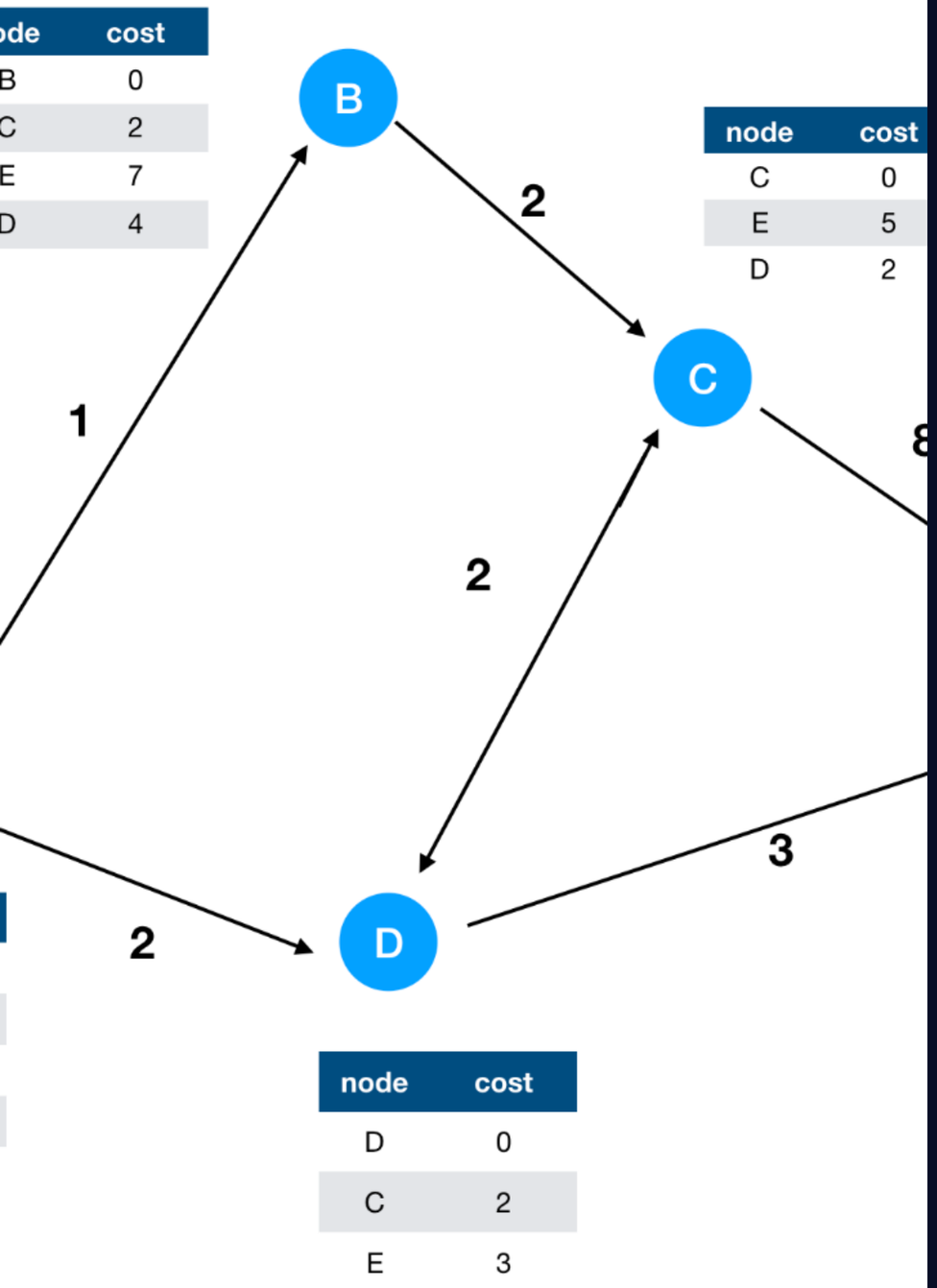


ode	cost
B	0
C	2
E	7
D	4

node	cost
C	0
E	5
D	2

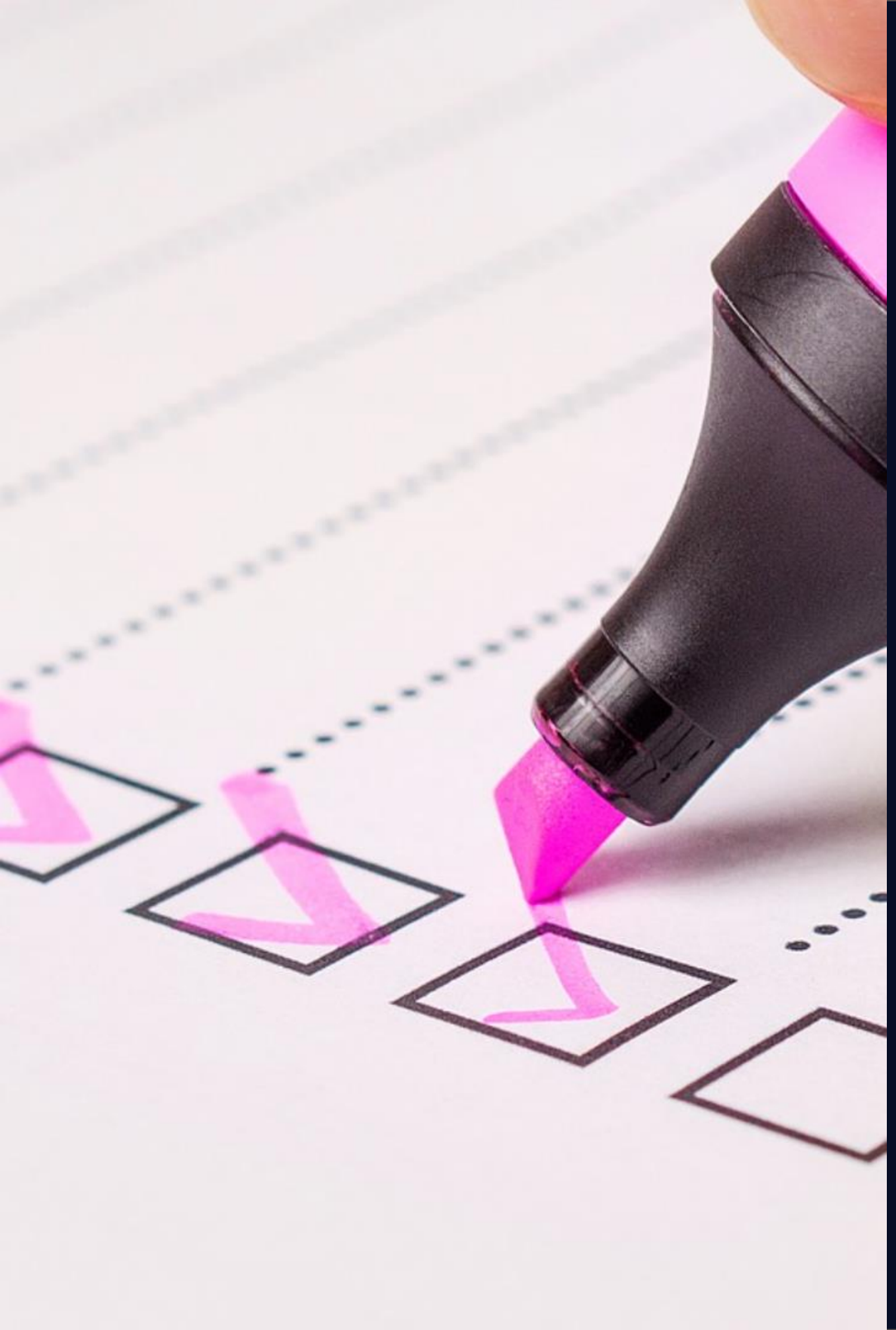
node	cost
D	0
C	2
E	3

Bellman Ford

Welcome to the world of Bellman Ford! In this presentation, we will explore the importance, functioning, time complexity, and various applications of this powerful algorithm.

Mehek K044 Mukund K046

Shetty K054 Rishabh K056



Overview and Introduction

Bellman Ford is a single-source shortest path algorithm that can find the shortest path in a weighted graph, even with negative edge weights.

Let's dive into the details of this algorithm and understand why it is crucial in the world of graph theory.

