**TABLE OF CONTENT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.N | Details | Submission Date | Signature | Remarks |
| 14. | Write a program to sort the given array of numbers in ascending order using bubble sort | 2081-12-25 |  |  |
| 15. | Write a program for creating MST from Kruskal’s algorithm.. | 2081-12-25 |  |  |
| 16 | Write a program for creating MST from Prims algorithm.. | 2081-12-25 |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

****

**Lab no: 14 Date: 2081/12/25**

**Title: Write a program to sort the given array of numbers in ascending order using bubble sort.**

**Bubble Sort:**

Bubble Sort is a simple comparison-based sorting algorithm. It repeatedly steps through the list, compares adjacent elements, and swaps them if they are in the wrong order. This process is repeated until the entire list is sorted.

**Algorithm:**

1. Start from the first element of the array.
2. Compare each adjacent pair of elements.
3. Swap them if they are in the wrong order (i.e., if the left element is greater than the right).
4. Move to the next pair and repeat until the last element.
5. After each pass, the largest element moves to its correct position.
6. Repeat the process for the remaining unsorted elements until the array is sorted.

**Pseudocode:**

bubbleSort(array)

for i = 0 to n-2

for j = 0 to n-2-i

if array[j] > array[j+1]

swap(array[j], array[j+1])

**Compiler: Dev C++**

**Language: C**

****

**Lab no: 15 Date: 2081/12/25**

**Title: Write a program for creating MST from Kruskal’s algorithm..**

**Kruskal’s Algorithm:**

Kruskal’s Algorithm is a **greedy algorithm** used to find the **Minimum Spanning Tree (MST)** of a graph. It works by **sorting all edges** in ascending order of their weight and **adding edges** one by one, ensuring that **no cycles** are formed, until all vertices are connected.

**Algorithm:**

1. Sort all edges of the graph in non-decreasing order of their weight.
2. Initialize an empty MST and a disjoint set for all vertices.
3. For each edge in the sorted list:
   1. If the edge connects two different components (i.e., doesn’t form a cycle), add it to the MST.
   2. Use the Union-Find data structure to detect cycles.
4. Stop when the MST contains exactly **V – 1** edges (where V is the number of vertices).

**Pseudocode:**

kruskalMST(graph)

sort edges in ascending order by weight

initialize disjointSet for all vertices

mst = empty list

for each edge (u, v) in sorted edge list

if find(u) ≠ find(v)

add (u, v) to mst

union(u, v)

return mst

**Compiler: Dev C++**

**Language: C**

****

**Lab no: 16 Date: 2081/12/25**

**Title: Write a program for creating MST from Prim’s algorithm.**

**Prim’s Algorithm:**

Prim’s Algorithm is a **greedy algorithm** that builds the MST by starting from an arbitrary vertex and **growing the MST** one edge at a time by always choosing the **minimum weight edge** that connects a vertex in the MST to a vertex outside it.

**Algorithm:**

1. Start from any vertex and add it to the MST.
2. Use a priority queue (min-heap) to keep track of the minimum weight edge that connects to a new vertex.
3. At each step, pick the smallest edge that connects a vertex in the MST to a vertex outside it.
4. Repeat until all vertices are included in the MST.

**Pseudocode:**

primMST(graph, start)

initialize visited set

initialize minHeap with (0, start)

mst = empty list

while minHeap is not empty and MST has less than V-1 edges

(weight, u) = extract min from minHeap

if u not in visited

add u to visited

add edge to mst

for each neighbor v of u

if v not in visited

add (weight, v) to minHeap

return mst

**Compiler: Dev C++**

**Language: C**

**Source Code:**

#include <stdio.h>

void printArray(int arr[], int n)

{

int k;

for (k = 0; k < n; k++)

{

printf("%4d", arr[k]);

}

printf("\n");

}

void bubbleSort(int arr[], int n)

{

int i, j, temp, swapped;

printf("Pass | Swap | Array State\n");

printf("----------------------------\n");

for (i = 0; i < n - 1; i++)

{

swapped = 0;

for (j = 0; j < n - 1 - i; j++)

{

if (arr[j] > arr[j + 1])

{

temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

swapped = 1;

printf("%4d | %4d-%d | ", i + 1, arr[j], arr[j + 1]);

printArray(arr, n);

}

}

if (!swapped) break;

