**Ford Fulkerson Network Flow**

**Iheb Gafsi**\*

INSAT Student

**Iheb.engineer@gmail.com**

**Definition:**

Ford-Fulkerson is a widely used algorithm for solving the maximum flow problem in network flow theory. The network flow problem involves finding the maximum amount of flow that can be sent from a source node to a sink node through a directed graph with capacities assigned to its edges. The main idea behind the Ford-Fulkerson algorithm is to find augmenting paths in the graph, which are paths from the source to the sink that can increase the flow. In each iteration, the algorithm finds an augmenting path using techniques like depth-first search or breadth-first search and computes the maximum amount of flow that can be pushed along this path. The process continues until no more augmenting paths can be found, and the maximum flow is achieved. The algorithm can be further improved using various techniques like the Edmonds-Karp algorithm, which guarantees a polynomial time complexity by using a shortest path search.

**Use cases:**

In network flow theory, a flow graph represents a network where the edges have capacities that indicate the maximum amount of flow they can carry. The goal is to find the maximum flow from a source node to a sink node while respecting the capacity constraints on each edge. The Ford-Fulkerson algorithm is an essential method to solve this problem. An augmenting path is a simple path from the source to the sink in the flow graph, and finding such paths is crucial in increasing the flow. The algorithm increases the flow along these paths until no more augmenting paths can be found, which ensures an optimal solution for the maximum flow problem. Network flow problems and the Ford-Fulkerson algorithm have numerous applications in real-world scenarios, such as traffic flow optimization, network bandwidth allocation, resource management in supply chains, and finding the maximum flow of data in computer networks. Additionally, the concept of flow graphs is used in various optimization problems like matching algorithms, image segmentation, and even in some mathematical proofs.

**Algorithm:**

1. Initialize the flow of all edges to zero.

2. While there exists an augmenting path from the source to the sink:

a. Find an augmenting path using a graph traversal algorithm like DFS or BFS.

b. Determine the minimum capacity (residual capacity) along this path. This is the maximum additional flow that can be pushed through the path.

c. Update the flow of each edge along the augmenting path by adding the minimum capacity.

d. Update the residual capacities of forward and backward edges. The residual capacity of a forward edge is the original capacity minus the flow, and for a backward edge, it is the flow.

3. When no more augmenting paths can be found, the algorithm terminates, and the current flow represents the maximum flow from the source to the sink.

1. # Constants

2. INF = float('inf')

3. # Edge Class

4. class Edge:

5.     def \_\_init\_\_(self, back, front, capactiy):

6.         self.back = back

7.         self.front = front

8.         self.capacity = capactiy

9.         self.residual = None

10.         self.flow = 0

11.     def isResidual(self):

12.         return self.capacity == 0

13.     def remaining\_capactiy(self):

14.         return self.capacity - self.flow

15.     def augment(self, bottleNeck):

16.         self.flow += bottleNeck

17.         self.residual.flow -= bottleNeck

18.

19. class FlowNetwork:

20.     def \_\_init\_\_(self, n, source, sink):

21.         self.n = n

22.         self.source = source

23.         self.sink = sink

24.         self.graph = [[] for \_ in range(n)]

25.         self.visited = [0] \* n

26.         self.visitedToken = 1

27.         self.max\_flow = 0

28.

29.     def add\_edge(self, back, front, capacity):

30.         edge = Edge(back, front, capacity)

31.         residual = Edge(front, back, 0)

32.         edge.residual = residual

33.         residual.residual = edge

34.         self.graph[back].append(edge)

35.         self.graph[front].append(residual)

36.     # Ford Fulkerson Algorithm

37.     def dfs(self, node, flow):

38.         if node == self.sink: return flow

39.

40.         self.visited[node] = self.visitedToken

41.         edges = self.graph[node]

42.         for edge in edges:

43.             if edge.remaining\_capactiy()>0 and self.visited[edge.front] != self.visitedToken:

44.                 bottlneck = self.dfs(edge.front, min(flow, edge.remaining\_capactiy()))

45.                 if bottlneck >0:

46.                     edge.augment(bottlneck)

47.                     return bottlneck

48.         return 0

49.

50.     def find\_max\_flow(self):

51.         f = self.dfs(self.source, INF)

52.         while f!=0:

53.             self.visitedToken += 1

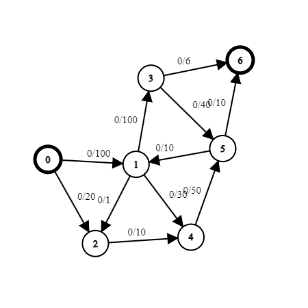
54.             self.max\_flow += f

55.             f = self.dfs(self.source, INF)

56.         return self.max\_flow

**Example:**

Here’s a small example illustrating an example of input outputs for the Ford Fulkerson Algorithm:



We will use the Python code down below to outline the output of the algorithm on this graph:

1. # Variables

2. INF = float('inf')

3. # Edge Class

4. class Edge:

5.     def \_\_init\_\_(self, back, front, capactiy):

6.         self.back = back

7.         self.front = front

8.         self.capacity = capactiy

9.         self.residual = None

10.         self.flow = 0

11.     def isResidual(self):

12.         return self.capacity == 0

13.     def remaining\_capactiy(self):

14.         return self.capacity - self.flow

15.     def augment(self, bottleNeck):

16.         self.flow += bottleNeck

17.         self.residual.flow -= bottleNeck

18.

19. class FlowNetwork:

20.     def \_\_init\_\_(self, n, source, sink):

21.         self.n = n

22.         self.source = source

23.         self.sink = sink

24.         self.graph = [[] for \_ in range(n)]

25.         self.visited = [0] \* n

26.         self.visitedToken = 1

27.         self.max\_flow = 0

28.

29.     def add\_edge(self, back, front, capacity):

30.         edge = Edge(back, front, capacity)

31.         residual = Edge(front, back, 0)

32.         edge.residual = residual

33.         residual.residual = edge

34.         self.graph[back].append(edge)

35.         self.graph[front].append(residual)

36.     # Ford Fulkerson Algorithm

37.     def dfs(self, node, flow):

38.         if node == self.sink: return flow

39.

40.         self.visited[node] = self.visitedToken

41.         edges = self.graph[node]

42.         for edge in edges:

43.             if edge.remaining\_capactiy()>0 and self.visited[edge.front] != self.visitedToken:

44.                 bottlneck = self.dfs(edge.front, min(flow, edge.remaining\_capactiy()))

45.                 if bottlneck >0:

46.                     edge.augment(bottlneck)

47.                     return bottlneck

48.         return 0

49.

50.     def find\_max\_flow(self):

51.         f = self.dfs(self.source, INF)

52.         while f!=0:

53.             self.visitedToken += 1

54.             self.max\_flow += f

55.             f = self.dfs(self.source, INF)

56.         return self.max\_flow

57.

58. #Application

59. ford = FlowNetwork(7, 0, 6)

60. ford.add\_edge(0, 1, 100)

61. ford.add\_edge(0, 2, 20)

62. ford.add\_edge(1, 2, 1)

63. ford.add\_edge(1, 3, 100)

64. ford.add\_edge(1, 4, 30)

65. ford.add\_edge(2, 4, 10)

66. ford.add\_edge(3, 5, 40)

67. ford.add\_edge(3, 6, 6)

68. ford.add\_edge(4, 5, 50)

69. ford.add\_edge(5, 1, 10)

70. ford.add\_edge(5, 6, 10)

71.

72. print(ford.find\_max\_flow())

The corresponding output is:

Python>> 16