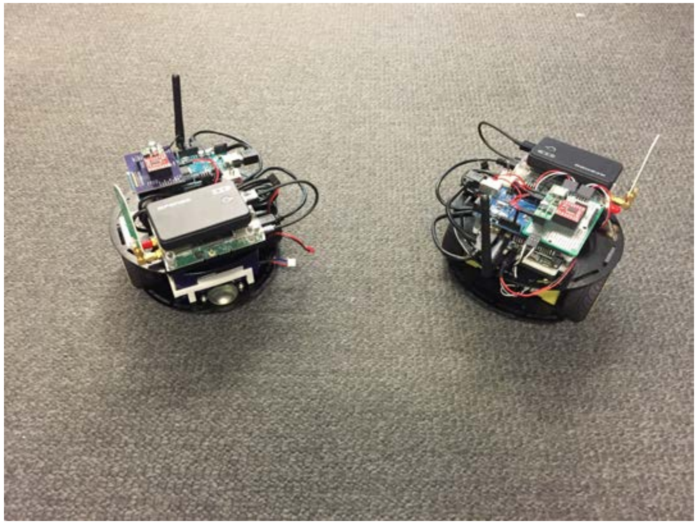
Video 9.1

CJ Taylor

The Motion Planning Problem

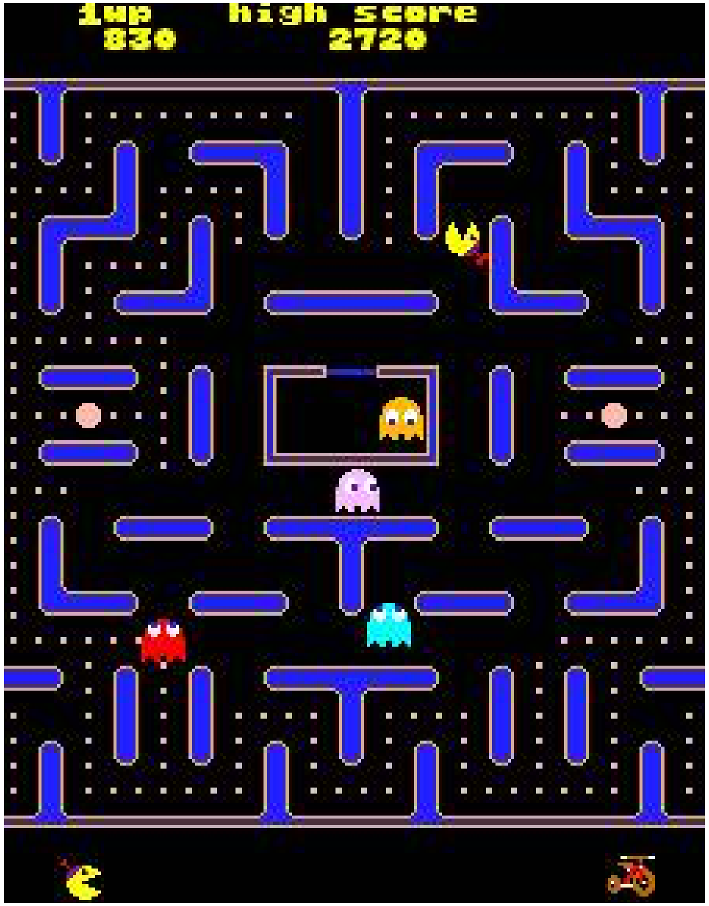
- A special case of the more general planning problem
- The goal is to develop techniques that would allow a robot or robots to automatically decide how to move from one position or configuration to another.
  - Specifically concerned with planning motions – get robot from place A to B

Motion Planning for Robotics



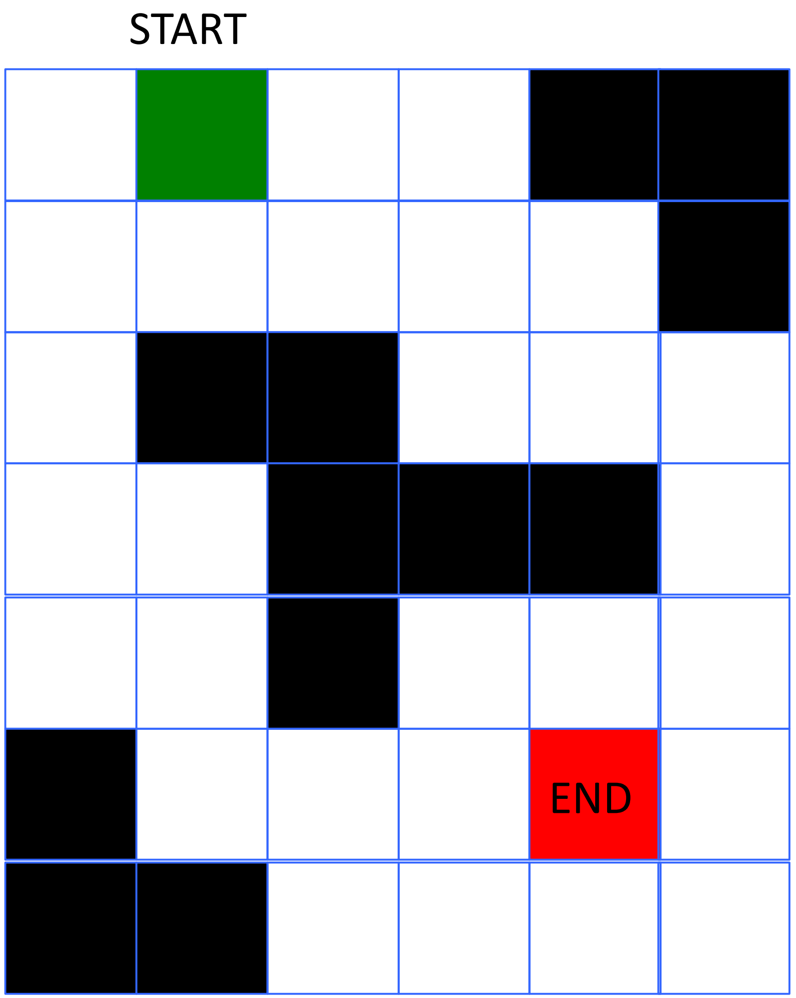
An Example – the PacMan problem

- How does the computer guide the ghosts back to their lair when they are eaten?



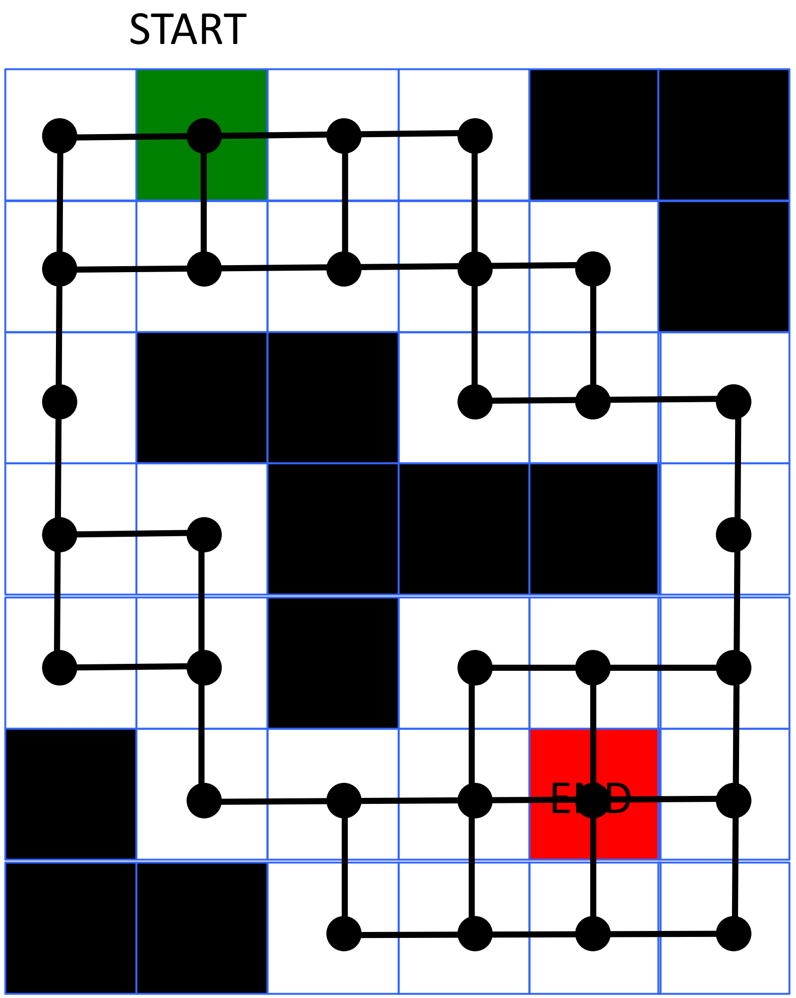
Planning on a grid

- In this example the robot can move between adjacent cells on the grid
- The dark squares indicate obstacles that the robot cannot traverse.



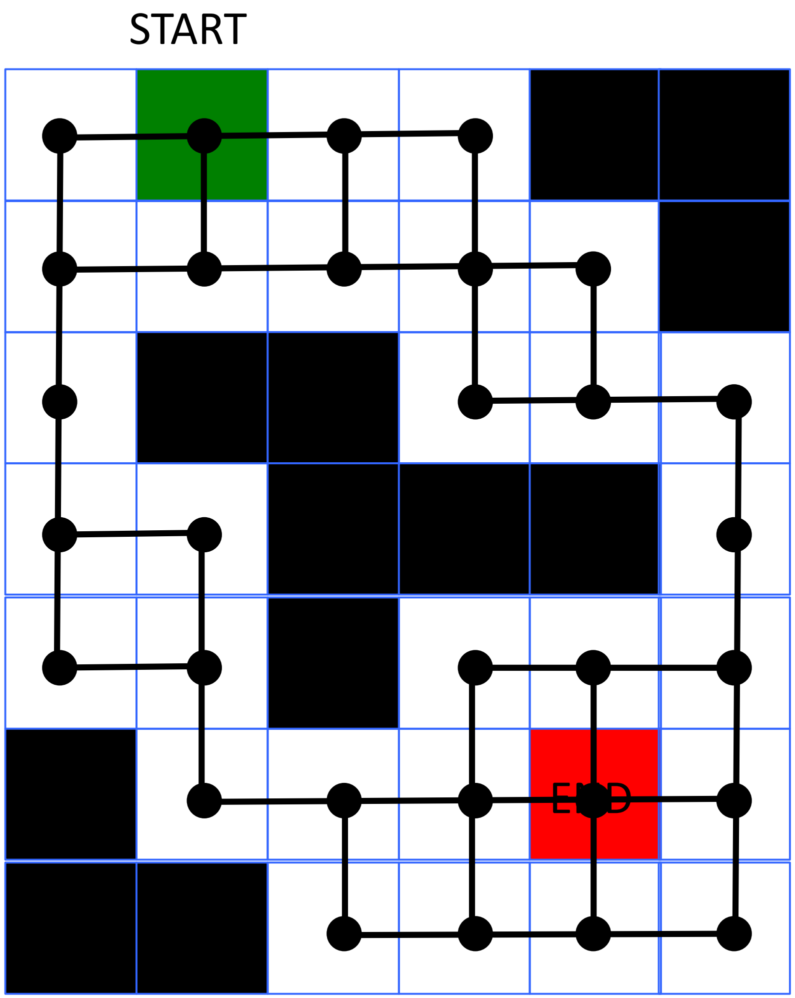
Graph Structure

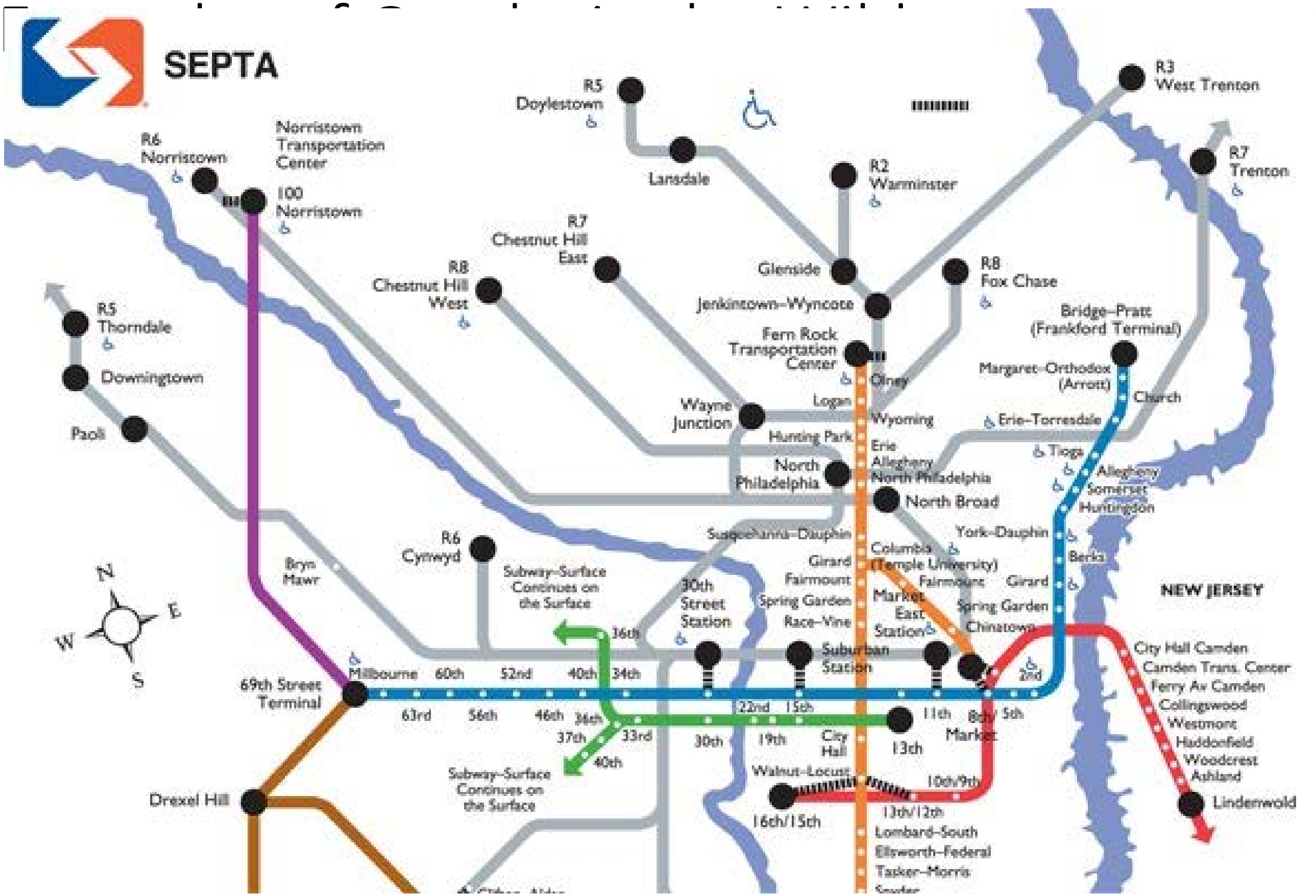
- We can think of the unoccupied cells as nodes and draw edges between adjacent cells as shown here.
- This set of nodes and edges constitutes a graph.



