**Capacity Scaling Algorithm**

**Iheb Gafsi**\*

INSAT Student

**Iheb.engineer@gmail.com**

**Definition:**

Capacity scaling is an optimization technique used to improve the Ford-Fulkerson algorithm's performance. It aims to reduce the number of iterations required to find the maximum flow in a flow network by efficiently selecting augmenting paths. In the traditional Ford-Fulkerson algorithm, each iteration may consider paths with different flow values, leading to slow convergence. However, capacity scaling ensures that only augmenting paths with flow values that are multiples of a power of two are considered, significantly reducing the number of iterations.

The capacity scaling algorithm starts with an initial flow value of zero and gradually increases it by powers of two until it reaches the maximum flow. During each iteration, it identifies an augmenting path using a depth-first search or breadth-first search, but it restricts the flow along this path to multiples of the current scaling factor. This restriction ensures that the flow increases more rapidly, and each iteration can quickly find the path with the largest possible flow.

The time complexity of the capacity scaling algorithm is , where E is the number of edges in the flow network, and C is the maximum capacity of any edge. This time complexity is much more efficient compared to the original Ford-Fulkerson algorithm, especially when the capacities are large. By exploiting the power of two scaling, capacity scaling efficiently converges to the maximum flow, making it a practical choice for solving maximum flow problems in large flow networks.

**Algorithm:**

1. # Variables

2. INF = float('inf')

3. # Edge Class

4. class Edge:

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

6.         self.back = back

7.         self.front = front

8.         self.capacity = capacity

9.         self.residual = None

10.         self.flow = 0

11.     def isResidual(self):

12.         return self.capacity == 0

13.     def remaining\_capacity(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.         self.delta = 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.         self.delta = max(self.delta, capacity)

38.     # Cs-Karp Algorithm

39.     def dfs(self, node, flow):

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

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

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

43.             if (edge.remaining\_capacity()>=self.delta and

44.                 self.visited[edge.front] != self.visitedToken):

45.                 bottleneck = self.dfs(edge.front, min(flow, edge.remaining\_capacity()))

46.                 if bottleneck > 0:

47.                     edge.augment(bottleneck)

48.                     return bottleneck

49.         return 0

50.

51.     def find\_max\_flow(self):

52.         self.delta = 1<<(self.delta.bit\_length()-1)

53.         while self.delta > 0:

54.             f = 1

55.             while f!=0:

56.                 # Mark all nodes as visited

57.                 self.visitedToken += 1

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

59.                 self.max\_flow += f

60.             self.delta//=2

61.         return self.max\_flow

62.

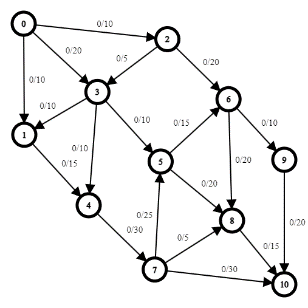
63. #Application

64. cs = FlowNetwork…

65. print(cs.find\_max\_flow())

**Example:**

Here’s a small example illustrating an example of input outputs for the Cs-Karp 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, capacity):

6.         self.back = back

7.         self.front = front

8.         self.capacity = capacity

9.         self.residual = None

10.         self.flow = 0

11.     def isResidual(self):

12.         return self.capacity == 0

13.     def remaining\_capacity(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.         self.delta = 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.         self.delta = max(self.delta, capacity)

38.     # Capacity Scaling Algorithm

39.     def dfs(self, node, flow):

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

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

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

43.             if (edge.remaining\_capacity()>=self.delta and

44.                 self.visited[edge.front] != self.visitedToken):

45.                 bottleneck = self.dfs(edge.front, min(flow, edge.remaining\_capacity()))

46.                 if bottleneck > 0:

47.                     edge.augment(bottleneck)

48.                     return bottleneck

49.         return 0

50.

51.     def find\_max\_flow(self):

52.         self.delta = 1<<(self.delta.bit\_length()-1)

53.         while self.delta > 0:

54.             f = 1

55.             while f!=0:

56.                 # Mark all nodes as visited

57.                 self.visitedToken += 1

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

59.                 self.max\_flow += f

60.             self.delta//=2

61.         return self.max\_flow

62.

63. #Application

64. cs = FlowNetwork(11, 0, 10)

65. cs.add\_edge(0, 2, 10)

66. cs.add\_edge(0, 3, 20)

67. cs.add\_edge(0, 1, 10)

68. cs.add\_edge(1, 4, 15)

69. cs.add\_edge(2, 6, 20)

70. cs.add\_edge(2, 3, 5)

71. cs.add\_edge(3, 5, 10)

72. cs.add\_edge(3, 4, 10)

73. cs.add\_edge(3, 1, 10)

74. cs.add\_edge(4, 7, 30)

75. cs.add\_edge(5, 6, 15)

76. cs.add\_edge(5, 8, 20)

77. cs.add\_edge(6, 9, 10)

78. cs.add\_edge(6, 8, 20)

79. cs.add\_edge(7, 5, 25)

80. cs.add\_edge(7, 8, 5)

81. cs.add\_edge(7, 10, 30)

82. cs.add\_edge(8, 10, 15)

83. cs.add\_edge(9, 10, 20)

84.

85. print(cs.find\_max\_flow())

The corresponding output is:

Python>> 40

