**Practical-7**: Implementation of operations on union-find data structures.

#include <iostream.h>

class Edge

{

public:

int src, dest;

};

class Graph

{

public:

int V, E;

Edge\* edge;

};

Graph\* createGraph(int V, int E)

{

Graph\* graph = new Graph();

graph->V = V;

graph->E = E;

graph->edge = new Edge[graph->E \* sizeof(Edge)];

return graph;

}

int find(int parent[], int i)

{

if (parent[i] == -1)

return i;

return find(parent, parent[i]);

}

void Union(int parent[], int x, int y)

{

int xset = find(parent, x);

int yset = find(parent, y);

if(xset != yset)

{

parent[xset] = yset;

}

}

int isCycle( Graph\* graph )

{

int \*parent = new int[graph->V \* sizeof(int)];

memset(parent, -1, sizeof(int) \* graph->V);

for(int i = 0; i < graph->E; ++i)

{

int x = find(parent, graph->edge[i].src);

int y = find(parent, graph->edge[i].dest);

if (x == y)

return 1;

Union(parent, x, y);

}

return 0;

}

int main()

{

int V = 3, E = 3;

Graph\* graph = createGraph(V, E);

graph->edge[0].src = 0;

graph->edge[0].dest = 1;

graph->edge[1].src = 1;

graph->edge[1].dest = 2;

graph->edge[2].src = 0;

graph->edge[2].dest = 2;

if (isCycle(graph))

cout<<"graph contains cycle";

else

cout<<"graph doesn't contain cycle";

return 0;

}

**Output:**

****