∞	
H	∞	∞	
I	∞	∞	

Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

A-Star Algorithm (from A to F)



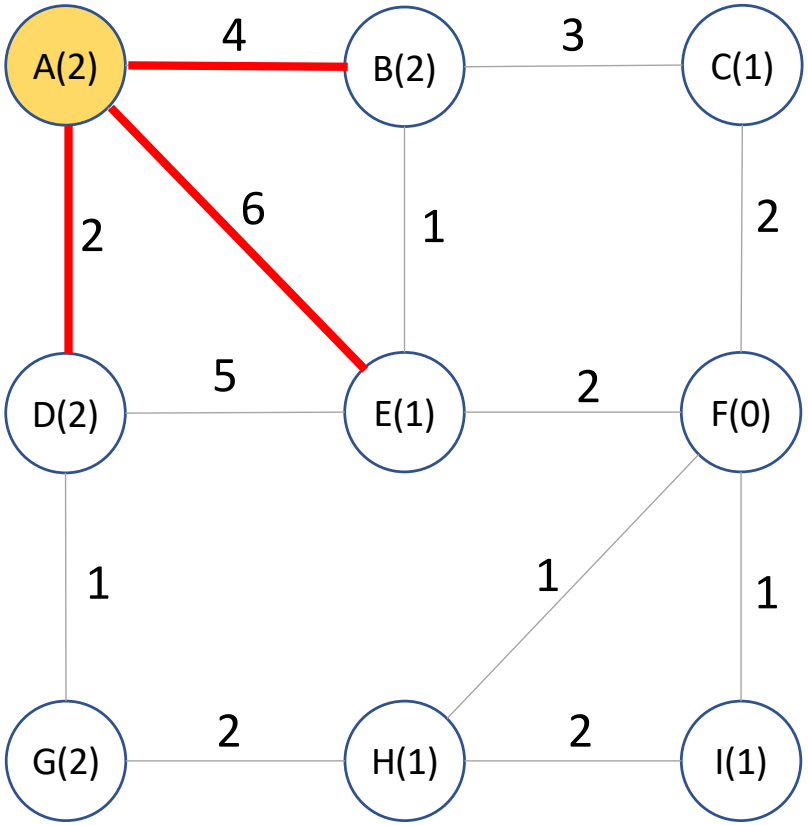
Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

1 Visit the unvisited vertex with the smallest cost

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	∞	∞	
C	∞	∞	
D	∞	∞	
E	∞	∞	
F	∞	∞	
G	∞	∞	
H	∞	∞	
I	∞	∞	

A-Star Algorithm (from A to F)



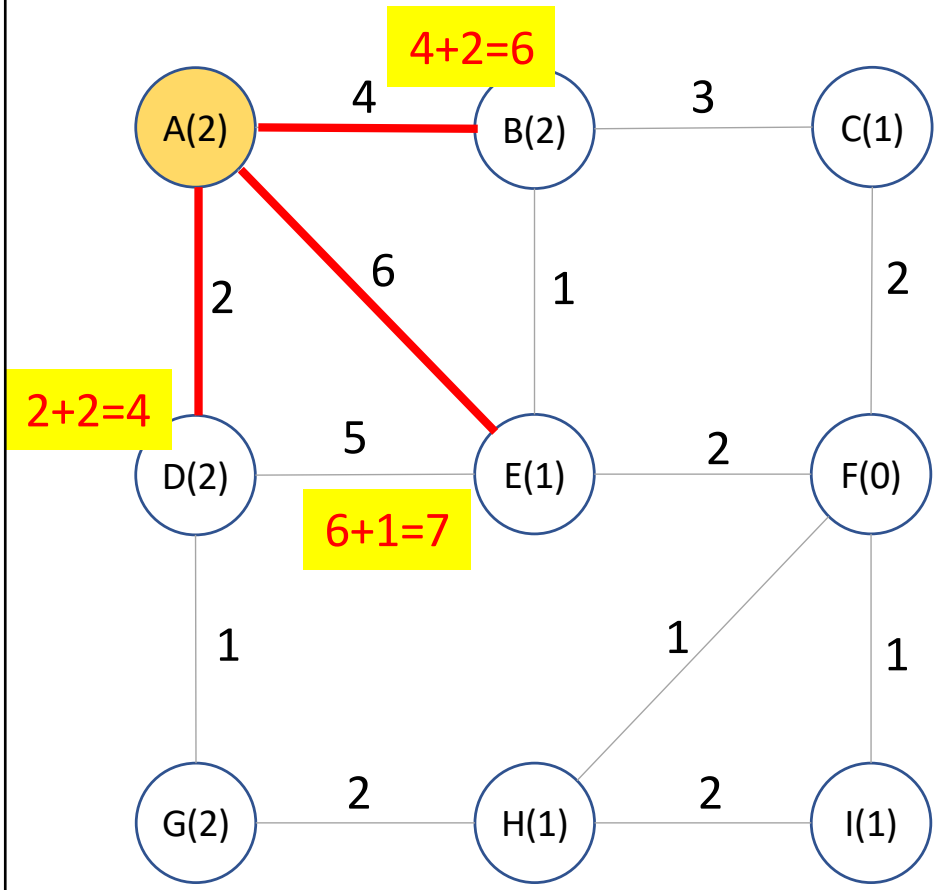
Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

2 For the current vertex, if it is not the destination, examine the unvisited neighbours

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	∞	∞	
C	∞	∞	
D	∞	∞	
E	∞	∞	
F	∞	∞	
G	∞	∞	
H	∞	∞	
I	∞	∞	

A-Star Algorithm (from A to F)



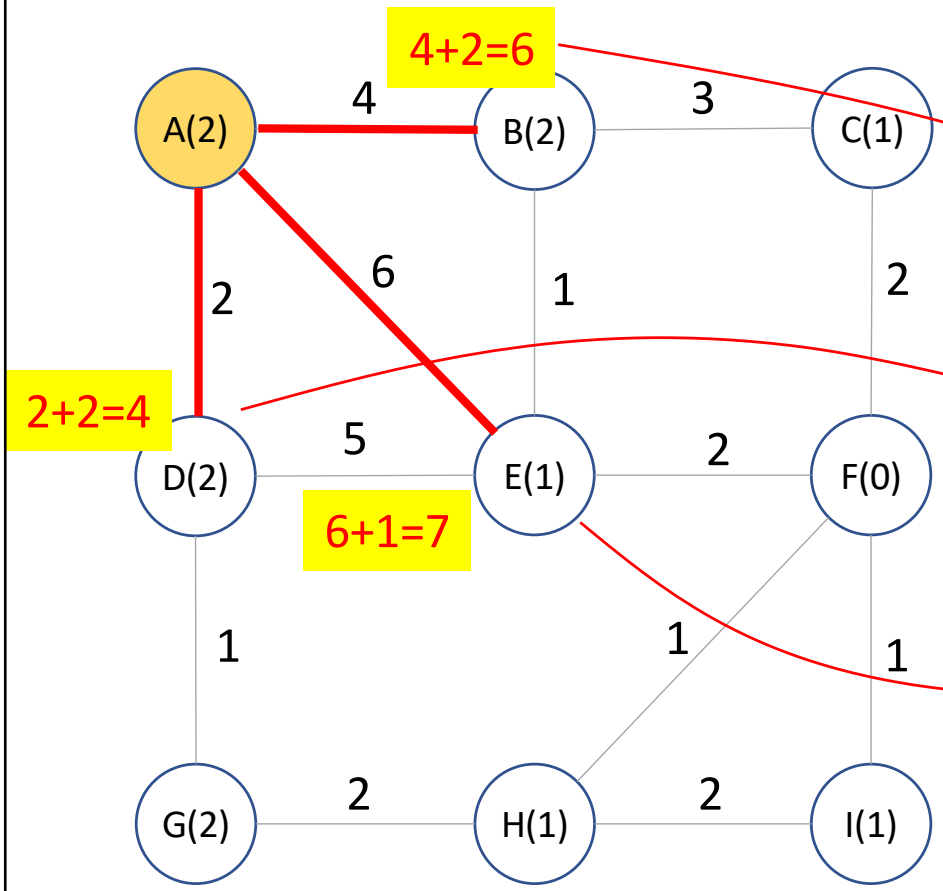
3 Calculate the cost of each neighbour

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	∞	∞	
C	∞	∞	
D	∞	∞	
E	∞	∞	
F	∞	∞	
G	∞	∞	
H	∞	∞	
I	∞	∞	

Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

A-Star Algorithm (from A to F)



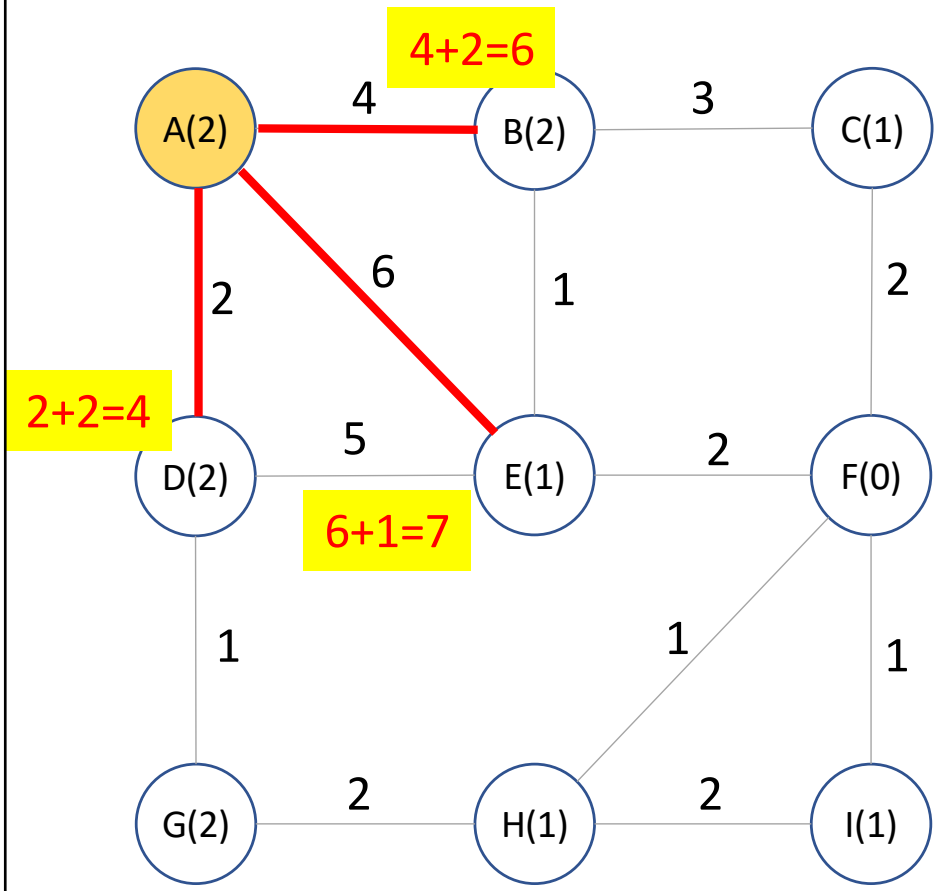
4 If the calculated cost is less than the known cost, update the lowest cost

