| Experiment **#13** | | Student ID | |
|---|---|---|---|
| Date | | Student Name | [@KLWKS_BOT] THANOS |

**Experiment Title: To implement programs on problem solving using Network Flow Algorithms.**

**Aim/Objective:** To understand the concept and implementation of programs on Network Flow Algorithms

**Description:** The students will understand the programs on Ford-Fulkerson Method, Edmonds-Karp Algorithm, Max-Flow Min-Cut Theorem, applying them to solve real-world problems.

**Pre-Requisites:**

**Knowledge:** Ford-Fulkerson Method, Edmonds-Karp Algorithm, Max-Flow Min-Cut Theorem

**Tools:** Code Blocks/Eclipse IDE.

**Pre-Lab:**

You are given a directed graph representing a flow network, where each edge has a capacity. Your task is to compute the maximum flow from a source node s to a sink node t using the Ford-Fulkerson method.

**Input Format**:

- An integer n, the number of nodes in the graph.
- An integer m, the number of edges in the graph.
- The next m lines each contain three integers u, v, and c, representing a directed edge from node u to node v with capacity c.
- Two integers s and t, the source and sink nodes.

**Output Format:**

- A single integer representing the maximum flow from s to t.

**Sample Input:**

- There are 5 cities: A, B, C, D, and E. The possible railway connections and their costs are:

**Constraints:**

- $2 \leq n \leq 100$
- $1 \leq m \leq 10^4$
- $1 \leq u,v \leq n$
- $0 \leq c \leq 10^9$
- There is at least one path from s to t.

**Sample Input:**

4 5

1 2 100

1 3 100

2 3 1

2 4 100

3 4 100

1 4

**Sample Output:**

200

- **Procedure/Program:**

```
#include <stdio.h>
#include <string.h>
#include <limits.h>

#define MAX_NODES 100

int capacity[MAX_NODES][MAX_NODES];
int flow[MAX_NODES][MAX_NODES];
int parent[MAX_NODES];
int n, m;

int bfs(int s, int t) {
   memset(parent, -1, sizeof(parent));
   parent[s] = -2;
   int queue[MAX_NODES], front = 0, back = 0;
   queue[back++] = s;
```

```
    while (front < back) {
      int current = queue[front++];
      for (int next = 1; next <= n; next++) {
        if (parent[next] == -1 && capacity[current][next] > flow[current][next]) {
          parent[next] = current;
          if (next == t) return 1;
          queue[back++] = next;
        }
      }
    }
    return 0;
}

int fordFulkerson(int s, int t) {
    int maxFlow = 0;

    while (bfs(s, t)) {
      int pathFlow = INT_MAX;
      for (int v = t; v != s; v = parent[v]) {
        int u = parent[v];
        pathFlow = pathFlow < (capacity[u][v] - flow[u][v]) ? pathFlow : (capacity[u][v] -
flow[u][v]);
      }

      for (int v = t; v != s; v = parent[v]) {
        int u = parent[v];
        flow[u][v] += pathFlow;
        flow[v][u] -= pathFlow;
      }
      maxFlow += pathFlow;
    }
    return maxFlow;
}
```

```
 int main() {
  n = 4; m = 5;
  capacity[1][2] = 100;
  capacity[1][3] = 100;
  capacity[2][3] = 1;
  capacity[2][4] = 100;
  capacity[3][4] = 100;

  int s = 1, t = 4;
  printf("%d\n", fordFulkerson(s, t));
  return 0;
}
```

- **Data and Results:**

## Data

Graph with 4 nodes, 5 edges, and given capacities.

## Result

Maximum flow from source to sink is calculated as 200.

- **Analysis and Inferences:**

## Analysis

Ford-Fulkerson algorithm finds augmenting paths using BFS traversal.

## Inferences

Graph flow increases with available paths and higher capacity edges.

**In-Lab:**

1. Consider a network consisting of n computers and m connections. Each connection specifies how fast a computer can send data to another computer. Kotivalo wants to download some data from a server. What is the maximum speed he can do this, using the connections in the network?

**Input Format:**

- The first input line has two integers n and m: the number of computers and connections. The computers are numbered 1,2,…,n. Computer 1 is the server and computer n is Kotivalo's computer.
- After this, there are m lines describing the connections. Each line has three integers a, b and c: computer a can send data to computer b at speed c.

**Output Format:**

- Print one integer: the maximum speed Kotivalo can download data.