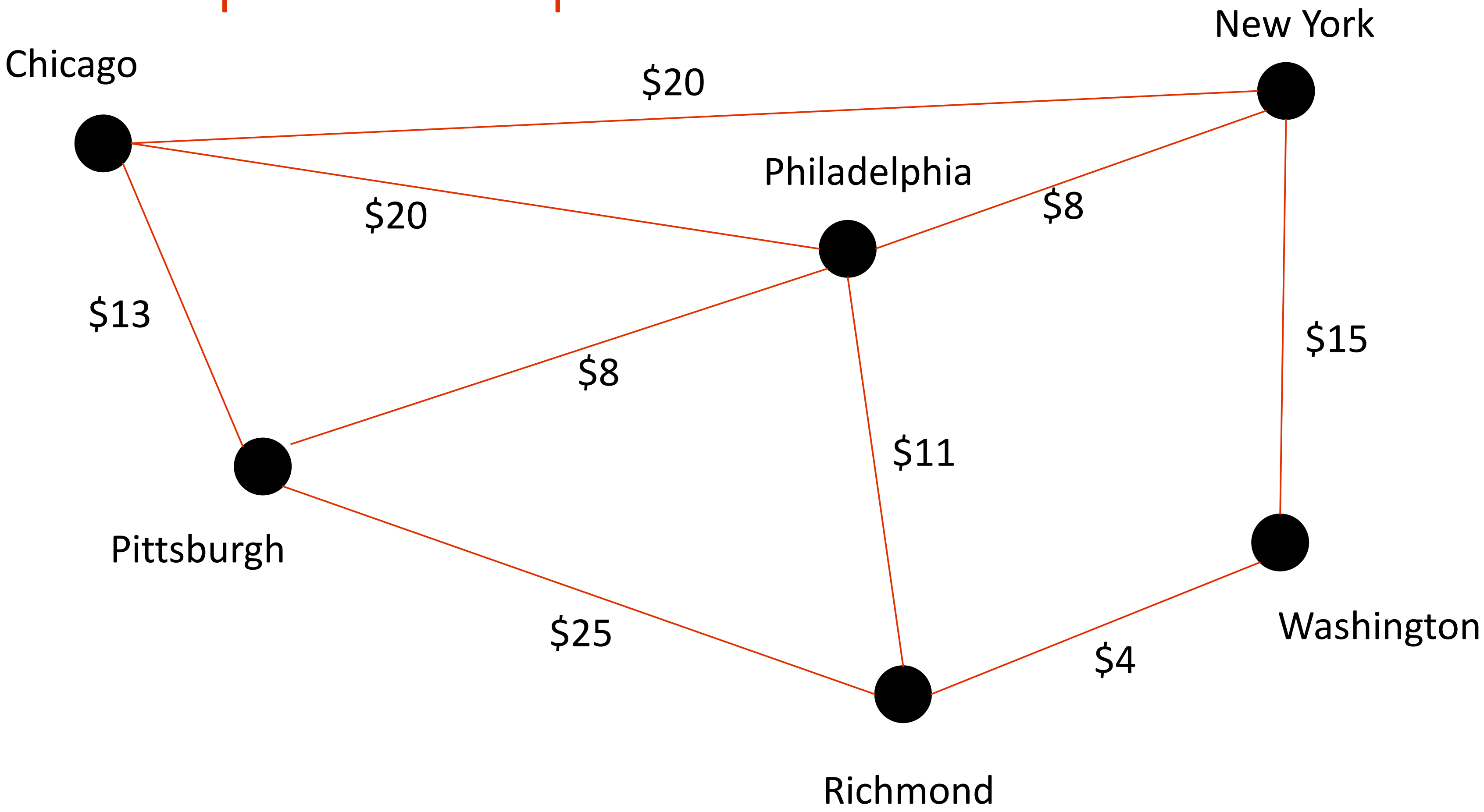
Graph Structure

- A graph, G, consists of a set of vertices, V, and a set of Edges, E, that link pairs of vertices.
- The edges are often annotated with numerical values to indicate relevant quantities like distances or costs.



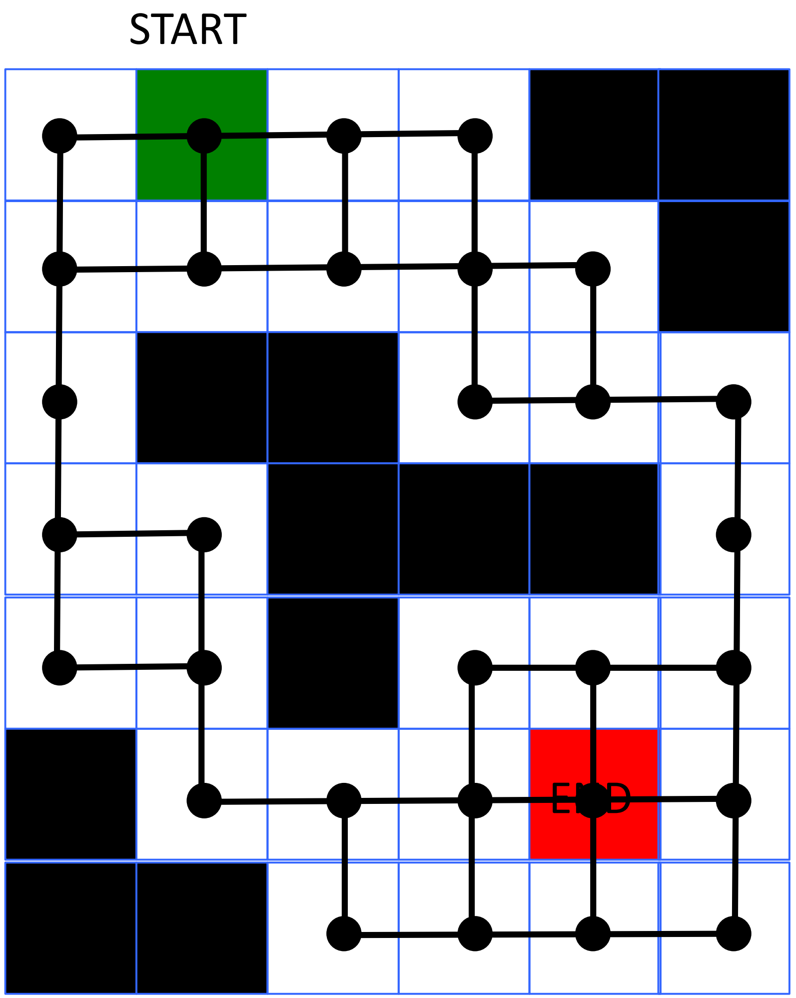

SEPTA
R5 Doylestown
Lansdale
R3 West Trenton
R7 Trenton
R2 Warminster
R6 Norristown
Norristown Transportation Center
100 Norristown
R7 Chestnut Hill East
R8 Chestnut Hill West
Glenside
Jenkintown–Wyncote
R8 Fox Chase
R5 Thorndale
Downingtown
Paoli
Fern Rock Transportation Center
Olney
Logan
Wyoming
Wayne Junction
Hunting Park
Erie
Allegheny
North Philadelphia
North Broad
Bridge–Pratt (Frankford Terminal)
Margaret–Orthodox (Arrott)
Church
Erie–Torresdale
Tioga
Allegheny
Somerset
Huntingdon
York–Dauphin
Berks
Girard
Susquehanna–Dauphin
Columbia (Temple University)
Fairmount
Spring Garden
Chinatown
Race–Vine
Market East Station
Suburban Station
30th Street Station
R6 Cynwyd
Bryn Mawr
Subway-Surface Continues on the Surface
NEW JERSEY
City Hall Camden
Camden Trans. Center
Ferry Av Camden
Collingswood
Westmont
Haddonfield
Woodcrest
Ashland
Lindenwold
Millbourne
69th Street Terminal
60th
52nd
40th
34th
36th
63rd
56th
46th
37th
33rd
40th
30th
22nd
15th
19th
City Hall
13th
11th
8th/Market
5th
2nd
Walnut–Locust
16th/15th
10th/9th
13th/12th
Lombard–South
Ellsworth–Federal
Tasker–Morris
Drexel Hill
N
S
E
W

Examples of Graphs in the Wild – Toll Chart



Graph Structure

- In this grid graph we will implicitly associate a cost or distance of 1 with every edge in the graph since they link adjacent cells.

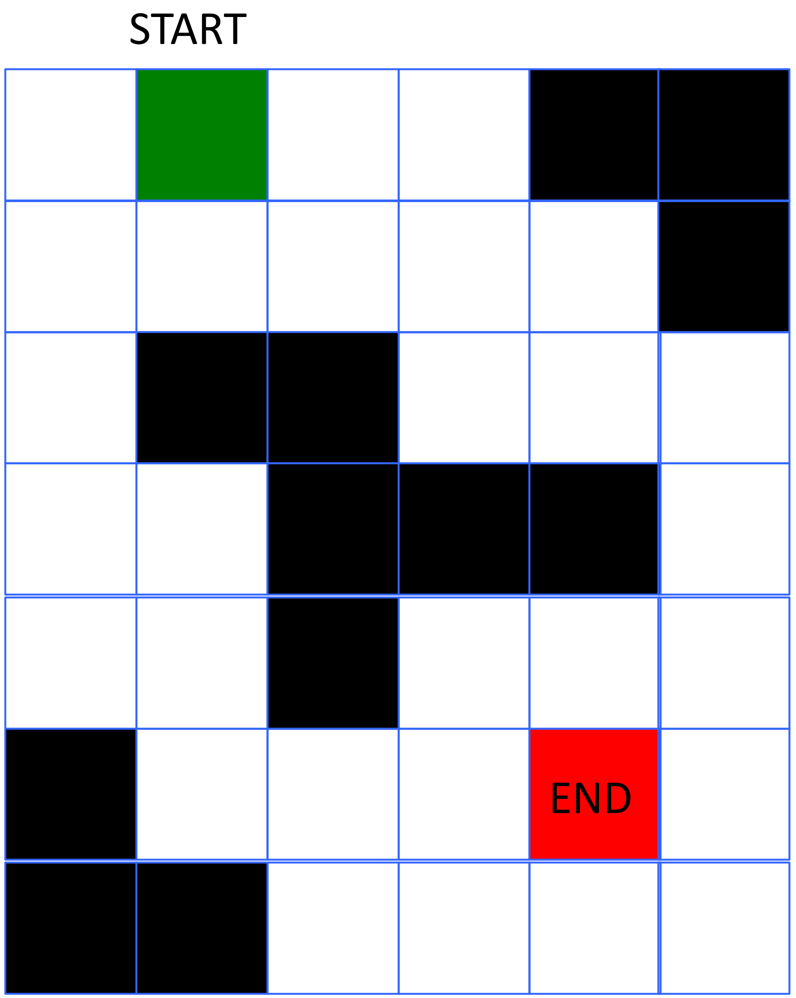


Video 9.2

CJ Taylor

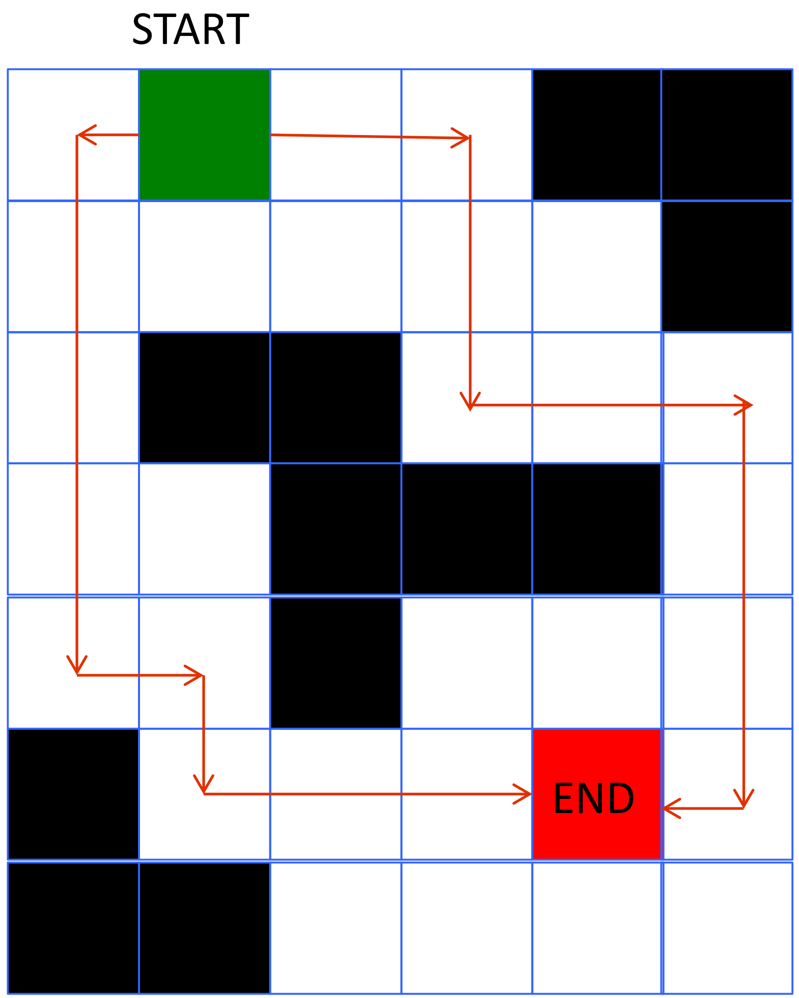
Planning on a grid

- The goal is to construct a path through the grid/graph from the start to the goal



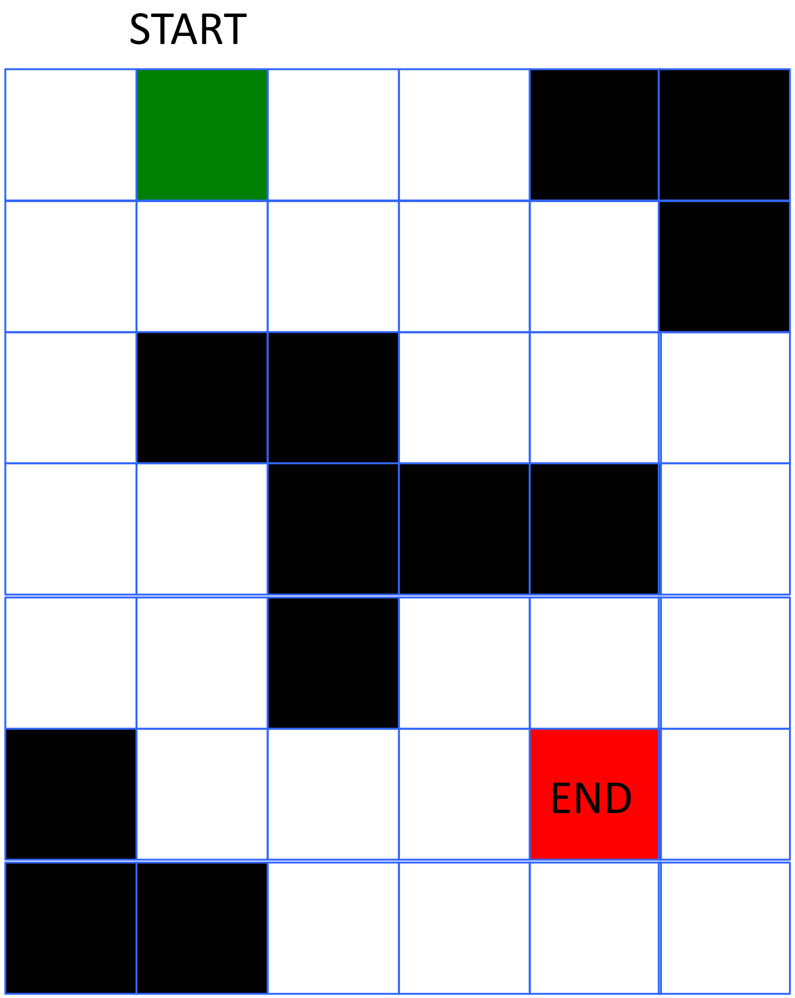
Planning on a grid

- Typically there are many possible paths between two nodes.
- We are usually interested in the shortest paths



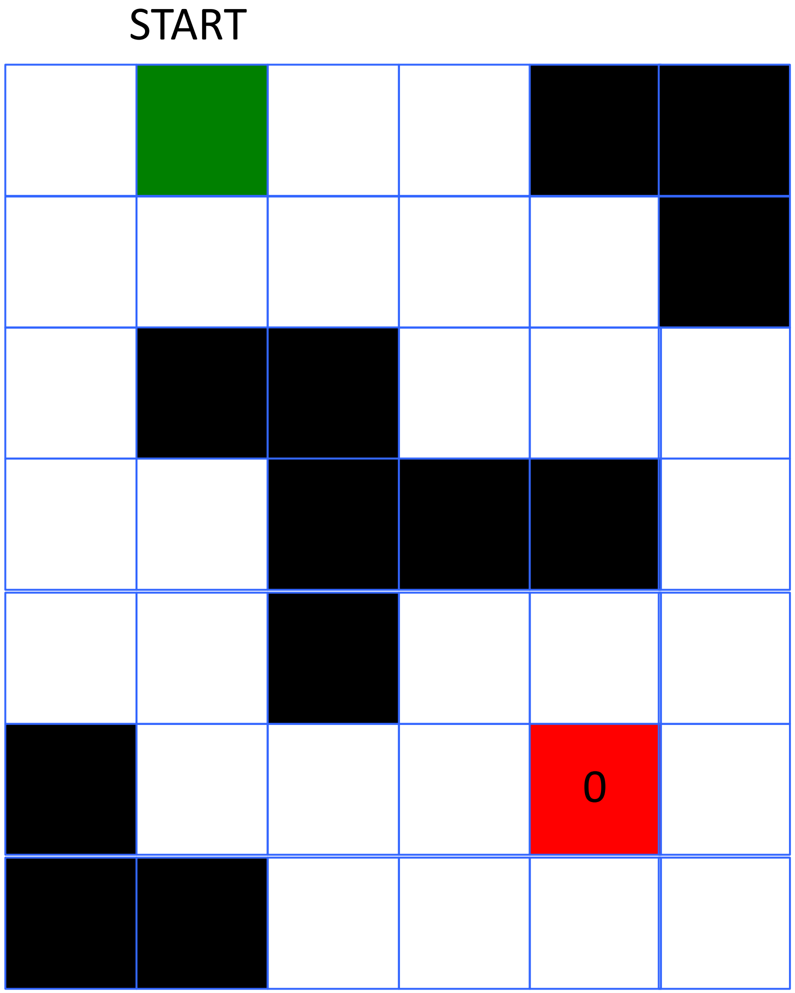
Planning on a grid

- Goal:
  - Construct the shortest path between the start and the goal location.