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	
C	∞	∞	
D	2	4	
E	6	7	
F	∞	∞	
G	∞	∞	
H	∞	∞	
I	∞	∞	

Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

A-Star Algorithm (from A to F)



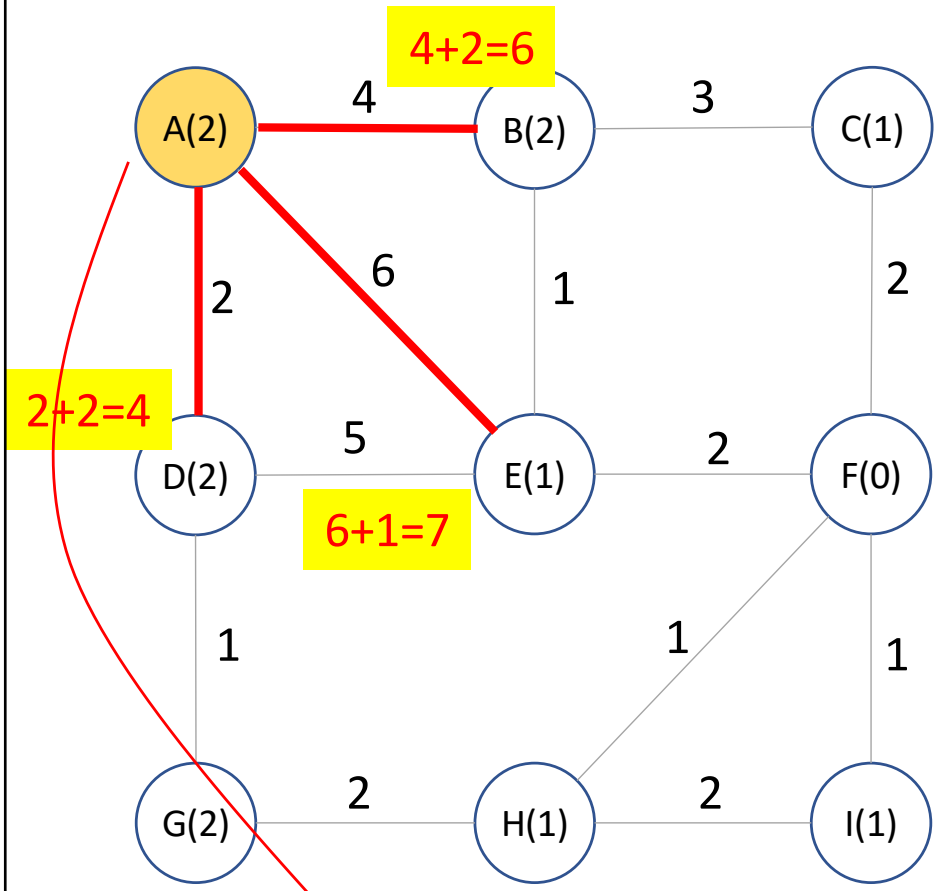
Visited = [ ]

Unvisited = [A, B, C, D, E, F, G, H, I]

5 Update the previous vertex for each updated lowest cost

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	∞	∞	
H	∞	∞	
I	∞	∞	

A-Star Algorithm (from A to F)



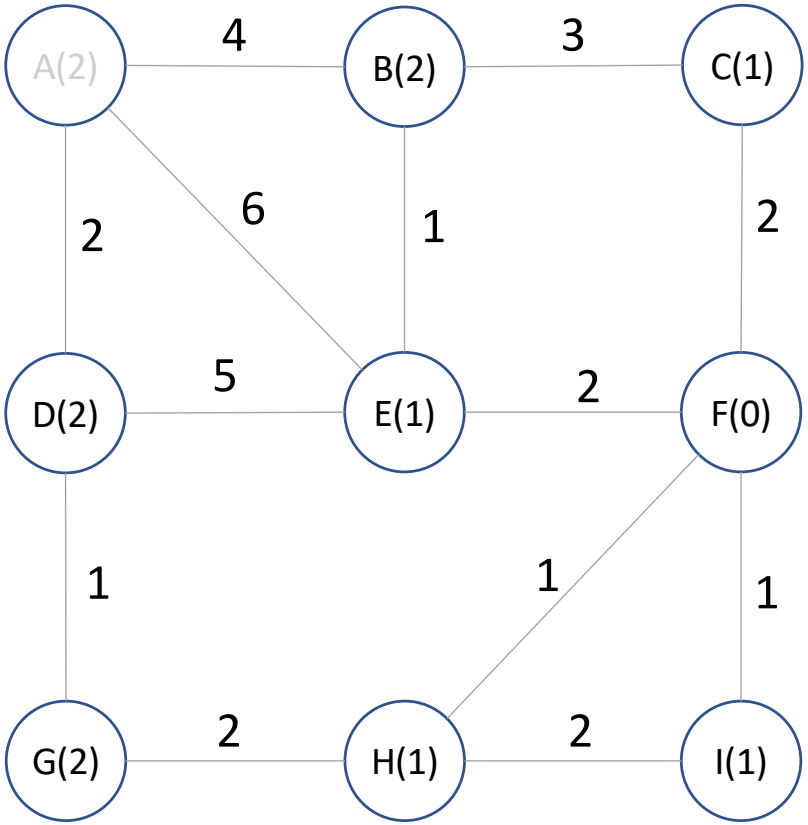
Visited = [A

Unvisited = [B, C, D, E, F, G, H, I]

6 Add the current vertex to the list of visited vertices

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	∞	∞	
H	∞	∞	
I	∞	∞	

A-Star Algorithm (from A to F)

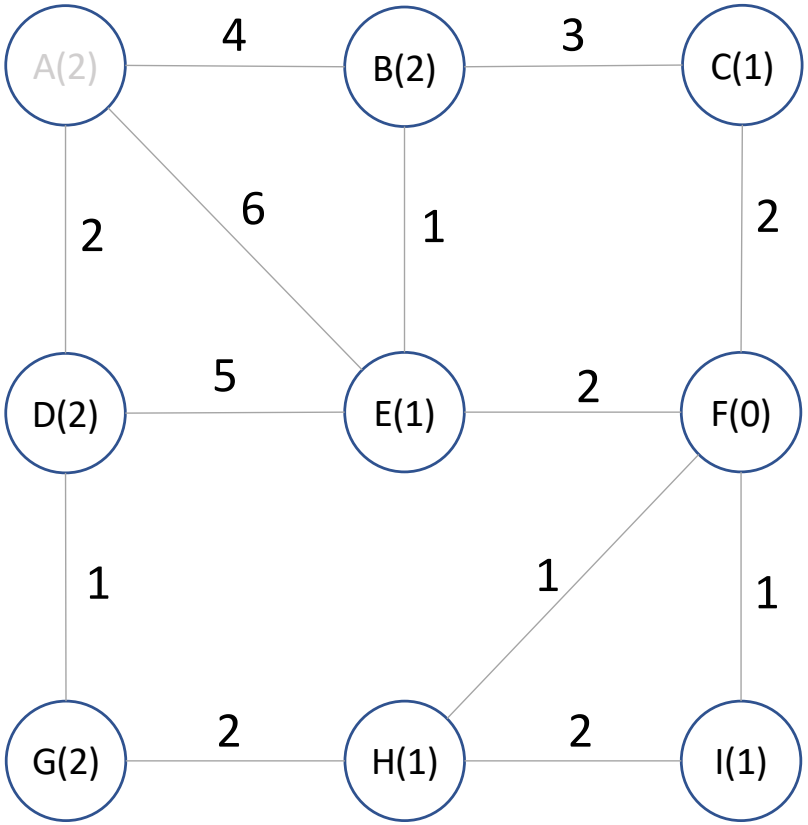


Visited = [A]

Unvisited = [B, C, D, E, F, G, H, I]

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	∞	∞	
H	∞	∞	
I	∞	∞	

A-Star Algorithm (from A to F)



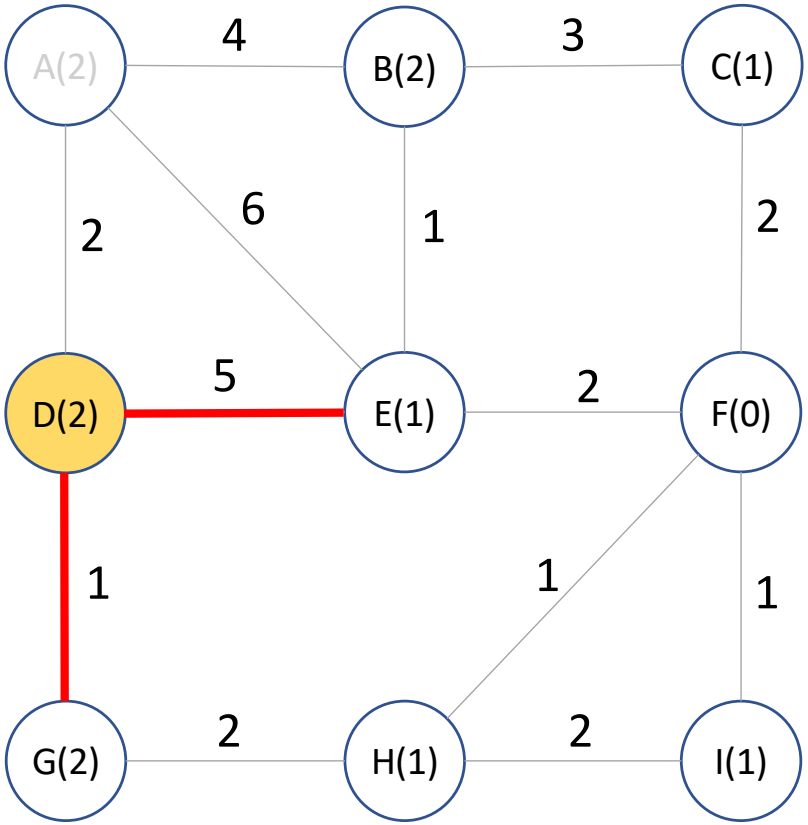
1 Visit the unvisited vertex with the smallest cost