**Constraints :**

- $1 \leq n \leq 500$
- $1 \leq m \leq 1000$
- $1 \leq a,b \leq n$
- $1 \leq c \leq 109$

**Example:**

**Input:**

4 5

1 2 3

2 4 2

1 3 4

3 4 5

4 1 3

**Output:**

6

- **Procedure/Program**:

```
#include <stdio.h>

#include <string.h>

#include <limits.h>


#define MAX_N 501


int maxFlow[MAX_N][MAX_N], parent[MAX_N];

int n = 4, m = 5;


int bfs(int source, int sink) {

    int visited[MAX_N] = {0}, queue[MAX_N], front = 0, back = 0;

    queue[back++] = source;

    visited[source] = 1;

    parent[source] = -1;


    while (front < back) {

        int u = queue[front++];

        for (int v = 1; v <= n; v++) {

            if (!visited[v] && maxFlow[u][v] > 0) {
```

```
            queue[back++] = v;
            visited[v] = 1;
            parent[v] = u;
            if (v == sink) return 1;
        }
      }
    }
    return 0;
}
int edmondsKarp(int source, int sink) {
    int totalFlow = 0;
    while (bfs(source, sink)) {
        int pathFlow = INT_MAX;
        for (int v = sink; v != source; v = parent[v]) {
            int u = parent[v];
            if (maxFlow[u][v] < pathFlow) pathFlow = maxFlow[u][v];
        }
        for (int v = sink; v != source; v = parent[v]) {
            int u = parent[v];
            maxFlow[u][v] -= pathFlow;
            maxFlow[v][u] += pathFlow;
```

```
      }
      totalFlow += pathFlow;
   }
   return totalFlow;
}
int main() {
   memset(maxFlow, 0, sizeof(maxFlow));
   int edges[][3] = {
      {1, 2, 3},
      {2, 4, 2},
      {1, 3, 4},
      {3, 4, 5},
      {4, 1, 3}
   };
     for (int i = 0; i < m; i++) {
      int a = edges[i][0], b = edges[i][1], c = edges[i][2];
      maxFlow[a][b] += c;
   }
   printf("%d\n", edmondsKarp(1, n));
   return 0;
}
```

- **Data and Results:**

## Data

Graph with 4 nodes, 5 edges, and assigned capacities provided.

## Result

Maximum flow from source to sink is calculated as 6.

- **Analysis and Inferences:**

## Analysis

Algorithm applies Edmonds-Karp method using BFS for augmenting paths.

## Inferences

Flow network optimization helps determine maximum capacity between nodes efficiently.

2. A game consists of n rooms and m teleporters. At the beginning of each day, you start in room 1 and you have to reach room n. You can use each teleporter at most once during the game. How many days can you play if you choose your routes optimally?

**Input Format:**

- The first input line has two integers n and m: the number of rooms and teleporters. The rooms are numbered 1,2,…,n.
- After this, there are m lines describing the teleporters. Each line has two integers a and b: there is a teleporter from room a to room b.
- There are no two teleporters whose starting and ending room are the same.

**Output Format:**

First print an integer k: the maximum number of days you can play the game. Then, print k route descriptions according to the example. You can print any valid solution.

**Constraints :**

- $2 \le n \le 500$
- $1 \le m \le 1000$
- $1 \le a,b \le n$

**Example:**

**Input:**

6 7

1 2

1 3

2 6

3 4

3 5

4 6

5 6

**Output:**

2

3

1 2 6

4

1 3 4 6

- **Procedure/Program**:

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

#define MAX_N 500
#define MAX_M 1000

int n = 6, m = 7;
int adj[MAX_N + 1][MAX_N + 1];
int path[MAX_N], path_size;
int routes[MAX_M][MAX_N], route_sizes[MAX_M], route_count = 0;



int input[][2] = {
   {1, 2}, {1, 3}, {2, 6}, {3, 4}, {3, 5}, {4, 6}, {5, 6}
};

bool find_path(int node) {
   if (node == n) {
      memcpy(routes[route_count], path, path_size * sizeof(int));
      route_sizes[route_count++] = path_size;
      return true;
   }
```

