**TASK 15**

**Iterative Improvements**

In a densely populated urban area, an unforeseen medical emergency has occurred, and there's an urgent need to distribute essential medical supplies to various locations within the city. The Ford-Fulkerson algorithm is employed to determine the maximum flow in the network, facilitating the fastest distribution of medical supplies.

**Input:**

The adjacency matrix:

0 10 0 10 0  0

0  0 4  2 8  0

0  0 0  0 0 10

0  0 0  0 9  0

0  0 6  0 0 10

0  0 0  0 0  0

**Output:**

Maximum flow is: 23

**Test Case 1:** Apply Ford-Fulkerson algorithm and find whether the augmenting path exists or not in the graph.

**Test Case 2:** Apply Ford-Fulkerson algorithm and find the updated residual capacity of a graph

**Aim:**

Write a C Program to apply Ford-Fulkerson algorithm to find maximum flow in a network.

**Algorithm**:

Step 1: Initialize the flow in all the edges to 0.

Step 2: While there is an augmenting path between the source and the sink, add this path to the flow.

Step 3:Update the residual graph

**Program:**

#include <stdio.h>

#define A 0

#define B 1

#define C 2

#define MAX\_NODES 1000

#define O 1000000000

int n;

int e;

int capacity[MAX\_NODES][MAX\_NODES];

int flow[MAX\_NODES][MAX\_NODES];

int color[MAX\_NODES];

int pred[MAX\_NODES];

int min(int x, int y) {

return x < y ? x : y;

}

int head, tail;

int q[MAX\_NODES + 2];

void enqueue(int x) {

q[tail] = x;

tail++;

color[x] = B;

}

int dequeue() {

int x = q[head];

head++;

color[x] = C;

return x;

}

// Using BFS as a searching algorithm

int bfs(int start, int target) {

int u, v;

for (u = 0; u < n; u++) {

color[u] = A;

}

head = tail = 0;

enqueue(start);

pred[start] = -1;

while (head != tail) {

u = dequeue();

for (v = 0; v < n; v++) {

if (color[v] == A && capacity[u][v] - flow[u][v] > 0) {

enqueue(v);

pred[v] = u;

}

}

}

return color[target] == C;

}

// Applying fordfulkerson algorithm

int fordFulkerson(int source, int sink) {

int i, j, u;

int max\_flow = 0;

for (i = 0; i < n; i++) {

for (j = 0; j < n; j++) {

flow[i][j] = 0;

}

}

// Updating the residual values of edges

while (bfs(source, sink)) {

int increment = O;

for (u = n - 1; pred[u] >= 0; u = pred[u]) {

increment = min(increment, capacity[pred[u]][u] - flow[pred[u]][u]);

}

for (u = n - 1; pred[u] >= 0; u = pred[u]) {

flow[pred[u]][u] += increment;

flow[u][pred[u]] -= increment;

}

// Adding the path flows

max\_flow += increment;

}

return max\_flow;

}

int main() {

for (int i = 0; i < n; i++) {

for (int j = 0; j < n; j++) {

capacity[i][j] = 0;

}

}

n = 6;

e = 7;

capacity[0][1] = 18;

capacity[0][4] = 13;

capacity[1][2] =19;

capacity[2][4] = 17;

capacity[2][5] =12;

capacity[3][5] = 15;

capacity[4][2] = 17;

capacity[4][3] = 14;

int s = 0, t = 5;

printf("Max Flow: %d\n", fordFulkerson(s, t));

}

**Output:**

**Max Flow: 26**