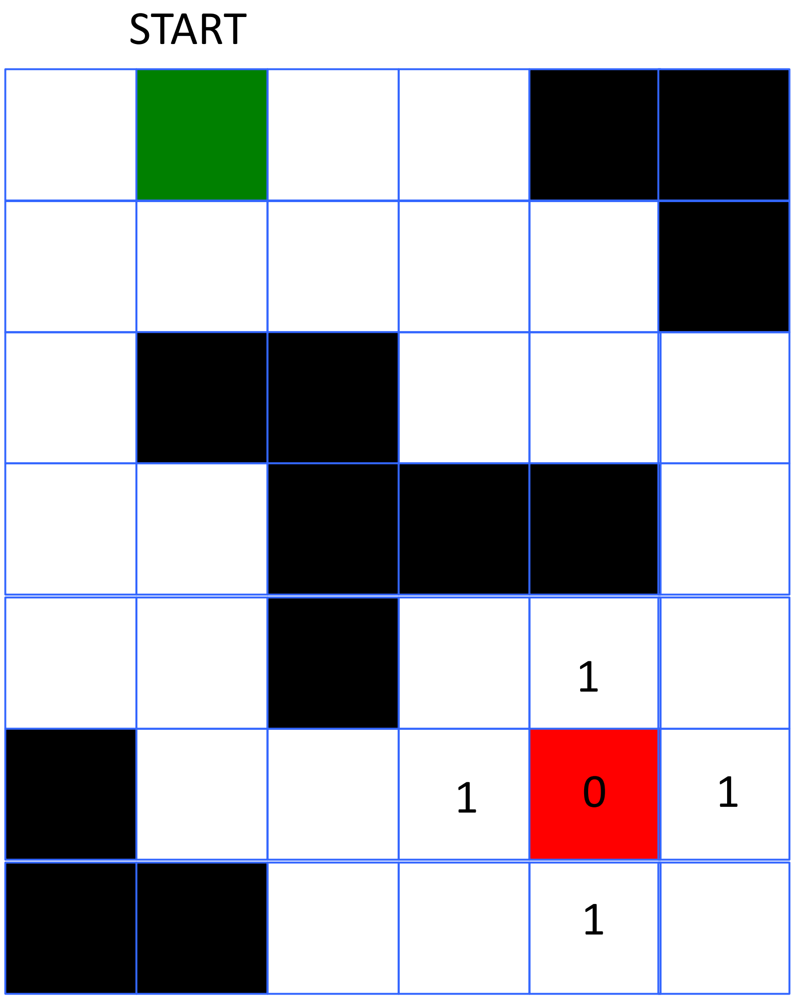
Planning Procedure – Grassfire Algorithm

- Begin by marking the destination node with a distance value of 0

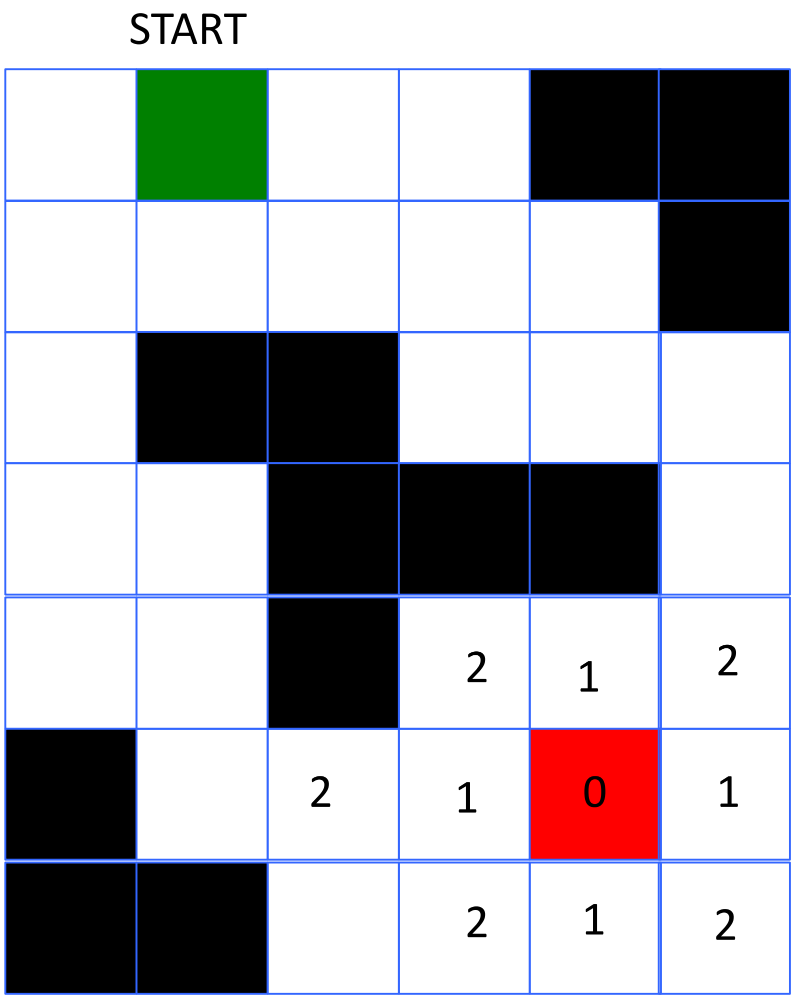


Planning Procedure – Grassfire Algorithm

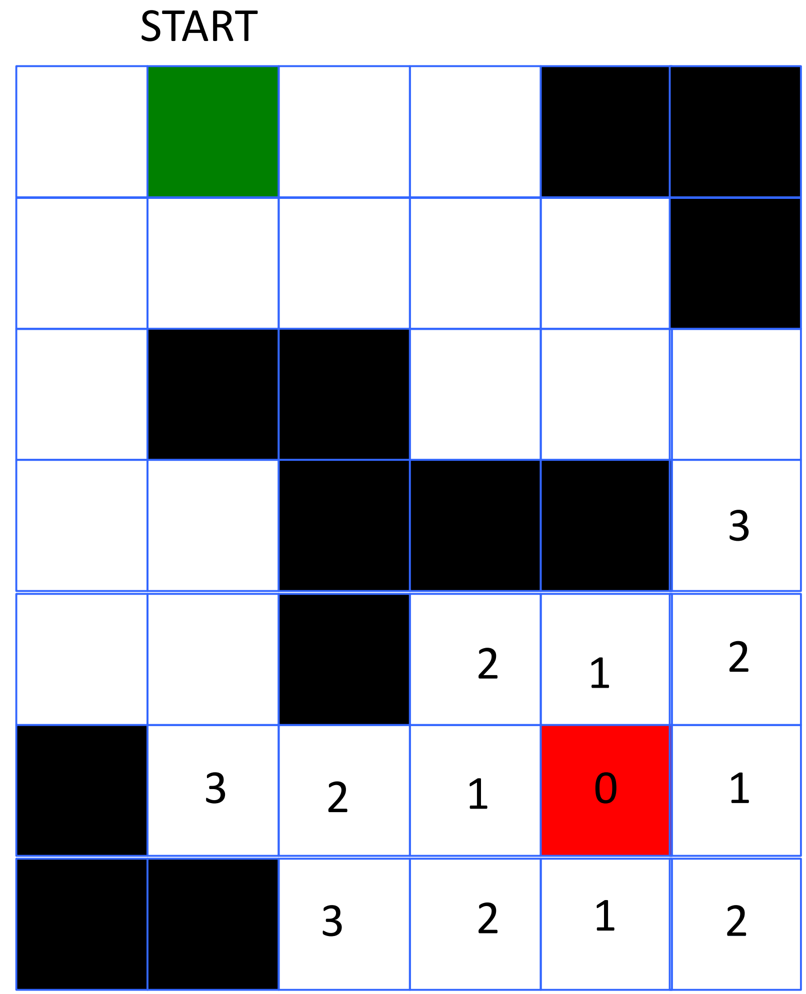
- On every iteration find all the unmarked nodes adjacent to marked nodes and mark them with that distance value + 1.



Planning Procedure – Grassfire Algorithm

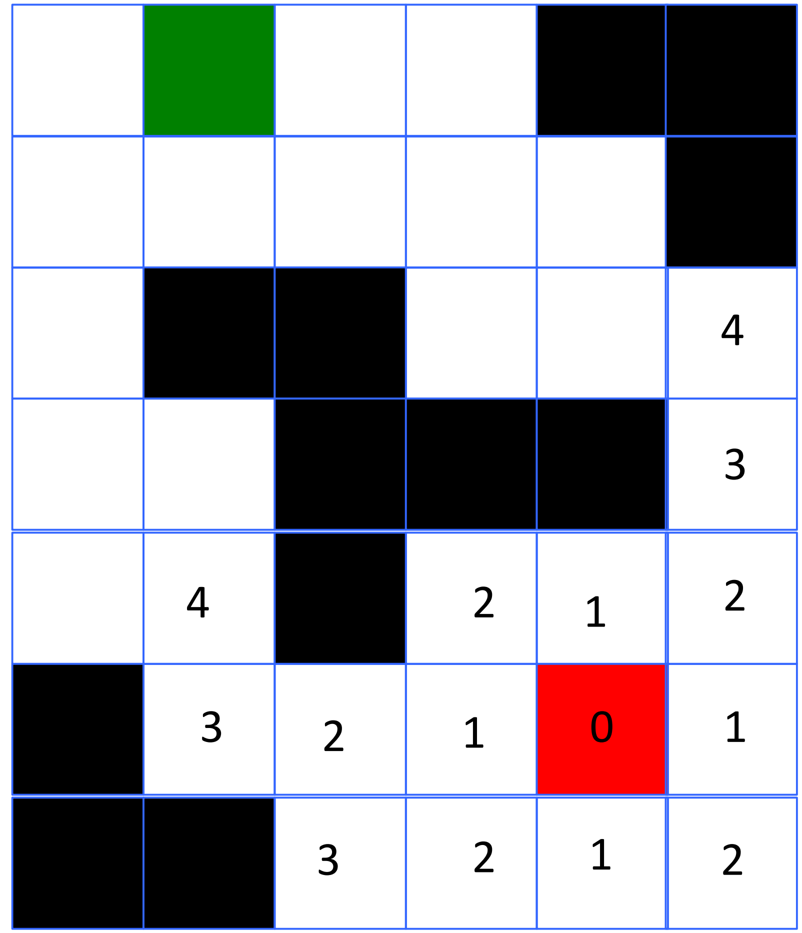


Planning Procedure – Grassfire Algorithm



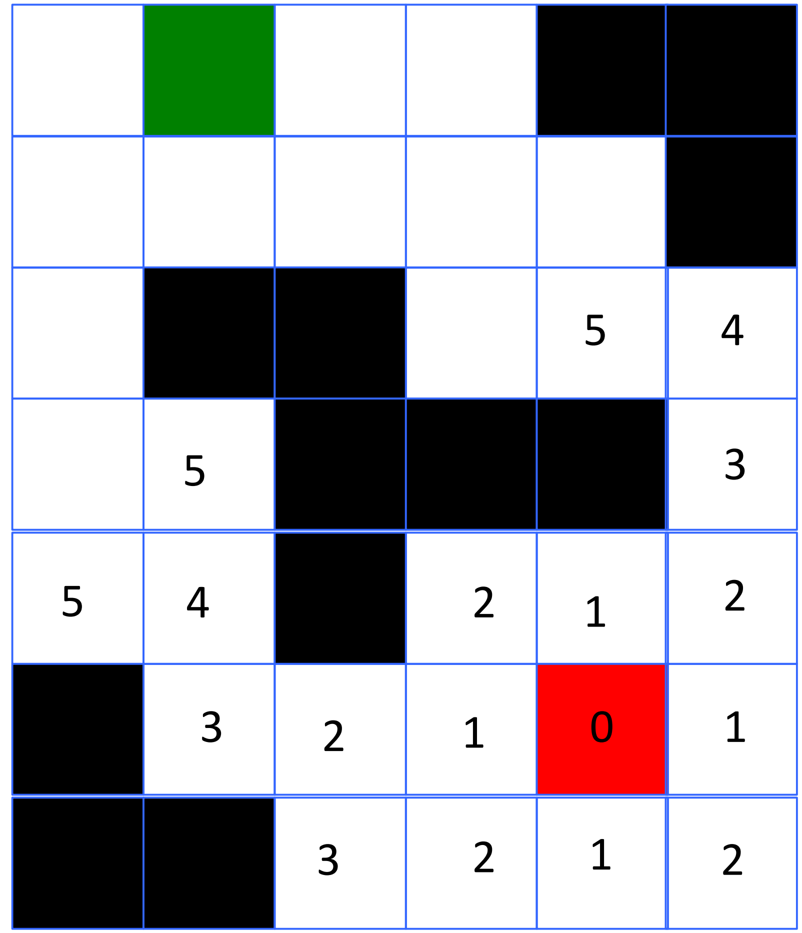
Planning Procedure – Grassfire Algorithm

START



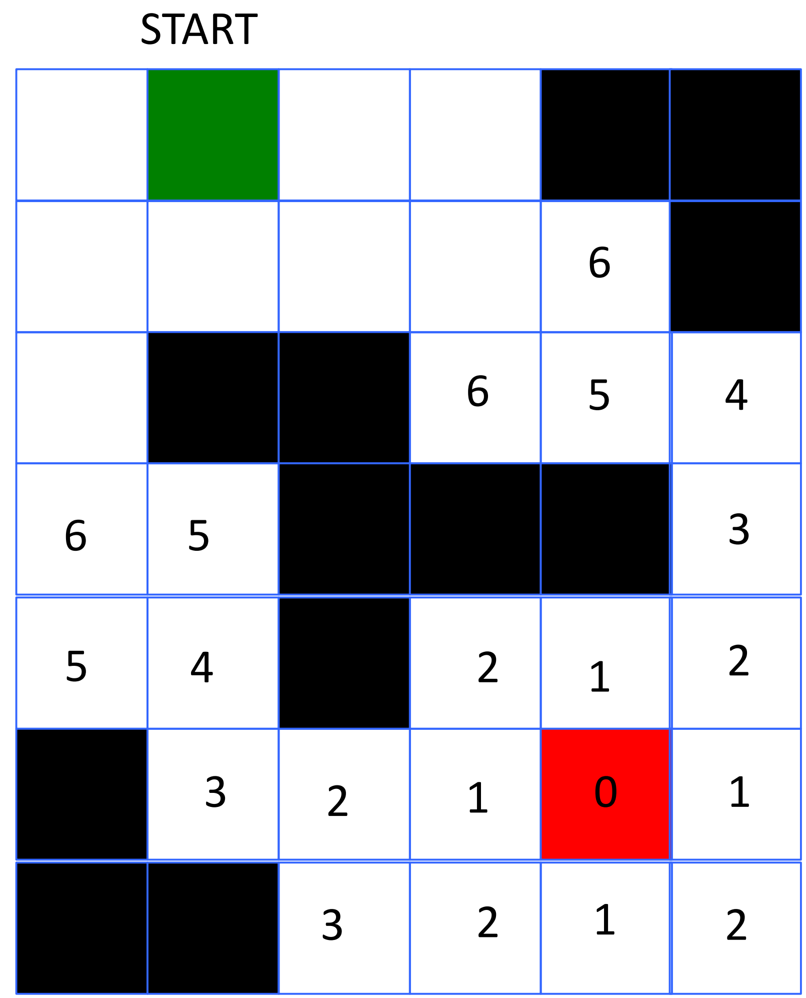
Planning Procedure – Grassfire Algorithm