}

}

int main()

{

int arr[] = {64, 34, 25, 12, 22, 11, 90};

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

printf("Original Array:\n");

printArray(arr, n);

printf("\nBubble Sort Steps:\n");

bubbleSort(arr, n);

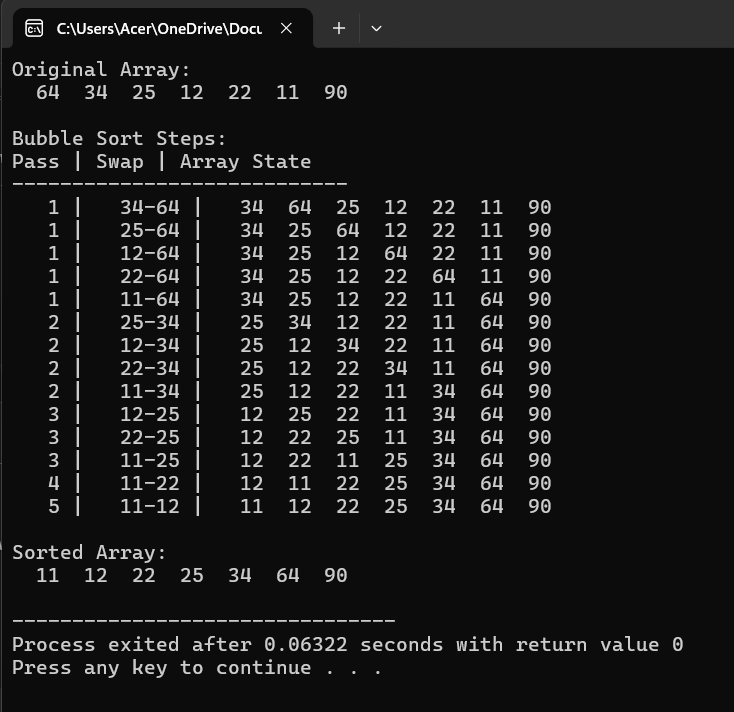
printf("\nSorted Array:\n");

printArray(arr, n);

return 0;

}

**Output:**



**Source Code:**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX 100

struct Edge {

int src, dest, weight;

char srcLabel, destLabel;

};

// Compare function for qsort

int compare(const void\* a, const void\* b) {

struct Edge\* e1 = (struct Edge\*)a;

struct Edge\* e2 = (struct Edge\*)b;

return e1->weight - e2->weight;

}

int find(int parent[], int i) {

if (parent[i] != i)

parent[i] = find(parent, parent[i]);

return parent[i];

}

void unionSets(int parent[], int rank[], int x, int y) {

int xroot = find(parent, x);

int yroot = find(parent, y);

if (rank[xroot] < rank[yroot])

parent[xroot] = yroot;

else if (rank[xroot] > rank[yroot])

parent[yroot] = xroot;

else {

parent[yroot] = xroot;

rank[xroot]++;

}

}

int getIndex(char label) {

return label - 'a';

}

int main() {

int V = 7; // Number of vertices a-g

int E = 12; // Number of edges (including eb)

struct Edge edges[] = {

{'a','b',5}, {'a','c',3}, {'e','f',3}, {'b','c',4},

{'f','g',4}, {'c','d',5}, {'e','g',5}, {'b','d',6},

{'d','e',6}, {'d','f',6}, {'c','f',6}, {'e','b',2} // NEW EDGE

};

printf("Original Graph (edges with weights):\n");

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

printf("%c-%c = %d\n", edges[i].src, edges[i].dest, edges[i].weight);

}

qsort(edges, E, sizeof(edges[0]), compare);

int parent[MAX];

int rank[MAX];

for (int v = 0; v < V; ++v) {

parent[v] = v;

rank[v] = 0;

}

struct Edge result[MAX];

int e = 0;

int i = 0;

int totalWeight = 0;

printf("\nStep-by-step process:\n");

while (e < V - 1 && i < E) {

struct Edge next\_edge = edges[i++];

int x = find(parent, getIndex(next\_edge.src));

int y = find(parent, getIndex(next\_edge.dest));

if (x != y) {

result[e++] = next\_edge;

totalWeight += next\_edge.weight;

unionSets(parent, rank, x, y);

printf("Accepted: %c-%c = %d\n", next\_edge.src, next\_edge.dest, next\_edge.weight);

