**Exercise - 9**

**Iterative Improvement**

**Aim:** To implement an iterative improvement strategy in Python to solve stable marriage and maxflow problems.

**Algorithm:**

**Stable Marriage Problem:**

1. Initialize an empty matching, where each element is unmatched initially.

2. While there exist unmatched elements in either set:

* Select an unmatched element from the first set (e.g., a man).
* Iterate through the preference list of the selected element.

1. If the element from the second set (e.g., a woman) is unmatched, assign them to each other and mark them as matched.
2. If the element from the second set is already matched, compare their current partner with the current proposer.

* If the current proposer is preferred over the current partner, assign the proposer and the second element to each other, and mark the previous partner as unmatched.
* If the current partner is preferred over the current proposer, continue to the next preference in the list.

3. Once all elements are matched, the algorithm terminates.

**Maxflow Problem:**

1. Initialize the flow in all edges to 0.

2. While there is an augmenting path from the source (s) to the sink (t) in the residual graph: a. Perform a BFS traversal to find an augmenting path from s to t in the residual graph. b. Determine the minimum capacity (min\_capacity) of the augmenting path.

3. If no augmenting path is found, terminate the algorithm, and the maximum flow has been reached.

4. Otherwise, update the flow in each edge along the augmenting path: a. Increase the flow in each edge along the path by min\_capacity. b. Decrease the flow in the reverse edges (back edges) by min\_capacity.

5. Go back to step 2.

**Source Code:**

**Stable Marriage:**

def stable\_marriage(n):

    men\_preferences = {}

    women\_preferences = {}

    for i in range(n):

        man = input(f"Enter the preferences for man {i+1} (separated by spaces): ").split()

        women\_preferences[i] = [int(woman) - 1 for woman in man]

    for i in range(n):

        woman = input(f"Enter the preferences for woman {i+1} (separated by spaces): ").split()

        men\_preferences[i] = [int(man) - 1 for man in woman]

    engaged = [-1] \* n

    men\_proposals = [0] \* n

    while -1 in engaged:

        man = engaged.index(-1)

        woman = men\_preferences[man][men\_proposals[man]]

        men\_proposals[man] += 1

        if engaged[woman] == -1:

            engaged[woman] = man

        else:

            current\_man = engaged[woman]

            if women\_preferences[woman].index(current\_man) >women\_preferences[woman].index(man):

                engaged[woman] = man

                engaged[current\_man] = -1

    print("\nEngagements:")

    for i, man in enumerate(engaged):

        print(f"Woman {i+1} is engaged to Man {man+1}")

if \_\_name\_\_ == '\_\_main\_\_':

    n = int(input("Enter the number of men/women: "))

    stable\_marriage(n)

**Maxflow:**

from collections import defaultdict

class Graph:

 def \_\_init\_\_(self, vertices):

     self.V = vertices

     self.graph = defaultdict(int)

 def add\_edge(self, u, v, capacity):

     self.graph[u, v] += capacity

 def bfs(self, source, sink, parent):

     visited = [False] \* self.V

     queue = []

     queue.append(source)

     visited[source] = True

     while queue:

         u = queue.pop(0)

         for v in range(self.V):

             if visited[v] is False and self.graph[u, v] > 0:

                 queue.append(v)

                 visited[v] = True

                 parent[v] = u

     return visited[sink]

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

     parent = [-1] \* self.V

     max\_flow = 0

     while self.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

vertices = int(input("Enter the number of vertices in the flow network: "))

edges = int(input("Enter the number of edges in the flow network: "))

g = Graph(vertices)

print("Enter the source, destination, and capacity for each edge:")

for \_ in range(edges):

 u, v, capacity = map(int, input().split())

 g.add\_edge(u, v, capacity)

source = int(input("Enter the source vertex: "))

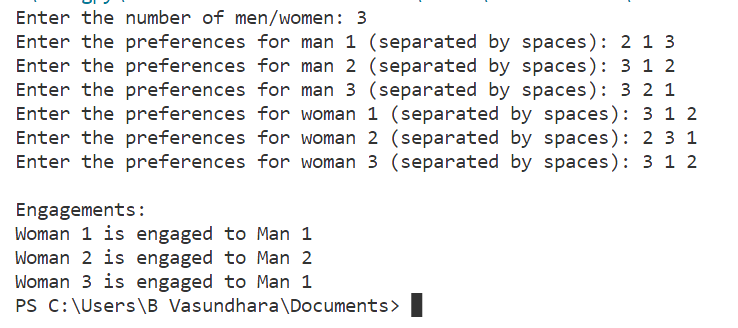
sink = int(input("Enter the sink vertex: "))

max\_flow = g.ford\_fulkerson(source, sink)

print("The maximum flow in the network is:", max\_flow)

**Sample Input and Output:**

**Stable Marriage:**

****

**Maxflow:**

A screenshot of a computer

Description automatically generated with medium confidence

**Result:**

Thus, an iterative improvement strategy in Python to solve maxflow problems and stable marriage problems has been successfully implemented and the output is verified.