START

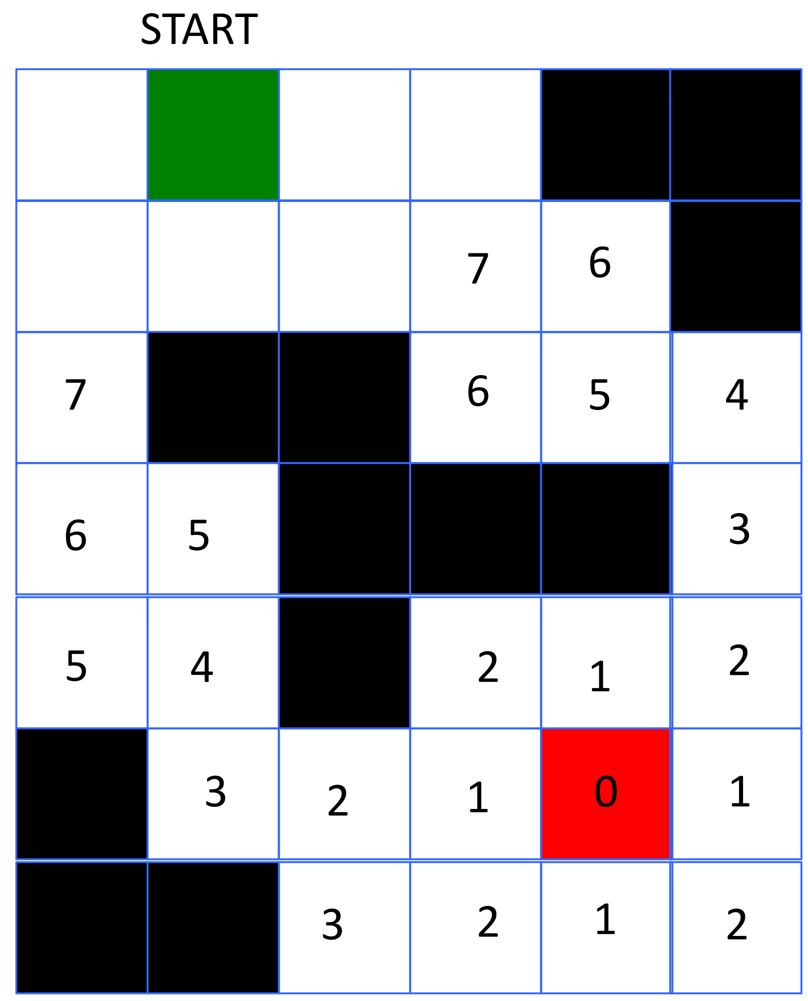


Planning Procedure – Grassfire Algorithm

- On every iteration the marking radiates outward from the destination like a fire spreading – hence the name

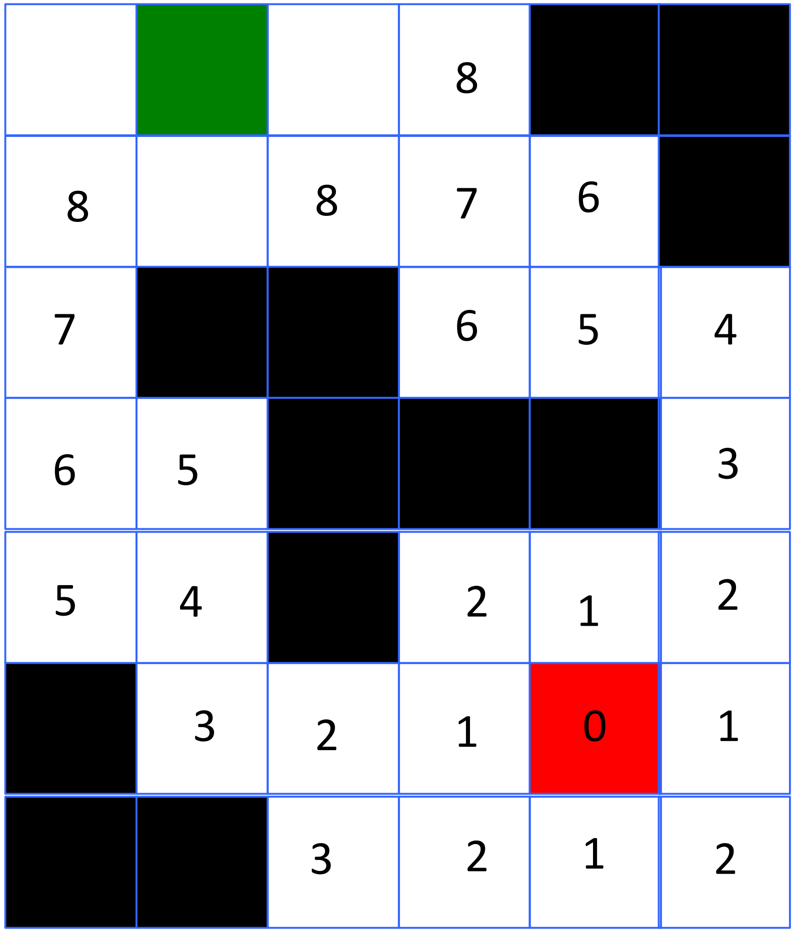


Planning Procedure – Grassfire Algorithm



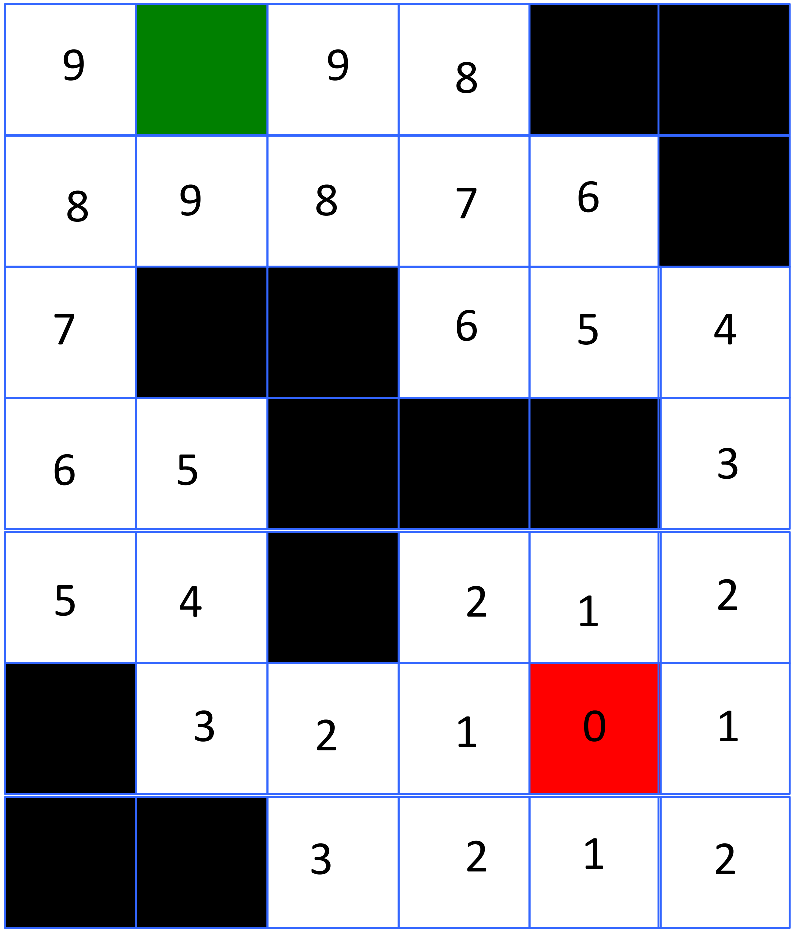
Planning Procedure – Grassfire Algorithm

START



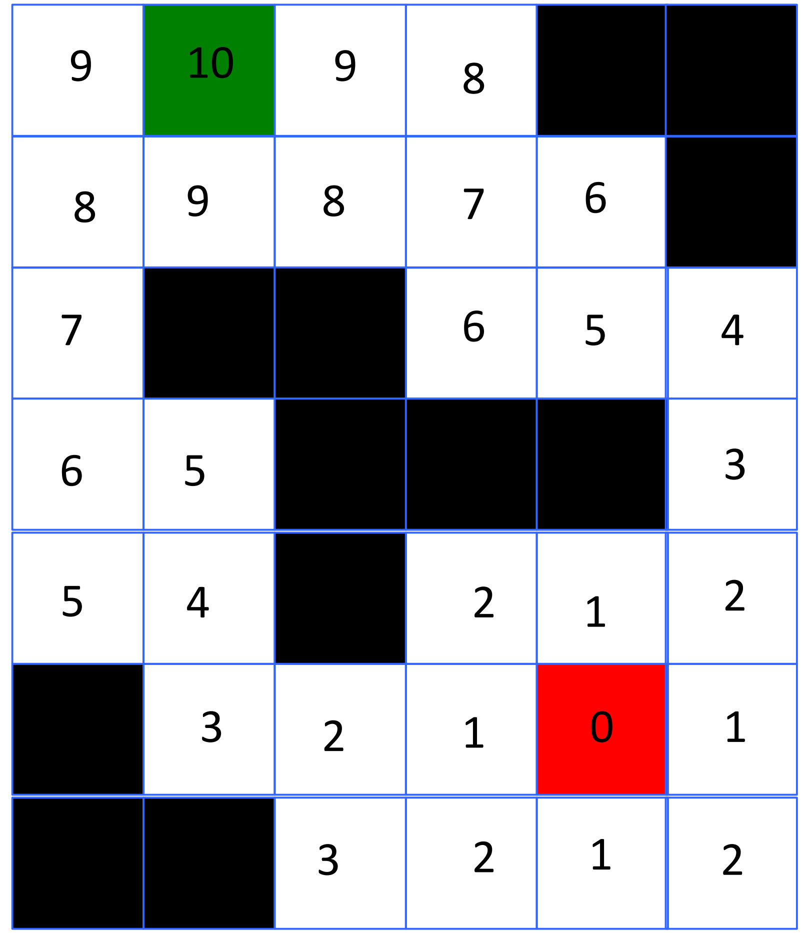
Planning Procedure – Grassfire Algorithm

START



Planning Procedure – Grassfire Algorithm

START



Planning Procedure – Grassfire Algorithm

- The distance values produced by the grassfire algorithm indicate the smallest number of steps needed to move from each node to the goal

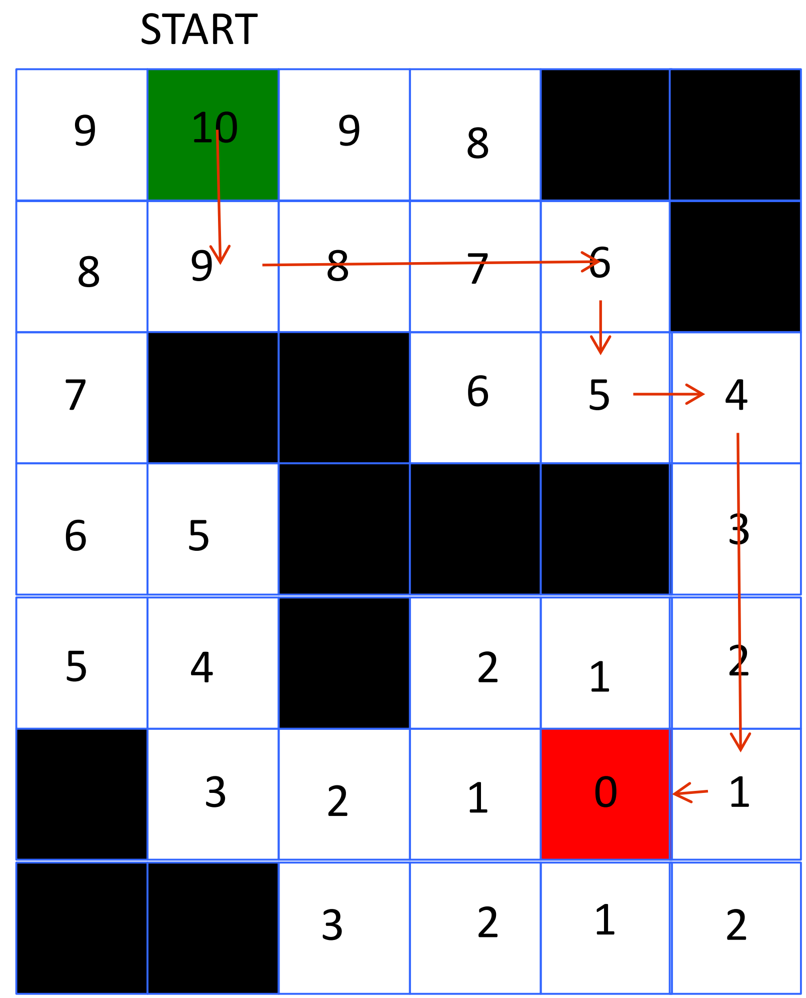
START

9	10	9	8		
8	9	8	7	6	
7			6	5	4
6	5				3
5	4		2	1	2
	3	2	1	0	1
		3	2	1	2

Grassfire algorithm – pseudo code

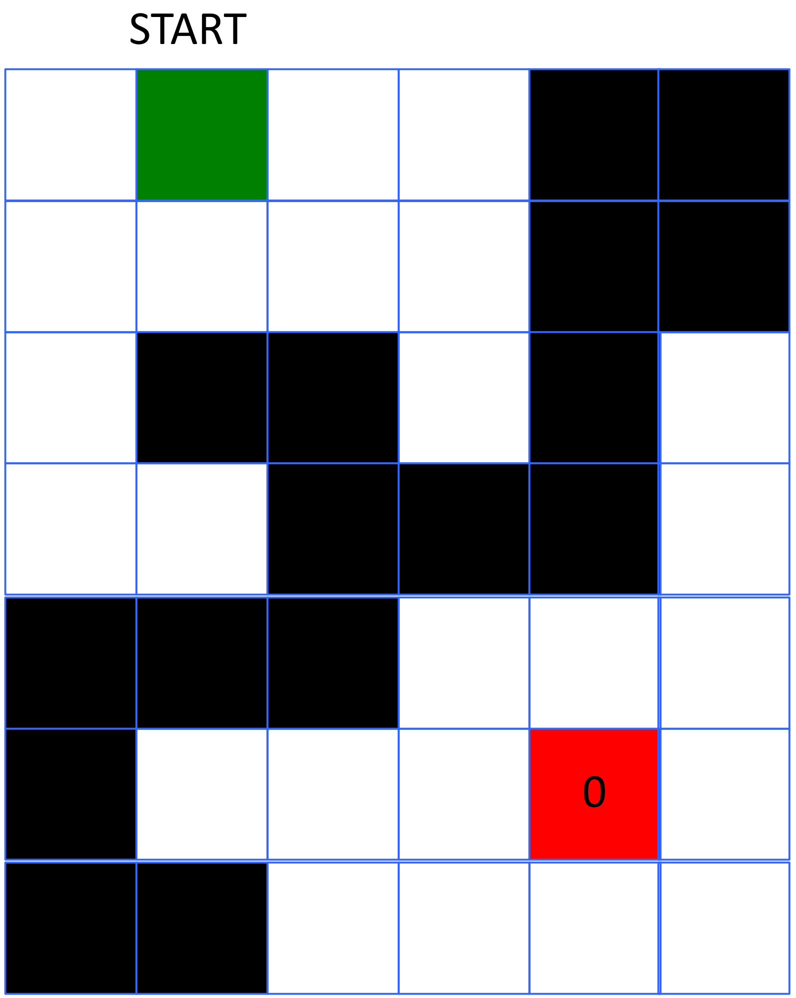
- For each node n in the graph
  - n.distance = Infinity
- Create an empty list.
- goal.distance = 0, add goal to list.
- While list not empty
  - Let current = first node in list, remove current from list
  - For each node, n that is adjacent to current
    - If n.distance = Infinity
    - n.distance = current.distance + 1
    - add n to the back of the list