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	∞	∞	
H	∞	∞	
I	∞	∞	

Visited = [A]

Unvisited = [B, C, D, E, F, G, H, I]

A-Star Algorithm (from A to F)



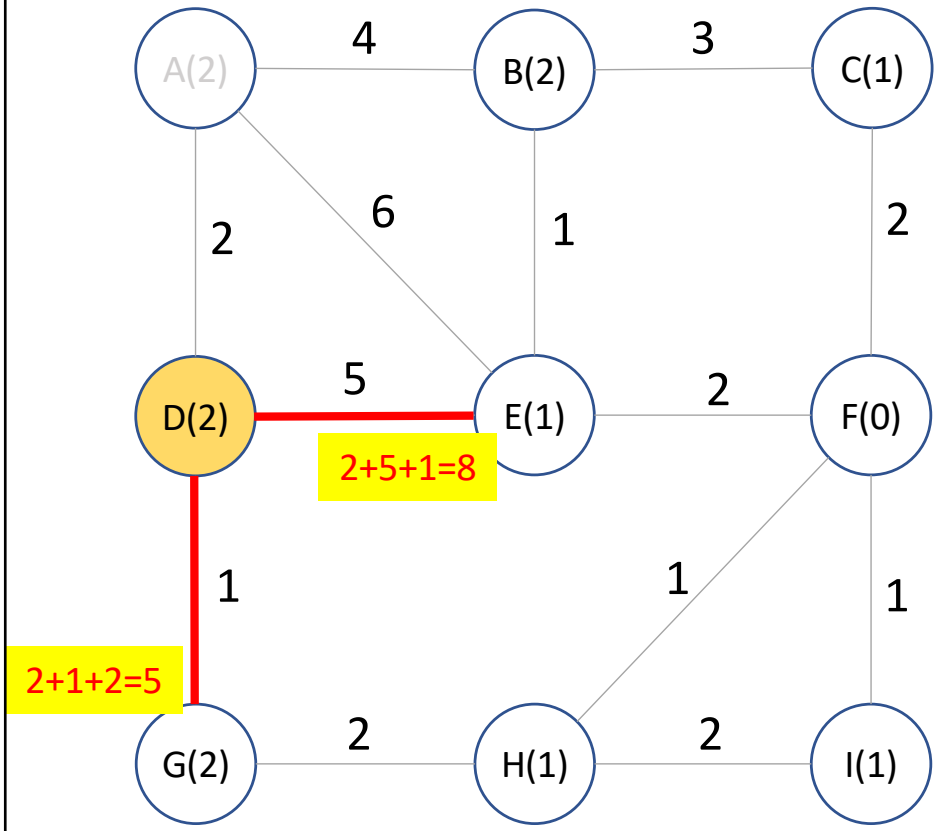
Visited = [A]

Unvisited = [B, C, D, E, F, G, H, I]

2 For the current vertex, if it is not the destination, examine the unvisited neighbours

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	∞	∞	
H	∞	∞	
I	∞	∞	

A-Star Algorithm (from A to F)



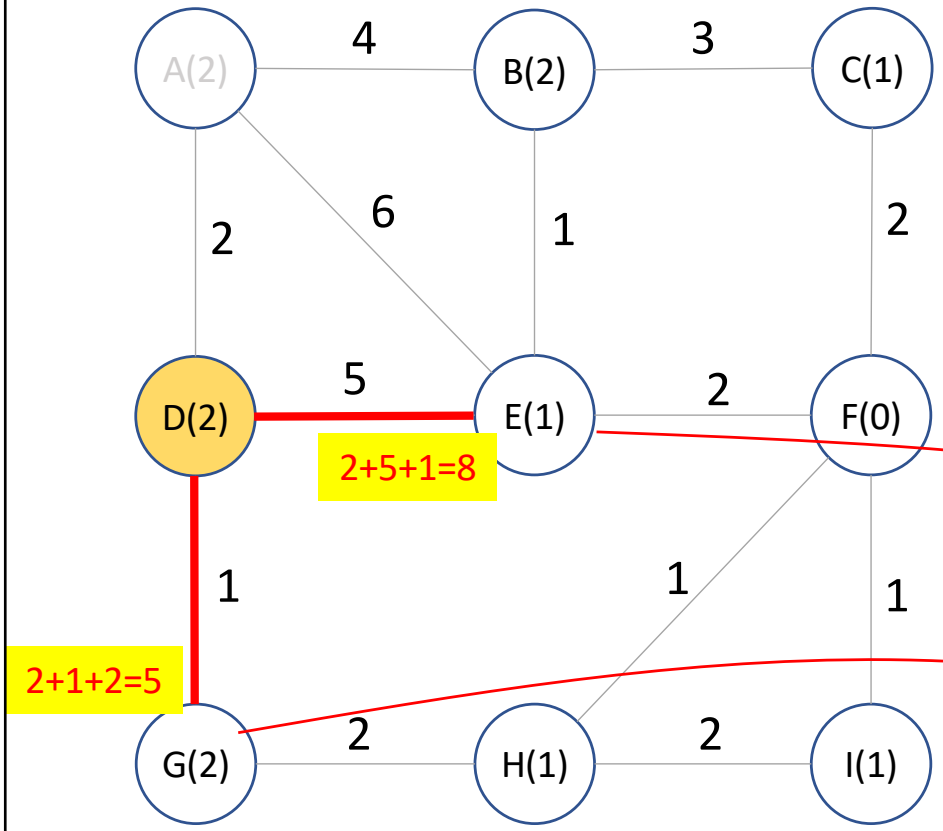
Visited = [A]

Unvisited = [B, C, D, E, F, G, H, I]

3 Calculate the cost of each neighbour

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	∞	∞	
H	∞	∞	
I	∞	∞	

A-Star Algorithm (from A to F)



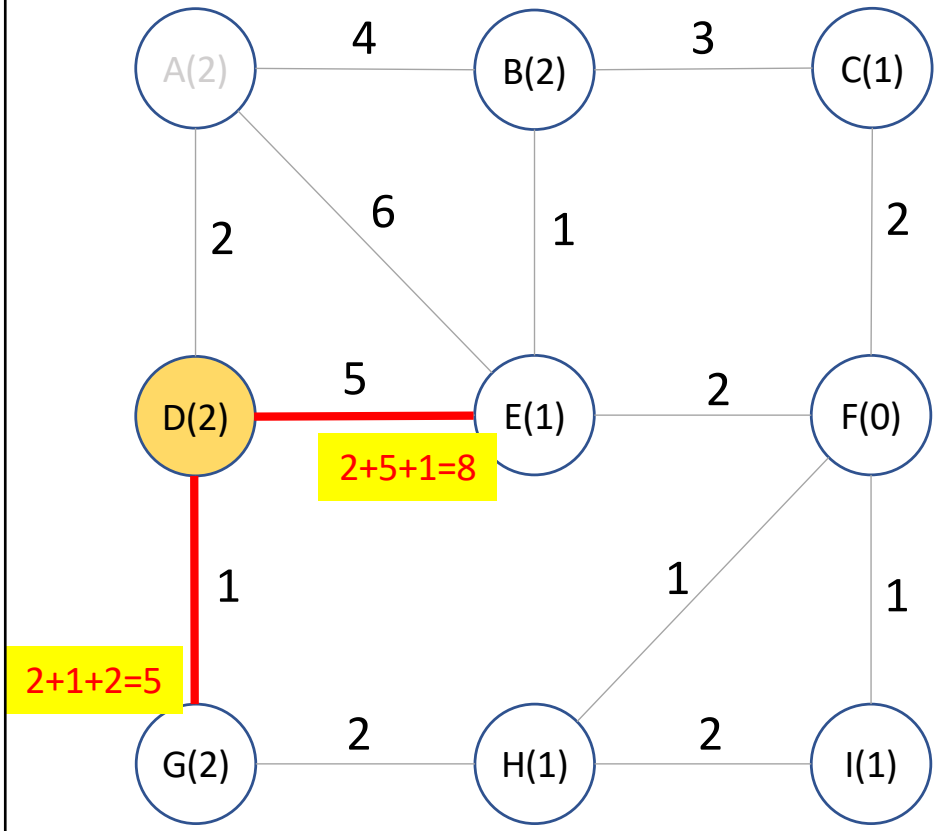
4 If the calculated cost is less than the known cost, update the lowest cost

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	3	5	
H	∞	∞	
I	∞	∞	

Visited = [A]

Unvisited = [B, C, D, E, F, G, H, I]

A-Star Algorithm (from A to F)



