**Implementation of iterative improvement strategy**

**for max flow problem**

**Aim:**

To implement the max flow algorithm using the iterative improvement strategy.

**Problem Description:**

The max flow algorithm, also known as the Ford-Fulkerson algorithm, is an iterative-based algorithm used to calculate the maximum flow that can pass from the source to the sink while considering capacity constraints.

**Algorithm:**

* + Start with the initial flow set to 0.
  + While there exists an augmenting path from the source to the sink:
    - Find an augmenting path using any path-finding algorithm, such as breadth-first search or depth-first search.
    - Determine the amount of flow that can be sent along the augmenting path, which is the minimum residual capacity along the edges of the path.
    - Increase the flow along the augmenting path by the determined amount.
  + Return the maximum flow.

**Code:**

from collections import defaultdict

class Graph:

    def \_\_init\_\_(self, graph):

        self.graph = graph

        self. ROW = len(graph)

    def searching\_algo\_BFS(self, s, t, parent):

        visited = [False] \* (self.ROW)

        queue = []

        queue.append(s)

        visited[s] = True

        while queue:

            u = queue.pop(0)

            for ind, val in enumerate(self.graph[u]):

                if visited[ind] == False and val > 0:

                    queue.append(ind)

                    visited[ind] = True

                    parent[ind] = u

        return True if visited[t] else False

    def ford\_fulkerson(self, source, sink):

        parent = [-1] \* (self.ROW)

        max\_flow = 0

        while self.searching\_algo\_BFS(source, sink, parent):

            path\_flow = float("Inf")

            s = sink

            while(s != source):

                path\_flow = min(path\_flow, self.graph[parent[s]][s])

                s = parent[s]

            max\_flow += path\_flow

            v = sink

            while(v != source):

                u = parent[v]

                self.graph[u][v] -= path\_flow

                self.graph[v][u] += path\_flow

                v = parent[v]

        return max\_flow

graph = [[0, 8, 0, 0, 3, 0],

         [0, 0, 9, 0, 0, 0],

         [0, 0, 0, 0, 7, 2],

         [0, 0, 0, 0, 0, 5],

         [0, 0, 7, 4, 0, 0],

         [0, 0, 0, 0, 0, 0]]

g = Graph(graph)

source = 0

sink = 5

print("\nMax Flow: %d " % g.ford\_fulkerson(source, sink))

**Output:**



**TIME COMPLEXITY:**

In the worst case, where capacities are integers, the time complexity can be O(E

\* |f\*|), where E is the number of edges and |f\*| is the maximum flow value. By

using efficient augmenting path search algorithms like Edmonds-Karp (with BFS),

the time complexity can be reduced to O(V \* E^2), where V is the number of

vertices and E is the number of edges.

**ALGORITHM ANALYSIS:**

The Ford-Fulkerson algorithm is commonly used to solve the maximum flow

problem. The time complexity of the algorithm depends on the specific

implementation and the choice of augmenting path search algorithm.

The Ford-Fulkerson algorithm is practical and efficient for many cases, but for

large networks or specialized requirements, other algorithms like the PushRelabel algorithm may offer better performance. The choice of algorithm

depends on the problem instance and efficiency requirements.

**Result:**

Thus, max flow problem has been executed successfully using iterative improvement.