COMP2119C
Data Structures & Algorithms

2024-25

Tutorial 1 – Graph Search

Jeff Siu

Motivation

Graphs are common in our daily life!

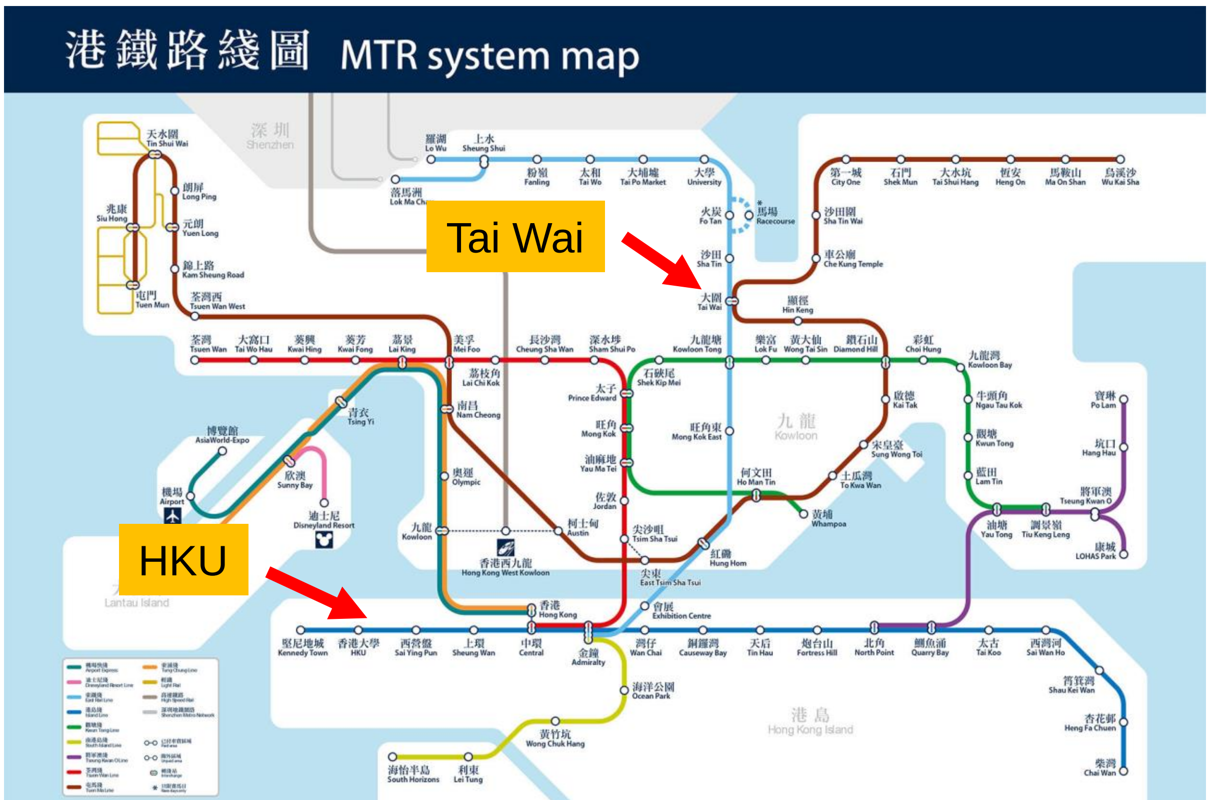
Social Network

Plane Routes

Motivation

My research interest since 3 years old!!!

Graph search problems are common in **transportation**:



One may ask:

1. What is the minimum time / lines needed to travel from HKU to Tai Wai?

2. What is the shortest path from station A to B without passing station C?

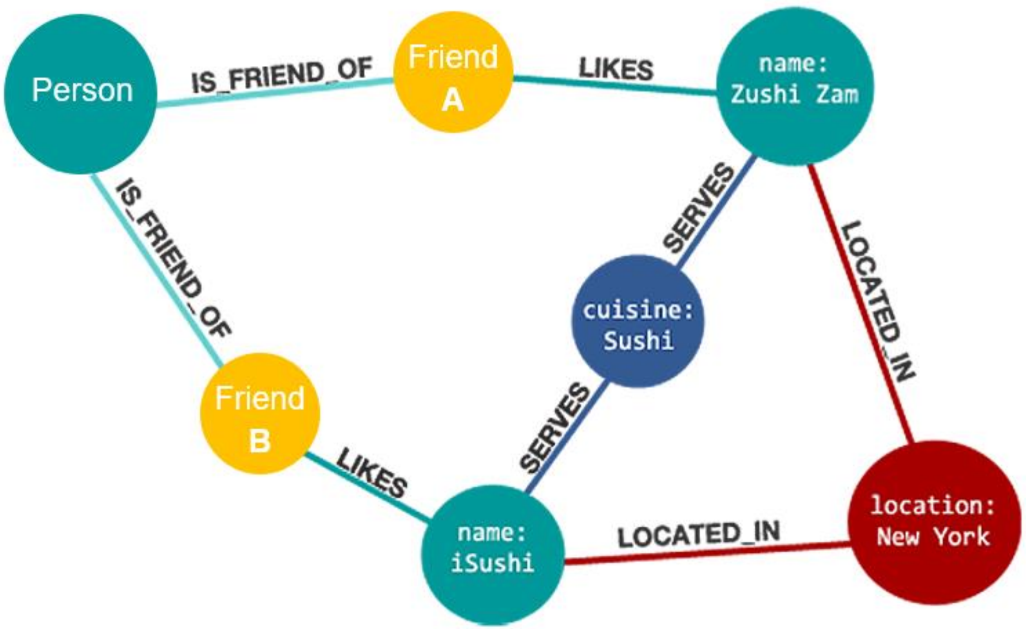
3. In terms of intermediate stations, which pair of stations are the farthest from each other?

4. (1000x more questions)

Motivation

My research interest since 3 years old!!!

Graph search problems are common in **database search engines**:



One may ask:

1. Who does Zushi Zam like?

2. What food does Friend B like?

3. How many friends like iSushi?

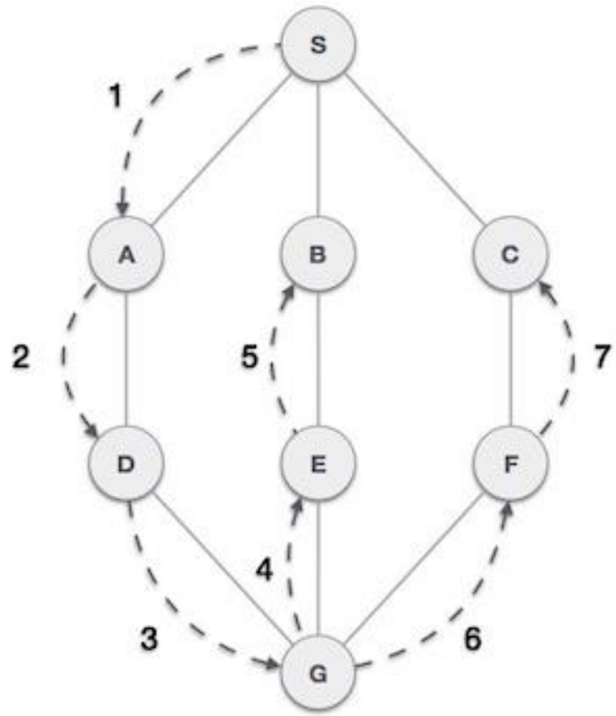
Outline

1. Different Graph Search Techniques
    (a) Depth First Search
    (b) Dijkstra's Algorithm

2. Take-home Exercises

Idea of DFS

Depth First Search (DFS) traverses a graph in a depthward motion and uses a stack to remember to get the next vertex to start a search, when a dead end occurs in any iteration.



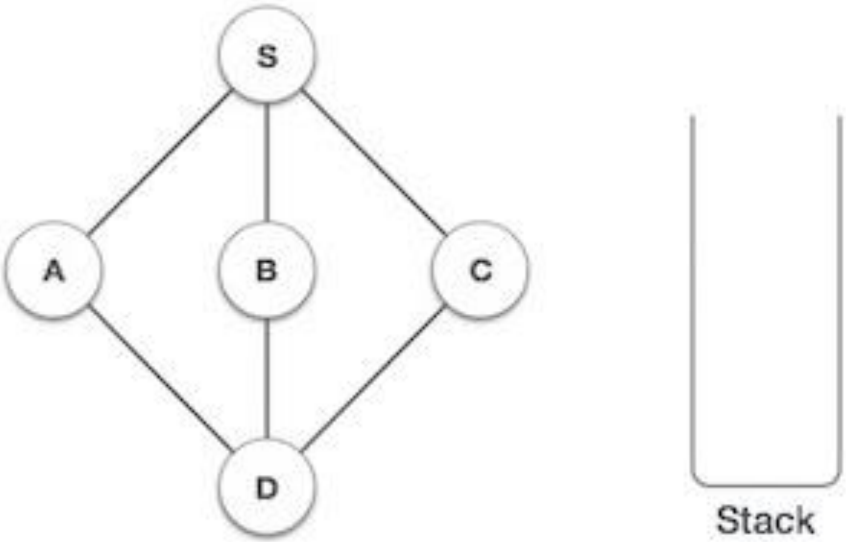
Idea of DFS

Step 1 – Visit the adjacent unvisited vertex. Mark it as visited. Display it. Push it in a stack.

Step 2 – If no adjacent vertex is found, pop up a vertex from the stack. (It will pop up all the vertices from the stack, which do not have adjacent vertices.)

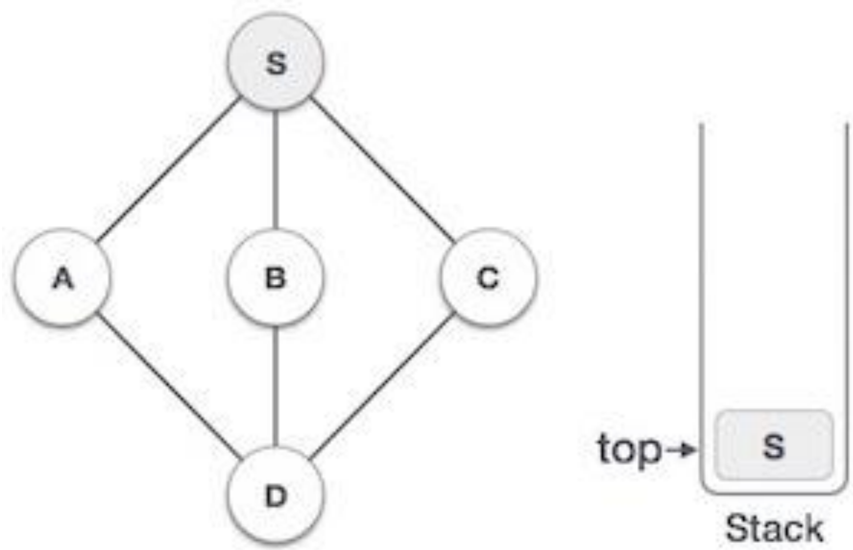
Step 3 – Repeat Rule 1 and Rule 2 until the stack is empty.

Example – DFS Illustration



Initialize the stack.

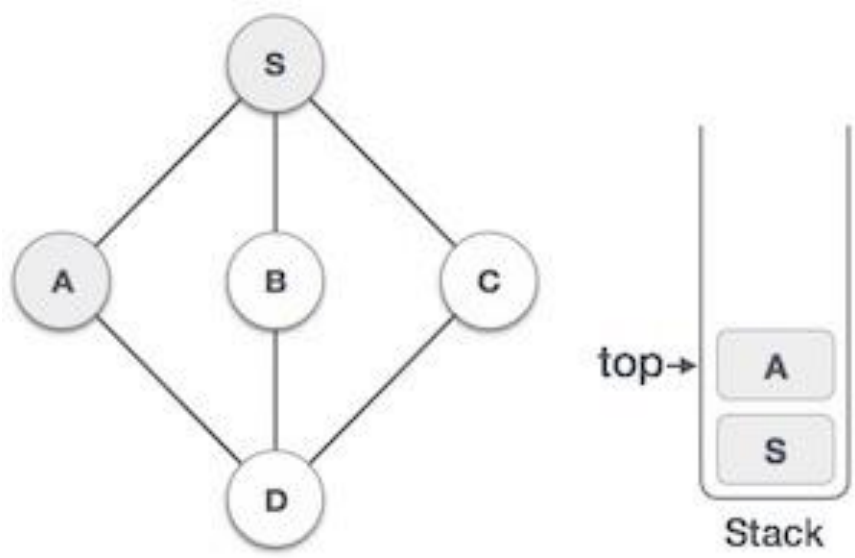
Example – DFS Illustration



Mark S as visited and put it onto the stack.

Explore any unvisited adjacent node from S in alphabetical order.

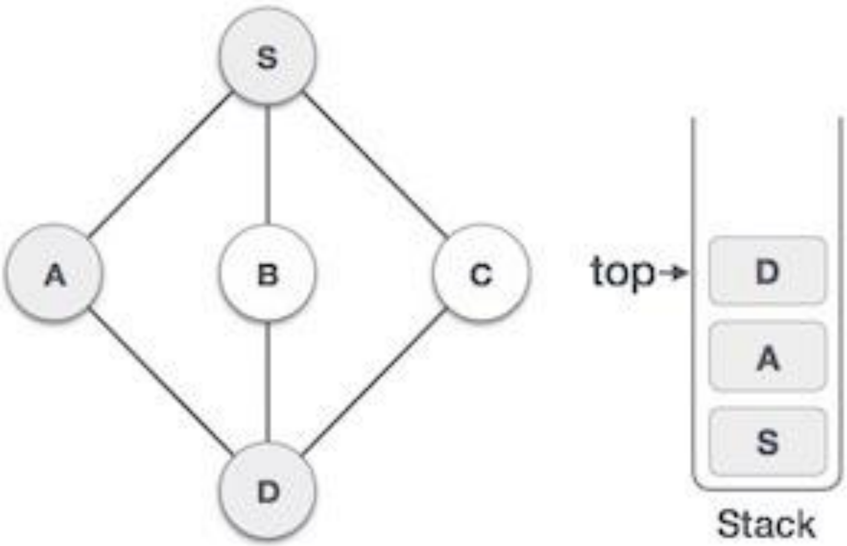
Example – DFS Illustration



Mark A as visited and put it onto the stack.

Explore any unvisited adjacent node from A. Both S and D are adjacent to A but we are concerned for unvisited nodes only.

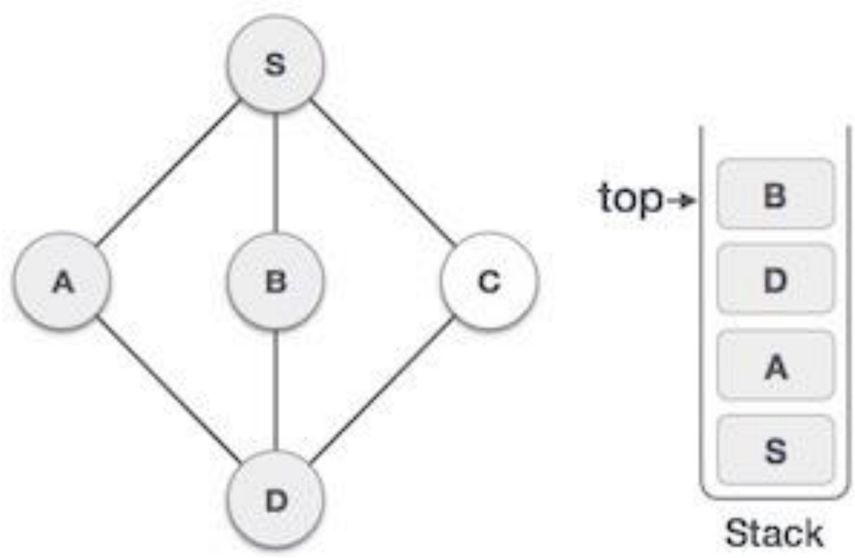
Example – DFS Illustration



Visit D and mark it as visited and put onto the stack.