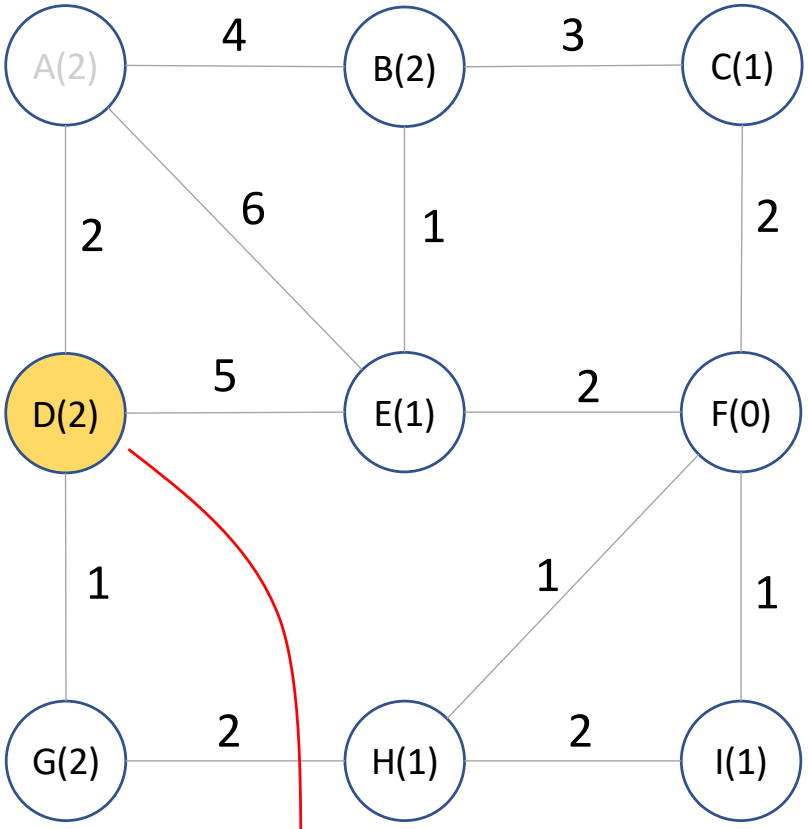
Visited = [A]

Unvisited = [B, C, D, E, F, G, H, I]

5 Update the previous vertex for each updated lowest cost

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	3	5	D
H	∞	∞	
I	∞	∞	

A-Star Algorithm (from A to F)



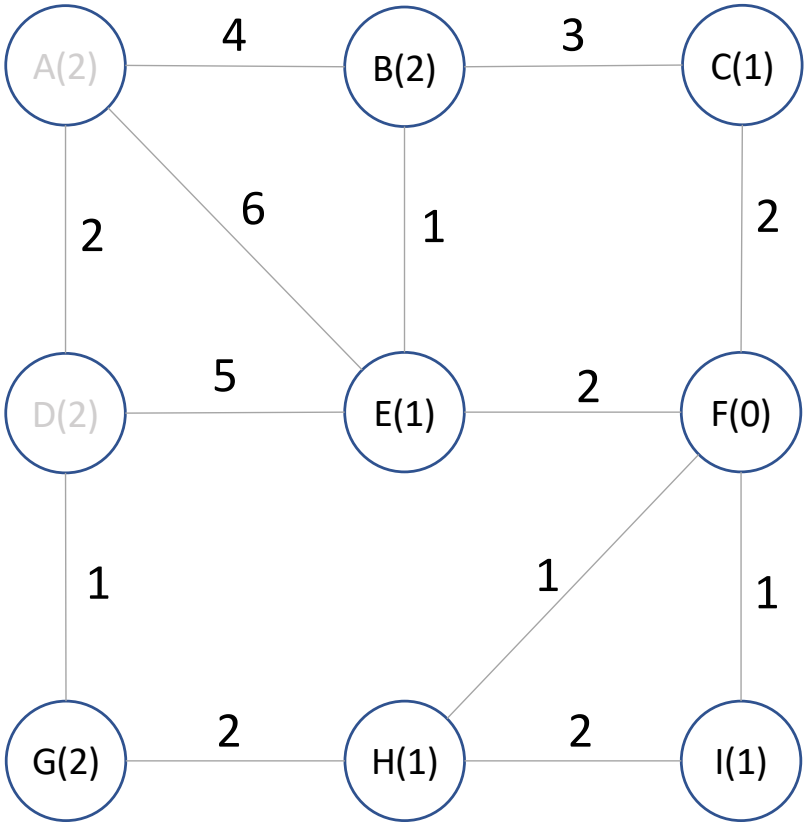
Visited = [A D

Unvisited = [B, C, E, F, G, H, I]

6 Add the current vertex to the list of visited vertices

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	3	5	D
H	∞	∞	
I	∞	∞	

A-Star Algorithm (from A to F)

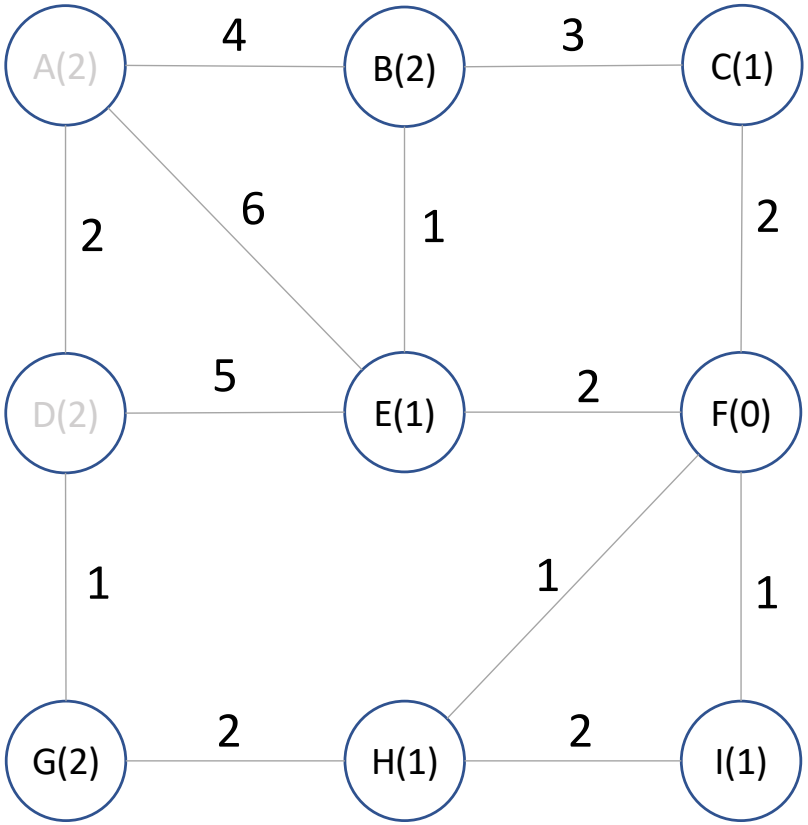


Visited = [A, D]

Unvisited = [B, C, E, F, G, H, I]

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	3	5	D
H	∞	∞	
I	∞	∞	

A-Star Algorithm (from A to F)



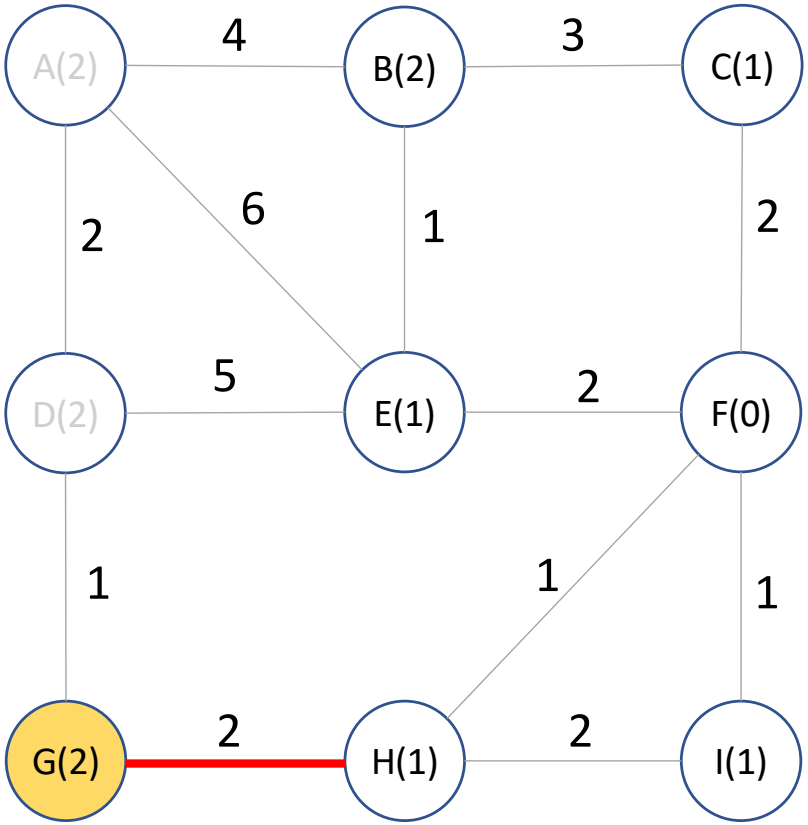
1 Visit the unvisited vertex with the smallest cost

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	3	5	D
H	∞	∞	
I	∞	∞	

Visited = [A, D]

Unvisited = [B, C, E, F, G, H, I]

A-Star Algorithm (from A to F)



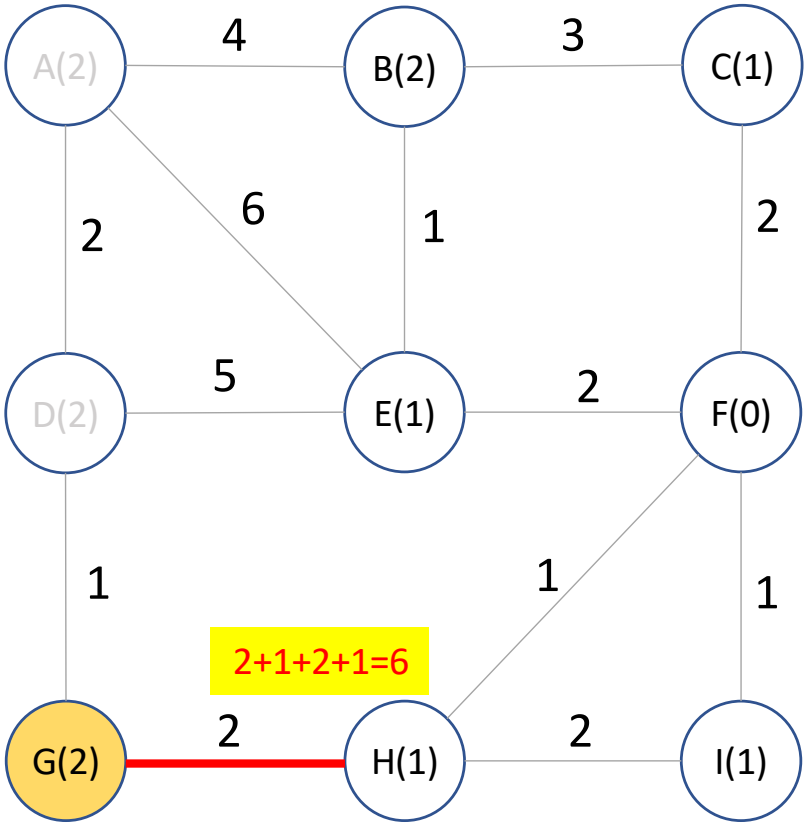
Visited = [A, D]

