**Ford Fulkerson Network Flow**

**Iheb Gafsi**\*

INSAT Student

**Iheb.engineer@gmail.com**

**Definition:**

The Edmonds-Karp algorithm is a specific implementation of the Ford-Fulkerson method for finding the maximum flow in a network flow graph. It efficiently solves the unspecific detail of choosing augmenting paths by employing a breadth-first search (BFS) strategy to find the shortest augmenting paths from the source to the sink. Unlike the original Ford-Fulkerson method that uses any augmenting path, the Edmonds-Karp algorithm ensures that the augmenting paths have the minimum number of edges, leading to a faster convergence.

The algorithm iteratively finds augmenting paths and updates the flow in the graph until no more augmenting paths can be found. It guarantees that the flow is non-decreasing at each step, and when no more augmenting paths are available, it produces the maximum flow.

By employing BFS to find augmenting paths, the Edmonds-Karp algorithm optimizes the search process by focusing on shorter paths first. This approach significantly improves the algorithm's performance, especially when the capacities of the edges are integral and relatively small. Due to its efficiency and guaranteed termination, the Edmonds-Karp algorithm has become one of the most widely used methods for solving network flow problems in various applications, including transportation, communication networks, and resource allocation...

The Edmonds-Karp algorithm reduces the time complexity of finding the maximum flow in a network flow graph from an unbounded in the Ford-Fulkerson algorithm to , where V is the number of vertices and E is the number of edges in the graph. The use of BFS to find shortest augmenting paths ensures that each augmenting path is of length at most V, and there can be at most E iterations to reach the maximum flow. This bounded complexity makes the Edmonds-Karp algorithm more efficient and practical for solving large-scale network flow problems compared to the unbounded complexity of the Ford-Fulkerson algorithm.

**Algorithm:**

1. from queue import Queue

2. # Variables

3. INF = float('inf')

4. # Edge Class

5. class Edge:

6.     def \_\_init\_\_(self, back, front, capacity):

7.         self.back = back

8.         self.front = front

9.         self.capacity = capacity

10.         self.residual = None

11.         self.flow = 0

12.     def isResidual(self):

13.         return self.capacity == 0

14.     def remaining\_capacity(self):

15.         return self.capacity - self.flow

16.     def augment(self, bottleNeck):

17.         self.flow += bottleNeck

18.         self.residual.flow -= bottleNeck

19.

20. class FlowNetwork:

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

22.         self.n = n

23.         self.source = source

24.         self.sink = sink

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

26.         self.visited = [0] \* n

27.         self.visitedToken = 1

28.         self.max\_flow = 0

29.

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

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

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

33.         edge.residual = residual

34.         residual.residual = edge

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

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

37.     # Edmonds-Karp Algorithm

38.     def bfs(self):

39.         queue = Queue()

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

41.         queue.put(self.source)

42.         prev = [None] \* self.n

43.         while not queue.empty():

44.             node = queue.get()

45.             if node == self.sink: break

46.             for edge in self.graph[node]:

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

48.                     self.visited[edge.front] = self.visitedToken

49.                     prev[edge.front] = edge

50.                     queue.put(edge.front)

51.         if prev[self.sink] == None: return 0

52.         bottleneck = INF

53.         edge = prev[self.sink]

54.         while edge!=None:

55.             bottleneck = min(bottleneck, edge.remaining\_capacity())

56.             edge = prev[edge.back]

57.         edge = prev[self.sink]

58.         while edge!=None:

59.             edge.augment(bottleneck)

60.             edge = prev[edge.back]

61.         return bottleneck

62.

63.     def find\_max\_flow(self):

64.         f = self.bfs()

65.         while f!=0:

66.             # Mark all nodes as visited

67.             self.visitedToken += 1

68.             self.max\_flow += f

69.             f = self.bfs()

70.         return self.max\_flow

71.

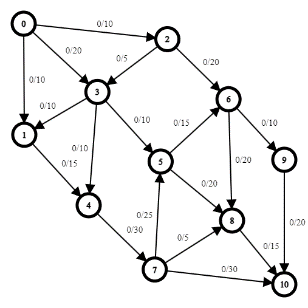
72. #Application

73. edmonds = FlowNetowrk…

74. print(edmonds.find\_max\_flow())

**Example:**

Here’s a small example illustrating an example of input outputs for the Edmonds-Karp Algorithm:



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

1. from queue import Queue

2. # Variables

3. INF = float('inf')

4. # Edge Class

5. class Edge:

6.     def \_\_init\_\_(self, back, front, capacity):

7.         self.back = back

8.         self.front = front

9.         self.capacity = capacity

10.         self.residual = None

11.         self.flow = 0

12.     def isResidual(self):

13.         return self.capacity == 0

14.     def remaining\_capacity(self):

15.         return self.capacity - self.flow

16.     def augment(self, bottleNeck):

17.         self.flow += bottleNeck

18.         self.residual.flow -= bottleNeck

19.

20. class FlowNetwork:

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

22.         self.n = n

23.         self.source = source

24.         self.sink = sink

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

26.         self.visited = [0] \* n

27.         self.visitedToken = 1

28.         self.max\_flow = 0

29.

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

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

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

33.         edge.residual = residual

34.         residual.residual = edge

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

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

37.     # Edmonds-Karp Algorithm

38.     def bfs(self):

39.         queue = Queue()

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

41.         queue.put(self.source)

42.         prev = [None] \* self.n

43.         while not queue.empty():

44.             node = queue.get()

45.             if node == self.sink: break

46.             for edge in self.graph[node]:

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

48.                     self.visited[edge.front] = self.visitedToken

49.                     prev[edge.front] = edge

50.                     queue.put(edge.front)

51.         if prev[self.sink] == None: return 0

52.         bottleneck = INF

53.         edge = prev[self.sink]

54.         while edge!=None:

55.             bottleneck = min(bottleneck, edge.remaining\_capacity())

56.             edge = prev[edge.back]

57.         edge = prev[self.sink]

58.         while edge!=None:

59.             edge.augment(bottleneck)

60.             edge = prev[edge.back]

61.         return bottleneck

62.

63.     def find\_max\_flow(self):

64.         f = self.bfs()

65.         while f!=0:

66.             # Mark all nodes as visited

67.             self.visitedToken += 1

68.             self.max\_flow += f

69.             f = self.bfs()

70.         return self.max\_flow

71.

72. #Application

73. edmonds = FlowNetwork(11, 0, 10)

74. edmonds.add\_edge(0, 2, 10)

75. edmonds.add\_edge(0, 3, 20)

76. edmonds.add\_edge(0, 1, 10)

77. edmonds.add\_edge(1, 4, 15)

78. edmonds.add\_edge(2, 6, 20)

79. edmonds.add\_edge(2, 3, 5)

80. edmonds.add\_edge(3, 5, 10)

81. edmonds.add\_edge(3, 4, 10)

82. edmonds.add\_edge(3, 1, 10)

83. edmonds.add\_edge(4, 7, 30)

84. edmonds.add\_edge(5, 6, 15)

85. edmonds.add\_edge(5, 8, 20)

86. edmonds.add\_edge(6, 9, 10)

87. edmonds.add\_edge(6, 8, 20)

88. edmonds.add\_edge(7, 5, 25)

89. edmonds.add\_edge(7, 8, 5)

90. edmonds.add\_edge(7, 10, 30)

91. edmonds.add\_edge(8, 10, 15)

92. edmonds.add\_edge(9, 10, 20)

93.

94. print(edmonds.find\_max\_flow())

The corresponding output is:

Python>> 40