Here, we have B and C nodes, which are adjacent to D and both are unvisited. However, we shall again choose in an alphabetical order.

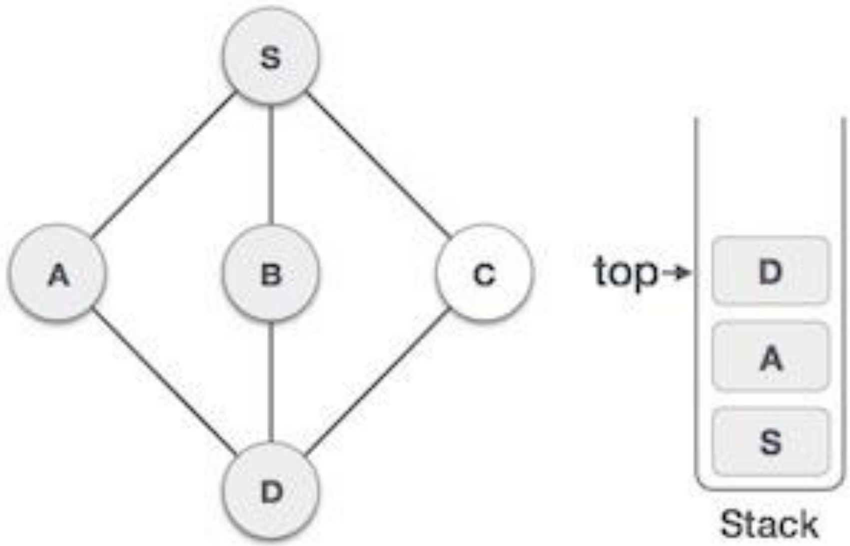
Example – DFS Illustration



We check the stack top for return to the previous node and check if it has any unvisited nodes.

Here, we find B to be on the top of the stack.

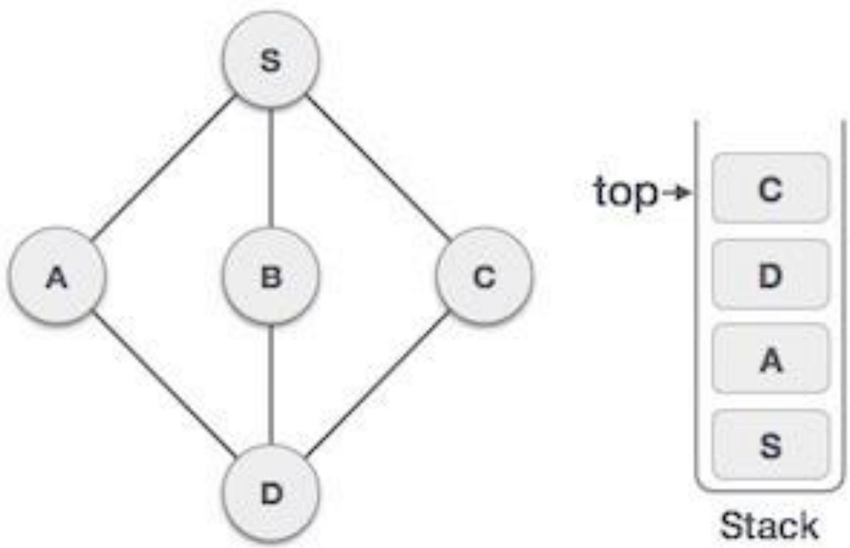
Example – DFS Illustration



We check the stack top for return to the previous node and check if it has any unvisited nodes.

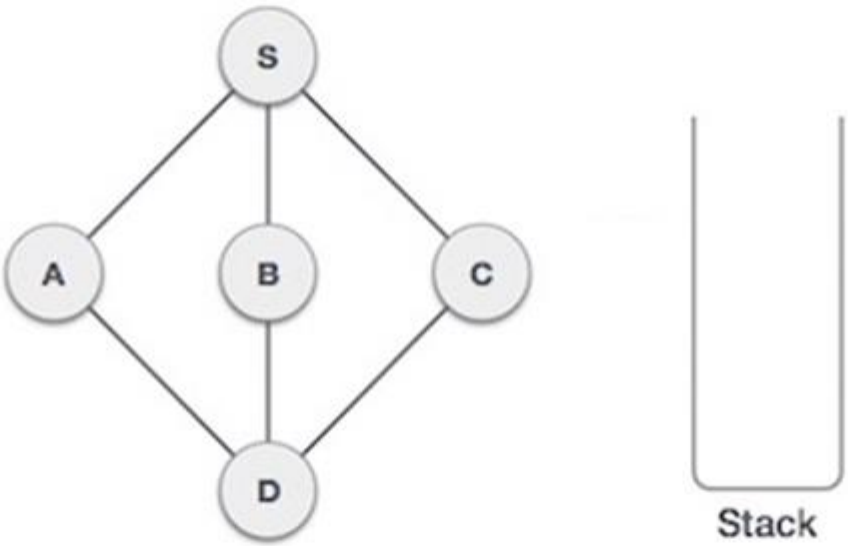
Here, we find D to be on the top of the stack.

Example – DFS Illustration



Only unvisited adjacent node from D is C now. So we visit C, mark it as visited and put it onto the stack.

Example – DFS Illustration



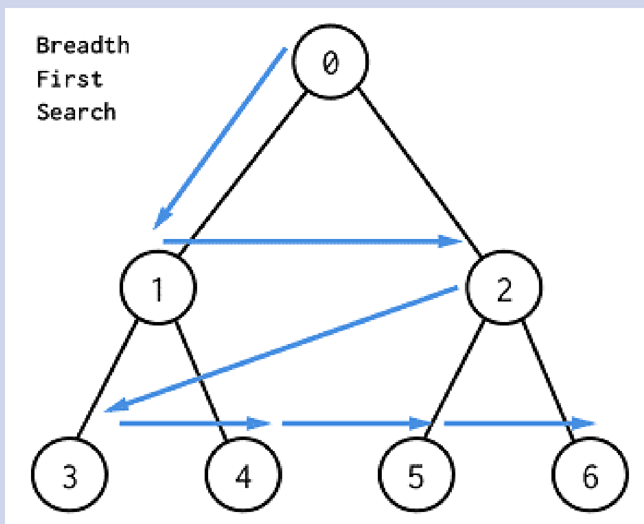
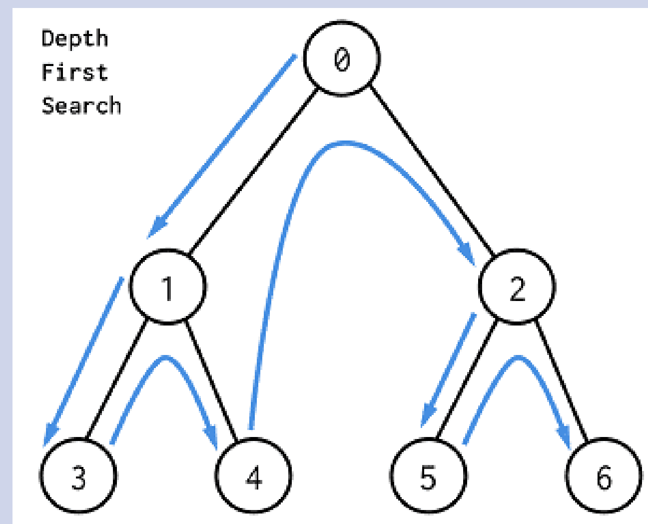
As C does not have any unvisited adjacent node so we keep popping the stack until we find a node that has an unvisited adjacent node.

In this case, there's none and we keep popping until the stack is empty.

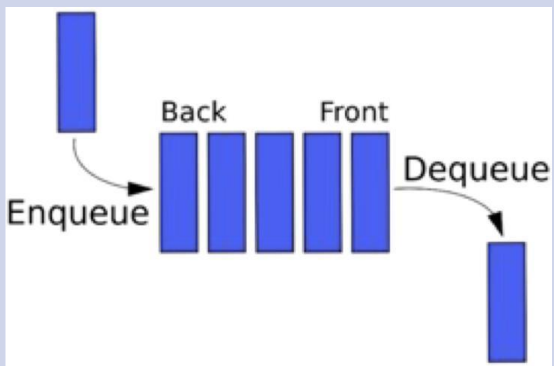
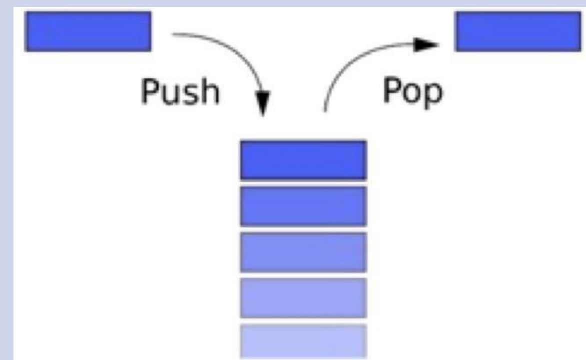
DFS Pseudocode

```
DFS(G, v)   (v is the vertex where the search starts) {
        Stack S;                        /* start with an empty stack
        bit visited[n];                 /* declare a bit vector visited */
        for each vertex u in G
                visited[u] = 0;
        push(S, v);
        visited[v] = 1;
        while (S is not empty) do
                u = pop(S);
                for each unvisited neighbour w of u
                        push(S, w);
                        visited[w] = 1;
        end while
}
```

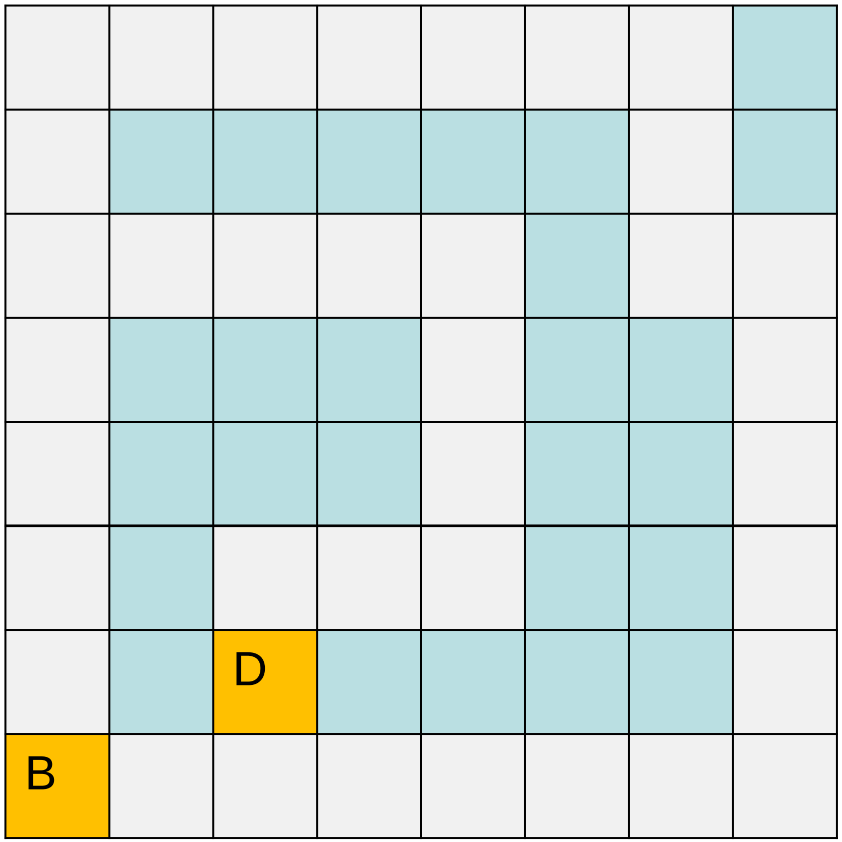
DFS vs BFS

	BFS	DFS
Principle	First walk through all nodes on the same level before moving on to the next level.  	First proceeds through the nodes as far as possible until we reach the node with no unvisited nearby nodes. 

DFS vs BFS

	BFS	DFS
Data Structure	Act like a queue to pick nodes by a first in first out manner. 	Act like a stack to pick nodes by a last in last out manner. 
Time Complexity	O(V + E)	O(V + E)
Suitable for	Find paths when the starting point is near the destination point	Find paths when the starting point is far away from the destination point

Example – DFS vs BFS in Path Search



Notes:

B Beginning postion

D Destination

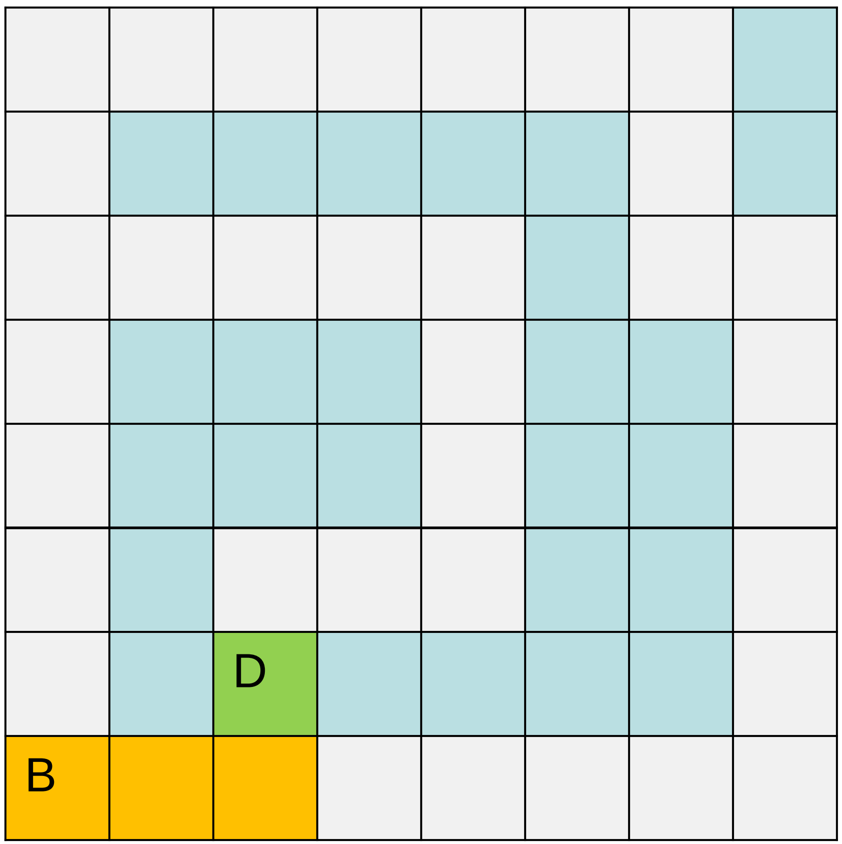
Walkable path

Wall

D At destination

Example – DFS vs BFS in Path Search

<u>DFS Path 1:</u>



Notes:

B Beginning postion

D Destination

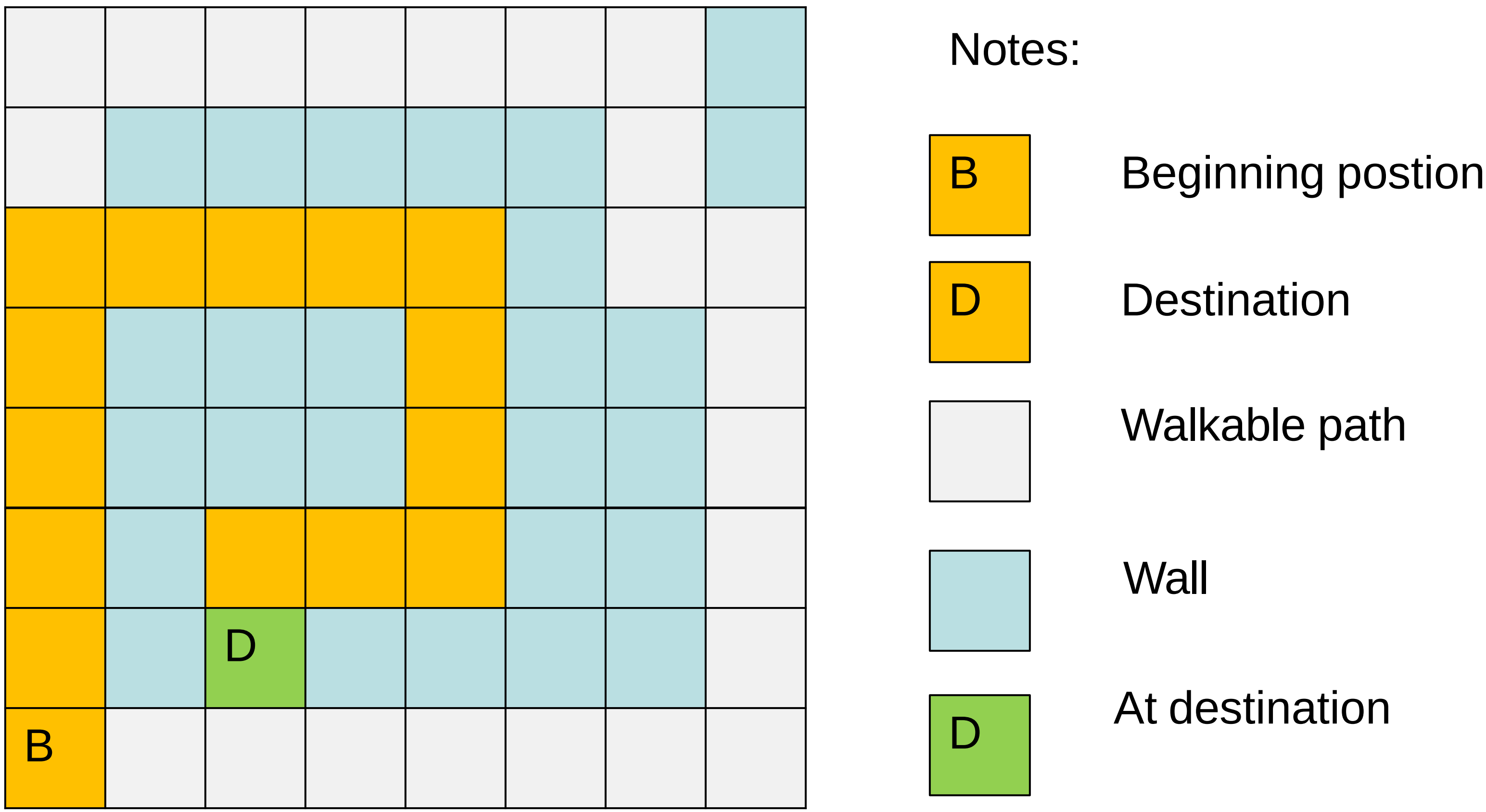
Walkable path

Wall

D At destination

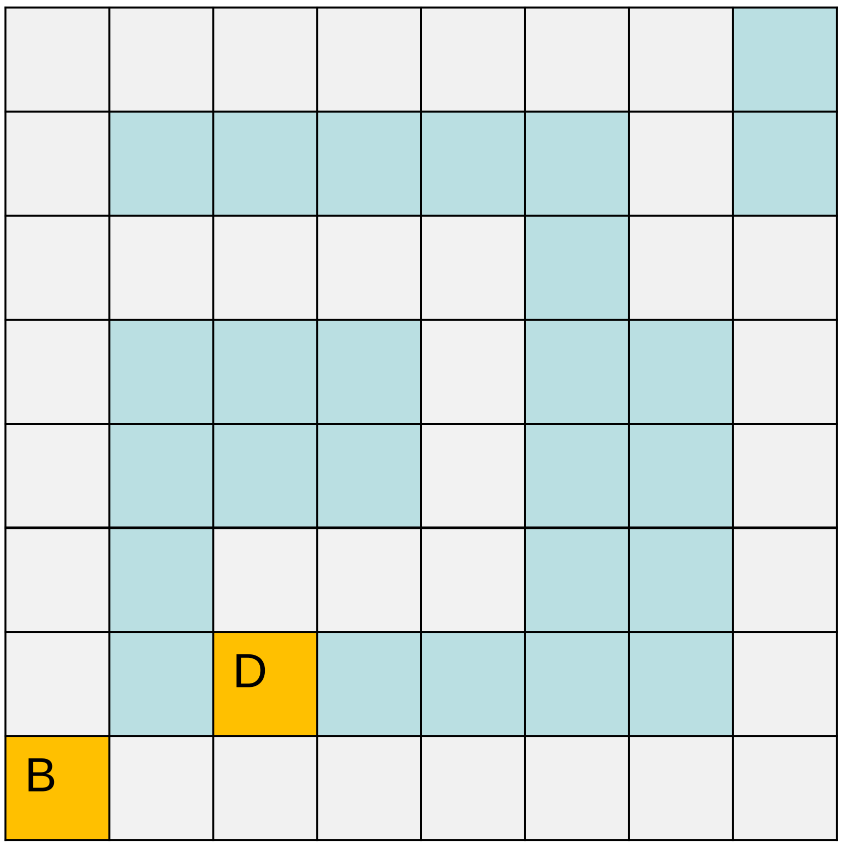
Example – DFS vs BFS in Path Search

DFS Path 2:



Example – DFS vs BFS in Path Search

BFS Path 1 & 2:

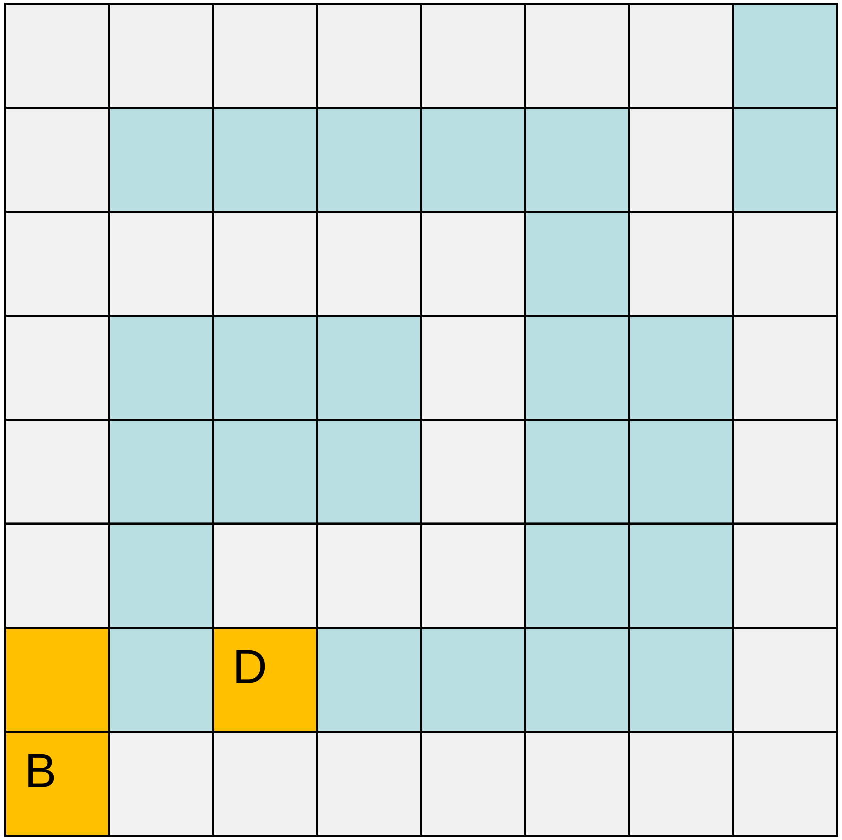


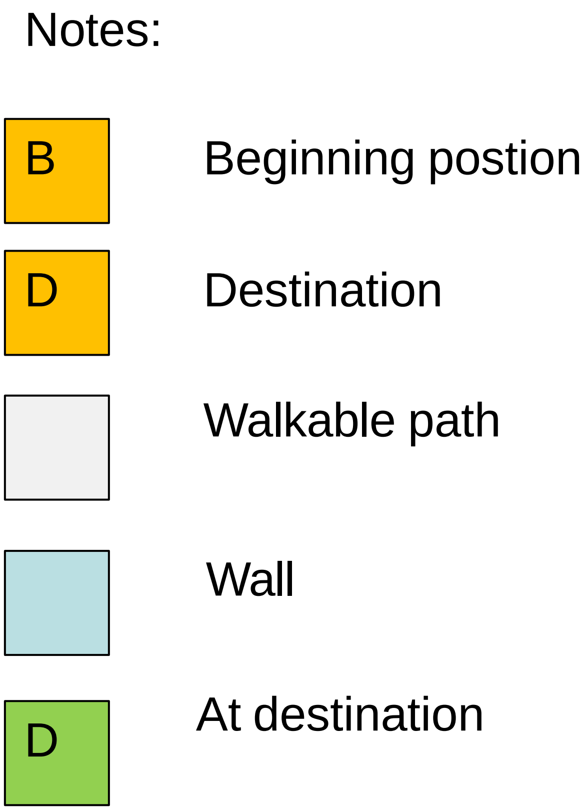
Notes:

- B — Beginning postion
- D — Destination
- Walkable path
- Wall
- D — At destination

Example – DFS vs BFS in Path Search

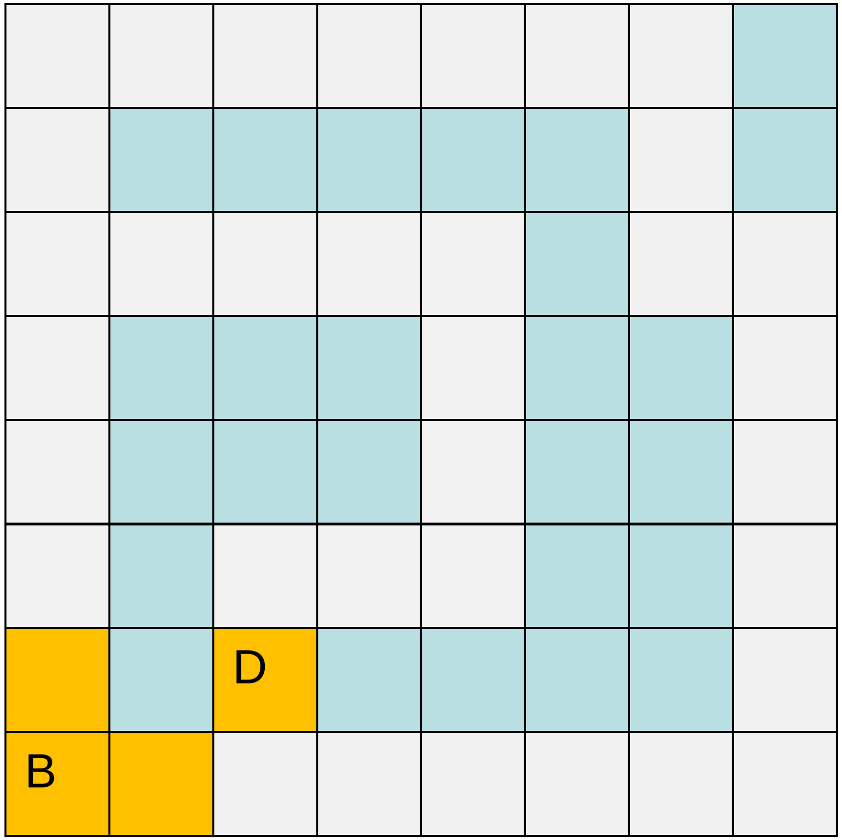
<u>BFS Path 1 & 2:</u>





Example – DFS vs BFS in Path Search

<u>BFS Path 1 & 2:</u>

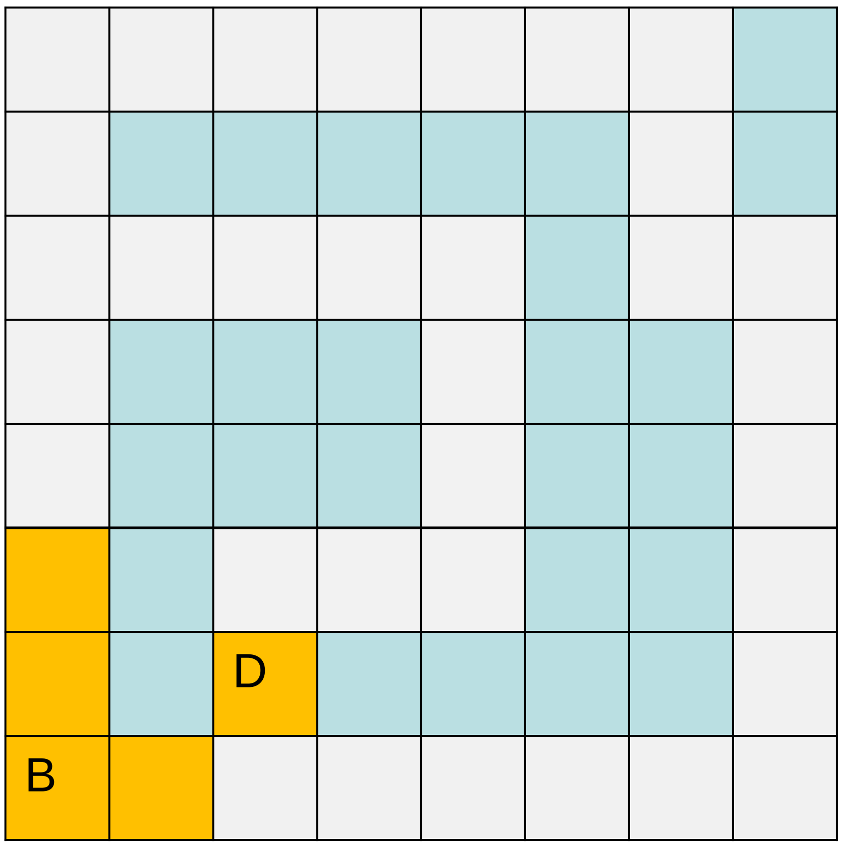


Notes:

- B Beginning postion
- D Destination
- Walkable path
- Wall
- D At destination

Example – DFS vs BFS in Path Search

<u>BFS Path 1 & 2:</u>

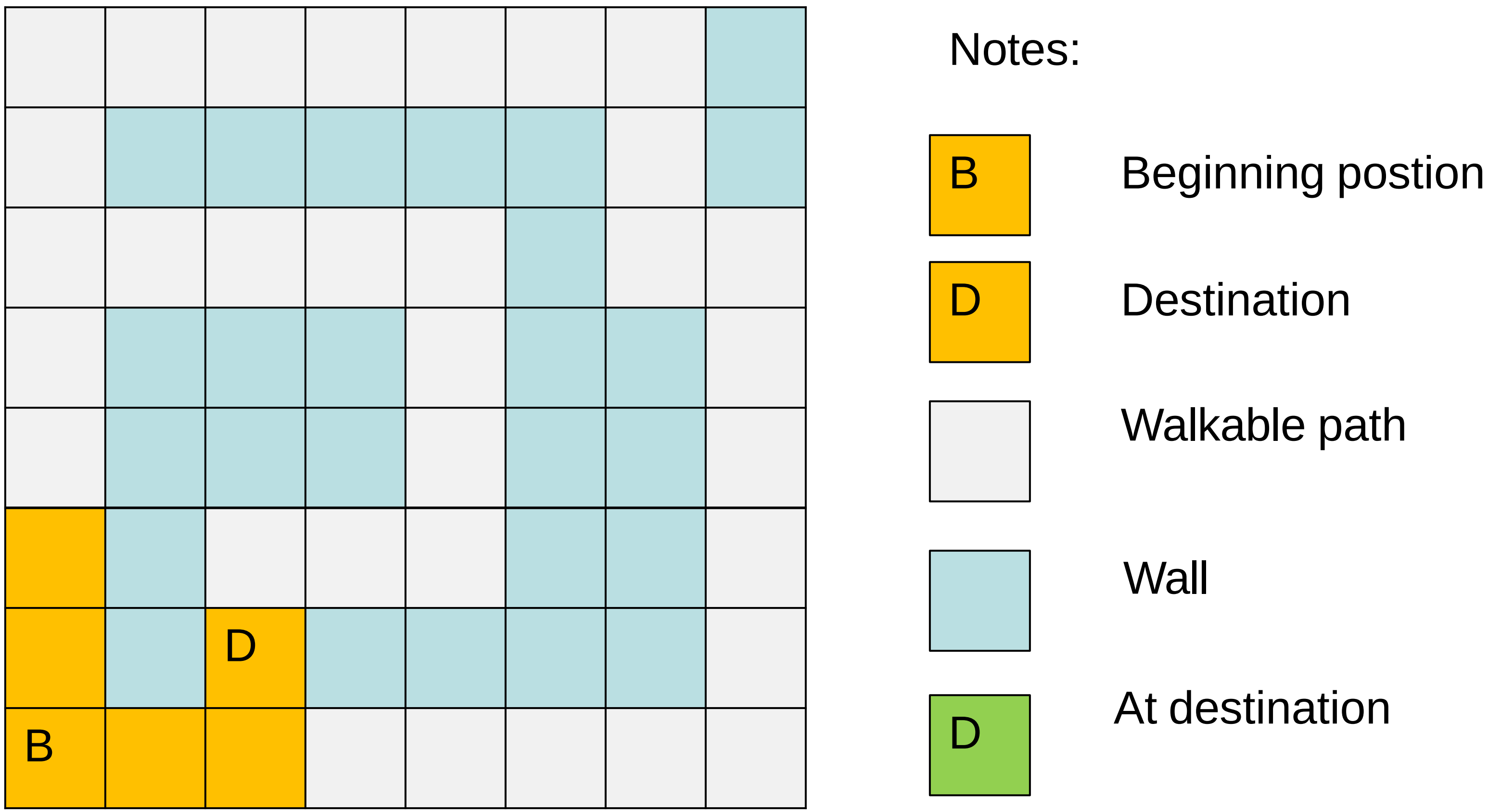


Notes:

B	Beginning postion
D	Destination
	Walkable path
	Wall
D	At destination

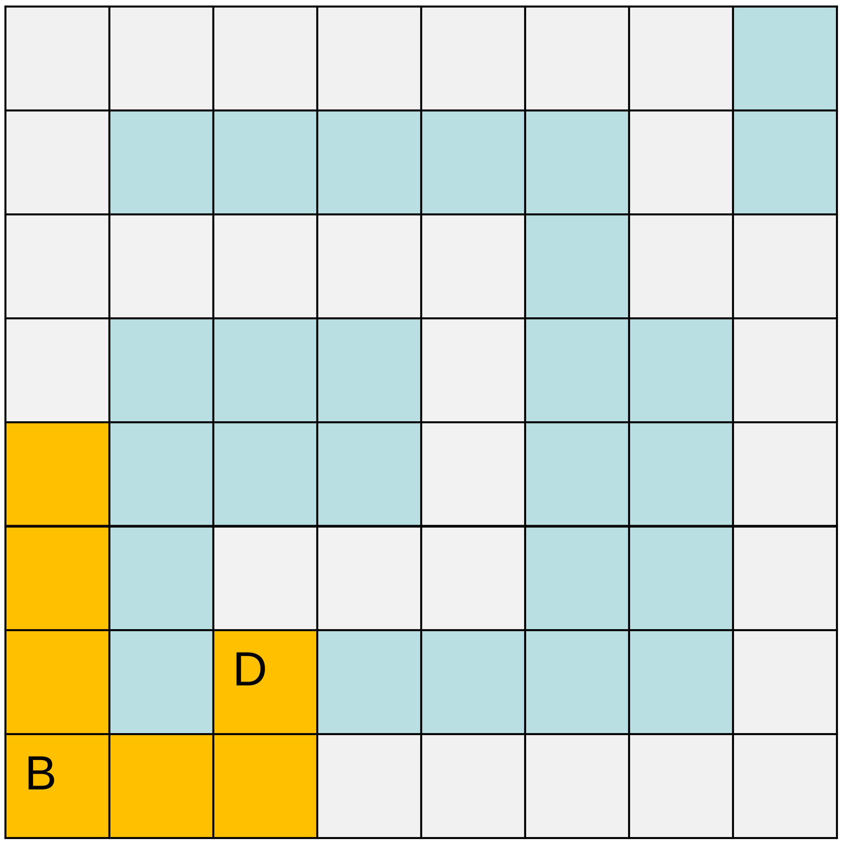
Example – DFS vs BFS in Path Search

BFS Path 1 & 2:



Example – DFS vs BFS in Path Search

BFS Path 1 & 2:



Notes:

B — Beginning postion

D — Destination

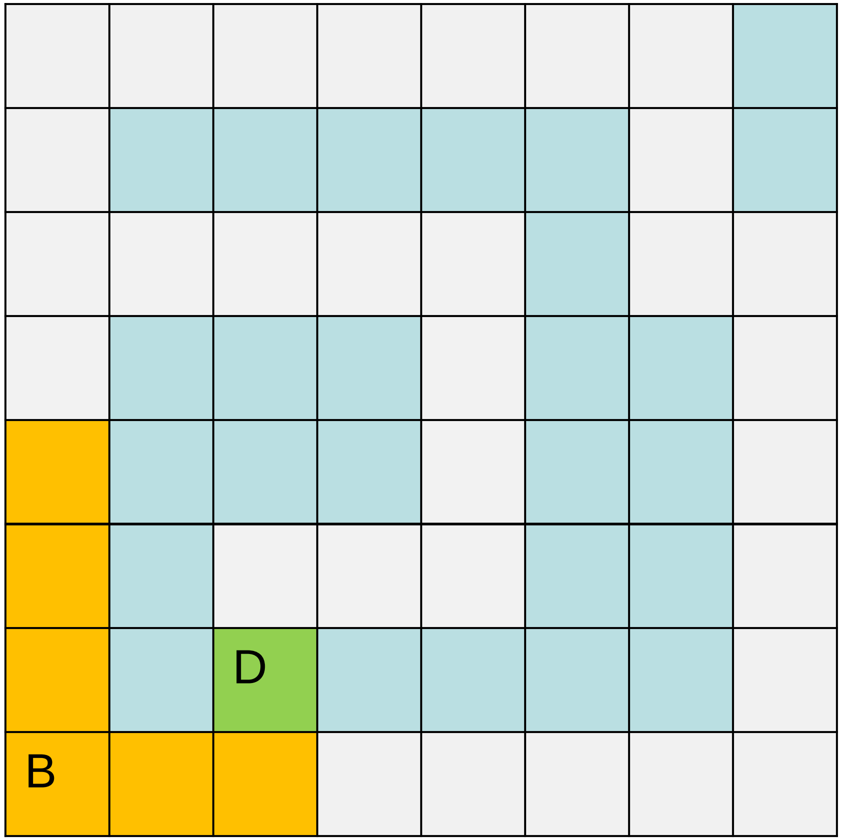
Walkable path

Wall

D — At destination

Example – DFS vs BFS in Path Search

BFS Path 1 & 2:



Notes:

- B — Beginning postion
- D — Destination
- Walkable path
- Wall
- D — At destination

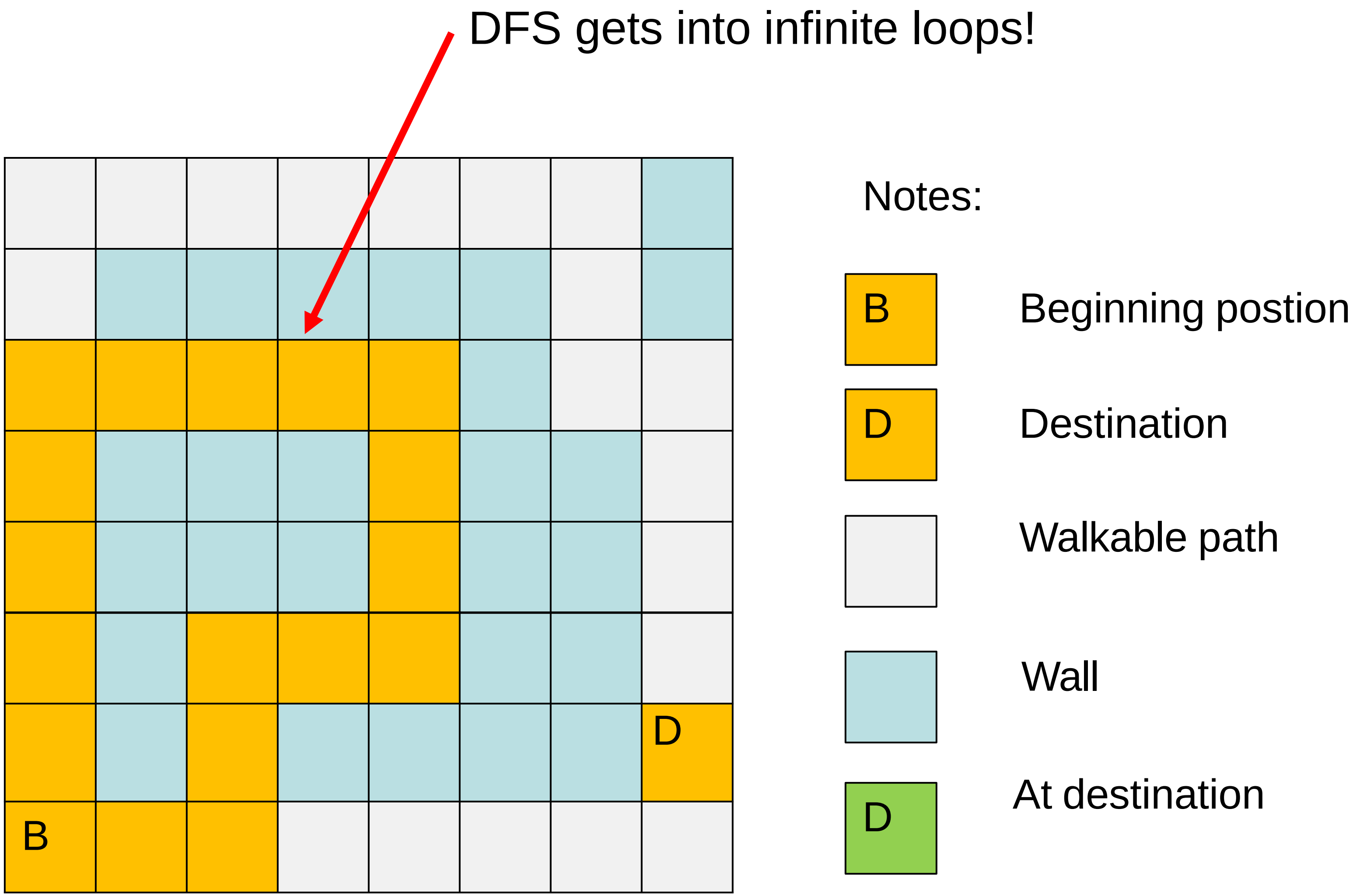
Limitations of DFS

From the example, we may notice that:

- If we apply DFS on the graph, and every time your partial path came to an intersection, you always searched the left-most street first. Then you might just keep going around the same block indefinitely.

- BFS guarantees success on finding a path (if it exists), as it explores all possible directions.

Limitations of DFS



Outline

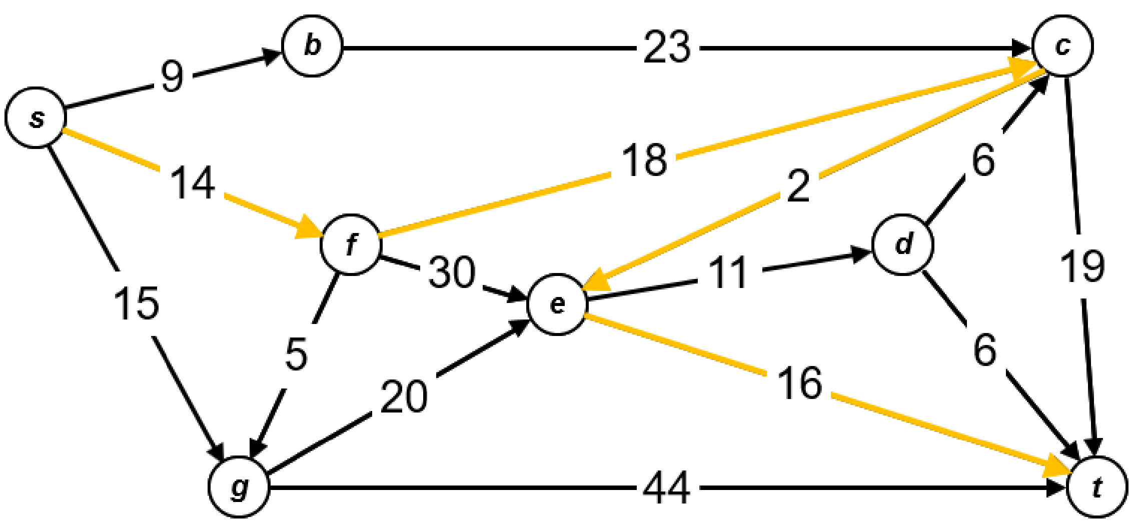
1. Different Graph Search Techniques
   (a) Depth First Search
   (b) Dijkstra’s Algorithm

2. Take-home Exercises

Motivation

- The BFS and DFS usually works well for un-weighted graph search problems, however, they are not the optimal algorithms to solve weighted graph search problems.

- Dijkstra's algorithm is a popular algorithm to solve shortest path finding problem in weighted graphs.