Unvisited = [B, C, E, F, G, H, I]

2 For the current vertex, if it is not the destination, examine the unvisited neighbours

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	3	5	D
H	∞	∞	
I	∞	∞	

A-Star Algorithm (from A to F)



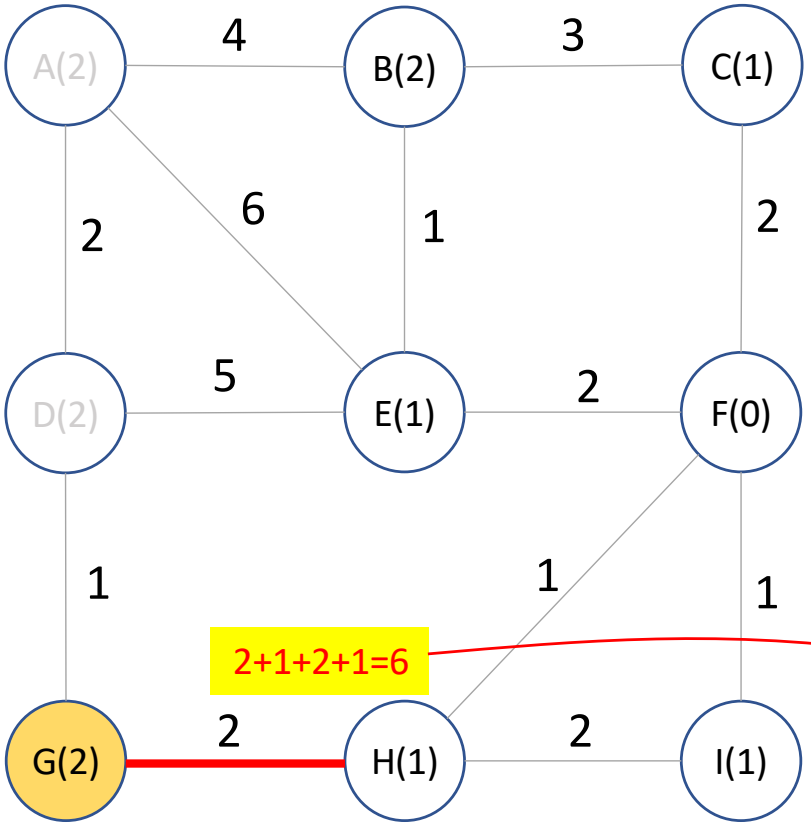
Visited = [A, D]

Unvisited = [B, C, E, F, G, H, I]

3 Calculate the cost of each neighbour from the start vertex

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	3	5	D
H	∞	∞	
I	∞	∞	

A-Star Algorithm (from A to F)



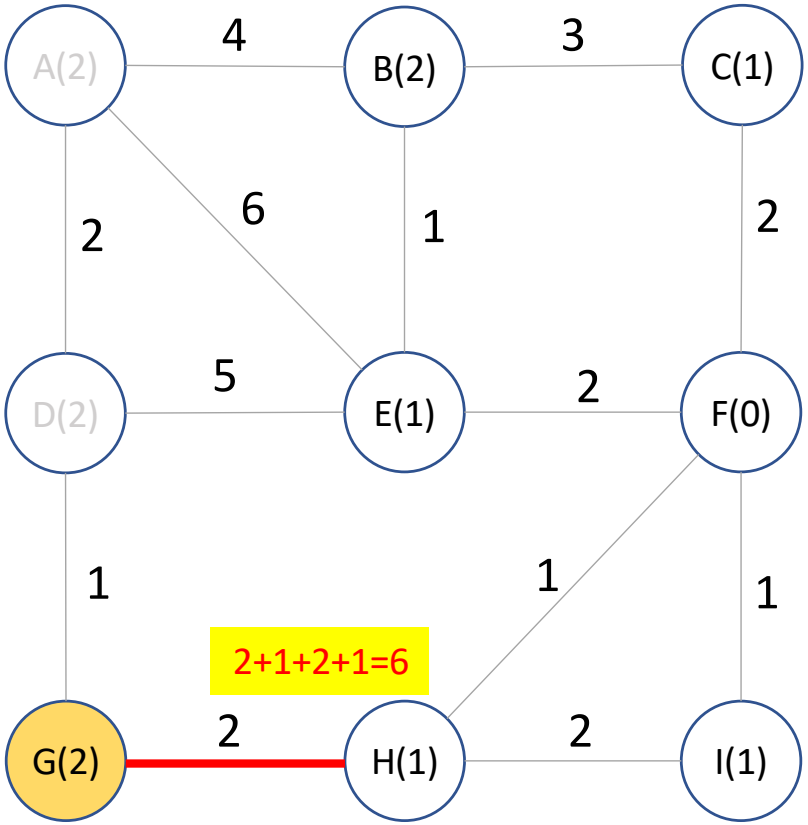
4 If the calculated cost is less than the known cost, update the lowest cost

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	3	5	D
H	5	6	
I	∞	∞	

Visited = [A, D]

Unvisited = [B, C, E, F, G, H, I]

A-Star Algorithm (from A to F)



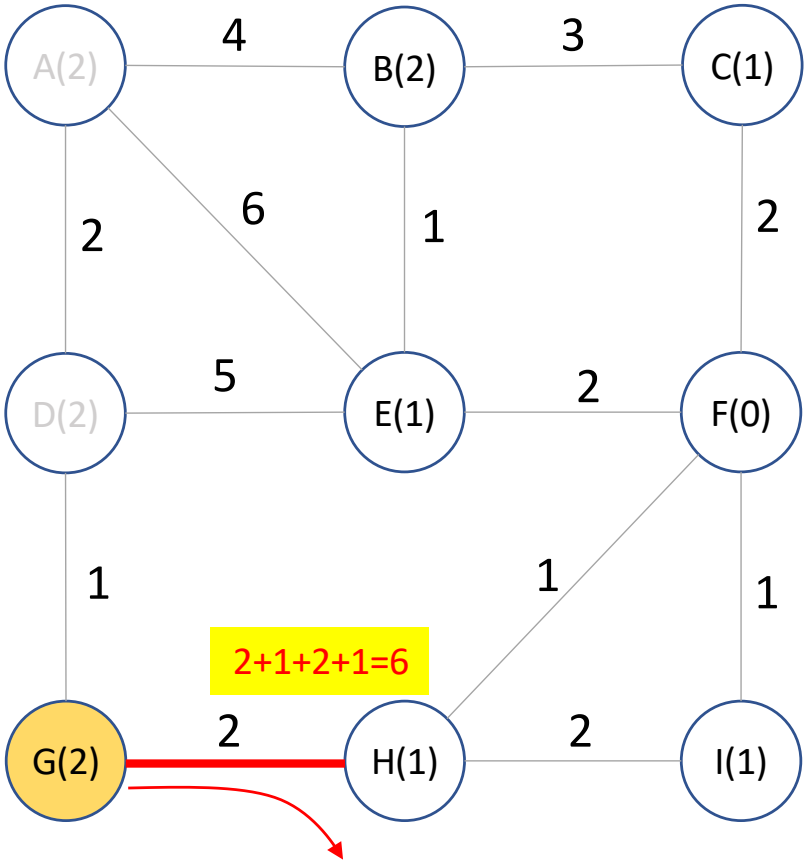
Visited = [A, D]

Unvisited = [B, C, E, F, G, H, I]

5 Update the previous vertex for each updated lowest cost

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	3	5	D
H	5	6	G
I	∞	∞	

A-Star Algorithm (from A to F)



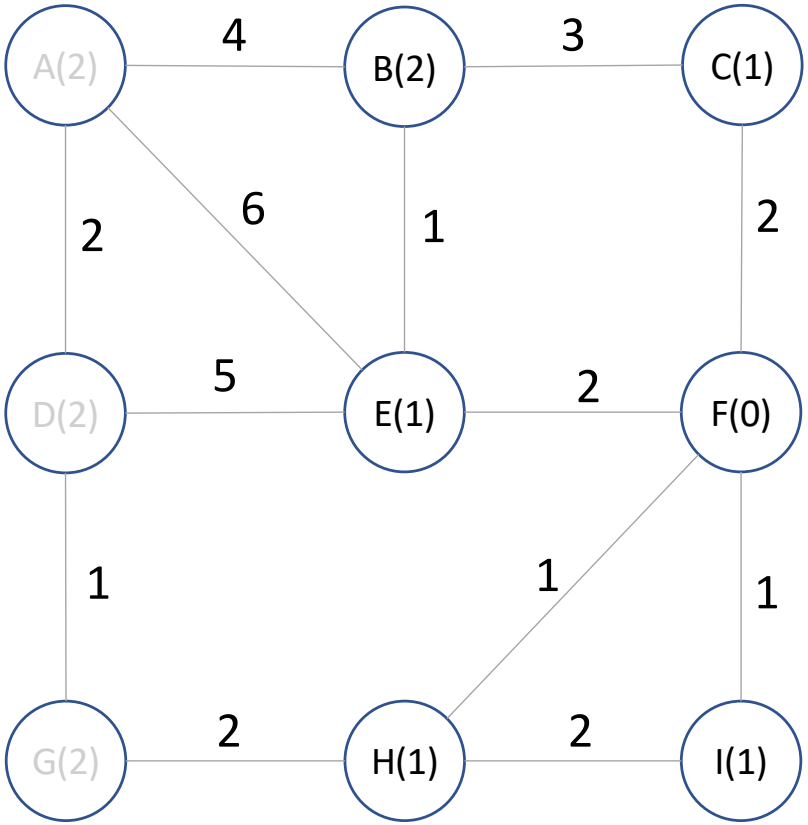
Visited = [A, D, G]

