Algorithms and their Applications
CS2004 (2020-2021)

Dr Mahir Arzoky

8.1 Graph Traversal – Dijkstra's Algorithm

NOTICES

COMPLETE THE YOUR VOICE MODULE SURVEY

brunel.ac.uk/yourvoice

Laboratory Worksheets and CodeRunner

- CodeRunner Tests
  - Class Test CR I: 124 attempts
  - Class Test CRII: 22 attempts
- Remember not to let these worksheets "pile up"
  - Do not leave any worksheet longer than 2 weeks
- No new laboratory worksheet this week – perfect opportunity to catch up!
- More Java Programming tutorial/help will be introduced from this week!

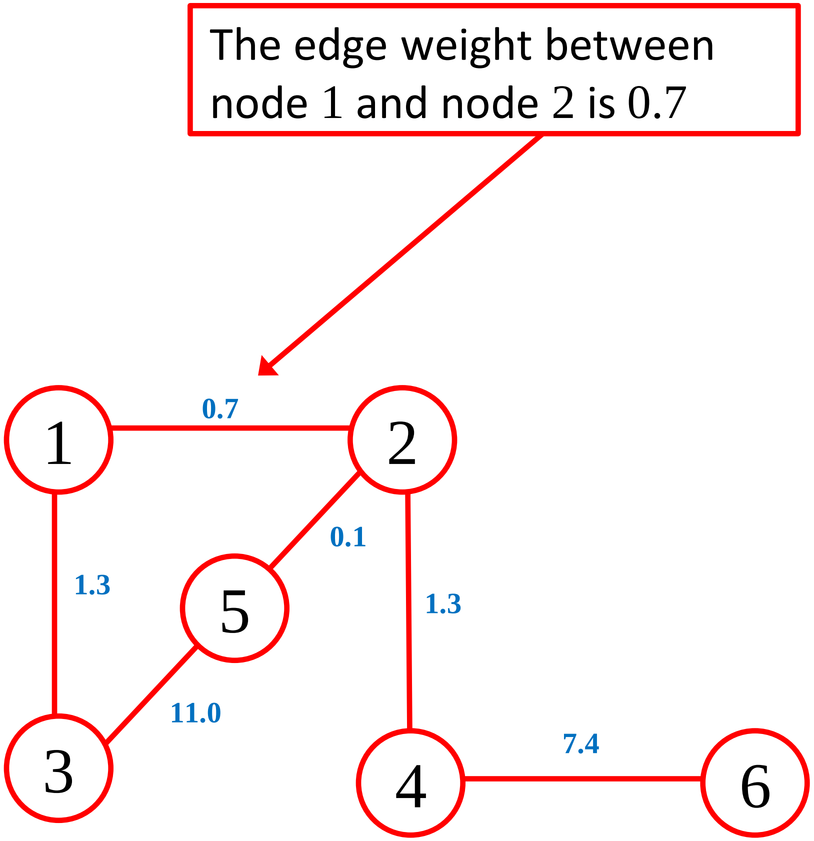
Previously On CS2004…

- So far we have looked at:
  - Concepts of Computation and Algorithms
  - Comparing algorithms
  - Some mathematical foundation
  - The Big-Oh notation
  - Computational Complexity
  - Data structures
  - Sorting algorithms
  - Last lecture we also looked at Graph Traversal algorithms

Graph Traversal – Dijkstra’s Algorithm

- Within this lecture we will discuss:
  - Shortest Path Problem
  - Dijkstra's Algorithm
  - Floyd–Warshall Algorithm

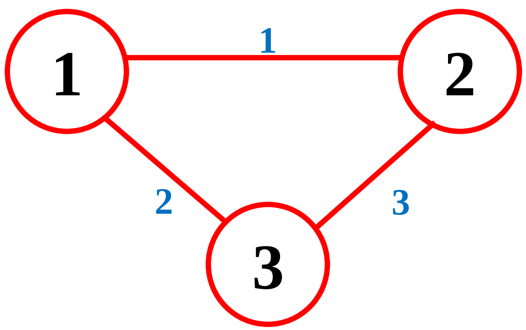
Weighted Graph - Recap



- A graph whose edges have some value associated to it (called weights)
- The weight usually represent a cost or a distance between two nodes
  - i.e. to go from node 1 to 2 there is some sort of cost/distance associated to it
- It can be represented as $G=(V,E,W)$
- $W(e)$ is called the weight of edge e

Adjacency Matrix - Recap

- We often represent a graph as a **matrix (2D array)** although other data structures can be used depending on the application
- If we have *N* nodes to represent
  - For an ***N* by *N* matrix *G* a non-zero value of g_{ij} (*i*th row *j*th column of *G*) means there is an edge between node *i* and *j***



$$G = \begin{array}{c c} \begin{array}{ccc} 1 & 2 & 3 \end{array} & \\ \begin{bmatrix} 0 & 1 & 2 \\ 1 & 0 & 3 \\ 2 & 3 & 0 \end{bmatrix} & \begin{array}{c} 1 \\ 2 \\ 3 \end{array} \end{array}$$

