Q. 1. Implement Sequential and Binary Search

/\*Implement Sequential Search.\*/

#include <iostream>

using namespace std;

int sequentialSearch(int array[], int size, int key) {

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

if (array[i] == key) {

return i; // return the index of the key if found

}

}

return -1; // return -1 if key is not found

}

int main() {

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

int size = sizeof(array) / sizeof(array[0]);

int key = 3;

int index = sequentialSearch(array, size, key);

if (index != -1) {

cout << "Key found at index " << index << endl;

} else {

cout << "Key not found" << endl;

}

return 0;

}

/\*Implement Binary Search.\*/

#include <iostream>

using namespace std;

int binarySearch(int arr[], int n, int key) {

int left = 0, right = n - 1;

while (left <= right) {

int mid = (left + right) / 2;

if (arr[mid] == key) {

return mid;

}

else if (arr[mid] < key) {

left = mid + 1;

}

else {

right = mid - 1;

}

}

return -1;

}

int main() {

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

int n = sizeof(arr) / sizeof(arr[0]);

int key = 3;

int index = binarySearch(arr, n, key);

if (index != -1) {

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

}

else {

cout << "Element not found" << 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;

}