Tracing a path to the destination

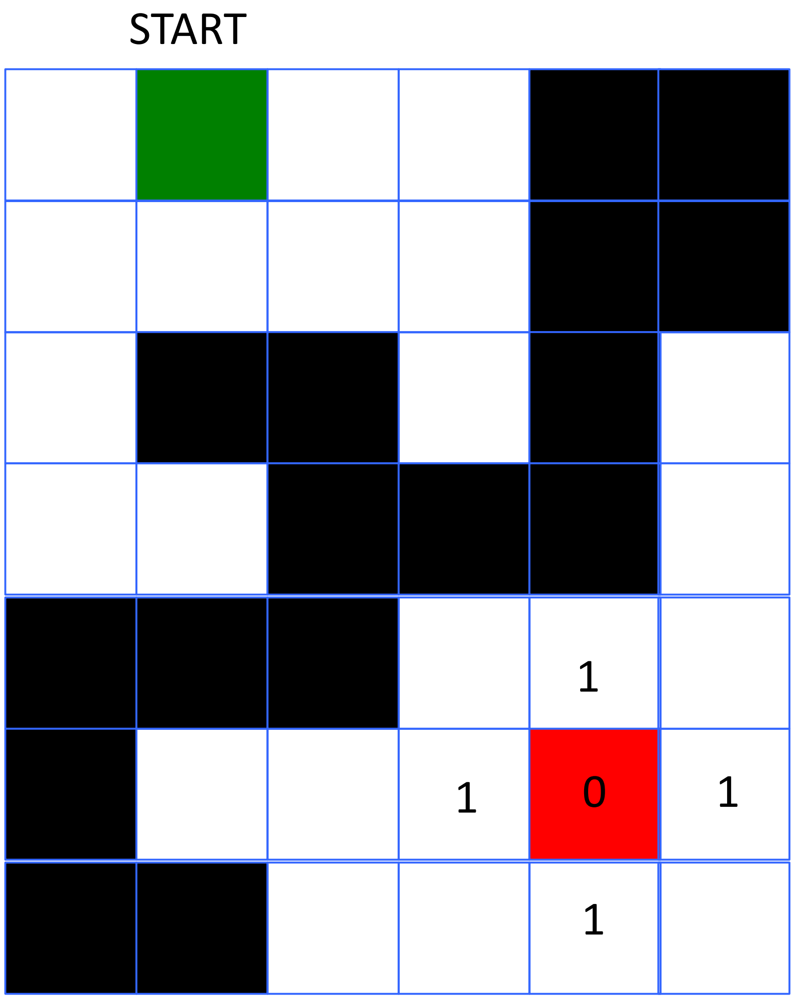


- To move towards the destination from any node simply move towards the neighbor with the smallest distance value, breaking ties arbitrarily.

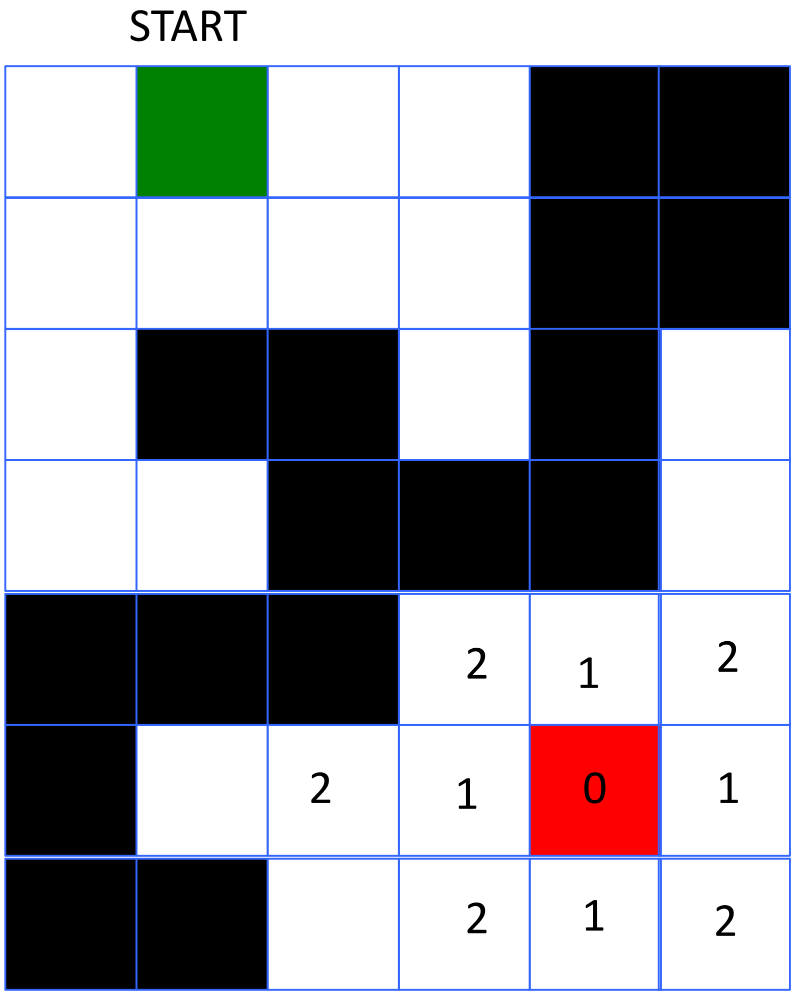
Another Example – Grassfire Algorithm



Another Example – Grassfire Algorithm

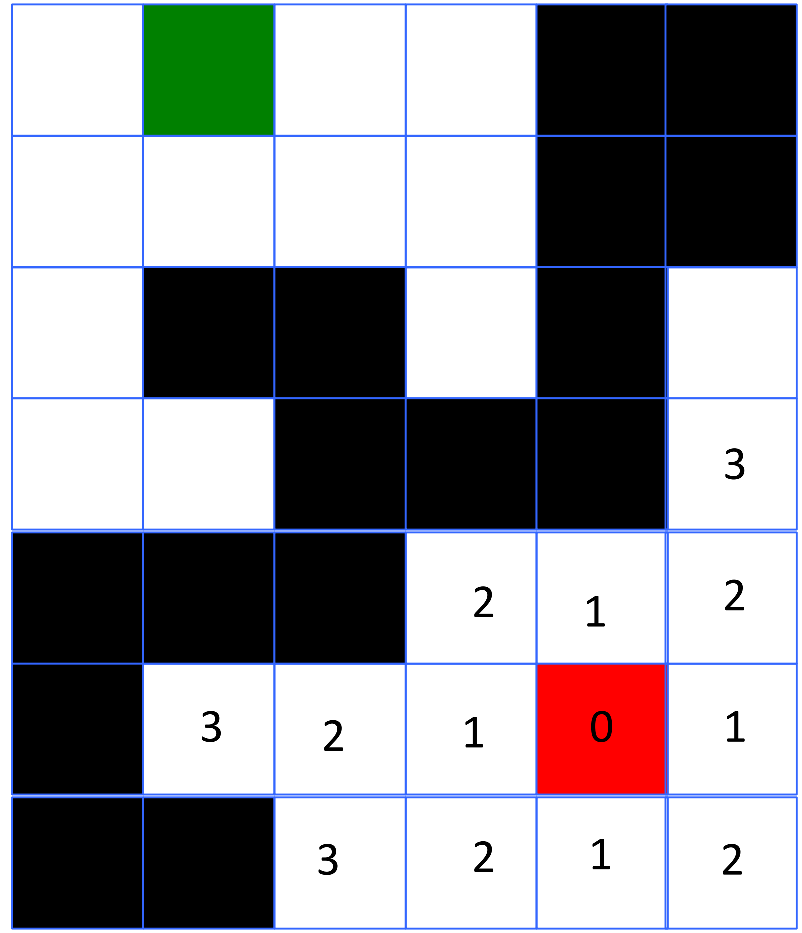


Planning Procedure – Grassfire Algorithm



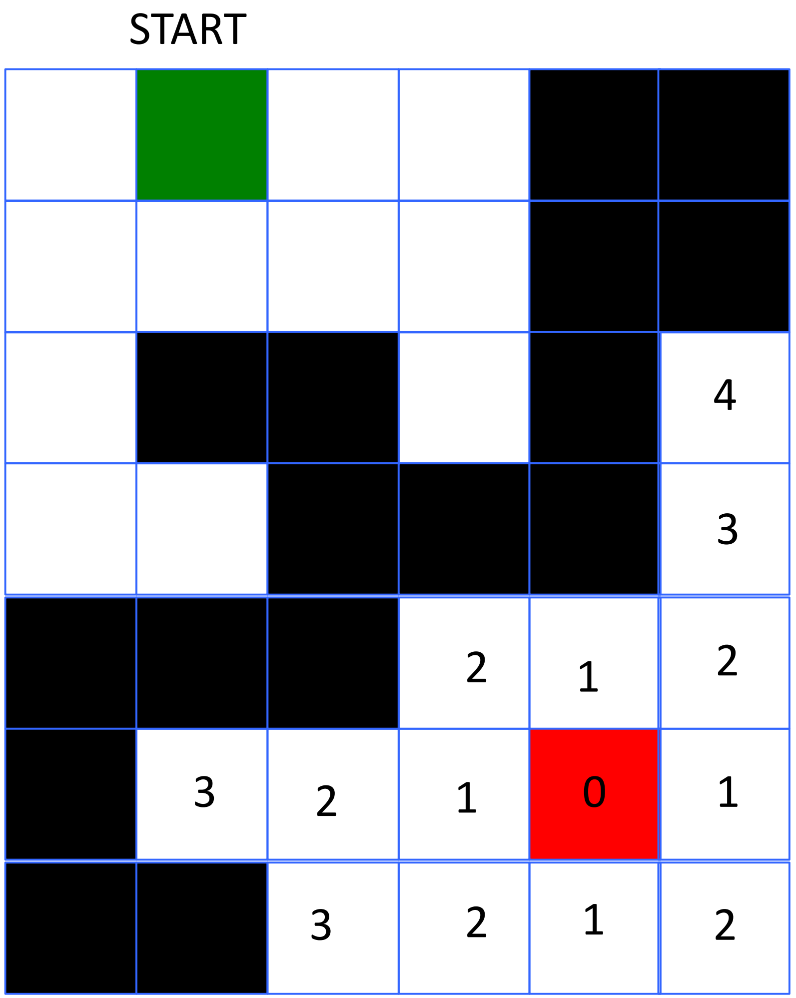
Planning Procedure – Grassfire Algorithm

START



Planning Procedure – Grassfire Algorithm

- In this case the procedure terminates before the start node is marked indicating that no path exists



Grassfire Algorithm

- It will find the shortest path between the start and the goal if one exists.
- If no path exists that fact will be discovered.

Computational Complexity - Grassfire

- The computational effort required to run the grassfire algorithm on a grid increases linearly with the number of edges.

- This can be expressed more formally as follows.

$$\mathcal{O}(|\mathbf{V}|) \tag{1}$$

Where $|\mathbf{V}|$ denotes the number of nodes in the graph

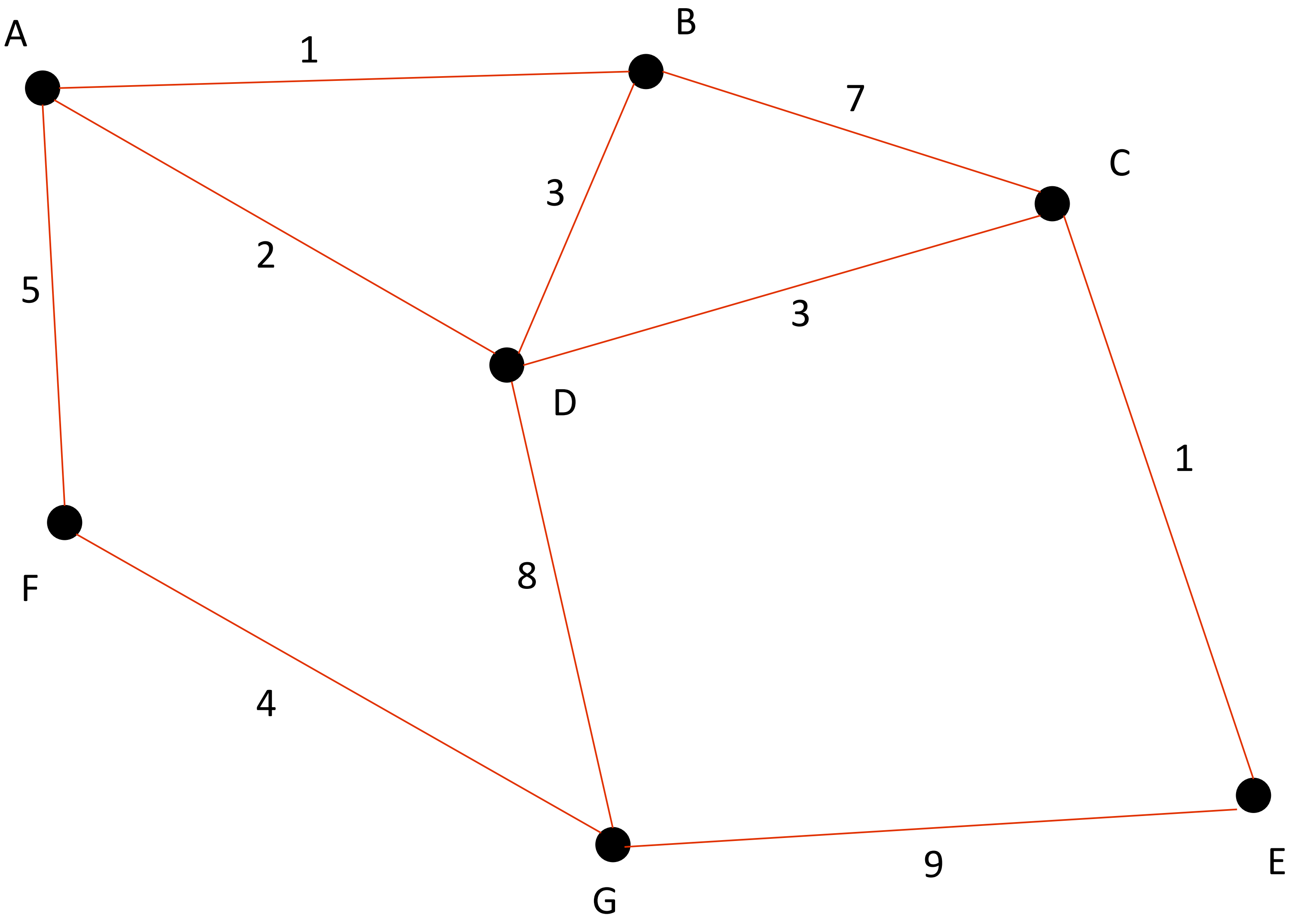
Computational Complexity - Grassfire

- Number of nodes in a 2D grid 100x100 = 10^4
- Number of nodes in a 3D grid 100x100x100 = 10^6
- Number of nodes in a 6D grid 100 cells on side = 10^{12}