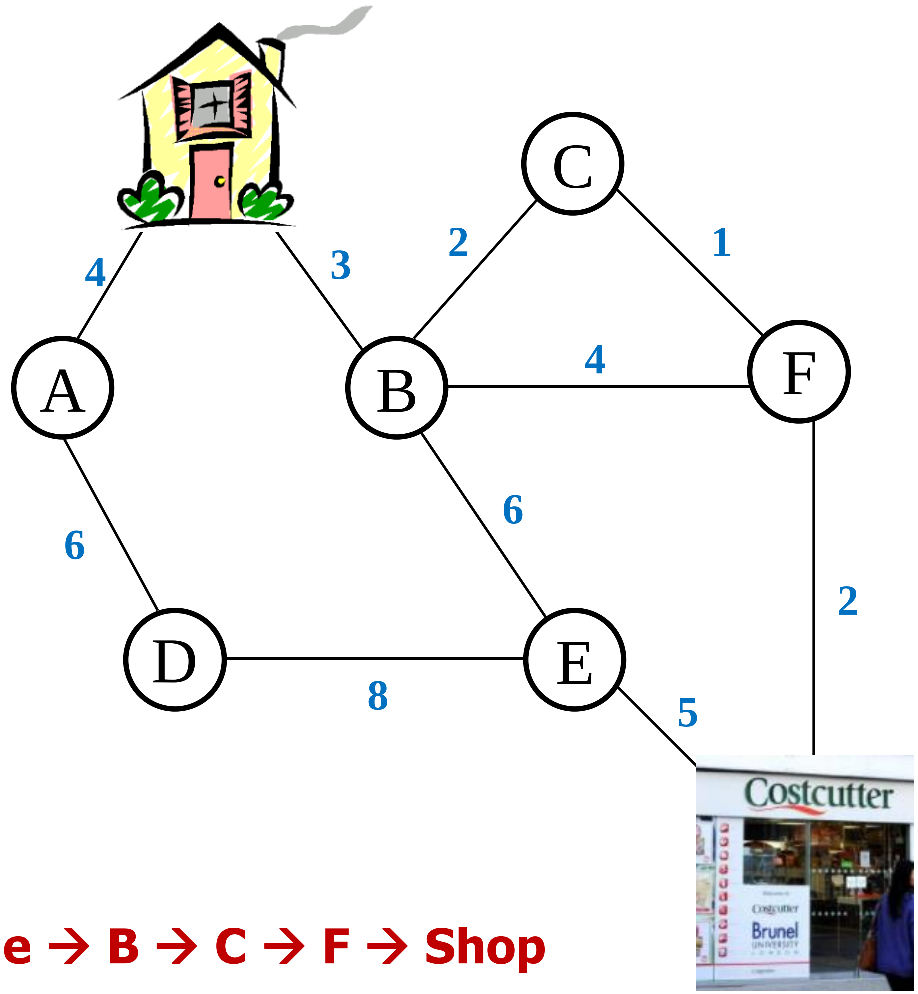
[5,5]

$$\begin{array}{c c} & \begin{array}{cccccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \\ 8 \\ 9 \\ 10 \end{array} & \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \end{array}$$

Shortest Path Problem

- The problem of finding a path between two nodes in a graph such that the sum of weights of its edges is minimised, e.g.:
  - Shortest path between two intersections on a road map
  - A network package being efficiently routed through the network

Shortest Path Example



Home → B → C → F → Shop

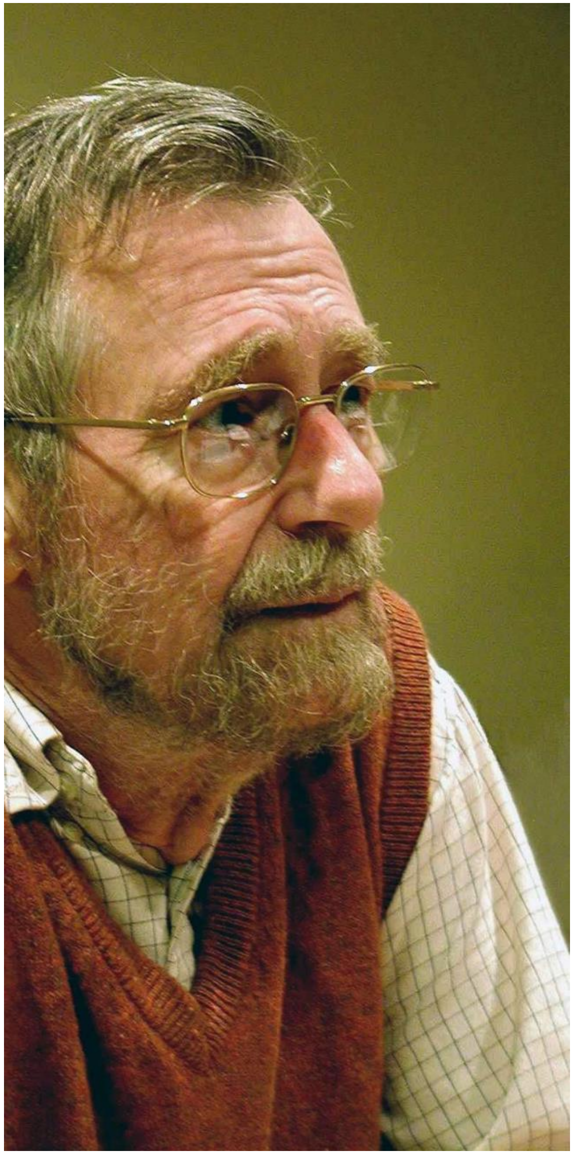
- You want to find shortest possible way from home to your favourite shop!
- You know that there are roads on the way that are more congested and difficult to use than others, and you want to avoid those
- Edges that are more difficult are given a large weight, whereas edges that are easier are given a lower edge weight
- The shortest path found by the algorithm will try and avoid edges with larger weights