Motivation

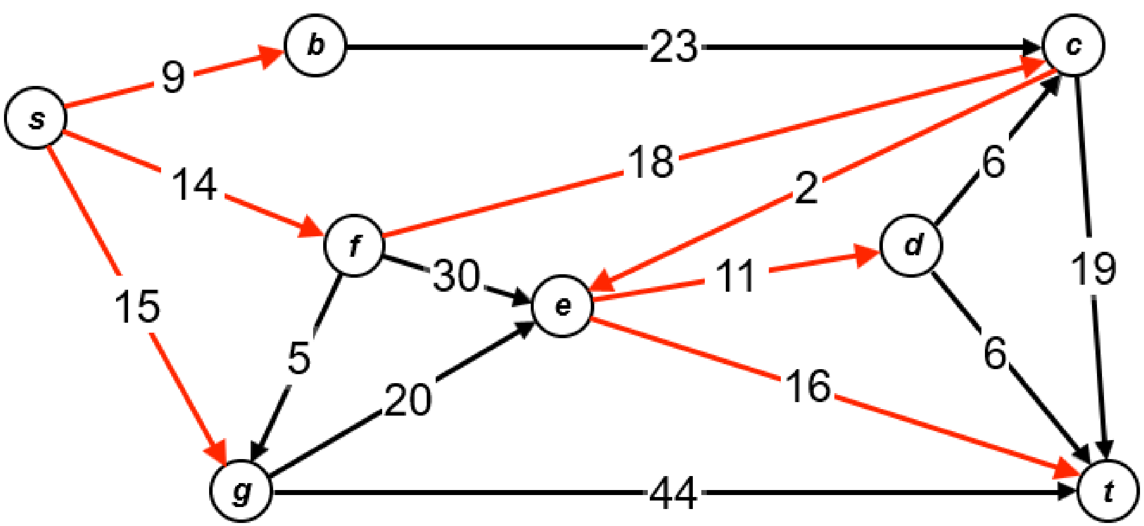
- Formal Problem Definition for the Dijkstra Algorithm:

  Given a graph with non-negative edge weights G = (V, E) and a distinguished source vertex, s ∈ V, determine the shortest distance from the source vertex to every vertex in the graph.

Idea of the Dijkstra Algorithm

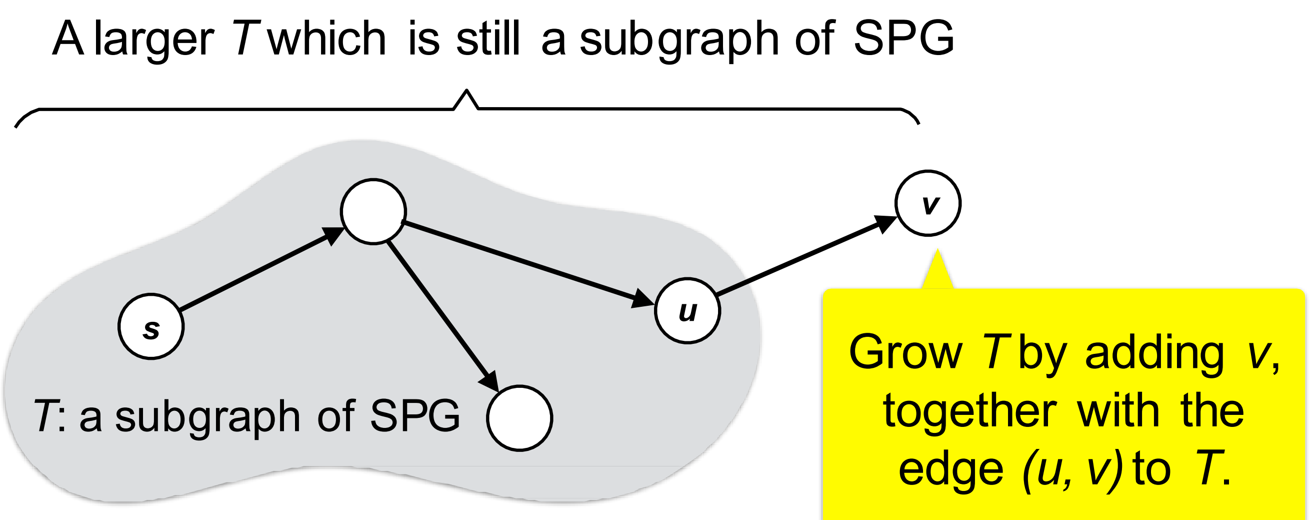
- To find the shortest path from source node *s* to any node *x* in the graph, Dijkstra Algorithm generates a shortest path graph (SPG) with source node *s* as the starting point.

- The SPG stores all the vertices and edges that will result in shortest path from node *s* to any vertex *v* in the graph.



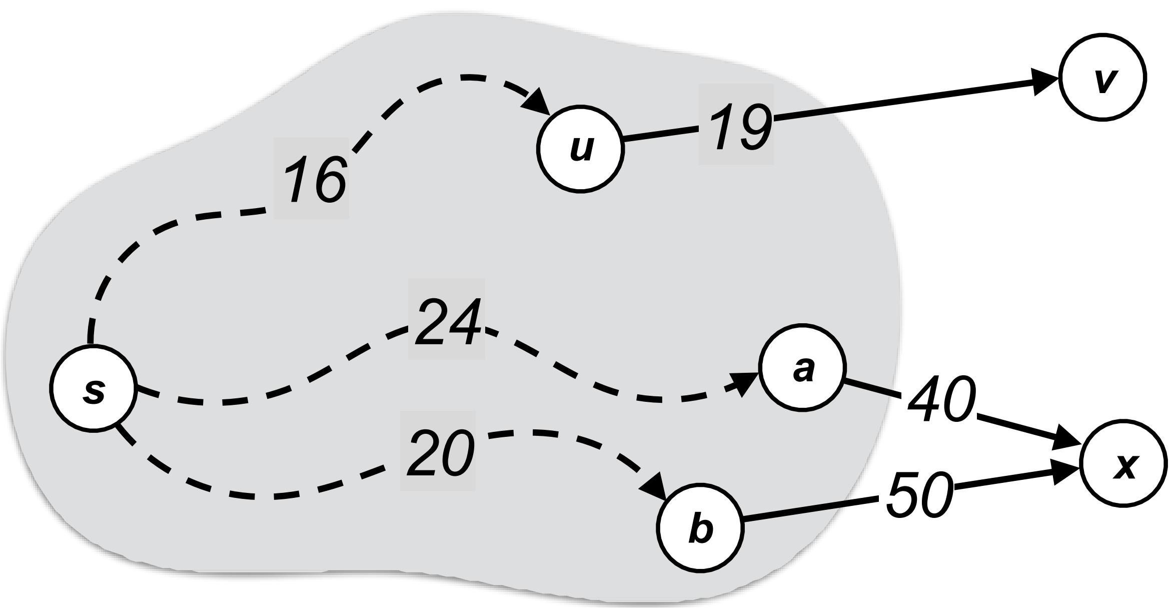
Idea of the Dijkstra Algorithm

- To generate the complete SPG, the algorithm starts from the smallest subgraph of SPG that contains only source node *s*.
- It iteratively attaches a "correct" edge and vertex to the subgraph such that the larger subgraph is still a subgraph of SPG.



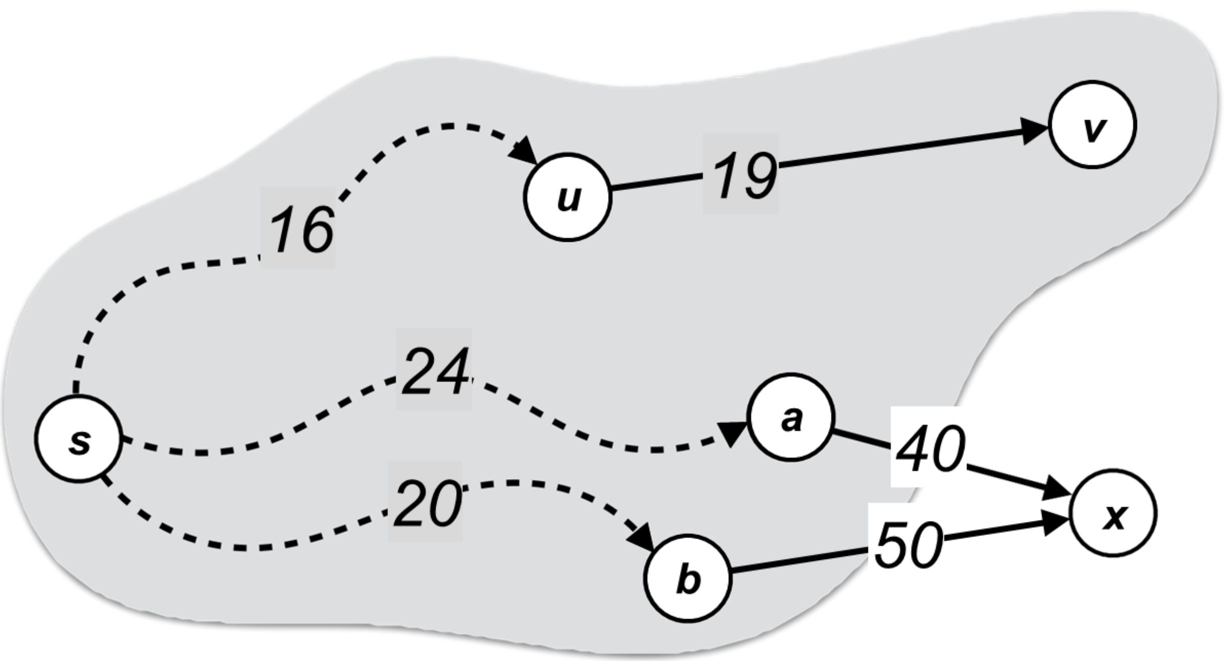
Idea of the Dijkstra Algorithm

- To pick the "correct" edge and vertex in each iteration, the algorithm chooses the vertex not in the SPG which has the "closest" distance with a node in the SPG.

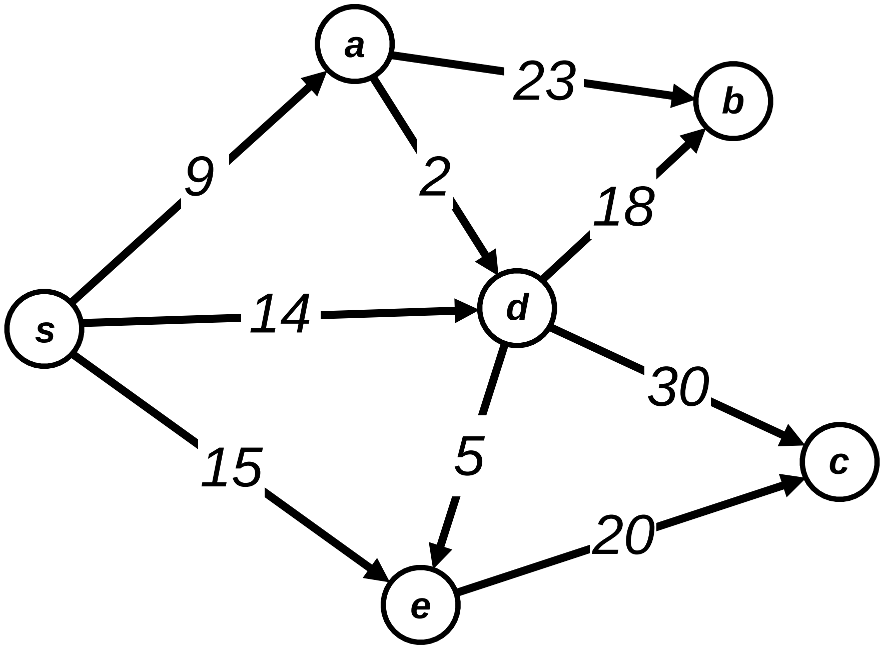


Idea of the Dijkstra Algorithm

- To pick the "correct" edge and vertex in each iteration, the algorithm chooses the vertex not in the SPG which has the "closest" distance with a node in the SPG.



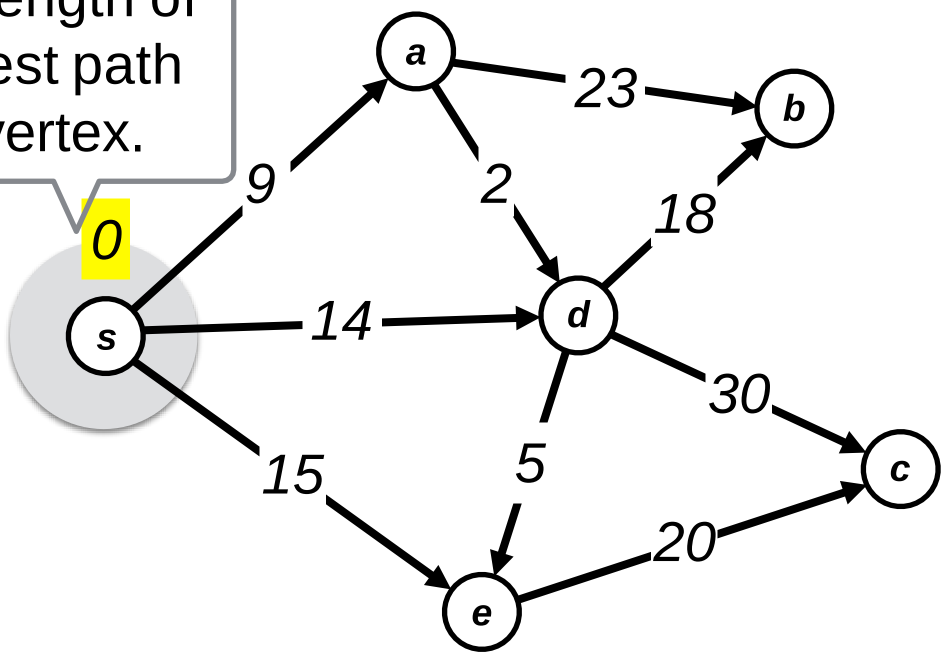
Example – Dijkstra Algorithm Illustration



$T = \{\}$

Example – Dijkstra Algorithm Illustration

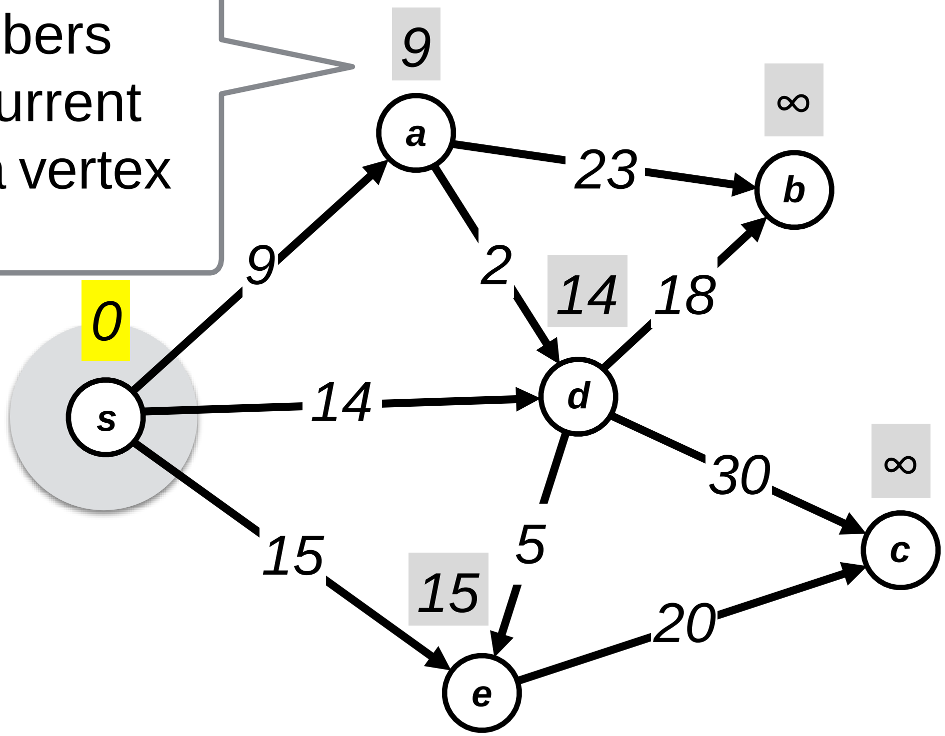
Highlighted numbers denote the length of of the shortest path from *s* to a vertex.