Unvisited = [B, C, E, F, H, I]

6 Add the current vertex to the list of visited vertices

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	3	5	D
H	5	6	G
I	∞	∞	

A-Star Algorithm (from A to F)

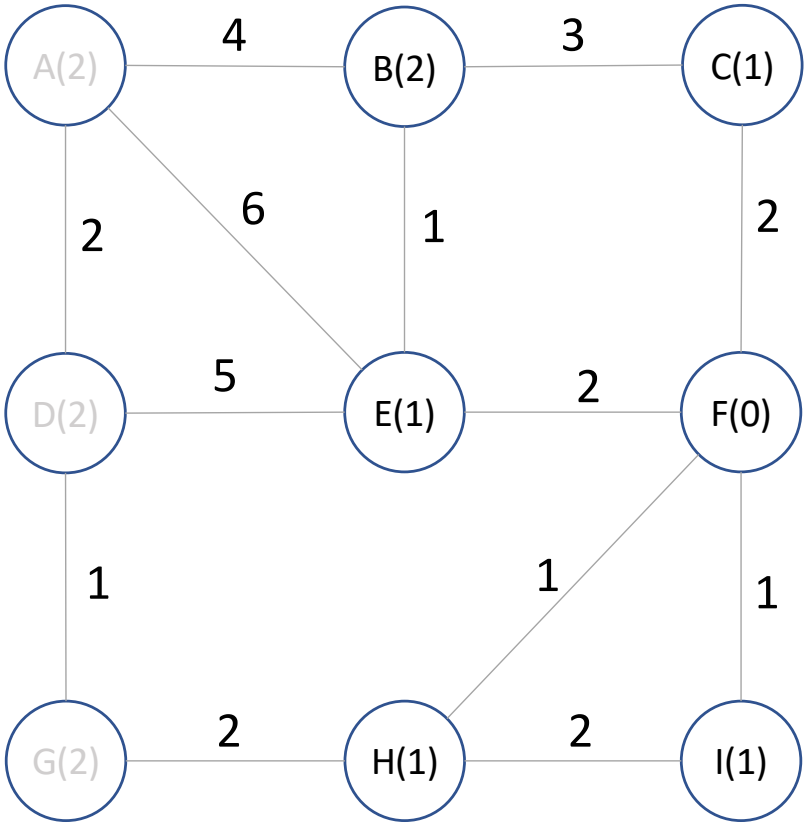


Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	3	5	D
H	5	6	G
I	∞	∞	

Visited = [A, D, G]

Unvisited = [B, C, E, F, H, I]

A-Star Algorithm (from A to F)



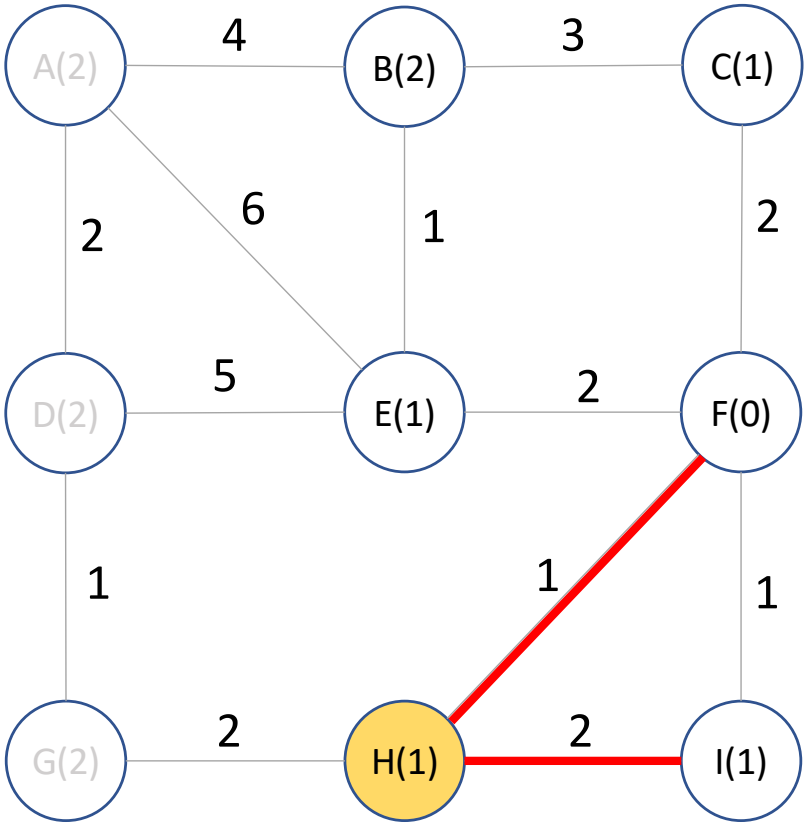
Visited = [A, D, G]

Unvisited = [B, C, E, F, H, I]

1 Visit the unvisited vertex with the smallest cost

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	3	5	D
H	5	6	G
I	∞	∞	

A-Star Algorithm (from A to F)



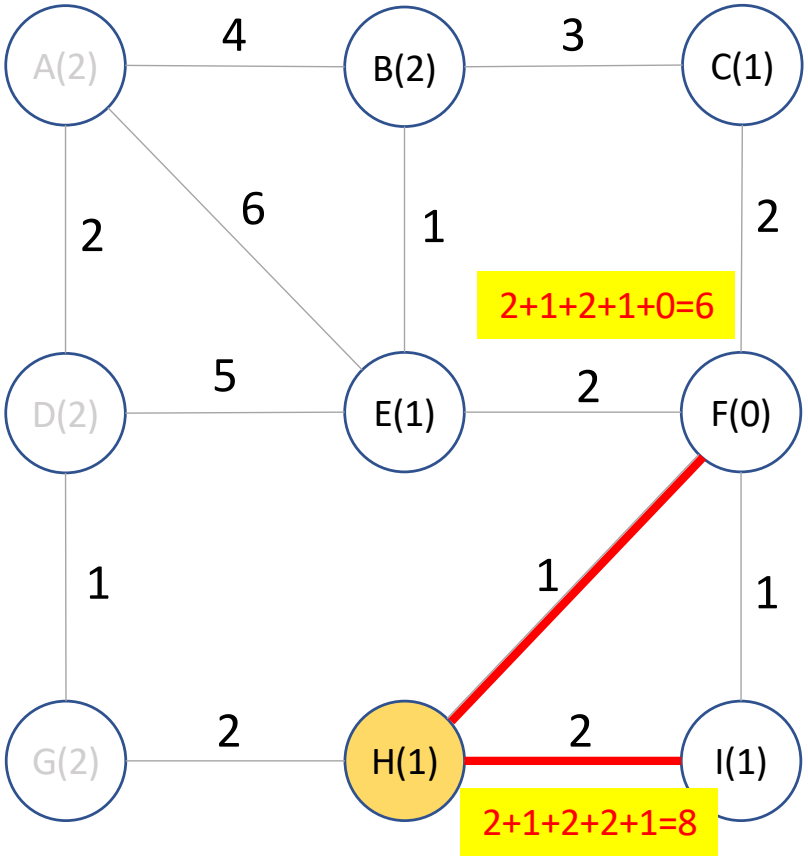
Visited = [A, D, G]

Unvisited = [B, C, E, F, H, I]

2 For the current vertex, if it is not the destination, examine the unvisited neighbours

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	3	5	D
H	5	6	G
I	∞	∞	

A-Star Algorithm (from A to F)



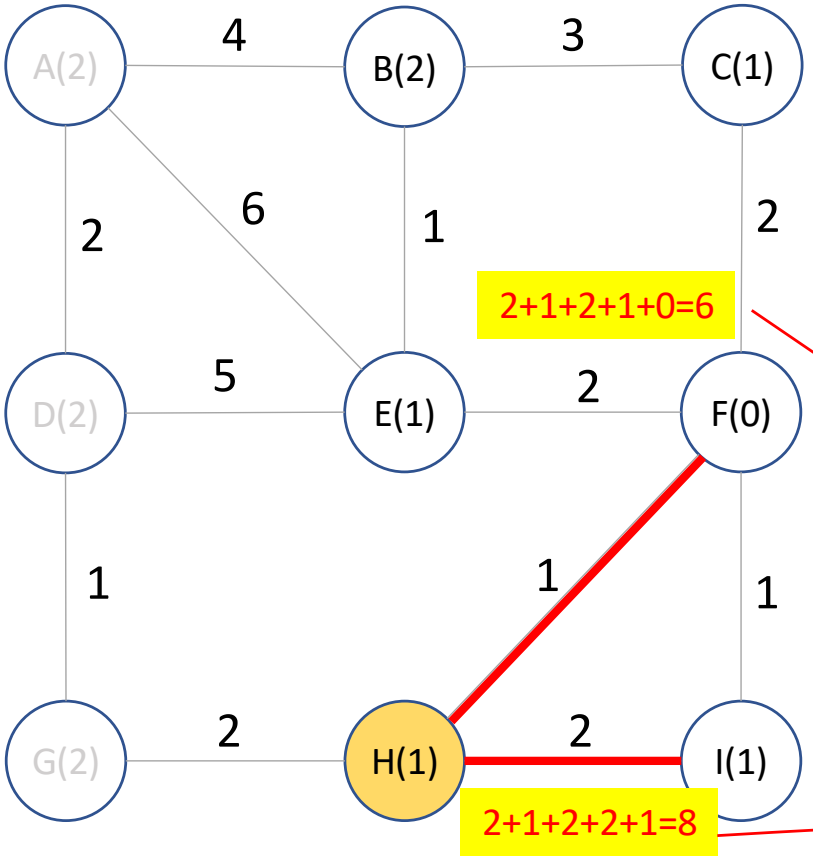
Visited = [A, D, G]