Shortest Path Problem Variations and Algorithms

- Problem Variations:
  - **Single-source shortest path problem**: shortest paths from source node to all other nodes in the graph
  - **Single-destination shortest path problem**: shortest paths from all nodes in the directed graph to a single destination node.
  - **All-pair shortest path problem**: shortest paths between every pair of nodes in the graph
- Various algorithms for solving the problem: Dijkstra’s algorithm, Bellman-Ford algorithm, A* search algorithm, Floyd-Warshall algorithm etc…

Dijkstra’s Algorithm

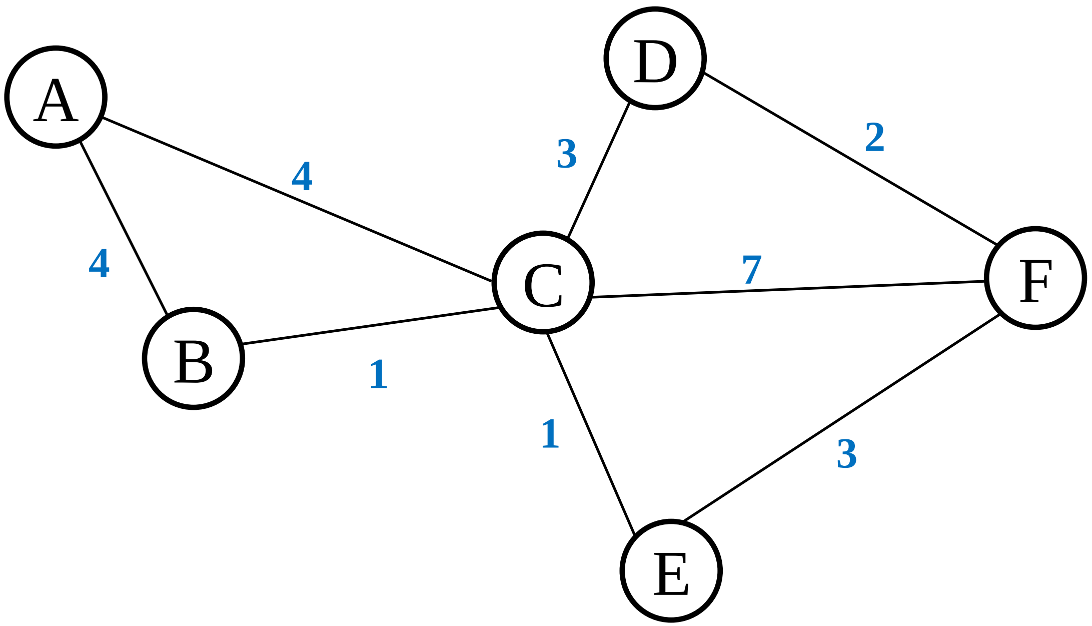
- Conceived by pioneer computer scientist Edsger Dijkstra
- The algorithm is a **greedy approach**
  - Making best (optimal) choice at each stage hoping that the end result is the best solution
- Given a start the algorithm finds the shortest paths between the start node and all the other nodes in a graph



Dijkstra’s Algorithm

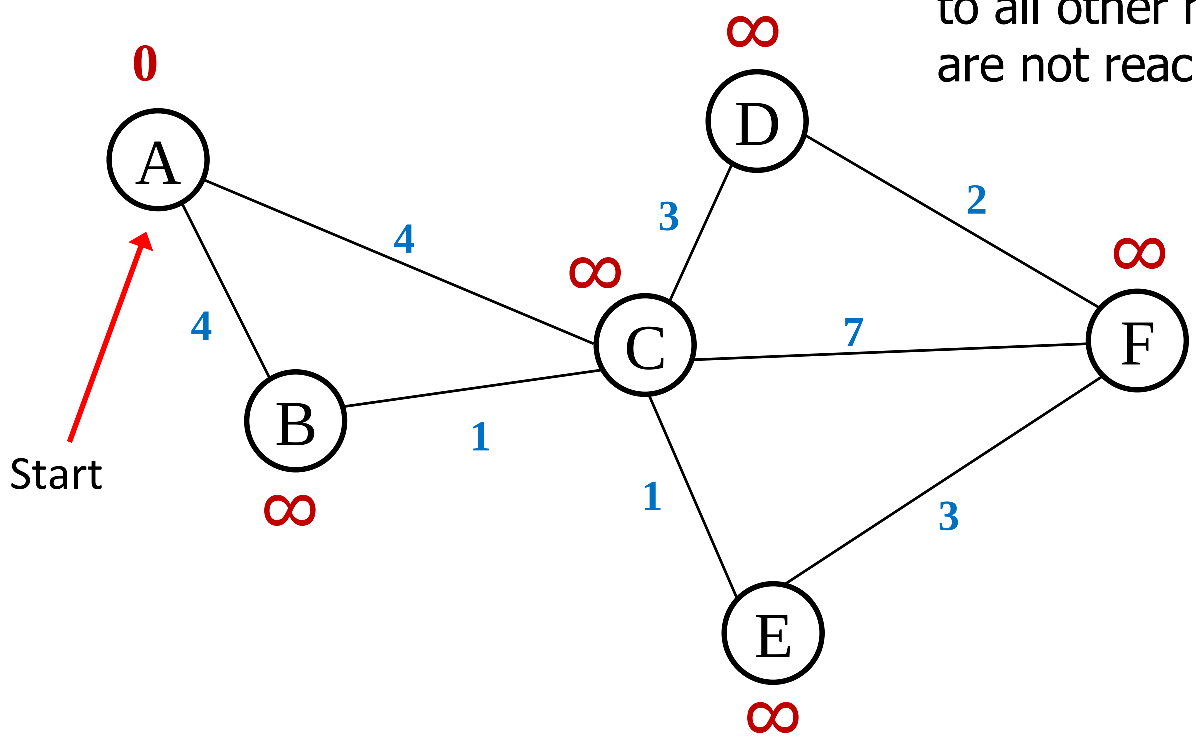
- Used to solve the shortest path problem
  - Both directed and undirected graphs
  - All edges must have non-negative weights
  - Graphs must be connected
- Many variants but the most common is to find the shortest paths from source node to all other nodes in the graph.
- Given an end destination, we can keep track of visited nodes and thus find the path