```
    for (int i = 1; i <= n; i++) {
      if (adj[node][i]) {
        adj[node][i] = 0;
        path[path_size++] = i;
        if (find_path(i)) return true;
        path_size--;
        adj[node][i] = 1;
      }
    }
    return false;
}

int main() {
    memset(adj, 0, sizeof(adj));

    for (int i = 0; i < m; i++) {
      adj[input[i][0]][input[i][1]] = 1;
    }

    while (1) {
      path_size = 1;
      path[0] = 1;
      if (!find_path(1)) break;
    }




    printf("%d\n", route_count);
    for (int i = 0; i < route_count; i++) {
      printf("%d\n", route_sizes[i]);
      for (int j = 0; j < route_sizes[i]; j++) {
        printf("%d%c", routes[i][j], j == route_sizes[i] - 1 ? '\n' : ' ');
      }
    }
    return 0;
}
```

- **Data and Results:**

**Data:**

The game has rooms, teleporters, and a goal to reach.

**Result:**

Optimal routes maximize gameplay days by selecting unique paths.

- **Analysis and Inferences:**

**Analysis:**

Graph traversal finds distinct paths from room one to n.

**Inferences:**

More teleporters increase possible routes and gameplay longevity.

**Post-Lab:**

There are n boys and m girls in a school. Next week a school dance will be organized. A dance pair consists of a boy and a girl, and there are k potential pairs. Your task is to find out the maximum number of dance pairs and show how this number can be achieved.

**Input Format**:

- The first input line has three integers n, m and k: the number of boys, girls, and potential pairs. The boys are numbered 1,2,….,n, and the girls are numbered 1,2,…,m.
- After this, there are k lines describing the potential pairs. Each line has two integers a and b: boy a and girl b are willing to dance together.

**Output Format:**

- First print one integer r: the maximum number of dance pairs. After this, print r lines describing the pairs. You can print any valid solution.

**Constraints:**

- $1 \leq n,m \leq 500$
- $1 \leq k \leq 1000$
- $1 \leq a \leq n$
- $1 \leq b \leq m$

**Sample Input:**

3 2 4
1 1
1 2
2 1
3 1

**Sample Output:**

2
1 2
3 1

- **Procedure/Program**:

```
#include <stdio.h>
#include <string.h>

#define MAX_N 500
#define MAX_M 500

int graph[MAX_N + 1][MAX_M + 1];
int paired[MAX_M + 1];
int visited[MAX_M + 1];

int can_match(int boy, int m) {
   for (int girl = 1; girl <= m; girl++) {
      if (graph[boy][girl] && !visited[girl]) {
         visited[girl] = 1;
         if (paired[girl] == -1 || can_match(paired[girl], m)) {
            paired[girl] = boy;
            return 1;
         }
      }
   }
   return 0;
}

int main() {
   int n = 3, m = 2, k = 4;
   int input[][2] = {{1, 1}, {1, 2}, {2, 1}, {3, 1}};

   memset(graph, 0, sizeof(graph));
   memset(paired, -1, sizeof(paired));

   for (int i = 0; i < k; i++) {
      int a = input[i][0], b = input[i][1];
      graph[a][b] = 1;
   }
```

```
    int max_pairs = 0;
    for (int boy = 1; boy <= n; boy++) {
        memset(visited, 0, sizeof(visited));
        if (can_match(boy, m)) max_pairs++;
    }

    printf("%d\n", max_pairs);
    for (int girl = 1; girl <= m; girl++) {
        if (paired[girl] != -1) {
            printf("%d %d\n", paired[girl], girl);
        }
    }

    return 0;
}
```

- **Data and Results:**

**Data**

The dataset contains boys, girls, and their potential dance pairs.

**Result**

The maximum number of dance pairs and their valid pairings.

- **Analysis and Inferences :**

**Analysis**

A bipartite matching approach ensures optimal boy-girl dance pairings.

**Inferences**

Matching efficiency depends on available pairs and compatibility constraints.

- **Sample VIVA-VOCE Questions (In-Lab):**

1. How do you construct the minimum cut from the final residual graph after finding the maximum flow??

Nodes reachable from the source form one set; edges crossing to unreachable nodes form the cut.

2. What is the time complexity of the Ford-Fulkerson method, and on what factors does it depend?

$O(E \cdot \text{max flow})$, depends on edge capacities and augmenting paths.

3. Describe the significance of the bottleneck capacity in an augmenting path.

Limits maximum flow increase per iteration.

4. How does the Ford-Fulkerson method ensure that flow conservation and capacity constraints are maintained?

Maintains inflow = outflow at nodes, updates respect capacities.

5. Compare and contrast the space complexity of the Edmonds-Karp algorithm with the standard Ford-Fulkerson implementation.

Both $O(V + E)$; Edmonds-Karp uses BFS, Ford-Fulkerson may use DFS.

| | |
|---|---|
| **Evaluator Remark (if Any):** | **Marks Secured ___out of 50** |
| | **Signature of the Evaluator with Date** |

**Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment.**