T = {s}

Example – Dijkstra Algorithm Illustration

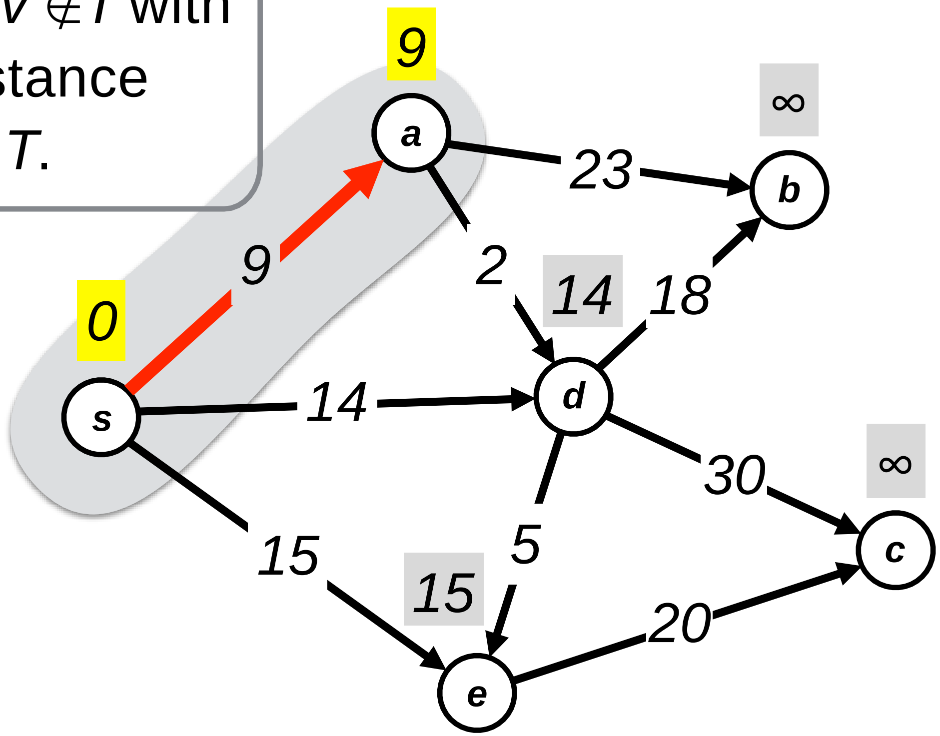
Shaded numbers denote the current distance of a vertex from *s*.



$T = \{s\}$

Example – Dijkstra Algorithm Illustration

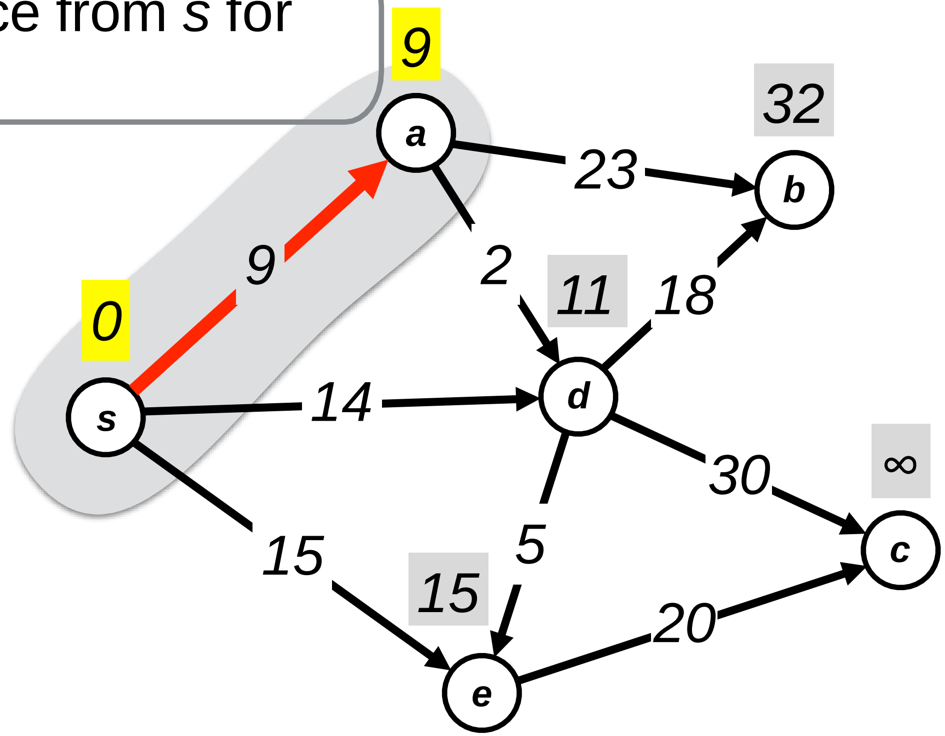
Add the vertex $v \notin T$ with the smallest distance from a node in T.



$T = \{s, a\}$

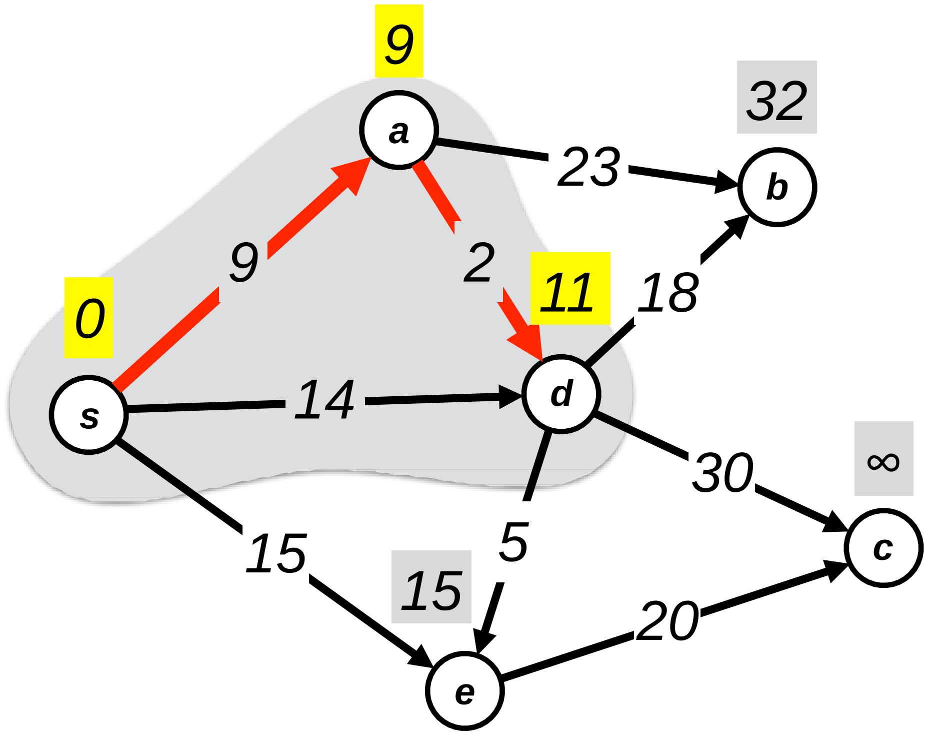
Example – Dijkstra Algorithm Illustration

Update distance from *s* for all $x \notin T$.