Dijkstra's Algorithm - Example



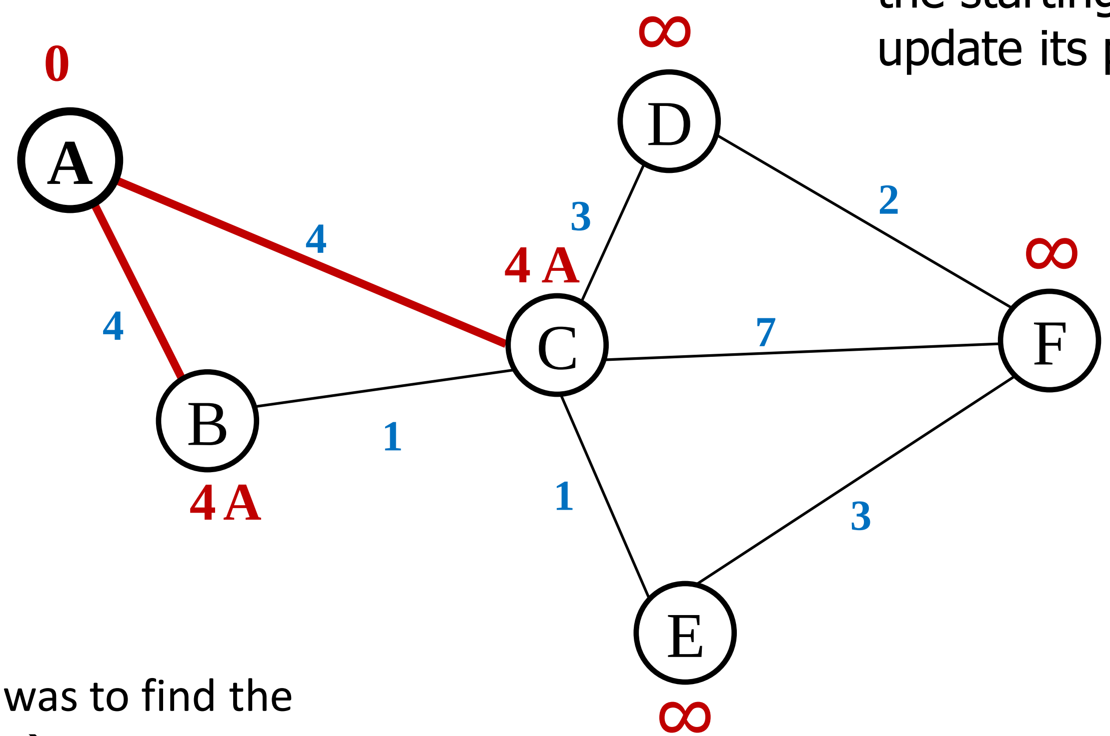
Dijkstra’s Algorithm - Example

- Choose a starting nodes
- Assign infinity path values to all other nodes as they are not reachable