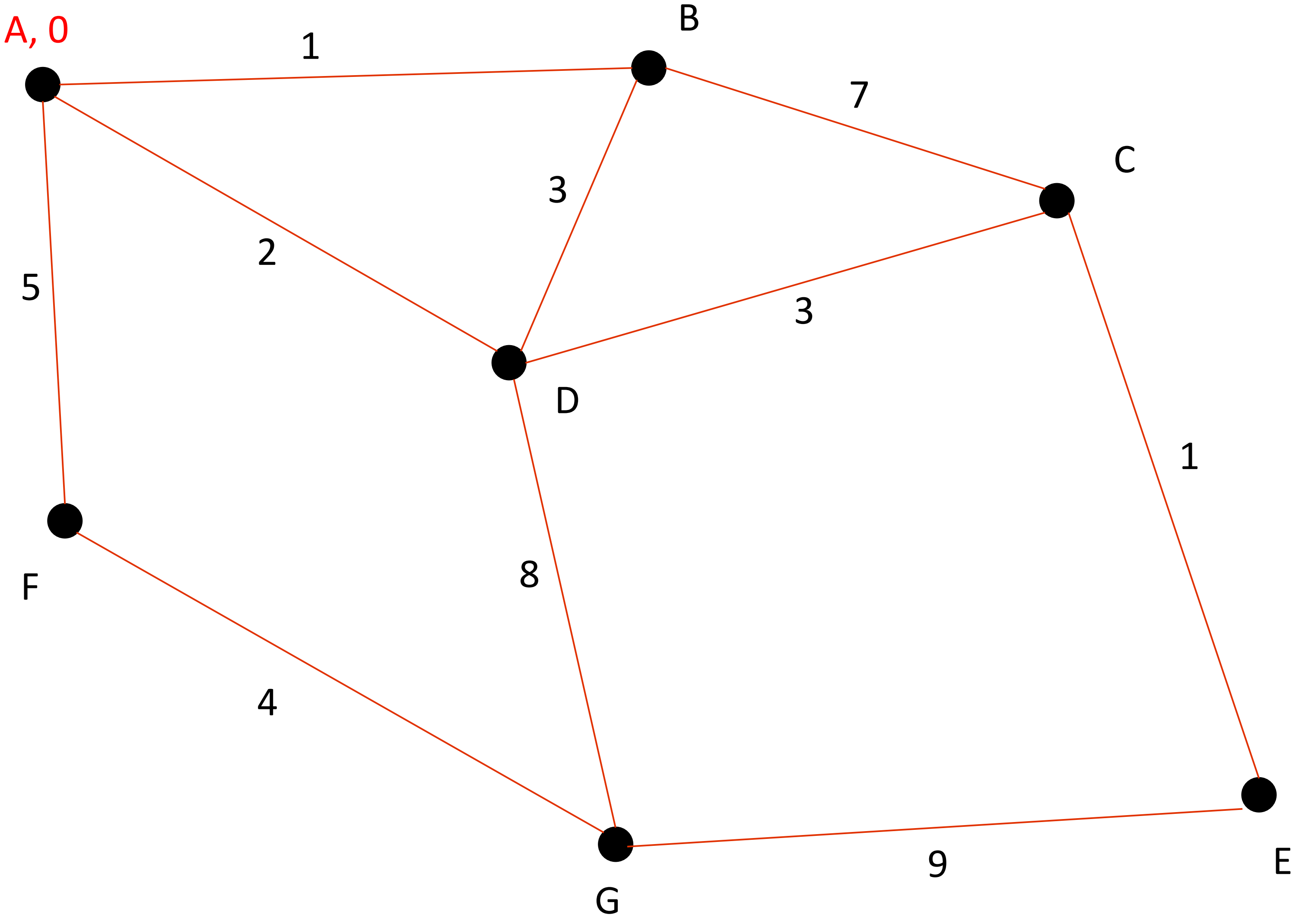
Video 9.3

CJ Taylor

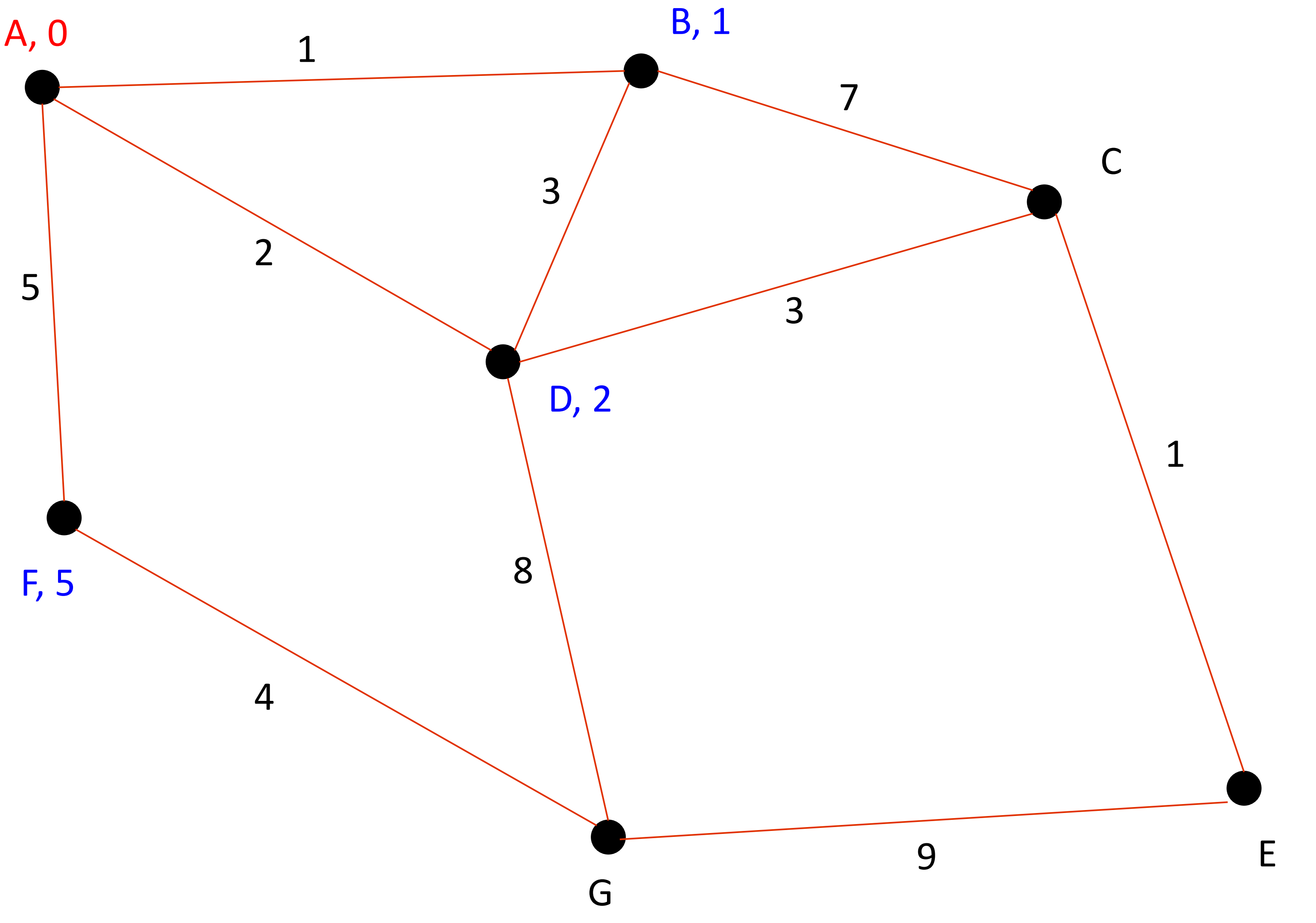
Planning shortest paths – Dijkstra's Algorithm



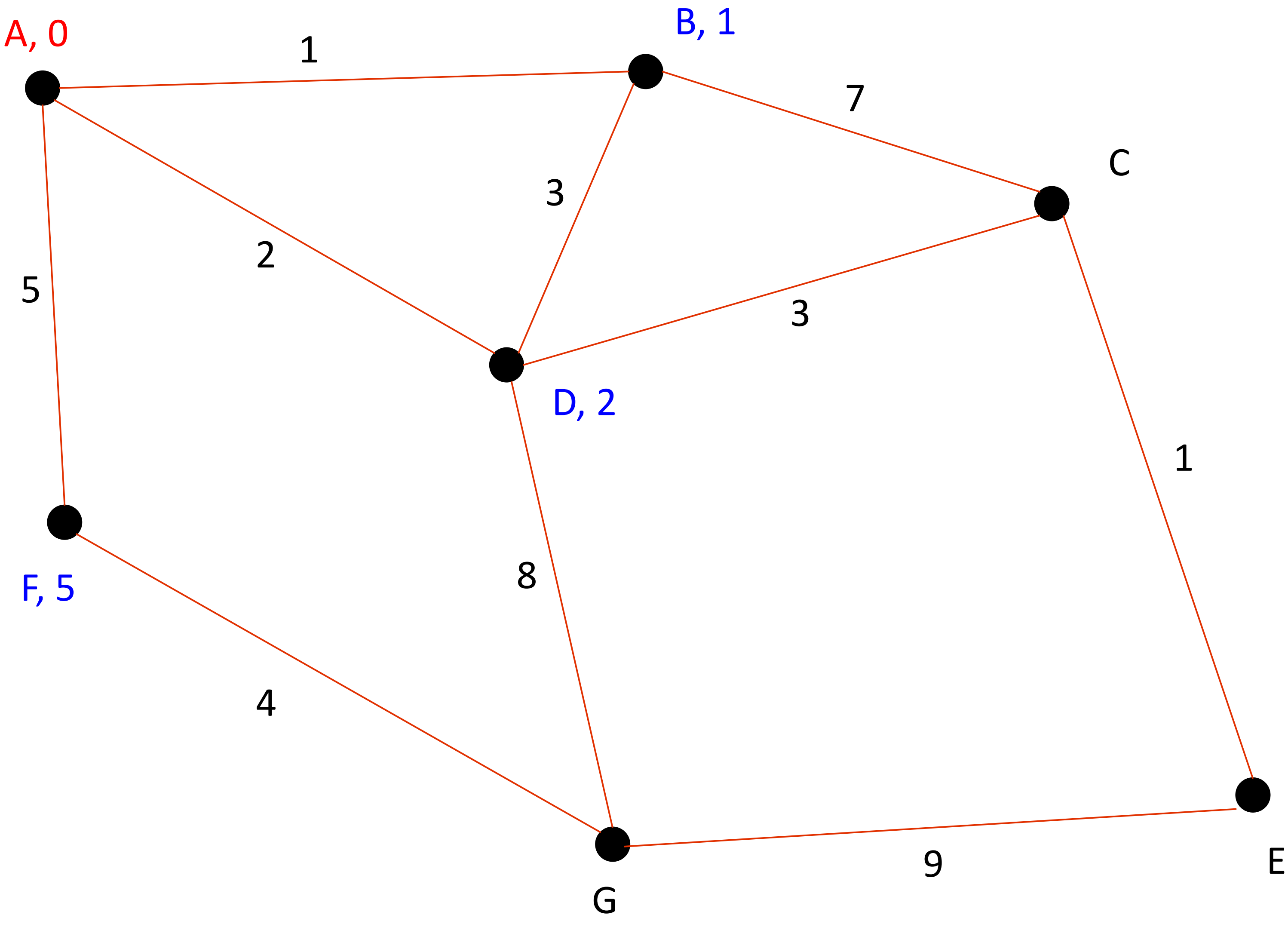
Planning shortest paths – Dijkstra's Algorithm



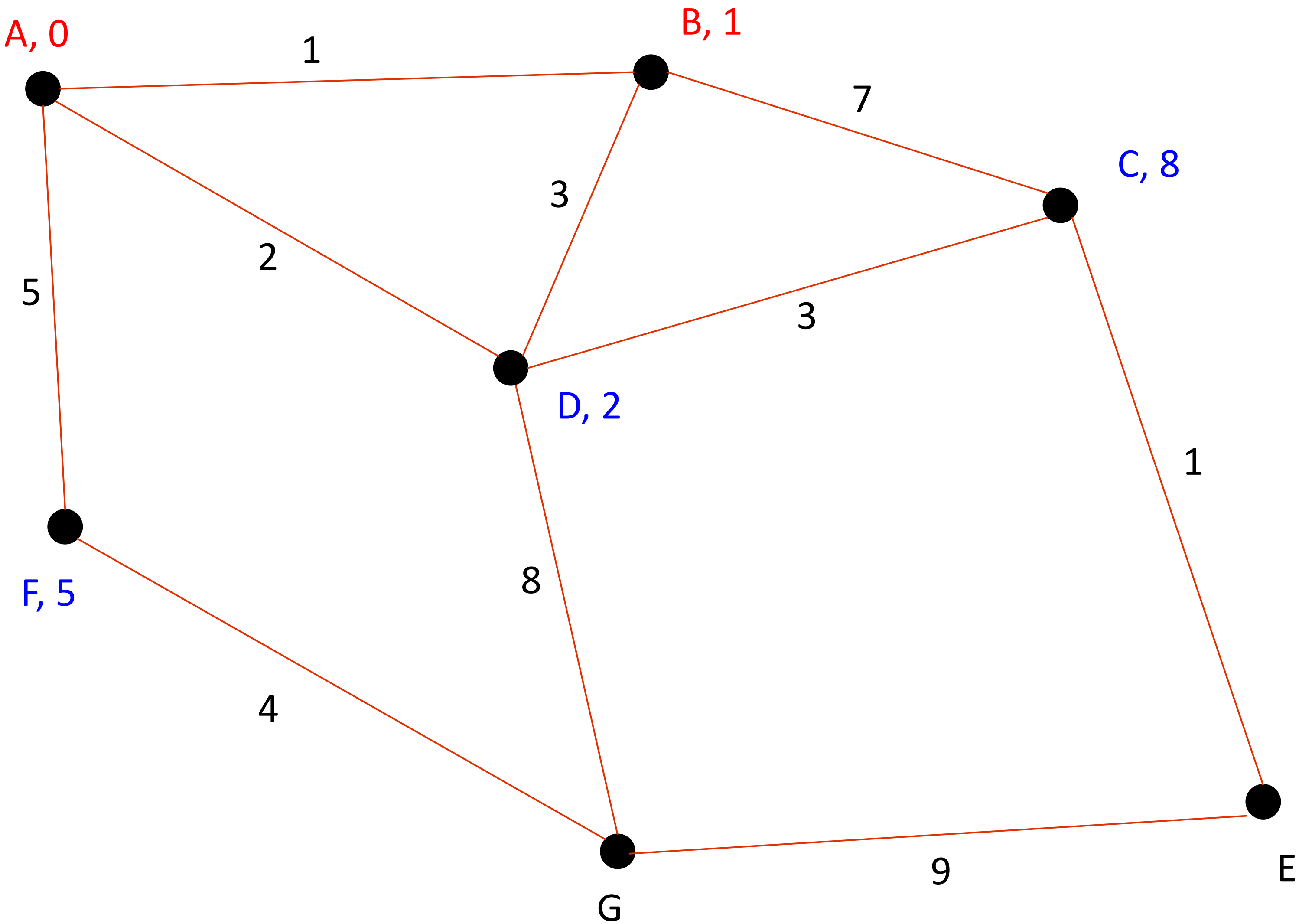
Planning shortest paths – Dijkstra's Algorithm



