Code:-

```
from collections import defaultdict


class Graph:
    def __init__(self, vertices):
        self.graph = defaultdict(dict)
        self.V = vertices


    def add_edge(self, u, v, w):
        self.graph[u][v] = w
        # Add reverse edge with 0 capacity if not present
        if v not in self.graph or u not in self.graph[v]:
            self.graph[v][u] = 0


    def _dfs(self, u, t, visited, path):
        visited.add(u)
        if u == t:
            return path
        for v in self.graph[u]:
            capacity = self.graph[u][v]
            if v not in visited and capacity > 0:
                res = self._dfs(v, t, visited, path + [(u, v)])
                if res:
                    return res
        return None


    def ford_fulkerson(self, source, sink):
        max_flow = 0
        while True:
            visited = set()
            path = self._dfs(source, sink, visited, [])
            if not path:
                break
```

```
            # Find minimum residual capacity in the path
            flow = min(self.graph[u][v] for u, v in path)

            # Update residual capacities
            for u, v in path:
                self.graph[u][v] -= flow
                self.graph[v][u] += flow

            max_flow += flow

        return max_flow
g = Graph(6)
g.add_edge(0, 1, 16)
g.add_edge(0, 2, 13)
g.add_edge(1, 2, 10)
g.add_edge(1, 3, 12)
g.add_edge(2, 1, 4)
g.add_edge(2, 4, 14)
g.add_edge(3, 2, 9)
g.add_edge(3, 5, 20)
g.add_edge(4, 3, 7)
g.add_edge(4, 5, 4)

print("Max Flow:", g.ford_fulkerson(0, 5))
```

Output:-

```
Max Flow: 23

=== Code Execution Successful ===
```