Dijkstra’s Algorithm - Example

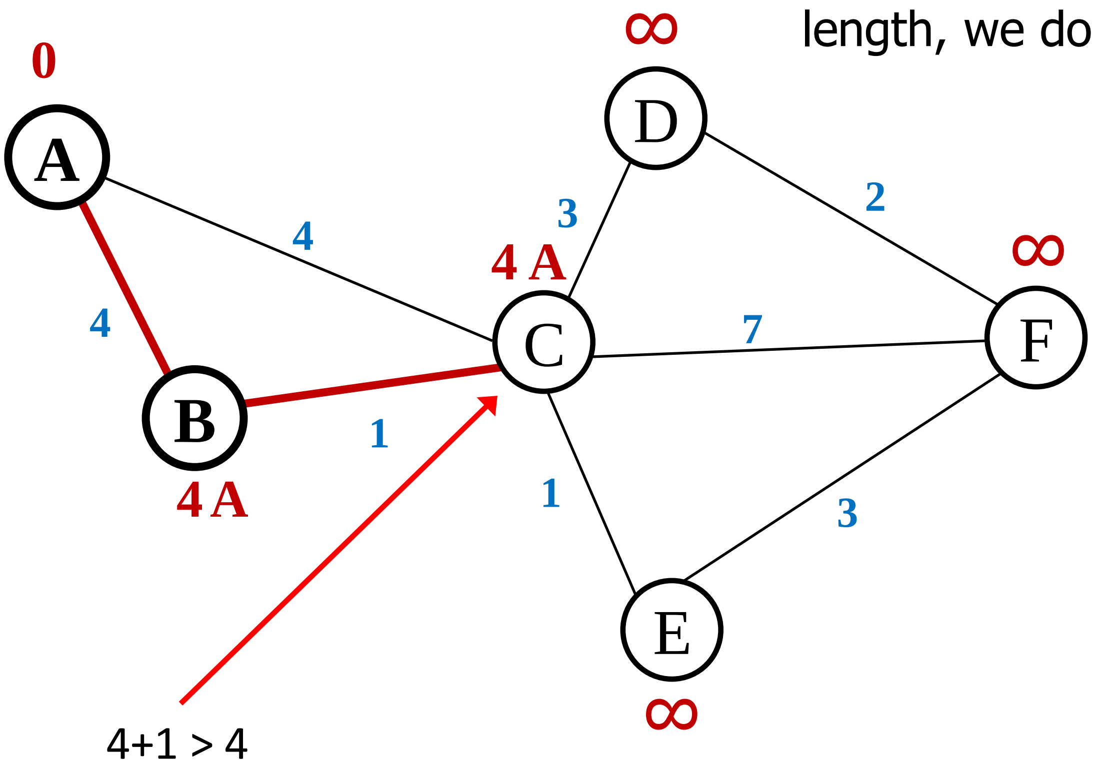
Go to each node adjacent to the starting node and update its path length



If the objective was to find the shortest path A → C, then it would be complete

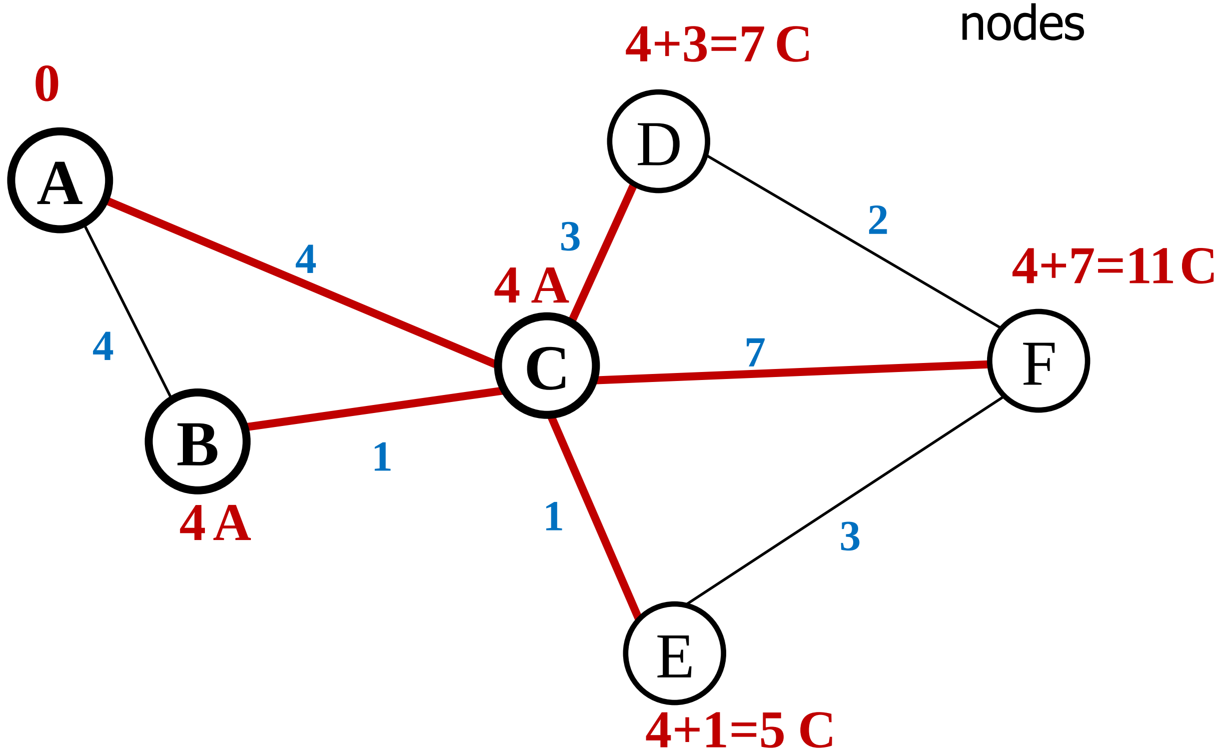
Dijkstra's Algorithm - Example

If the path length of adjacent node is less than the new path length, we don't update!