Unvisited = [B, C, E, F, H, I]

3 Calculate the cost of each neighbour

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	∞	∞	
G	3	5	D
H	5	6	G
I	∞	∞	

A-Star Algorithm (from A to F)



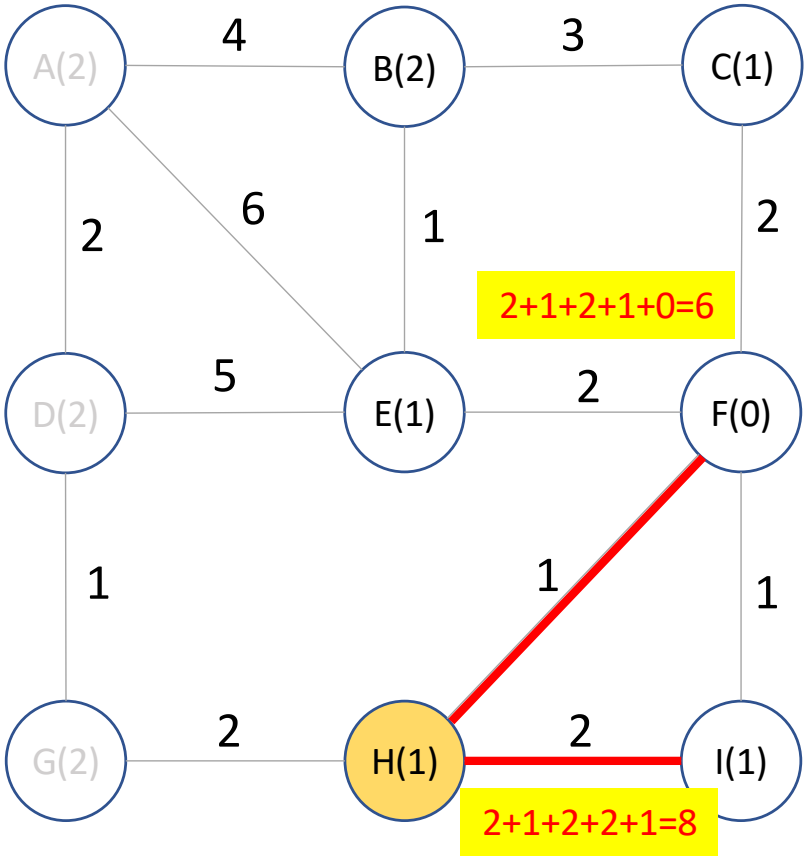
4 If the calculated cost is less than the known cost, update the lowest cost

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	1	6	
G	3	5	D
H	5	6	G
I	2	8	

Visited = [A, D, G]

Unvisited = [B, C, E, F, H, I]

A-Star Algorithm (from A to F)



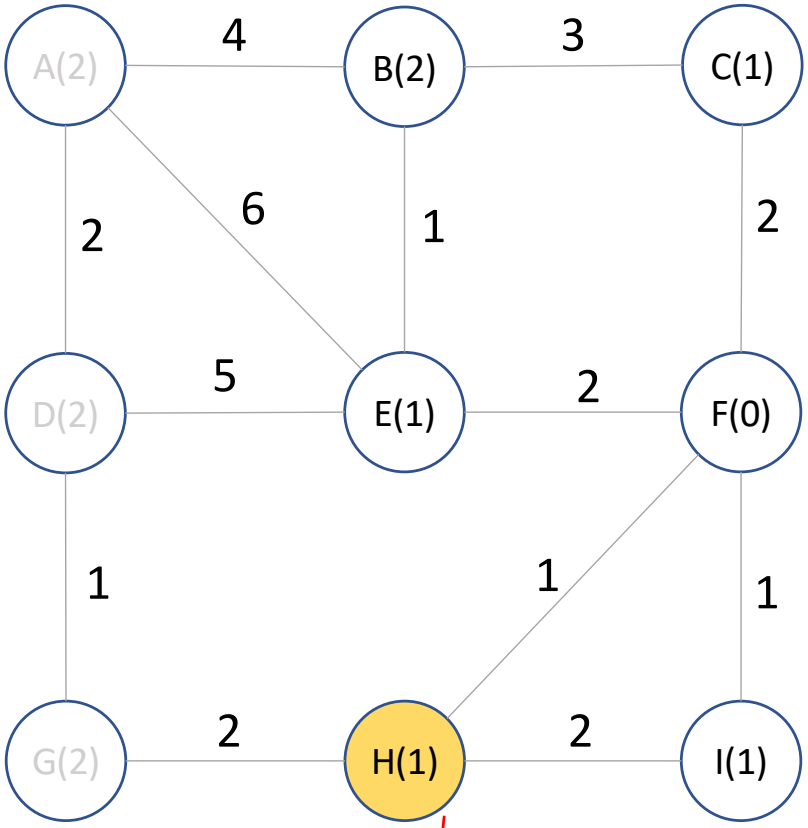
Visited = [A, D, G]

Unvisited = [B, C, E, F, H, I]

5 Update the previous vertex for each updated lowest cost

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	1	6	H
G	3	5	D
H	5	6	G
I	2	8	H

A-Star Algorithm (from A to F)



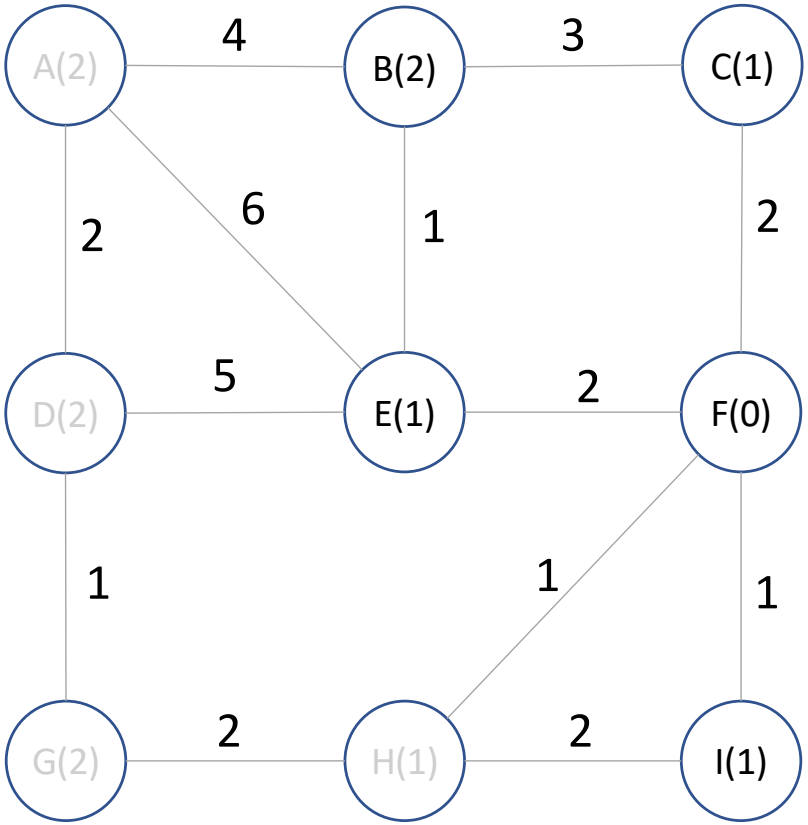
Visited = [A, D, G H]

Unvisited = [B, C, E, F, I]

6 Add the current vertex to the list of visited vertices

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	1	6	H
G	3	5	D
H	5	6	G
I	2	8	H

A-Star Algorithm (from A to F)

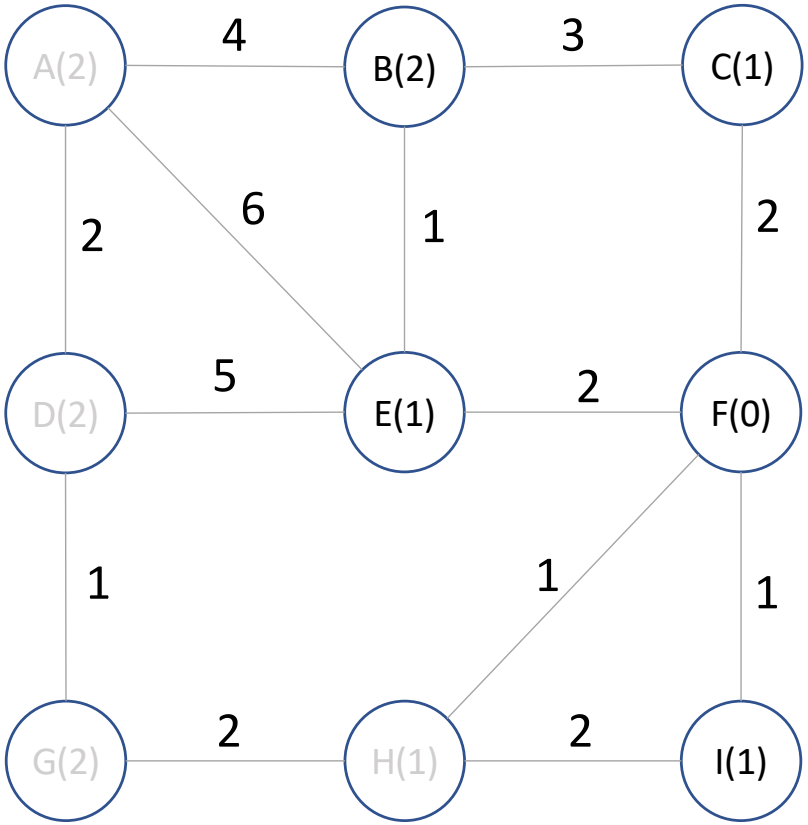


Visited = [A, D, G, H]

Unvisited = [B, C, E, F, I]

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	1	6	H
G	3	5	D
H	5	6	G
I	2	8	H

A-Star Algorithm (from A to F)