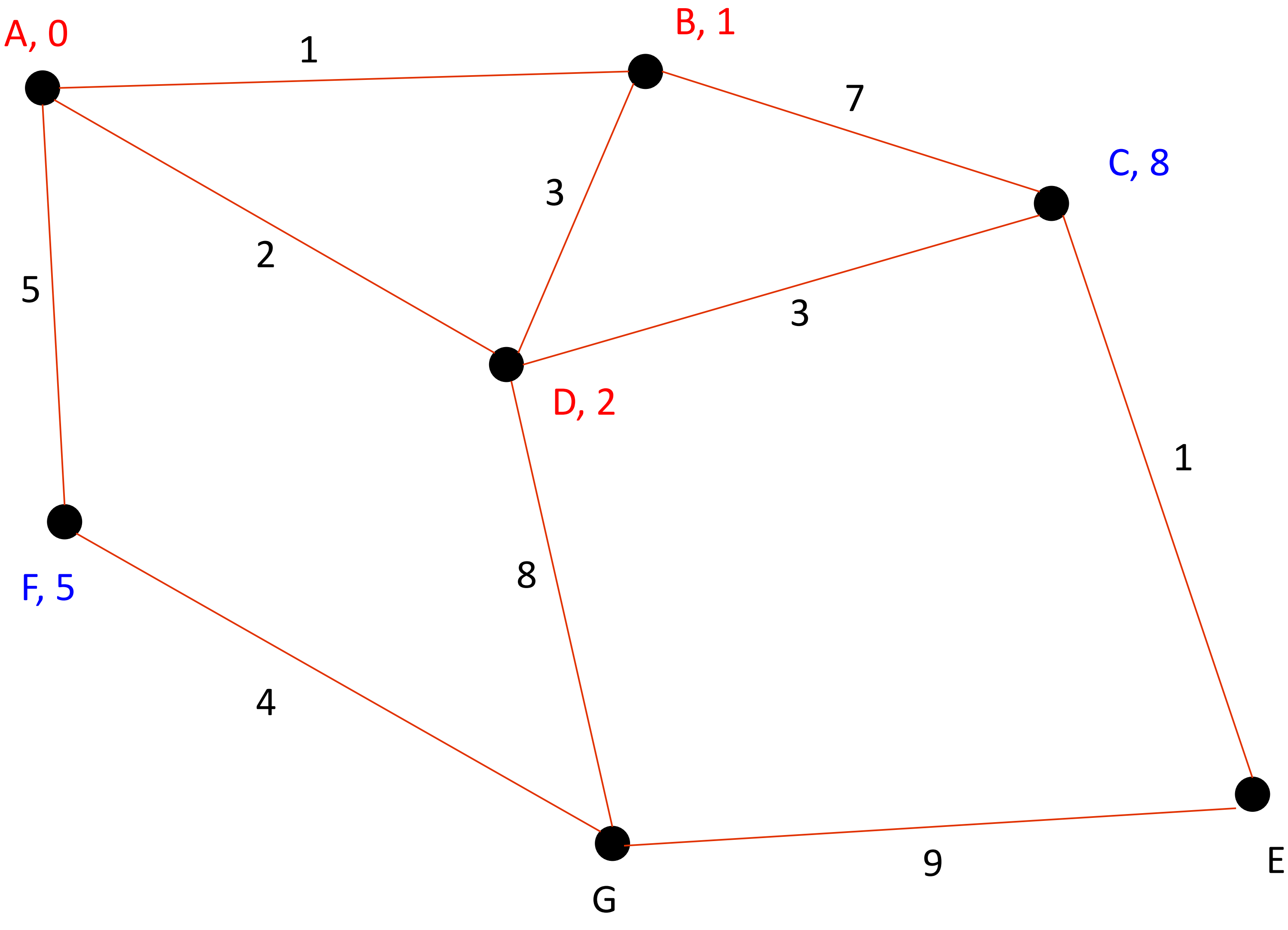
Planning shortest paths – Dijkstra's Algorithm



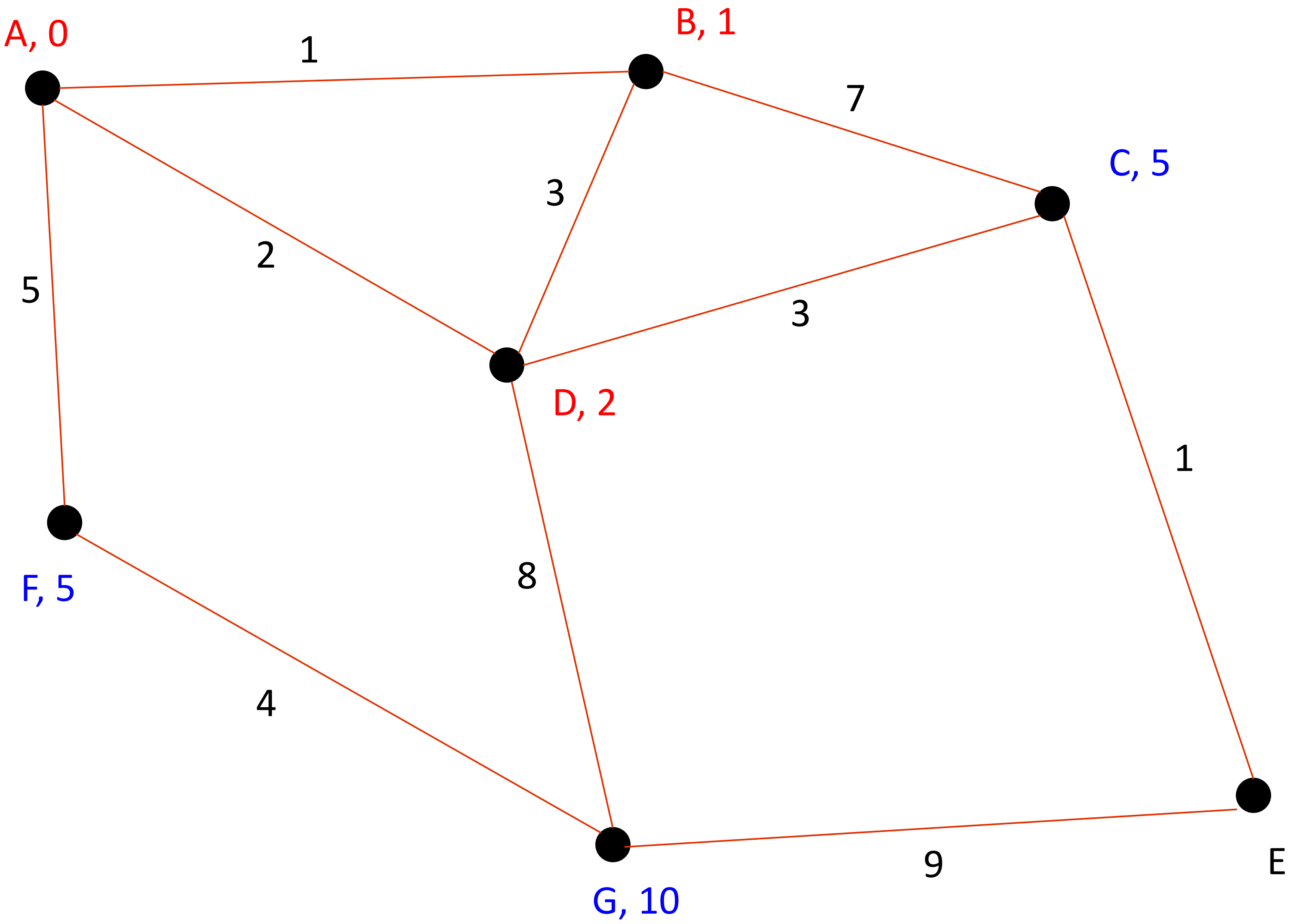
Planning shortest paths – Dijkstra's Algorithm



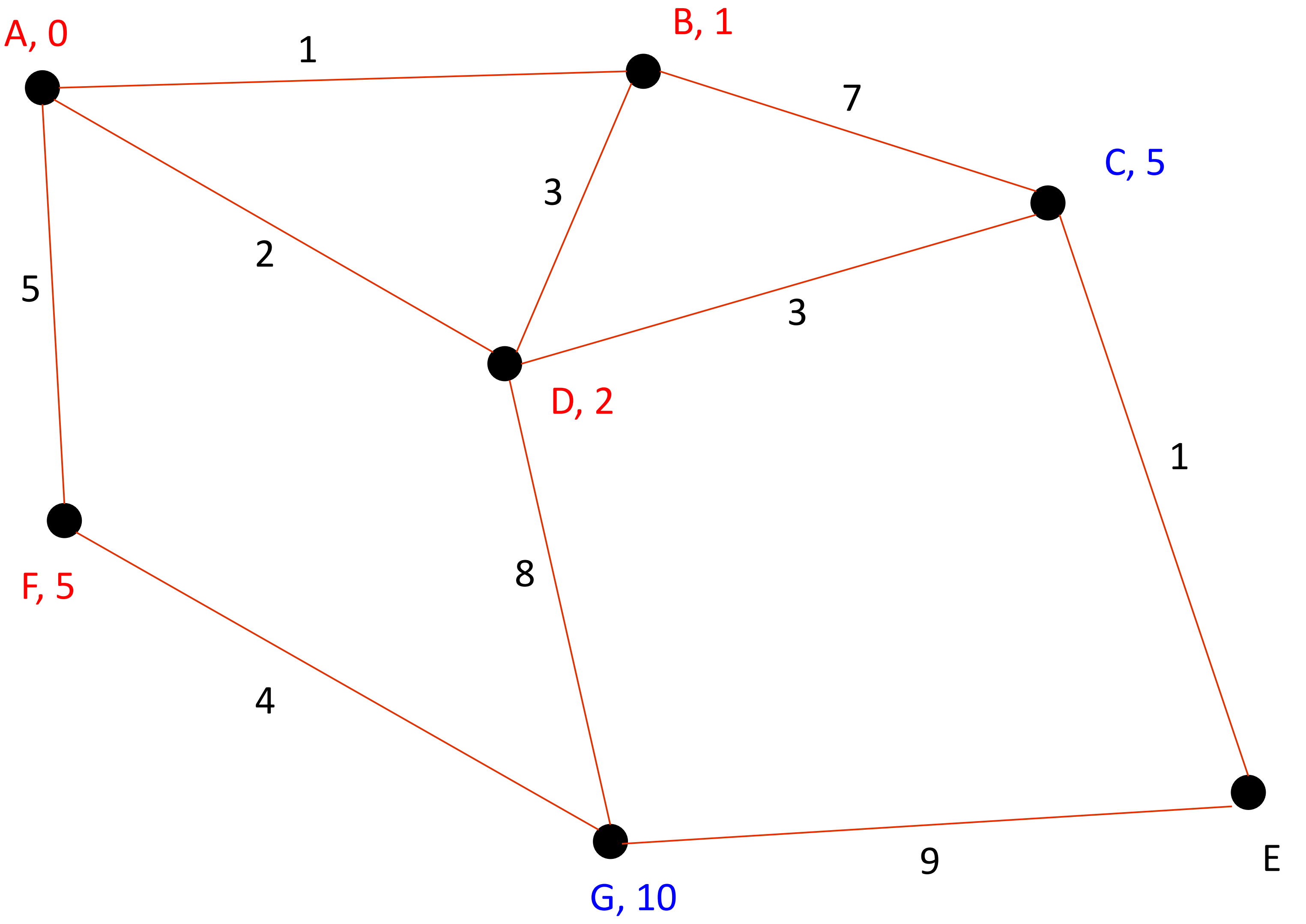
Planning shortest paths – Dijkstra's Algorithm



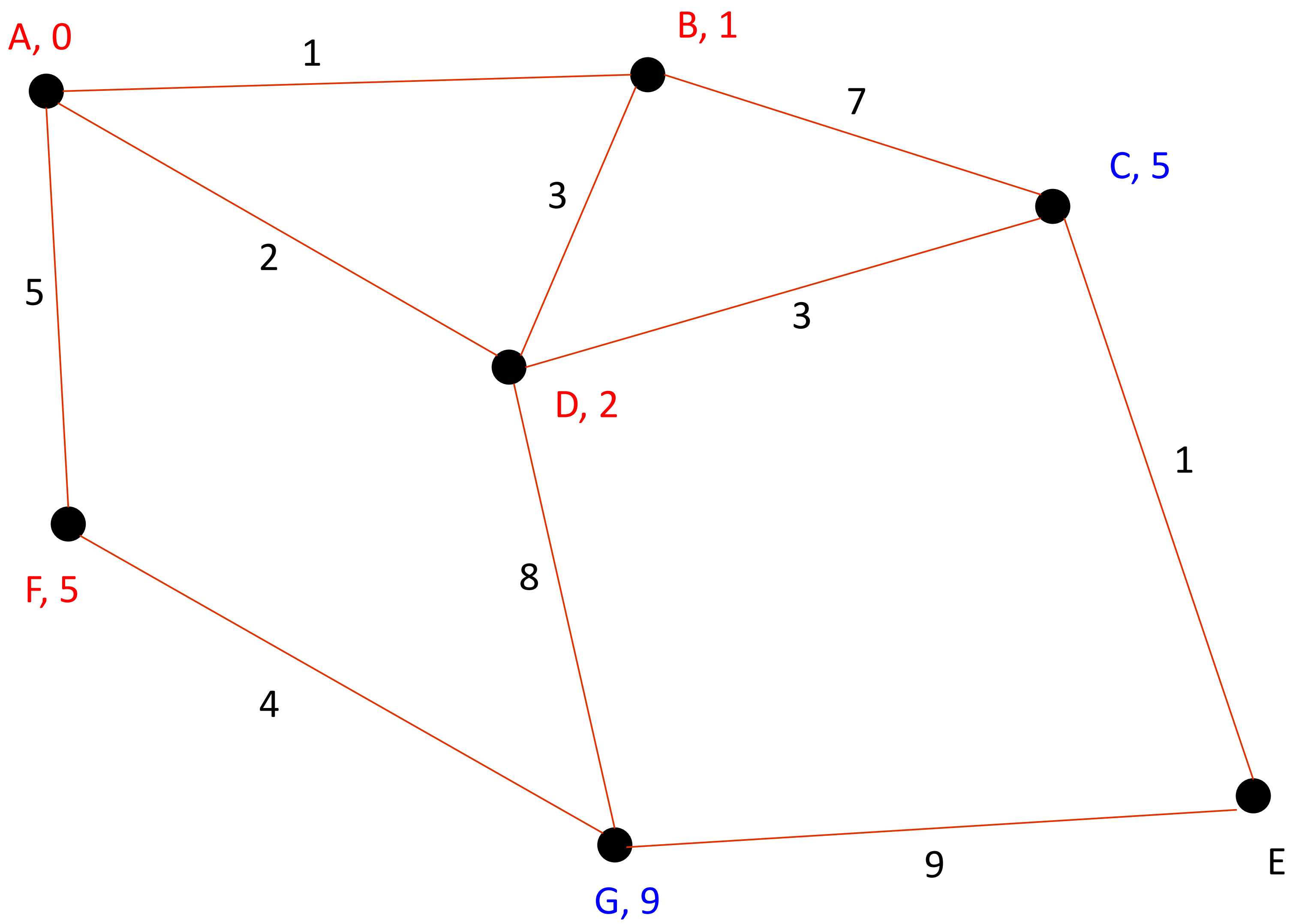
Planning shortest paths – Dijkstra's Algorithm



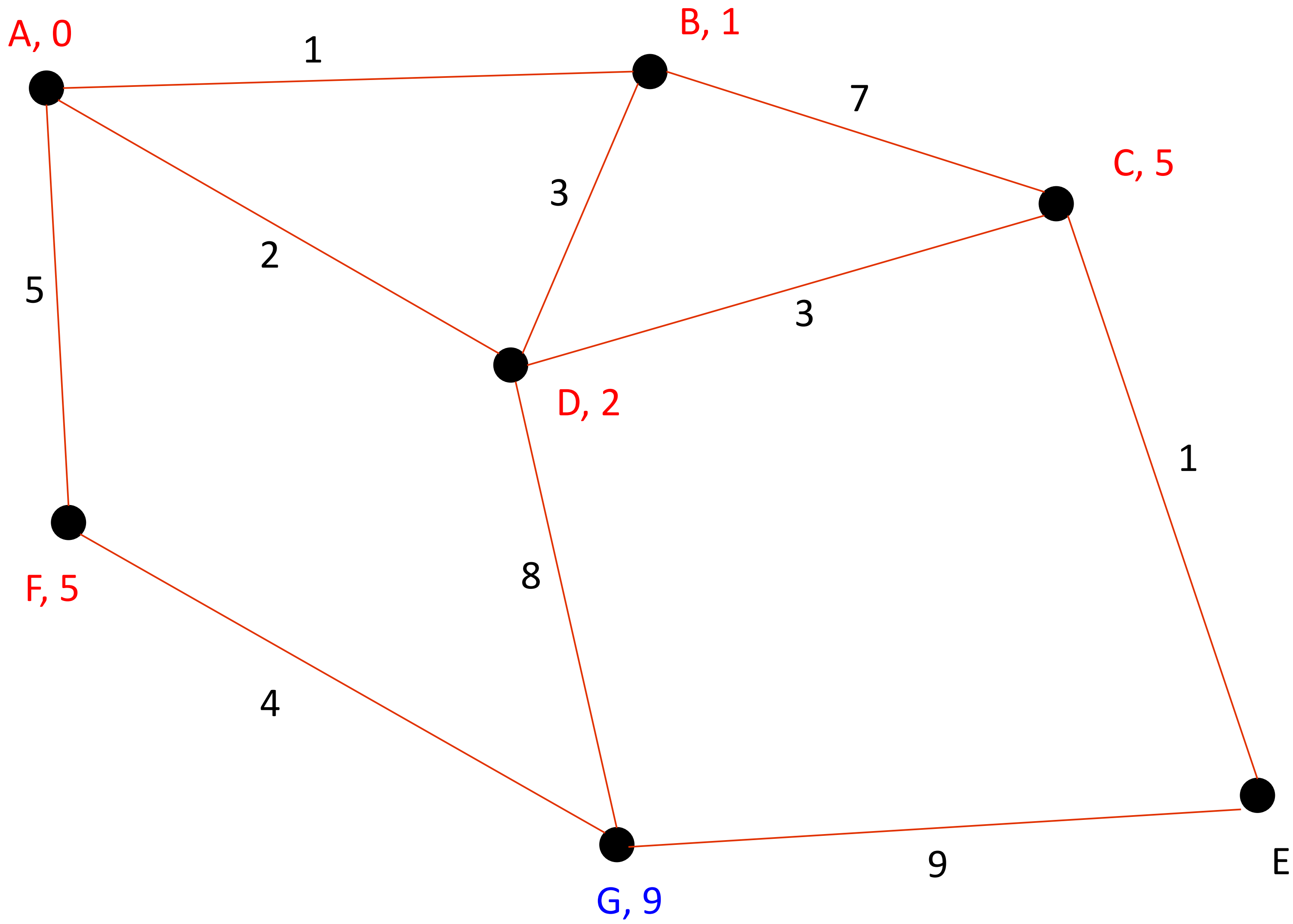
Planning shortest paths – Dijkstra's Algorithm