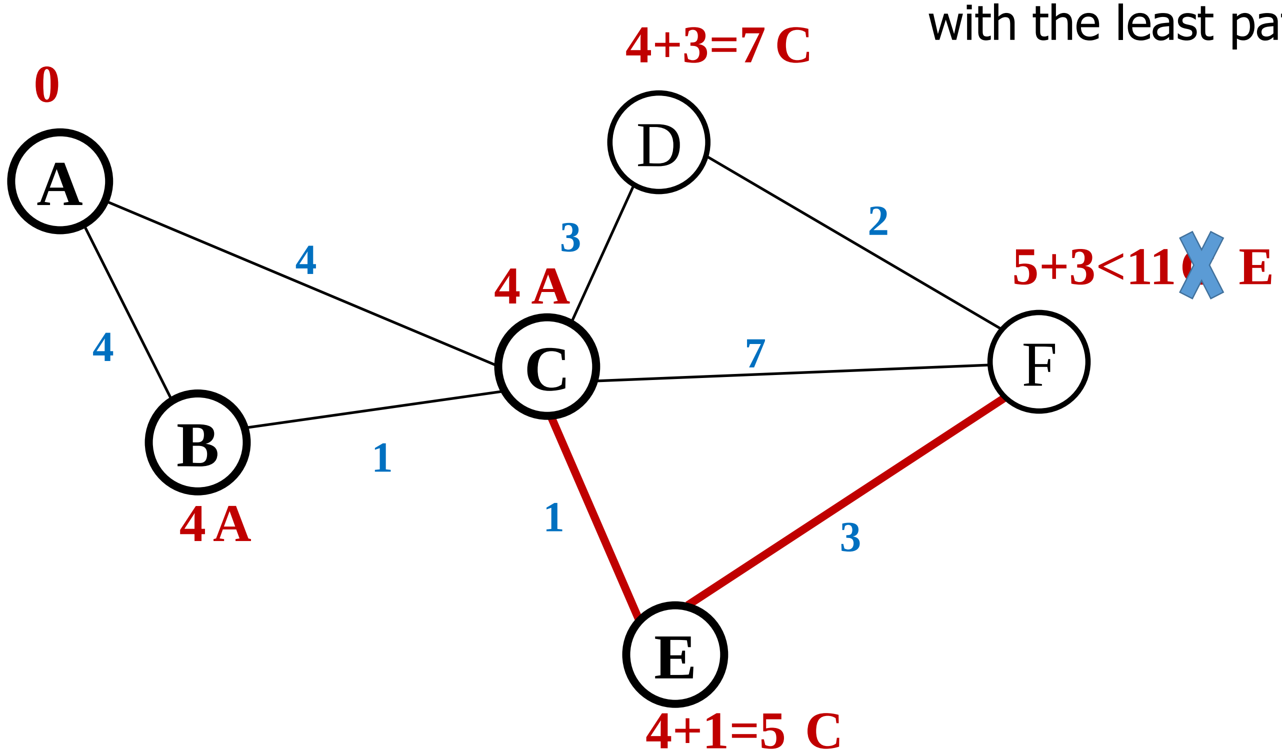
Dijkstra’s Algorithm - Example

We don’t update path lengths of already visited nodes



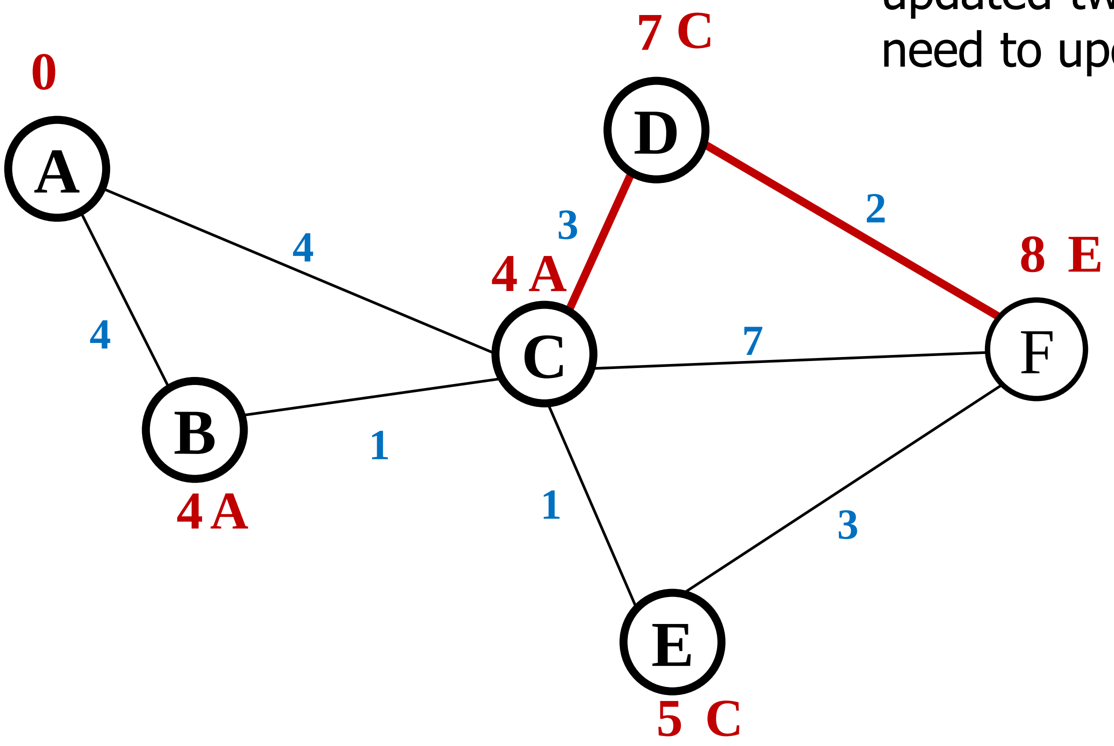
Dijkstra’s Algorithm - Example

After each iteration, we pick the unvisited nodes with the least path length

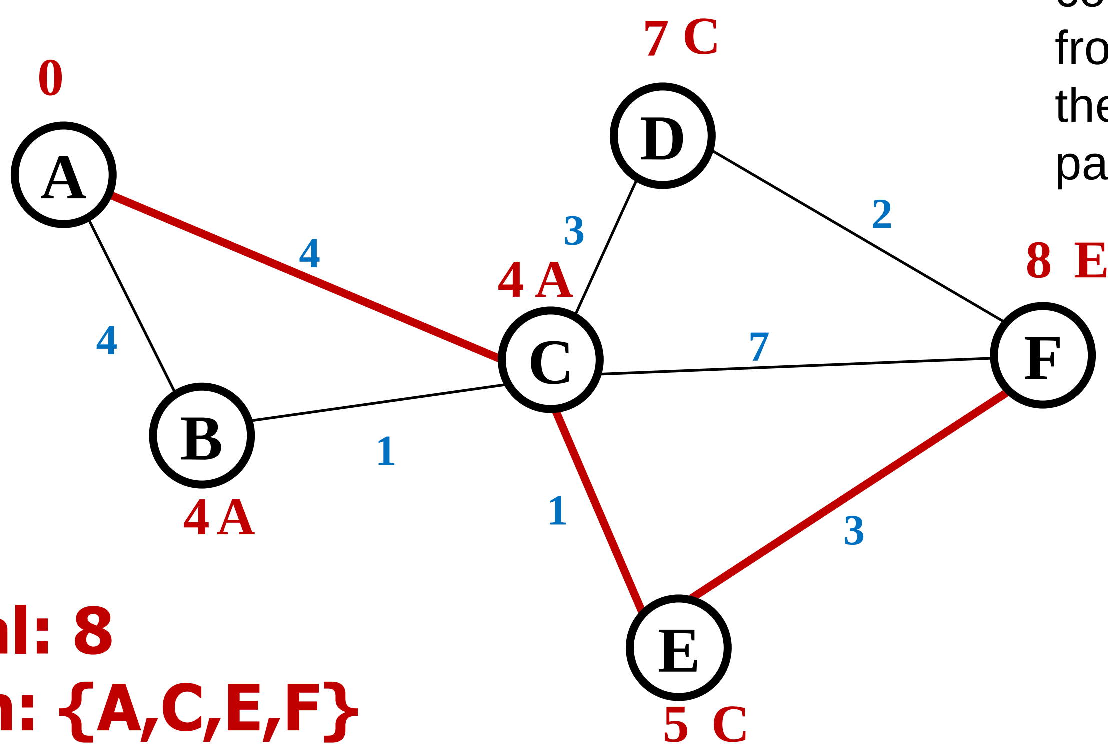


Dijkstra’s Algorithm - Example

Node F has already been updated twice, but we don’t need to update it again!



Dijkstra’s Algorithm - Example



Once the algorithm is complete, we can backtrack from the destination node to the starting node to find the path

Total: 8
Path: {A,C,E,F}

Dijkstra’s algorithm demonstration

https://www.cs.usfca.edu/~galles/visualization/Dijkstra.html

Dijkstra’s Algorithm Pseudocode

```
Algorithm 1. Dijkstra(G,S)
Input: G- The graph being searched
       S- The start node
1) Let Unvisited = all unvisited nodes from G
2) Let distance of start node = 0
3) Let distance of all other nodes = ∞