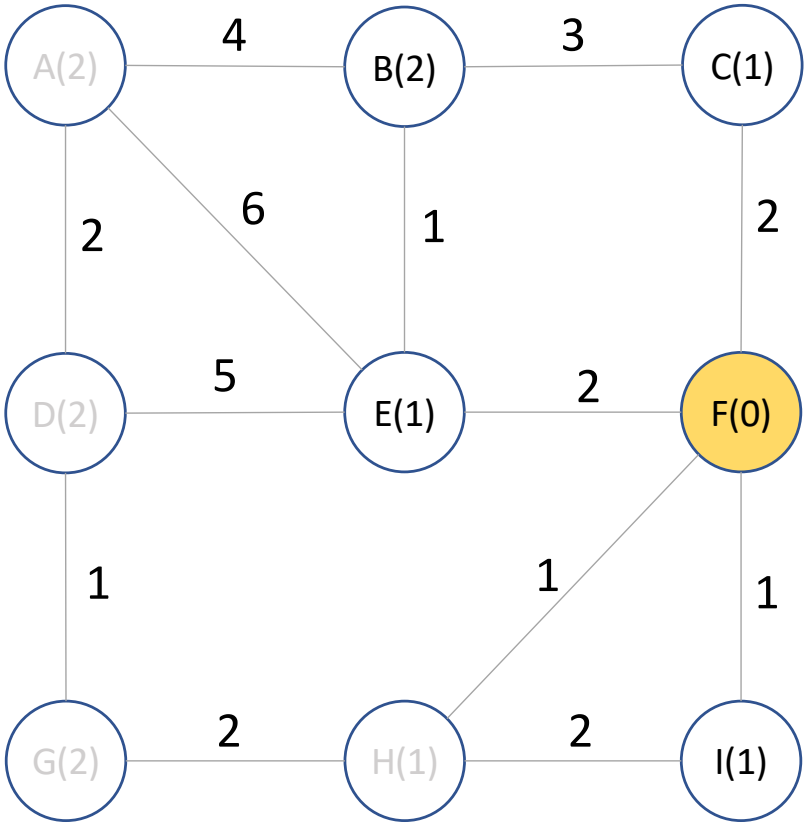
Visited = [A, D, G, H]

Unvisited = [B, C, E, F, I]

1 Visit the unvisited vertex with the smallest cost

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	1	6	H
G	3	5	D
H	5	6	G
I	2	8	H

A-Star Algorithm (from A to F)



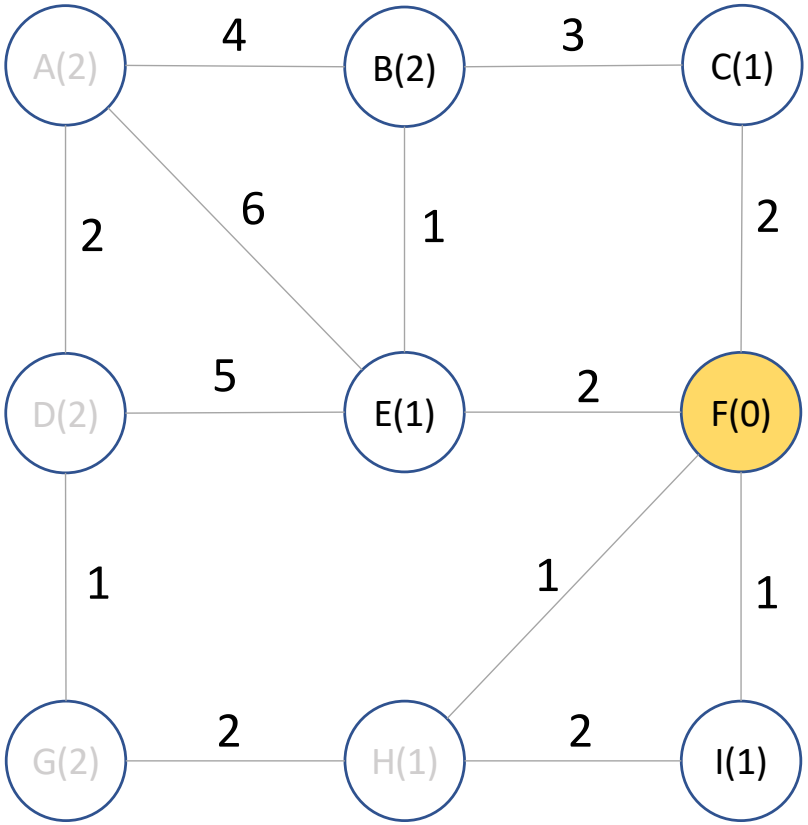
Visited = [A, D, G, H]

Unvisited = [B, C, E, F, I]

2 For the current vertex, if it is not the destination, examine the unvisited neighbours

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	1	6	H
G	3	5	D
H	5	6	G
I	2	8	H

A-Star Algorithm (from A to F)



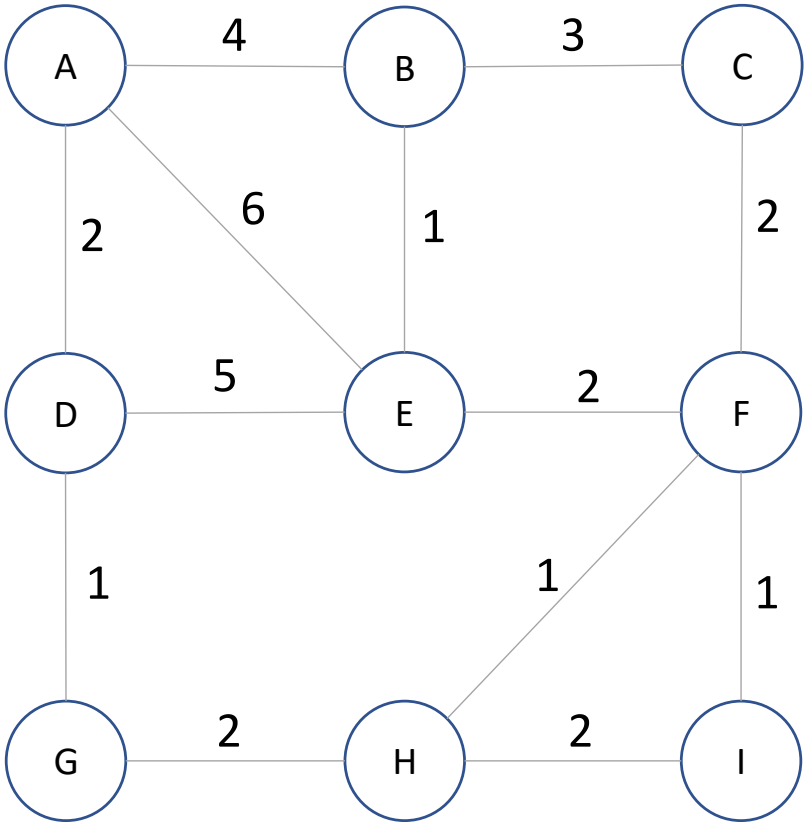
Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	1	6	H
G	3	5	D
H	5	6	G
I	2	8	H

Visited = [A, D, G, H]

Unvisited = [B, C, E, F, I]

Comparing Graph Traversing

DFS and BFS Algorithms (from A to F)



A → F

DFS: A → B → C → F

Vertices: 4, Cost: 9

BFS: A → E → F

Vertices: 2, Cost: 8

Dijkstra's and A-star Algorithms (from A to F)

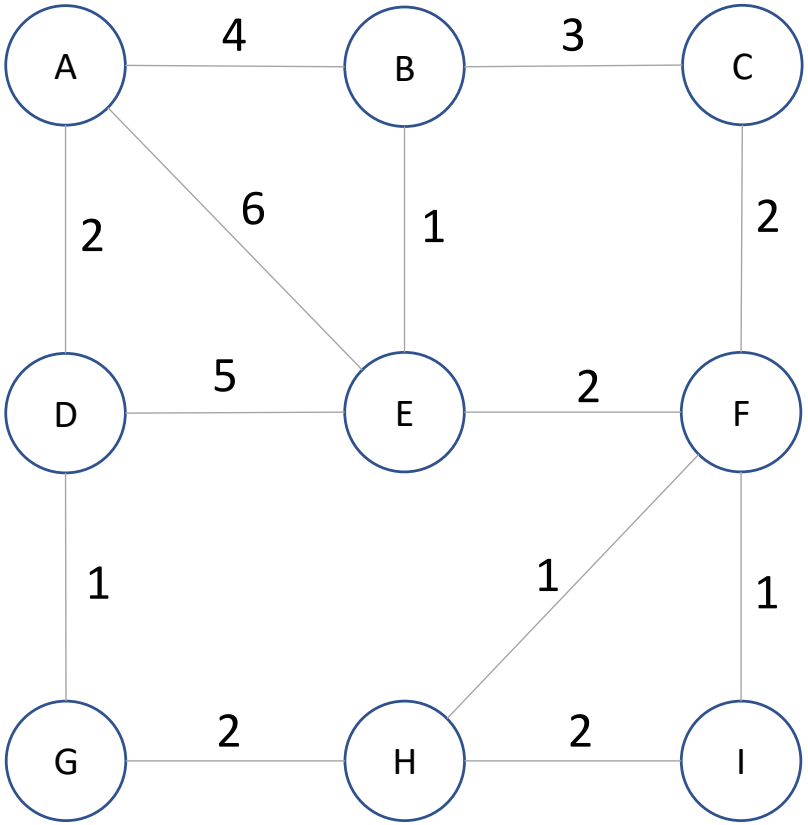
Dijkstra's

Vertex	Lowest cost from origin	Previous vertex
A	0	
B	4	A
C	7	B
D	2	A
E	5	B
F	6	H
G	3	D
H	5	G
I	7	H

A-star

Vertex	Lowest cost from origin	Lowest combined cost	Previous vertex
A	0	2	
B	4	6	A
C	∞	∞	
D	2	4	A
E	6	7	A
F	1	6	H
G	3	5	D
H	5	6	G
I	2	8	H

DFS, BFS, Dijkstra's and A-Star



A → F

DFS: A → B → C → F

Vertices: 4, Cost: 9

BFS: A → E → F

Vertices: 2, Cost: 8

Dijkstra's: A → D → G → H → F

Vertices: 5, Cost: 6

A-Star: A → D → G → H → F (faster)

Vertices: 5, Cost: 6

Thank you