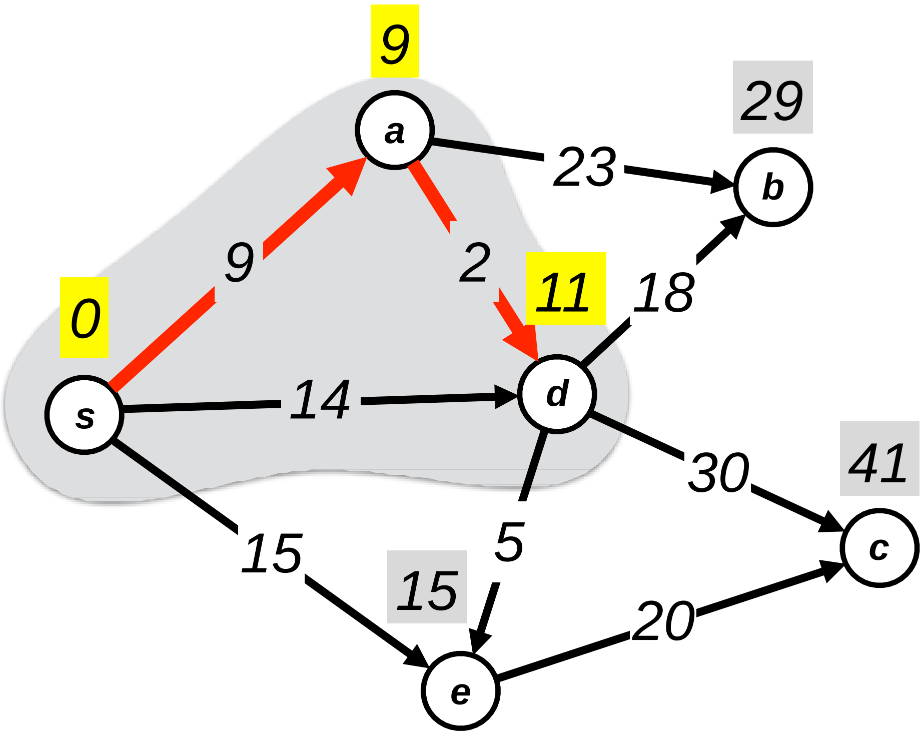
$T = \{s, a\}$

Example – Dijkstra Algorithm Illustration



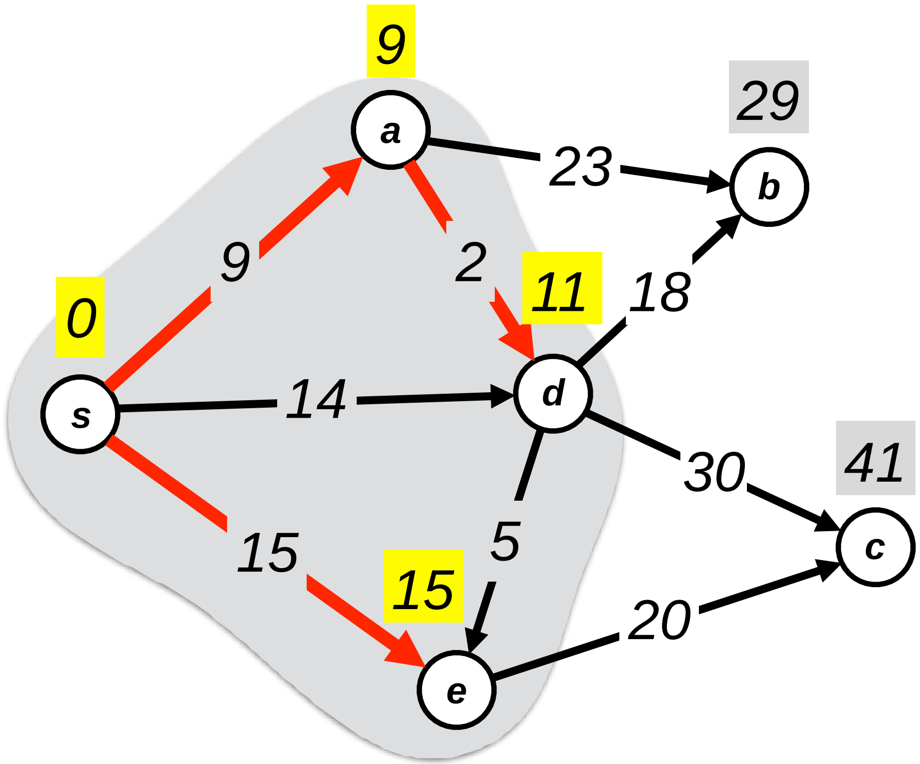
$T = \{s, a, d\}$

Example – Dijkstra Algorithm Illustration



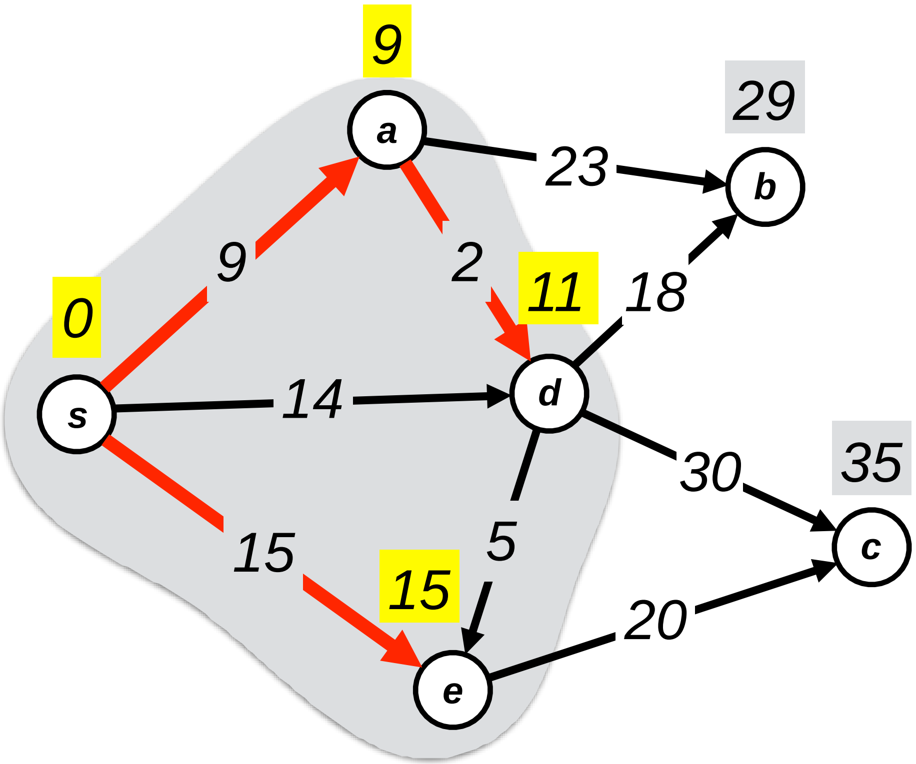
$T = \{s, a, d\}$

Example – Dijkstra Algorithm Illustration



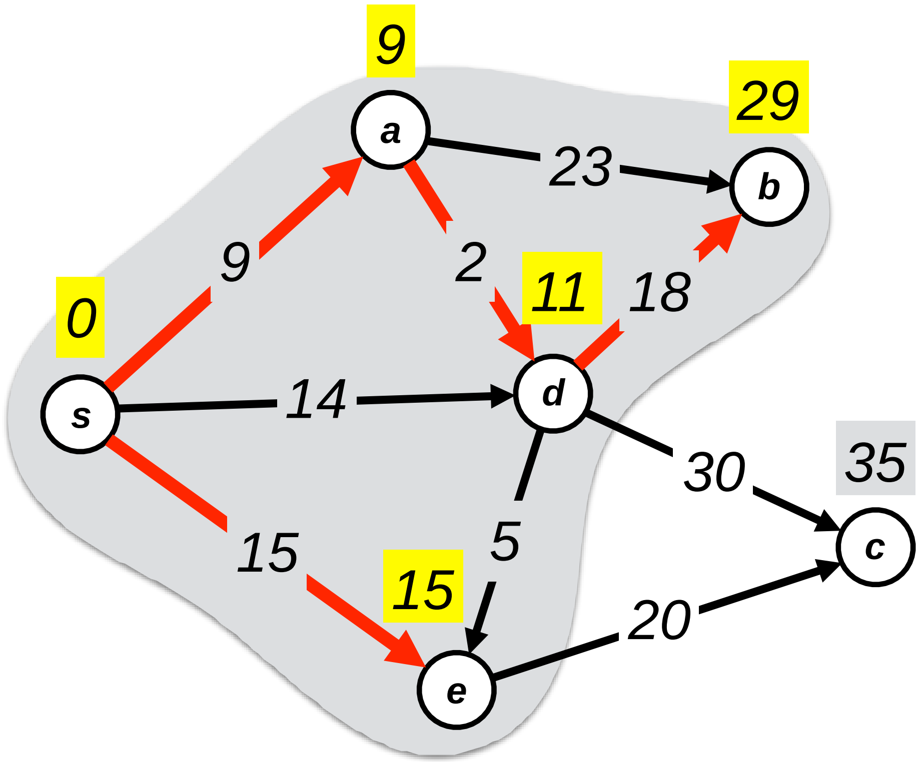
$T = \{s, a, d, e\}$

Example – Dijkstra Algorithm Illustration



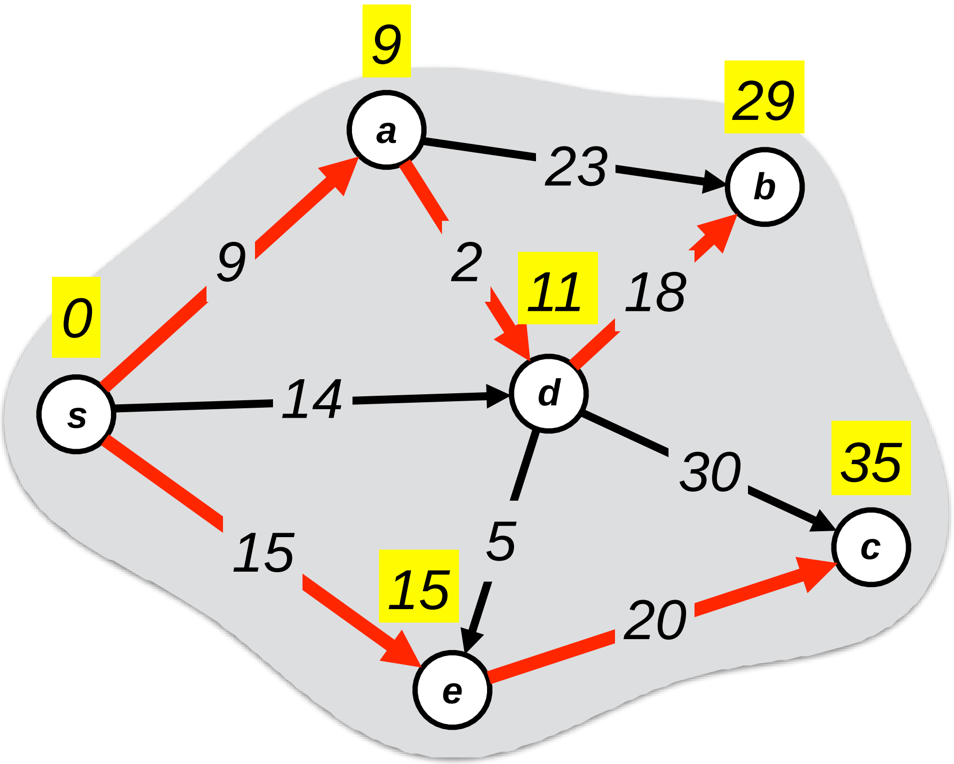
$T = \{s, a, d, e\}$

Example – Dijkstra Algorithm Illustration



$T = \{s, a, d, e, b\}$

Example – Dijkstra Algorithm Illustration



$T = \{s, a, d, e, b, c\}$

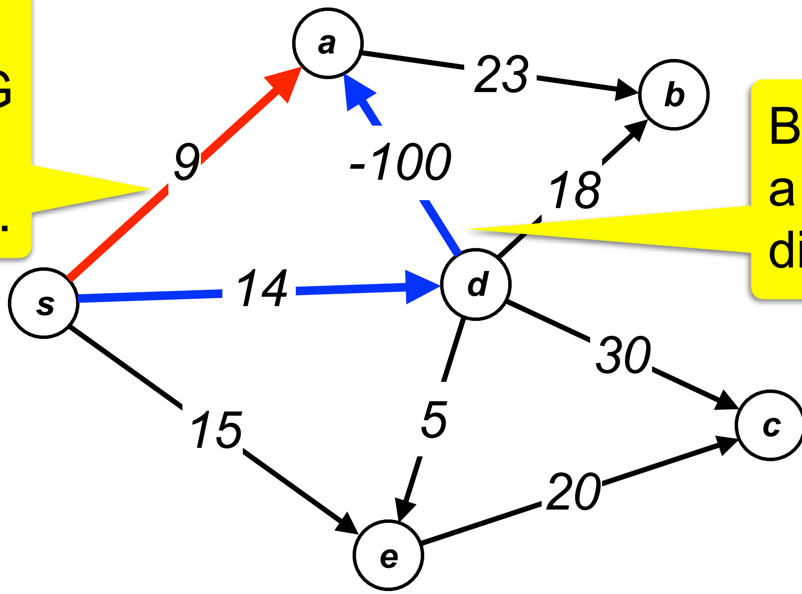
Dijkstra Algorithm Pseudocode (Naïve Version)

```
Dijkstra(G, s)   (s is the vertex where the search starts) {
    /* dist(x) stores the shortest distance found from s to vertex x /*
    initialize dist(s) = 0, and dist(x) = ∞ for other vertices x;
    T = { };
    while T ≠ G do
        pick the node v ∉ T with the smallest dist(v);
        add v to T;
        for all edges (v, x) ∈ E
            if dist(x) > dist(v) + Length(v, x)
                dist(x) = dist(v) + Length(v, x);
}
```

Limitations of Dijkstra Algorithm

- Dijkstra's algorithm cannot handle graphs with negative edges

Dijkstra will add this edge to SPG at step 1, and assert dist(*a*) = 9.



But there is a path with distance -86

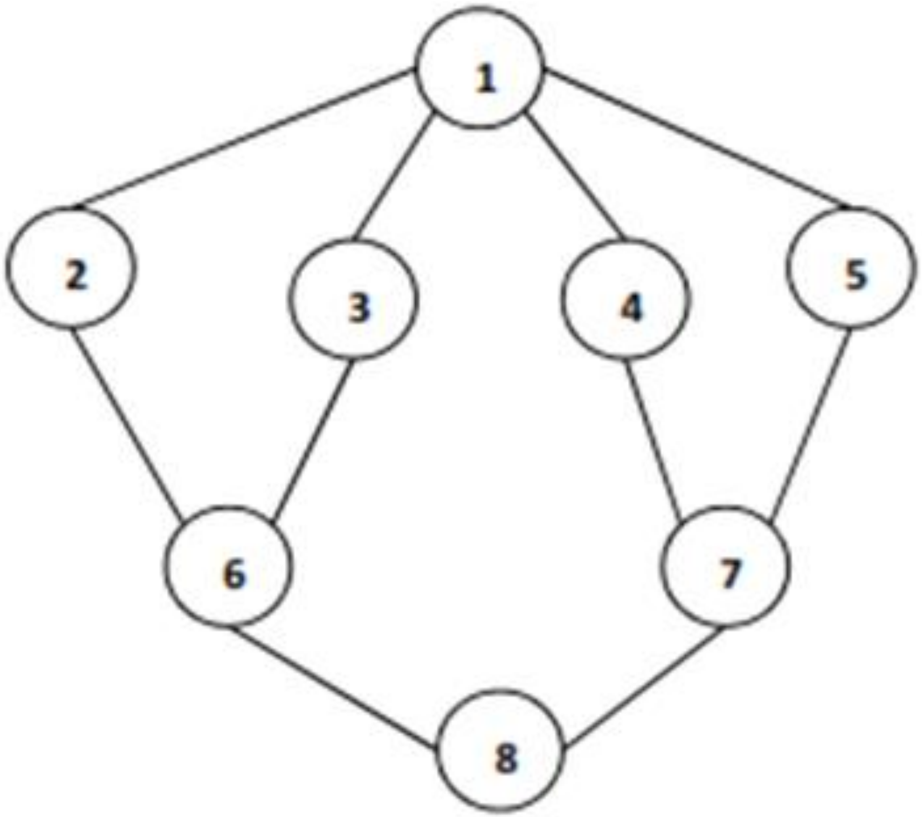
Outline

1. Different Graph Search Techniques
    (a) Depth First Search
    (b) Dijkstra’s Algorithm

2. Take-home Exercises

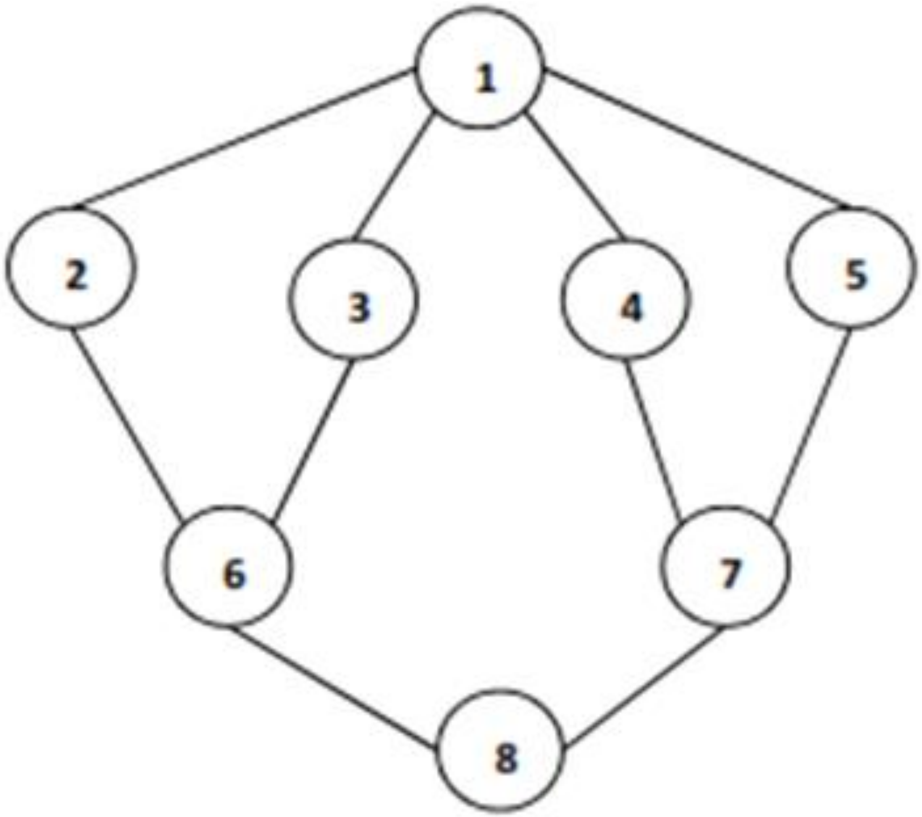
Exercise 1

Give the BFS and DFS traversal order of the following graph.



Exercise 1

Give the BFS and DFS traversal order of the following graph.



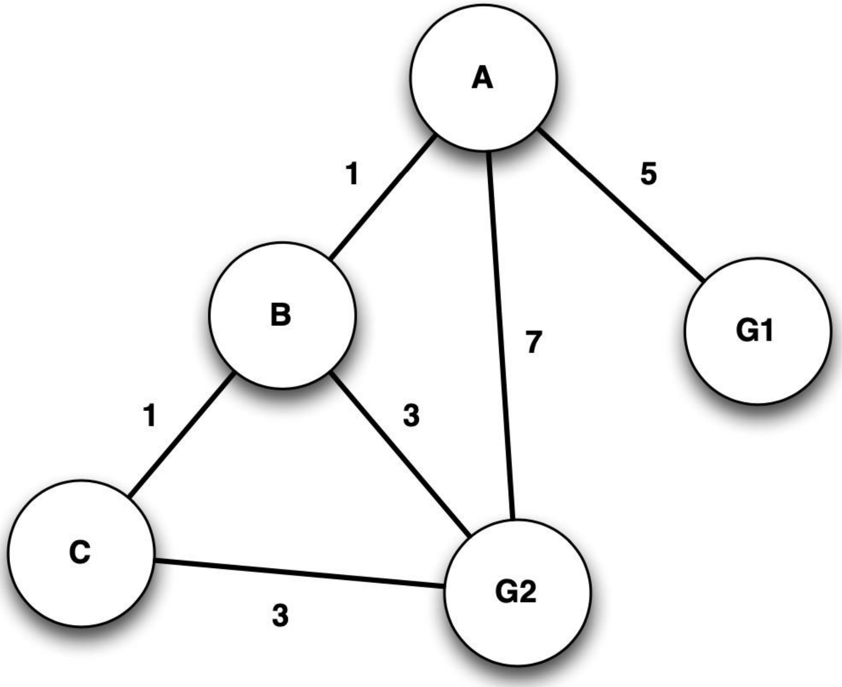
Solution:

BFS: 1,2,3,4,5,6,7,8

DFS: 1,2,6,3,8,7,4,5

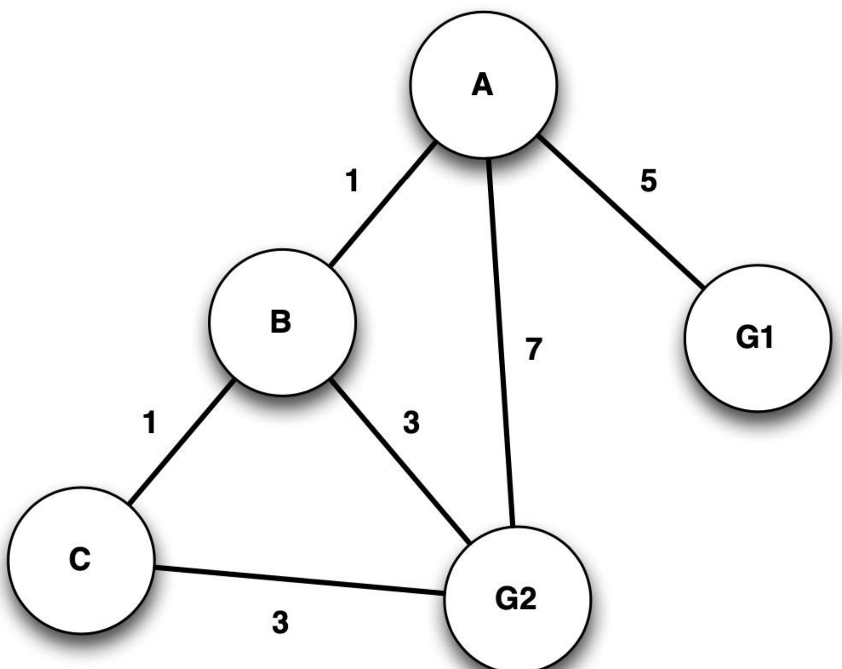
Exercise 2

G1 and G2 are goal nodes, and A is the start node. Find the shortest distance and shortest path from A to G1 or G2.