4) while Unvisited is not empty
5)    let currentNode = node with the smallest distance
6)    remove currentNode from Unvisited
7)   If currentNode's position is the goal
8)     Backtrack to get path
9)   For each neighbour (still in Unvisited) to the currentNode
10)      let newDist = currentNode's distance + distance of
         currentNode and neighbour
11)      If newDist < currentNode's distance
12)         set neighbour's distance to newDist
13)         set neighbour's parent to currentNode
14)   End For
15) End While
Output: The shortest paths between start node and all other nodes
```

Dijkstra's Time Analysis

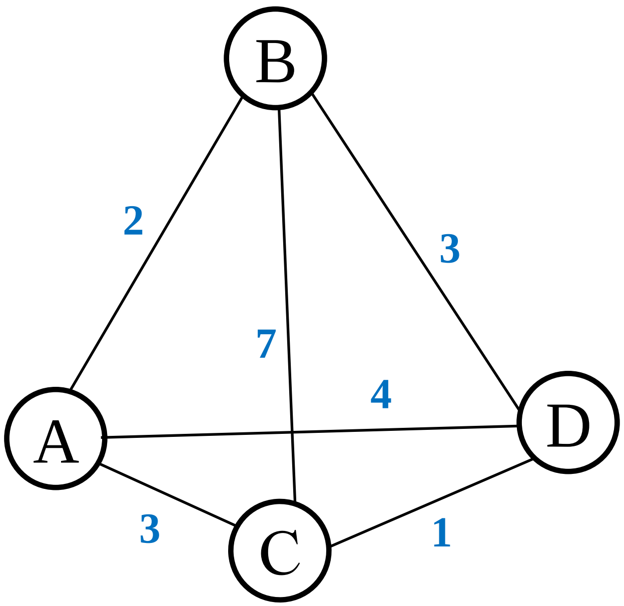
- The complexity of the algorithm depends on the implementation and the data structure:
  - The classic implementation is O(n^2), n being the number of nodes of the graph
  - Using other implementations like heaps: O(E + n log n), E is the number of edges and n is the number of nodes of the graph

