Q. 1. Implement circular queue using arrays.

/\*Implement circular queue using arrays. \*/

#include <iostream>

using namespace std;

class CircularQueue {

int \*queue, size, front, rear;

public:

CircularQueue(int s) {

size = s;

queue = new int[size];

front = rear = -1;

}

void enqueue(int x);

int dequeue();

void display();

};

void CircularQueue::enqueue(int x) {

if ((front == 0 && rear == size - 1) || (front == rear + 1)) {

cout << "Queue is full\n";

return;

}

else if (front == -1) {

front = rear = 0;

}

else if (rear == size - 1 && front != 0) {

rear = 0;

}

else {

rear++;

}

queue[rear] = x;

}

int CircularQueue::dequeue() {

if (front == -1) {

cout << "Queue is empty\n";

return -1;

}

int x = queue[front];

if (front == rear) {

front = rear = -1;

}

else if (front == size - 1) {

front = 0;

}

else {

front++;

}

return x;

}

void CircularQueue::display() {

if (front == -1) {

cout << "Queue is empty\n";

return;

}

if (rear >= front) {

for (int i = front; i <= rear; i++)

cout << queue[i] << " ";

}

else {

for (int i = front; i < size; i++)

cout << queue[i] << " ";

for (int i = 0; i <= rear; i++)

cout << queue[i] << " ";

}

}

int main() {

CircularQueue q(5);

q.enqueue(1);

q.enqueue(2);

q.enqueue(3);

q.enqueue(4);

q.enqueue(5);

q.enqueue(6);

q.display();

cout << endl;

q.dequeue();

q.dequeue();

q.display();

cout << endl;

return 0;

}

Q. 2. Implement minimum cost spanning tree algorithm.

#include <iostream>

#include <vector>

#include <utility>

#include <algorithm>

using namespace std;

const int MAX = 1000;

int id[MAX], nodes, edges; //array id is use for check the parent of vertex;

pair <long long, pair<int, int> > p[MAX];

//initialise the parent array id[]

void init()

{

for(int i = 0;i < MAX;++i)

id[i] = i;

}

int root(int x)

{

while(id[x] != x) //if x is not itself parent then update its parent

{

id[x] = id[id[x]];

x = id[x];

}

return x; //return the parent

}

//function for union

void union1(int x, int y)

{

int p = root(x);

int q = root(y);

id[p] = id[q];

}

//function to find out the edges in minimum spanning tree and its cost

long long kruskal(pair<long long, pair<int, int> > p[])

{

int x, y;

long long cost, minimumCost = 0;

for(int i = 0;i < edges;++i)

{

x = p[i].second.first;

y = p[i].second.second;

cost = p[i].first;

if(root(x) != root(y))

{

minimumCost += cost;

cout<<x<<" ----> "<<y<<" :"<<p[i].first<<endl;//print the edges contain in

spanning tree

union1(x, y);

}

}

return minimumCost;

}

int main()

{

int x, y;

long long weight, cost, minimumCost;

init();

cout <<"Enter Nodes and edges"<<endl;

cin >> nodes >> edges;

//enter the vertex and cost of edges

for(int i = 0;i < edges;++i)

{

cout<<"Enter the value of X, Y and edges"<<endl;

cin >> x >> y >> weight;

p[i] = make\_pair(weight, make\_pair(x, y));

}

//sort the edges according to their cost

sort(p, p + edges);

minimumCost = kruskal(p);

cout <<"Minimum cost is "<< minimumCost << endl;

return 0;

}