Exercise 2

G1 and G2 are goal nodes, and A is the start node. Find the shortest distance and shortest path from A to G1 or G2.



Solution:

Step 1:

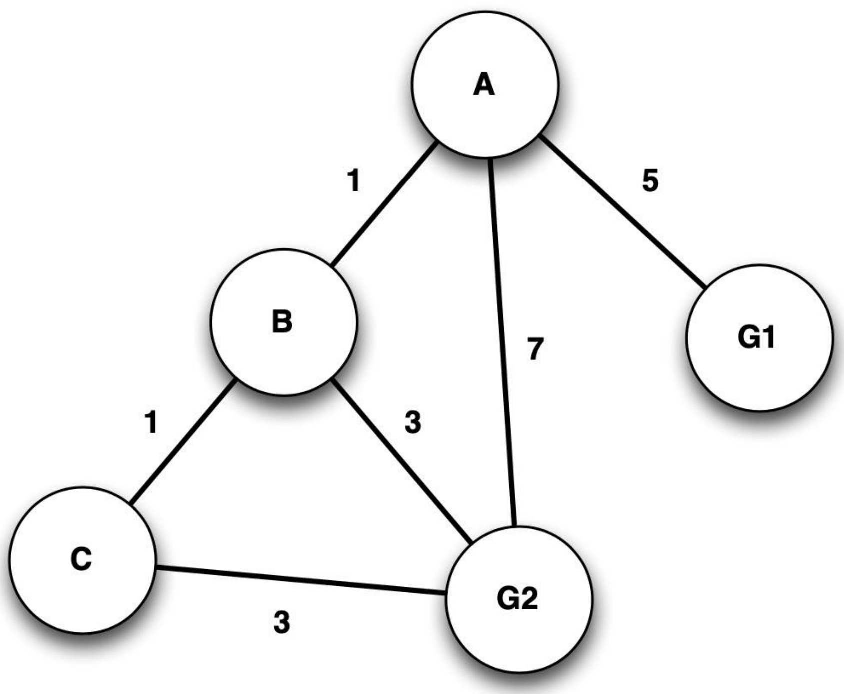
T = {}

dist(x) = {0, ∞, ∞, ∞, ∞}

A, B, C, G1, G2

Exercise 2

G1 and G2 are goal nodes, and A is the start node. Find the shortest distance and shortest path from A to G1 or G2.



Solution:

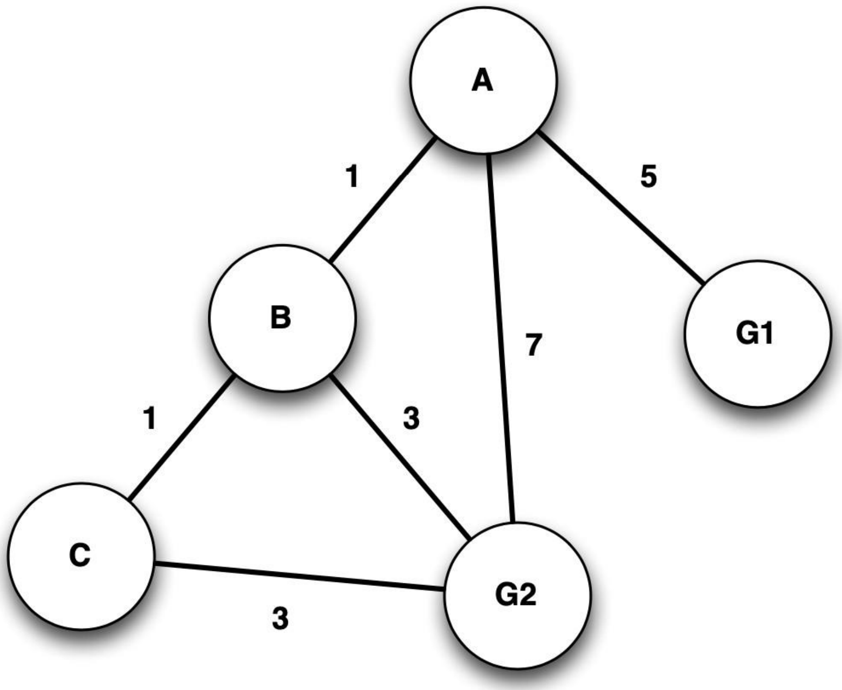
Step 2:

T = {A}

dist(x) = {0, 1, ∞, 5, 7}

Exercise 2

G1 and G2 are goal nodes, and A is the start node. Find the shortest distance and shortest path from A to G1 or G2.

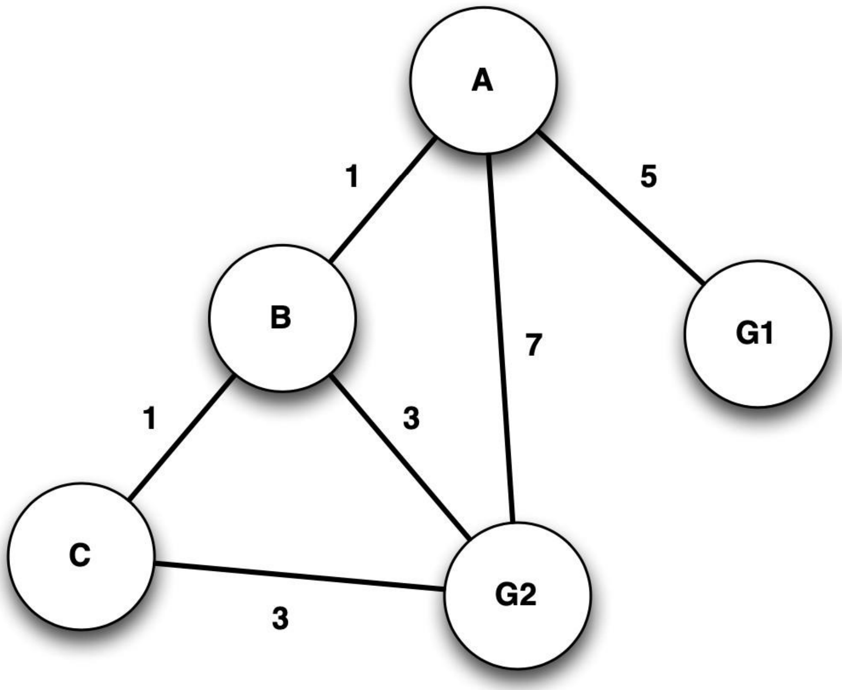


Solution:

Step 3:
T = {A, B}
dist(x) = {0, 1, 2, 5, 4}

Exercise 2

G1 and G2 are goal nodes, and A is the start node. Find the shortest distance and shortest path from A to G1 or G2.

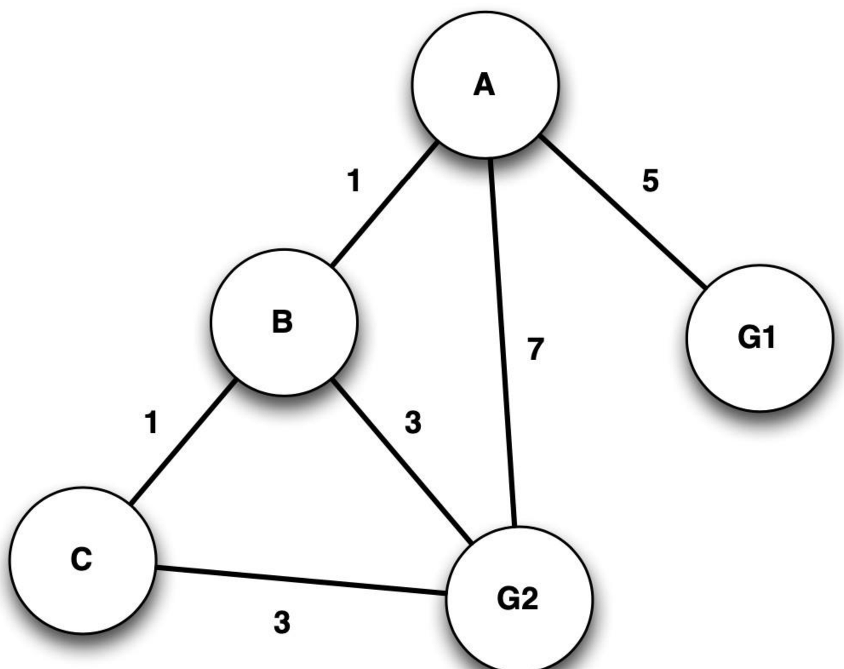


Solution:

Step 4:
T = {A, B, C}
dist(x) = {0, 1, 2, 5, 4}

Exercise 2

G1 and G2 are goal nodes, and A is the start node. Find the shortest distance and shortest path from A to G1 or G2.



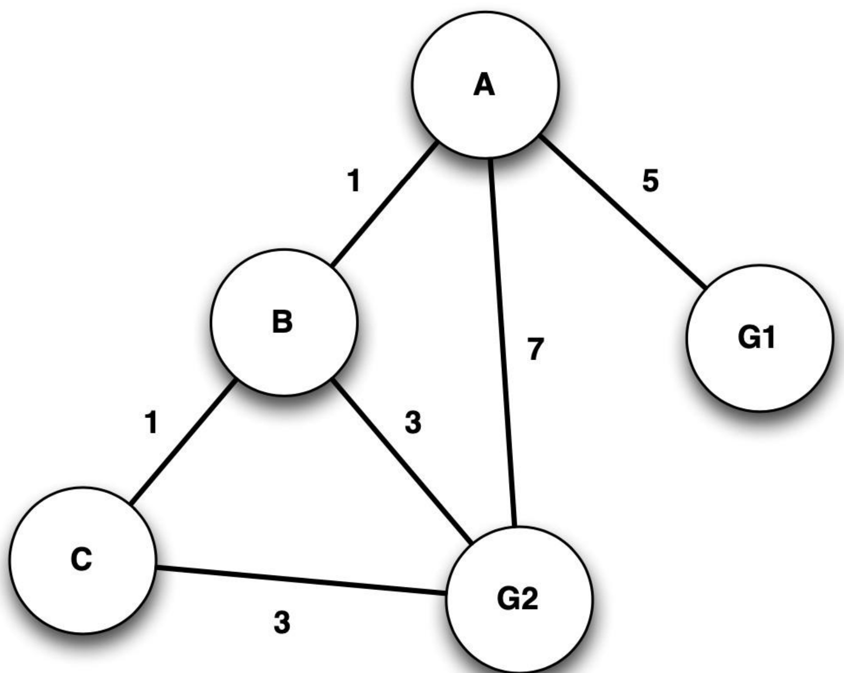
Solution:

Step 5:
T = {A, B, C, G2}
dist(x) = {0, 1, 2, 5, 4}

Distance to G2: 4
Path to G2: A → B → G2

Exercise 2

G1 and G2 are goal nodes, and A is the start node. Find the shortest distance and shortest path from A to G1 or G2.

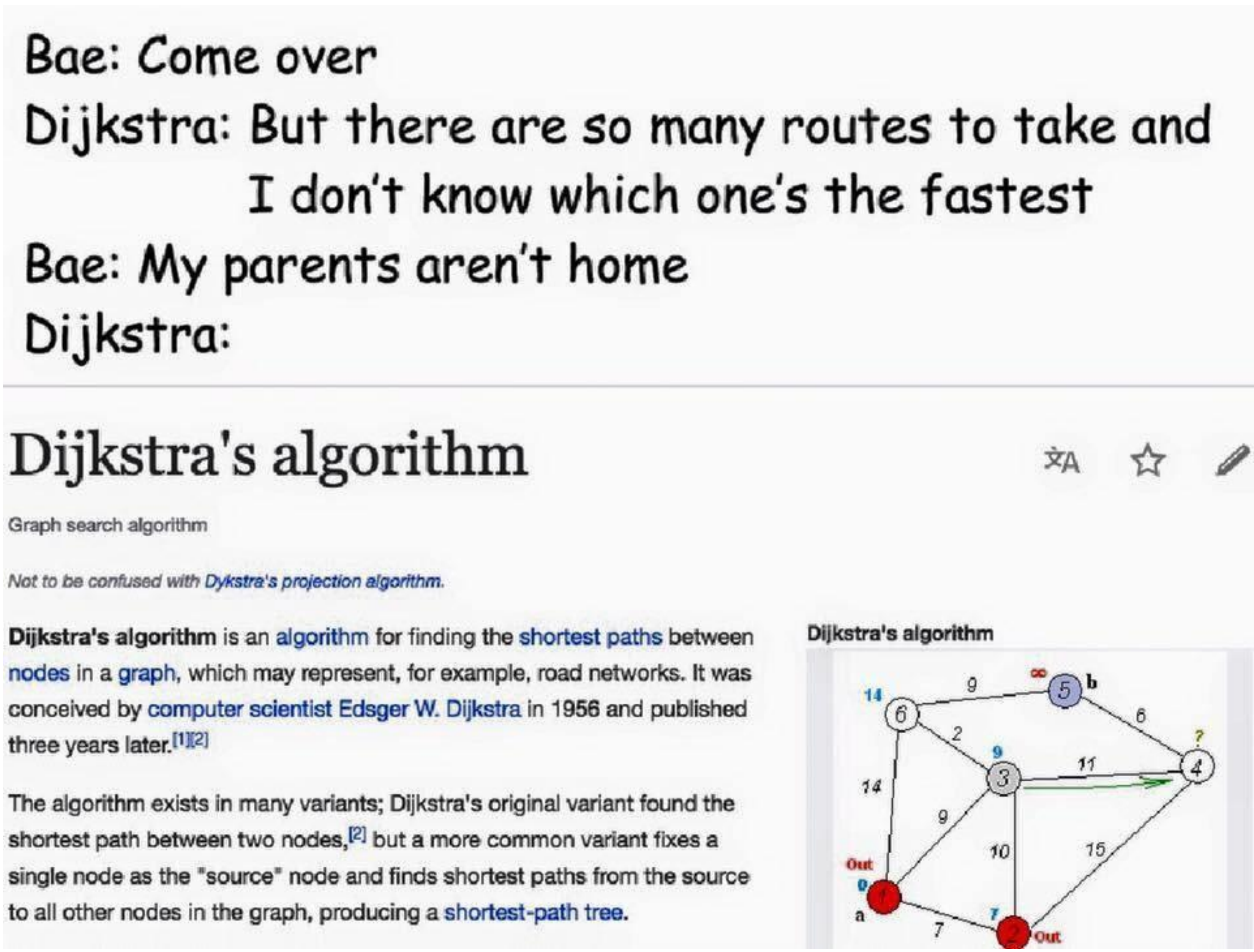


<u>Solution:</u>

Step 6:
T = {A, B, C, G2, G1}
dist(x) = {0, 1, 2, 5, 4}

Distance to G1: 5
Path to G1: A → G1

The End


Bae: Come over
Dijkstra: But there are so many routes to take and I don't know which one's the fastest
Bae: My parents aren't home
Dijkstra:
Dijkstra's algorithm
Graph search algorithm
Not to be confused with Dykstra's projection algorithm.
Dijkstra's algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later.[1][2]
The algorithm exists in many variants; Dijkstra's original variant found the shortest path between two nodes,[2] but a more common variant fixes a single node as the "source" node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.
Dijkstra's algorithm