Q. 1. 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;

}

Q. 2. Write a menu driven program to perform following operations on singly linked list: Create,

reverse, search, count and Display

/\*Write a menu driven program to perform following

operations on singly linked list: Create, reverse,

search, count and Display.

\*/

#include <iostream>

using namespace std;

struct Node {

int data;

Node\* next;

};

class LinkedList {

private:

Node\* head;

int count;

public:

LinkedList() {

head = NULL;

count = 0;

}

void create() {

int data;

cout << "Enter the data for the node: ";

cin >> data;

Node\* newNode = new Node();

newNode->data = data;

newNode->next = head;

head = newNode;

count++;

}

void reverse() {

Node\* prev = NULL;

Node\* current = head;

Node\* next = NULL;

while (current != NULL) {

next = current->next;

current->next = prev;

prev = current;

current = next;

}

head = prev;

}

int search(int key) {

Node\* current = head;

int index = 0;

while (current != NULL) {

if (current->data == key) {

return index;

}

current = current->next;

index++;

}

return -1;

}

int countNodes() {

return count;

}

void display() {

Node\* current = head;

while (current != NULL) {

cout << current->data << " ";

current = current->next;

}

cout << endl;

}

};

int main() {

int choice;

LinkedList list;

while (true) {

cout << "1. Create Node" << endl;

cout << "2. Reverse List" << endl;

cout << "3. Search Element" << endl;

cout << "4. Count Nodes" << endl;

cout << "5. Display List" << endl;

cout << "6. Exit" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

list.create();

break;

case 2:

list.reverse();

break;

case 3: {

int key;

cout << "Enter the element to be searched: ";

cin >> key;

int index = list.search(key);

if (index == -1) {

cout << "Element not found." << endl;

} else {

cout << "Element found at index: " << index << endl;

}

break;

}

case 4:

cout << "Number of nodes: " << list.countNodes() << endl;

break;

case 5:

list.display();

break;

case 6:

return 0;

default:

cout << "Invalid choice. Please enter a valid choice." << endl;

}

}

return 0;

}