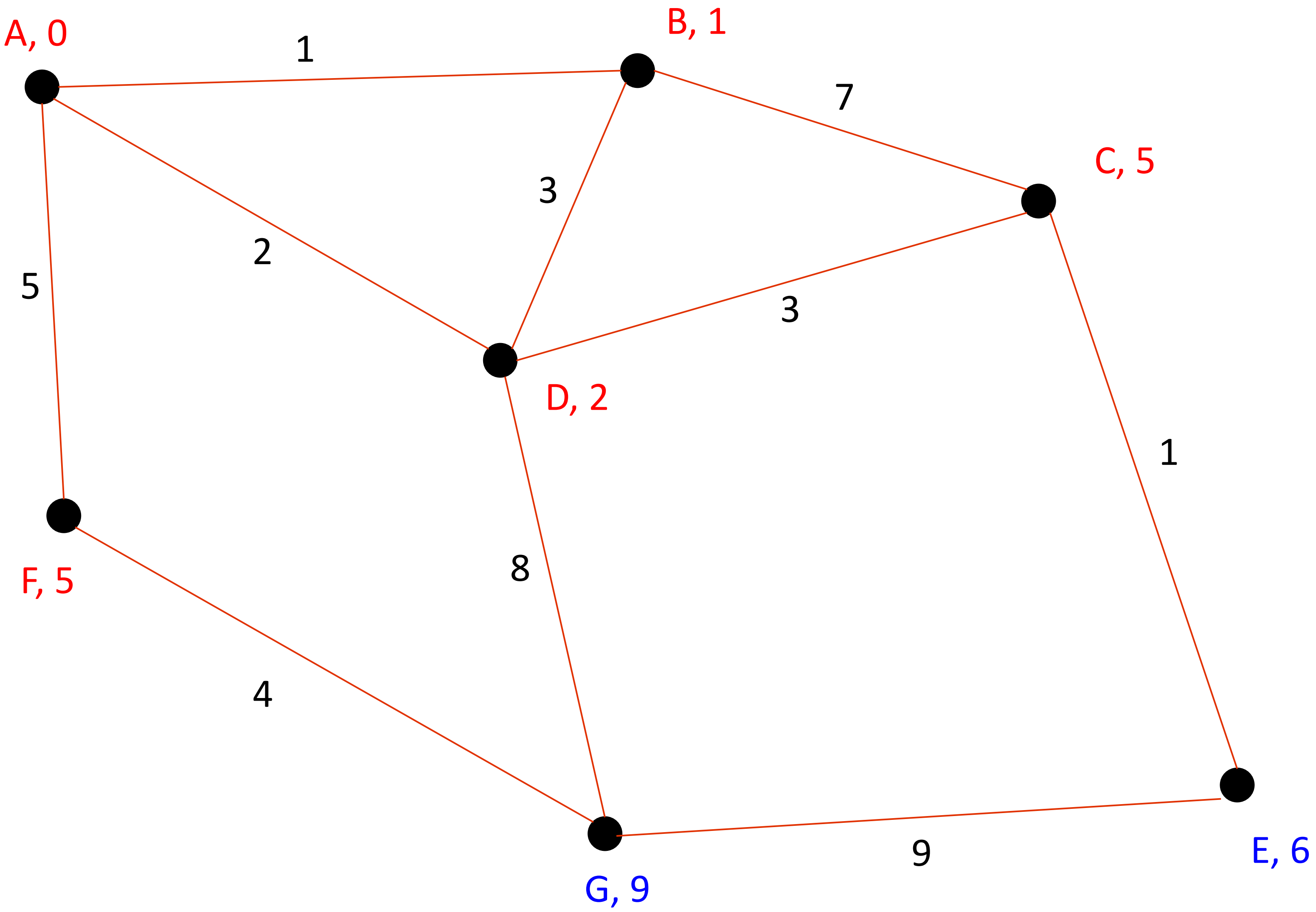
Planning shortest paths – Dijkstra's Algorithm



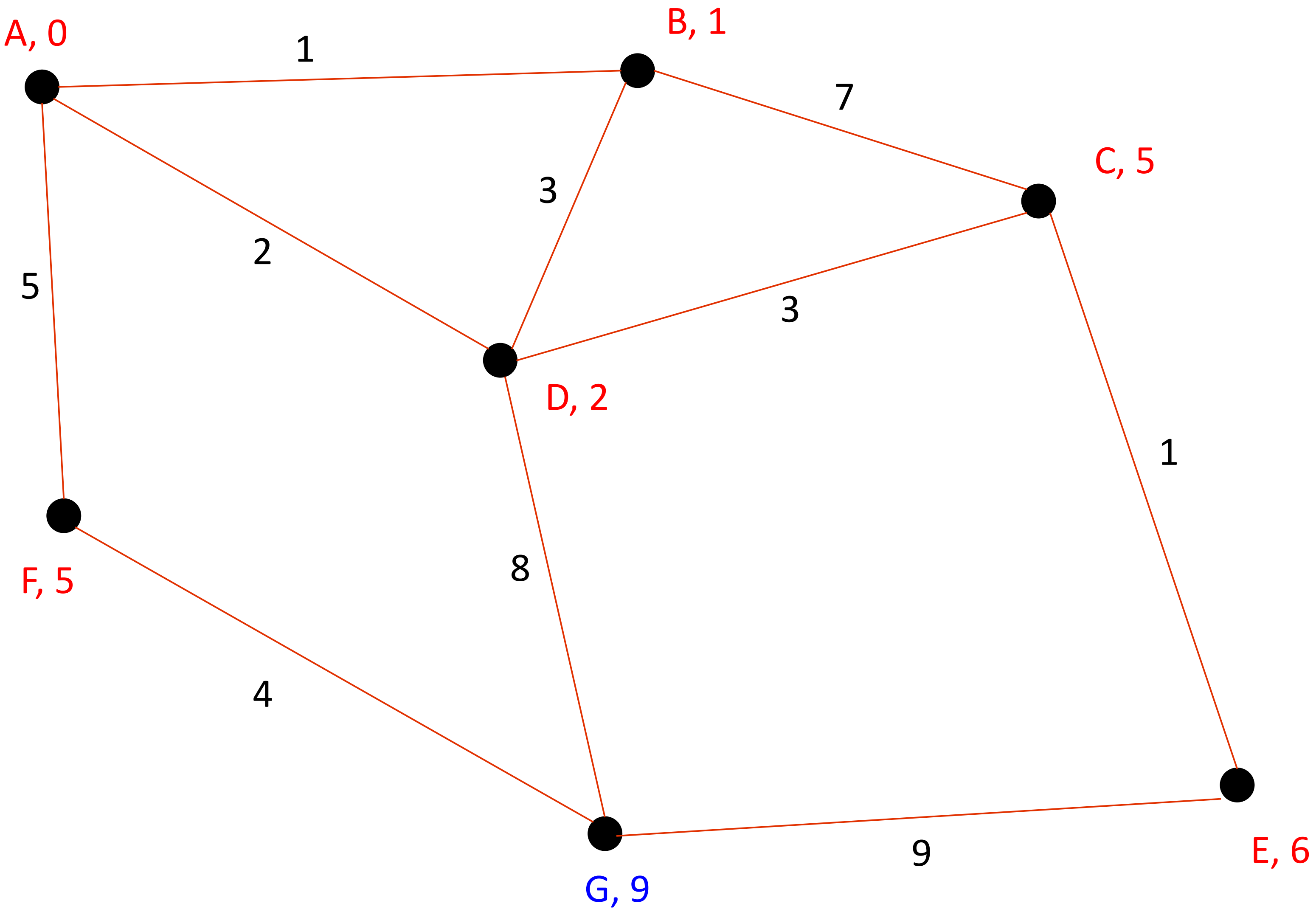
Planning shortest paths – Dijkstra's Algorithm



Planning shortest paths – Dijkstra's Algorithm



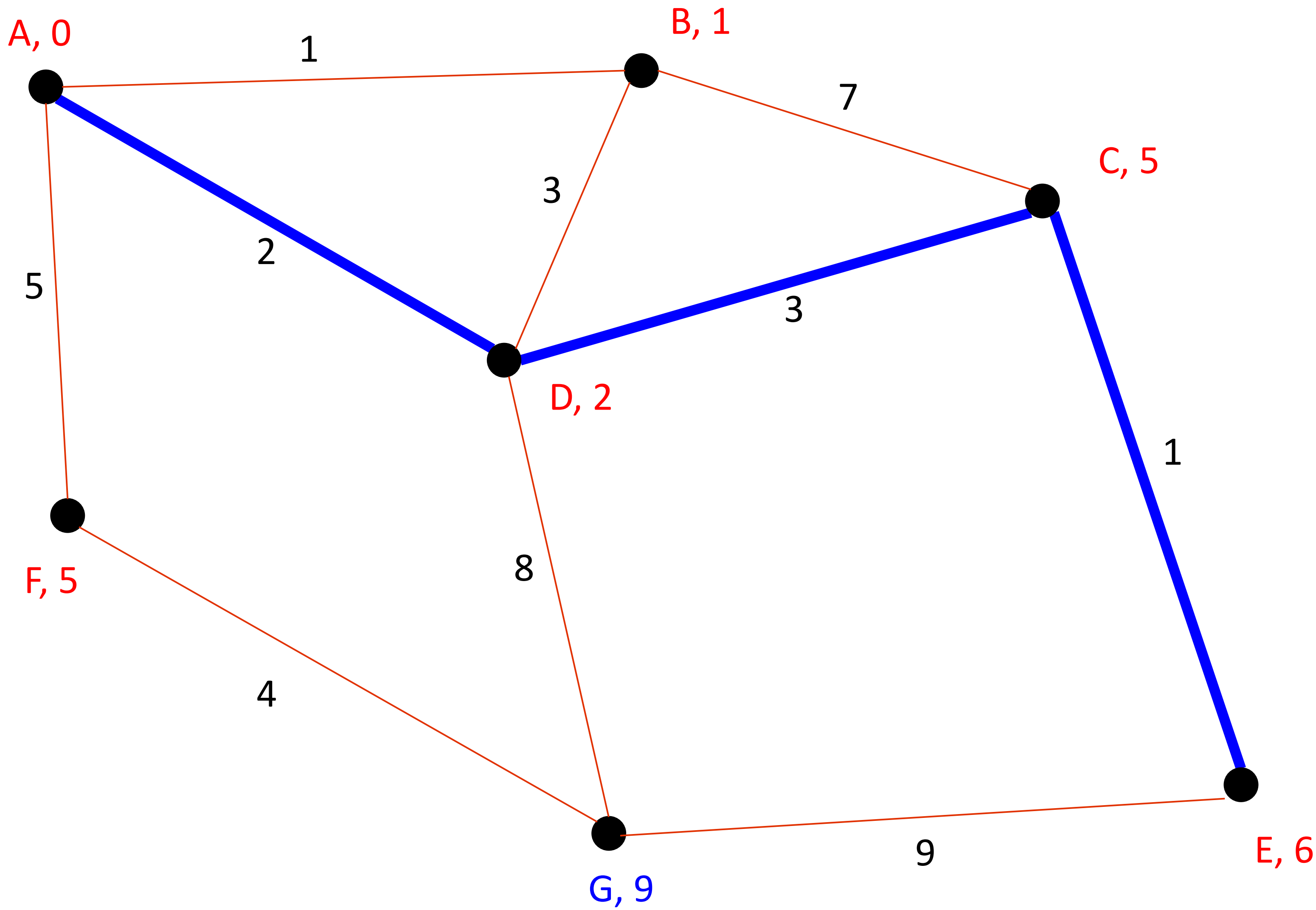
Planning shortest paths – Dijkstra's Algorithm



Dijkstra's algorithm – pseudo code

- For each node n in the graph
  - n.distance = Infinity
- Create an empty list.
- start.distance = 0, add start to list.
- While list not empty
  - Let current = node in the list with the smallest distance, remove current from list
  - For each node, n that is adjacent to current
    - If n.distance > current.distance + length of edge from n to current
    - n.distance = current.distance + length of edge from n to current
    - n.parent = current
    - add n to list if it isn't there already

Planning shortest paths – Dijkstra's Algorithm



Computational Complexity of Dijkstra's algorithm

- A naive version of Dijkstra's algorithm can be implemented with a computational complexity that grows quadratically with the number of nodes.

$$\mathcal{O}(|\mathbf{V}|^{\mathbf{2}}) \tag{1}$$

- By keeping the list of nodes sorted using a clever data structure known as a priority queue the computational complexity can be reduced to something that grows more slowly

$$\mathcal{O}((|\mathbf{E}| + |\mathbf{V}|)\log(|\mathbf{V}|)) \tag{2}$$

- $|\mathbf{V}|$ denotes the number of nodes in the graph and $|\mathbf{E}|$ denotes the number of edges

Video 9.4

CJ Taylor

A* Procedure

- Improving on Dijkstra/Grassfire using heuristic search
- Example of Best First search strategy

Dijkstra/Grassfire Algorithm

- When applied on a grid graph where all of the edges have the same length, Dijkstra's algorithm and the grassfire procedure have similar behaviors.
- They both explore nodes in order based on their distance from the starting node until they encounter the goal.

A* Search

- A* Search attempts to improve upon the performance of grassfire and Dijkstra by incorporating a heuristic function that guides the path planner.

Heuristic Functions

- Heuristic functions are used to map every node in the graph to a non-negative value
- Heuristic Function Criteria:
  - H(goal) = 0
  - For any 2 adjacent nodes x and y
    - H(x) <= H(y) + d(x,y)
    - d(x,y) = weight/length of edge from x to y
- These properties ensure that for all nodes, n
  - H(n) <= length of shortest path from n to goal.

Example Heuristic Functions

- For path planning on a grid the following 2 heuristic functions are often used
  - Euclidean Distance

$$H(x_n, y_n) = \sqrt{((x_n - x_g)^2 + (y_n - y_g)^2)} \tag{1}$$

  - Manhattan Distance

$$H(x_n, y_n) = |(x_n - x_g)| + |(y_n - y_g)| \tag{2}$$

  - where (x_n, y_n) denotes the coordinates of the node n and (x_g, y_g) denotes the coordinate of the goal

A* algorithm – pseudo code

- For each node n in the graph
  - n.f = Infinity, n.g = Infinity
- Create an empty list.
- start.g = 0, start.f = H(start) add start to list.
- While list not empty
  - Let current = node in the list with the smallest f value, remove current from list
  - If (current == goal node) report success
  - For each node, n that is adjacent to current
    - If (n.g > (current.g + cost of edge from n to current))
    - n.g = current.g + cost of edge from n to current
    - n.f = n.g + H(n)
    - n.parent = current
    - add n to list if it isn't there already