Applications of Bellman Ford

Routing Protocols

Bellman Ford plays a crucial role in routing protocols, where it helps determine the best path for data packets to travel between networks.

Negative Cycle Detection

The algorithm can also identify the presence of negative cycles, which are important to detect in certain scenarios to avoid infinite loops or unreliable computations.

Algorithm

The Bellman Ford algorithm is a fundamental graph algorithm used to find the shortest path from a source vertex to all other vertices in a weighted graph. It works by iteratively relaxing the edges of the graph until the optimal distances are found. The algorithm also detects negative weight cycles, which can cause infinite loops in path calculations. By using dynamic programming principles, the Bellman Ford algorithm provides an efficient solution for pathfinding in various

Code

This Python code demonstrates the Bellman-Ford algorithm. It's within a Graph class, adding edges, finding paths, finding paths, handling negative cycles, and displaying shortest distances from the source vertex to all others.
all others.

```python
class Network:
  def __init__(self, num_nodes):
      self.num_nodes = num_nodes
      self.edges = []

  def add_edge(self, src, dest, weight):
      self.edges.append((src, dest, weight))

  def bellman_ford(self, source):
      distance = [float('inf')] * self.num_nodes
      distance[source] = 0

      for _ in range(self.num_nodes - 1):
          for src, dest, weight in self.edges:
              if distance[src] + weight < distance[dest]:
                  distance[dest] = distance[src] + weight

      negative_cycle = False
      for src, dest, weight in self.edges:
          if distance[src] + weight < distance[dest]:
              negative_cycle = True
              break

      if negative_cycle:
          print("Negative weight cycle detected. Cannot find reliable routes.")
      else:
          print("Shortest distance from the source node:")
          for node, dist in enumerate(distance):
              if dist == float('inf'):
                  print(f"Node {node}: Not reachable")
              else:
                  print(f"Node {node}: Distance {dist}")
```

```python
if __name__ == "__main__":
  num_nodes = int(input("Enter the number of nodes: "))
  network = Network(num_nodes)

  num_edges = int(input("Enter the number of edges: "))
  for _ in range(num_edges):
      src = int(input("Enter the source node for the edge: "))
      dest = int(input("Enter the destination node for the edge:
"))
      weight = int(input("Enter the weight of the edge: "))
      network.add_edge(src, dest, weight)

  source_node = int(input("Enter the source node for the
Bellman-Ford algorithm: "))
  network.bellman_ford(source_node)
```

Output

```
Enter the destination node for the edge: 1
Enter the weight of the edge: -2
Enter the source node for the edge: 3
Enter the destination node for the edge: 4
Enter the weight of the edge: 9
Enter the source node for the Bellman-Ford algorithm: 1
Shortest distance from the source node:
Node 0: Not reachable
Node 1: Distance 0
Node 2: Distance 5
Node 3: Distance 8
Node 4: Distance -4
```

Advantages

1 Advantages of the Algorithm

Bellman Ford handles graph with negative negative edge weights, making it a versatile versatile algorithm with a wide range of applications.

2 Advantage 2

Bellman Ford can handle negative cycles in a graph and detect them.

3 Advantage 3

The algorithm can handle graphs with multiple edges between the same vertices.

Limitations

1 Limitations and When to Avoid Using It

Although powerful, the algorithm has a higher higher time complexity compared to other other shortest path algorithms, making it less less efficient for larger graphs.

2 Limitation 2

The algorithm may not work correctly if the the graph contains a negative cycle.

3 Limitation 3

Bellman Ford is not the most efficient algorithm for graphs with non-negative edge weights.

Conclusion

Bellman Ford is a fundamental algorithm in graph theory, allowing us to find the shortest path in diverse scenarios. Its ability to handle negative edge weights and detect negative cycles makes it a valuable tool in various applications.

Now that we have explored the intricacies of Bellman Ford, we hope you have gained a deeper understanding of understanding of this remarkable algorithm!

Bibliography

Bellman–Ford Algorithm - GeeksforGeeks

Bellman Ford's Algorithm

Bellman-Ford Algorithm - javatpoint

Bellman–Ford Algorithm | Scaler Topics


THANK YOU