Dijkstra's Algorithm Applications

- Finding Shortest Path!
- GPS System
  - A geographical map as a graph
  - Locations in the map are nodes/ vertices
  - Road between locations are edges
  - Weights of edges are distance between two locations
- Network routing
- Commercial Shipping
- Etc…

Floyd–Warshall Algorithm

- This algorithm computes how far each node is from every other node in the input graph
- All-pairs shortest path
- Computes 1-step paths, 2-step paths, etc...
  - Is it shorter to go to a node via a node rather than directly to it?
- It does not return the paths [but it can do…]
- Negative edges allowed



Dynamic Programming

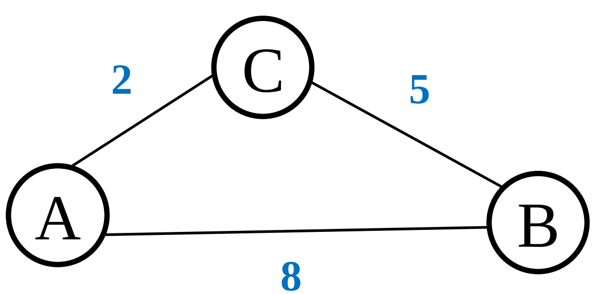
- Floyd-Warshall relies on Dynamic Programming
- It is a problem-solving approach
- Systematically pre-compute and store the answers to sub-problems to build up the solution to a complex problem
  - Reuses those recorded results instead of recomputing them

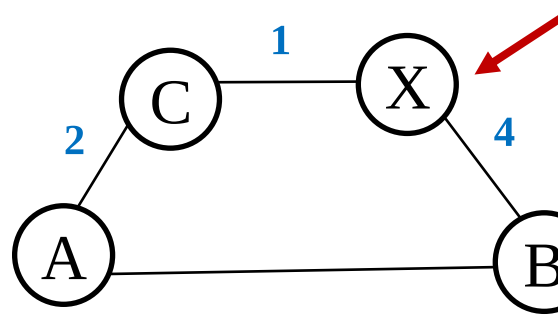
Floyd–Warshall - Steps

- The optimal way to represent the graph is with 2D adjacency matrix
- If there is no edge between two nodes we set the edge weight to infinity
  - Indicating two nodes are not directly connected to each other

Floyd–Warshall - Steps

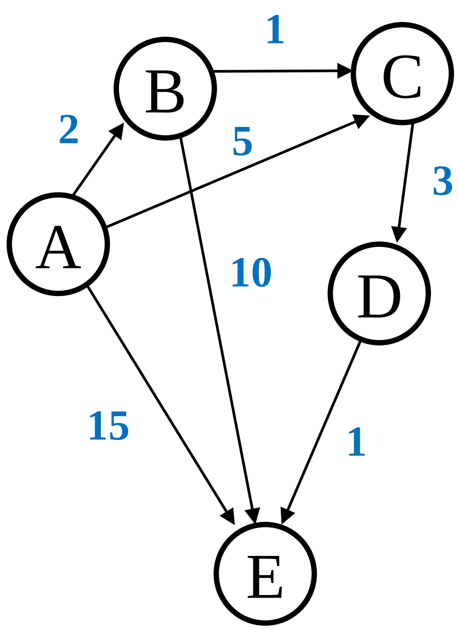
- The main goal is to consider and compute all intermediate paths between two nodes
- For example, we want to go from node A to node B
- If distance of (A → C) + (C → B) is better than A → B, then its better to go through node C
- The optimal path from A → B is:
  - A→ C, C→ B, routing through another node labelled "X"
  - We already computed the optimal path C→ B and it involves intermediate node





We may have longer paths with more intermediate nodes and smaller costs

Floyd–Warshall - Steps



- We make use of dynamic programming to store previous optimal solutions
- Find all pair shortest paths that use 0 intermediate node, then the shortest paths that use 1 intermediate node... until using all N nodes as intermediate nodes.
- Construct a matrix (*n* X *n)*, where matrix(*i*,*j*) = shortest path from node *i* to node *j*
- We route through nodes {0,1,2,…,*k*} computing *k*=0, *k*=1, *k*=2, etc…
- This builds the solution routing through 0, then all solutions through 0 and 1, then all solutions through 0,1,2, and so on…
- Until we covered all nodes and thus solving the shortest path problem

Floyd-Warshall algorithm demonstration

https://www.cs.usfca.edu/~galles/visualization/Floyd.html

Floyd–Warshall - Pseudocode

Algorithm 2. FW(D)

Input: D, an n by n matrix representing distance pairs between nodes, if there is no edge then $d_{ij} = +\infty$

1) Let P = D
2) For k = 1 to n
3)   For i = 1 to n
4)   For j = 1 to n
5) $p_{ij} = \mathrm{Min}(p_{ij}, p_{ik} + p_{kj})$
6)   End For
7)   End For
8) End For

Output: P, the shortest paths between all nodes

Floyd–Warshall Time Analysis

- What is the time complexity of this algorithm?
  - O(n^3), where n is the number of nodes
  - Because of the three nested `for` loops
- Ideal for smaller graphs
  - No larger than few hundred nodes

Next Lecture

- We will be looking at Search and Fitness

This Weeks Laboratory

- No new laboratory worksheet
- Use this week's laboratory session to:
  - Catch up with your worksheets
  - Do your Class Tests