} else {

printf("Rejected (cycle): %c-%c = %d\n", next\_edge.src, next\_edge.dest, next\_edge.weight);

}

}

printf("\nMinimum Spanning Tree (MST):\n");

for (i = 0; i < e; ++i)

printf("%c-%c = %d\n", result[i].src, result[i].dest, result[i].weight);

printf("Total Weight of MST = %d\n", totalWeight);

return 0;

}

**Output:**

****

**Source Code:**

#include <stdio.h>

#include <limits.h>

#include <stdbool.h>

#define V 5

int minKey(int key[], bool mstSet[]) {

int min = INT\_MAX, min\_index;

for (int v = 0; v < V; v++)

if (!mstSet[v] && key[v] < min)

min = key[v], min\_index = v;

return min\_index;

}

void printMST(int parent[], int graph[V][V], char labels[]) {

int totalWeight = 0;

printf("\nEdges in MST:\n");

for (int i = 1; i < V; i++) {

printf("%c - %c = %d\n", labels[parent[i]], labels[i], graph[i][parent[i]]);

totalWeight += graph[i][parent[i]];

}

printf("Total Weight of MST = %d\n", totalWeight);

}

void printGraph(int graph[V][V], char labels[]) {

printf("\nGraph Representation (Adjacency Matrix):\n ");

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

printf("%c ", labels[i]);

printf("\n");

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

printf("%c |", labels[i]);

for (int j = 0; j < V; j++) {

if (graph[i][j] != 0)

printf("%2d ", graph[i][j]);

else

printf(" . ");

}

printf("\n");

}

}

void primMST(int graph[V][V], char labels[]) {

int parent[V];

int key[V];

bool mstSet[V];

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

key[i] = INT\_MAX, mstSet[i] = false;

key[0] = 0;

parent[0] = -1;

printf("\nStep-by-step process:\n");

for (int count = 0; count < V - 1; count++) {

int u = minKey(key, mstSet);

mstSet[u] = true;

printf("\nSelected Vertex: %c\n", labels[u]);

for (int v = 0; v < V; v++) {

if (graph[u][v] && !mstSet[v]) {

if (graph[u][v] < key[v]) {

printf(" ? Accepting edge %c-%c (%d)\n", labels[u], labels[v], graph[u][v]);

parent[v] = u;

key[v] = graph[u][v];

} else {

printf(" ? Rejecting edge %c-%c (%d), better edge exists\n", labels[u], labels[v], graph[u][v]);

}

}

}

}

printMST(parent, graph, labels);

}

int main() {

char labels[] = {'a', 'b', 'c', 'd', 'e'};

int graph[V][V] = {

// a b c d e

{ 0, 5,15,20, 0}, // a

{ 5, 0,25, 0, 0}, // b

{15,25, 0,30,37}, // c

{20, 0,30, 0,35}, // d

{ 0, 0,37,35, 0} // e

};

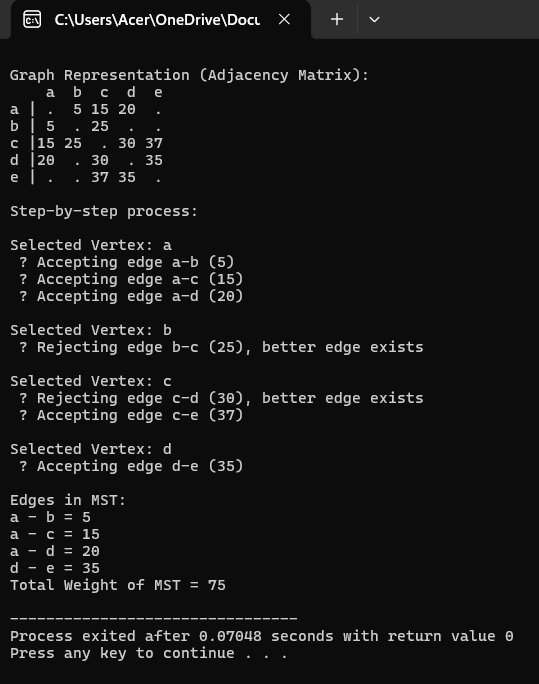
printGraph(graph, labels);

primMST(graph, labels);

return 0;

}

**Output:**

****