

**M.Sc. (Five Year Integrated) in Computer Science  
(Artificial Intelligence & Data Science)**

**Third Semester**

**Laboratory Record**

**23-813-0306: ALGORITHMS LAB**

*Submitted in partial fulfillment  
of the requirements for the award of degree in  
Master of Science (Five Year Integrated)  
in Computer Science (Artificial Intelligence & Data Science) of  
Cochin University of Science and Technology (CUSAT)  
Kochi*



*Submitted by*

**NANDAKISHORE V J  
(81323017)**

**DEPARTMENT OF COMPUTER SCIENCE  
COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY (CUSAT)  
KOCHI-682022**

**DECEMBER 2024**

**DEPARTMENT OF COMPUTER SCIENCE**  
**COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY (CUSAT)**  
**KOCHI, KERALA-682022**



*This is to certify that the software laboratory record for **23-813-0306: ALGORITHMS LAB** is a record of work carried out by **NANDAKISHORE V J(81323017)**, in partial fulfillment of the requirements for the award of degree in **Master of Science (Five Year Integrated) in Computer Science (Artificial Intelligence & Data Science)** of Cochin University of Science and Technology (CUSAT), Kochi. The lab record has been approved as it satisfies the academic requirements in respect of the second semester laboratory prescribed for the Master of Science (Five Year Integrated) in Computer Science degree.*

**Faculty Member in-charge**

Dr. Jereesh A S  
Assistant Professor  
Department of Computer Science  
CUSAT

Dr. Madhu S Nair  
Professor and Head  
Department of Computer Science  
CUSAT

# Contents

Sl.No	Title	Page no
1.	Linear search	1
2.	Binary Search	3
3.	Bubble Sort	5
4.	Insertion Sort	7
5.	Selection Sort	9
6.	Merge Sort	11
7.	Quick Sort	14
8.	Heap Sort	16
9.	Graph (BFS & DFS traversal)	19
10.	Prim's algorithm	23
11.	Kruskal's algorithm	26
12.	Dijkstra's algorithm	30
13.	Bellman Ford algorithm	33
14.	Floyd Warshall Algorithm	36
15.	Dynamic Fibonacci Series	39
16.	Coin Row problem	40
17.	Coin Change Making Problem	42
18.	Coin Collecting Problem	45
19.	Minimum Cost Path of a Matrix	48
20.	0/1 Knapsack Problem	51
21.	Longest Common Subsequence	54

## 1.LINEAR SEARCH

**DATE:**

### ALGORITHM

```
1. FUNCTION LinearSearch(arr, target, index)
2.     IF index >= LENGTH(arr) THEN
3.         RETURN -1 // Target not found
4.     END IF
5.
6.     IF arr[index] == target THEN
7.         RETURN index // Target found
8.     END IF
9.
10.    RETURN LinearSearch(arr, target, index + 1)
11. END FUNCTION
```

### PROGRAM

```
#include<iostream>
using namespace std;

int Linear_Search(int arr[], int target, int index, int size) {
    if (index >= size) {
        return -1; // Target not found
    }

    if (arr[index] == target) {
        return index; // Target found
    }

    return Linear_Search(arr, target, index + 1, size);
}

int main() {
    int n;
    cout << "Enter the number of elements you want to enter: ";
    cin >> n;
```

```
int arr[n];

cout << "Enter the elements one by one:" << endl;
for (int i = 0; i < n; i++) {
    cin >> arr[i];
}

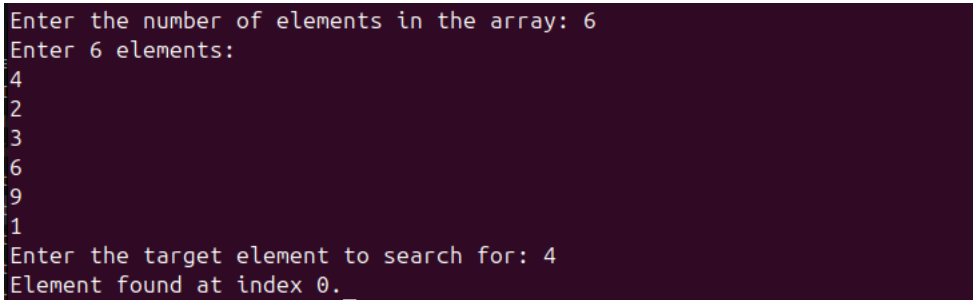
int el;
cout << "Enter the element to search: ";
cin >> el;

int result = Linear_Search(arr, el, 0, n);

if (result != -1) {
    cout << "Element found at index: " << result << endl;
} else {
    cout << "Element not found..." << endl;
}

return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
Enter the number of elements in the array: 6
Enter 6 elements:
4
2
3
6
9
1
Enter the target element to search for: 4
Element found at index 0.
```

## 2. BINARY SEARCH

**DATE:**

### ALGORITHM

```
1. function binarySearch(arr, LB, UB, key)
2. while LB <= UB
3. mid = LB + (UB - LB) / 2
4. if arr[mid] == key
5. return mid
6. else if arr[mid] < key
7. LB = mid + 1
8. else
9. UB = mid - 1
10. end while
11. return -1 // key not found
12. end
```

### PROGRAM

```
#include<iostream>
using namespace std;

void Binary_search(int arr[], int LB, int UB, int key) {
    while (LB <= UB) {
        int mid = LB + (UB - LB) / 2;

        if (arr[mid] == key) {
            cout << "Element found at index: " << mid << endl;
            return;
        } else if (arr[mid] < key) {
            LB = mid + 1;
        } else {
            UB = mid - 1;
        }
    }
    cout << "Element not found" << endl;
}

int main() {
    int n;
```

```
    cout << "Enter the number of elements you want to enter: ";
    cin >> n;

    int arr[n];

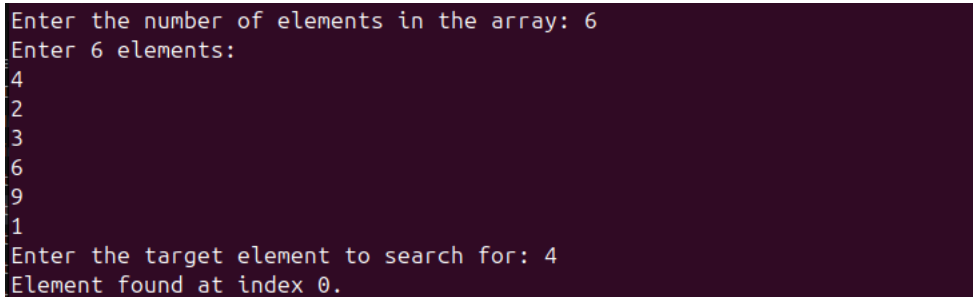
    cout << "Enter the elements one by one in sorted order:" << endl;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    int el;
    cout << "Enter the element to search: ";
    cin >> el;

    Binary_search(arr, 0, n - 1, el);

    return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
Enter the number of elements in the array: 6
Enter 6 elements:
4
2
3
6
9
1
Enter the target element to search for: 4
Element found at index 0.
```

### 3.BUBBLE SORT

**DATE:**

**ALGORITHM**

```
1.  FUNCTION bubbleSort(arr, LB, UB)
2.      n = UB - LB + 1
3.      FOR i = 0 TO n - 1
4.          FOR j = 0 TO n - i - 1
5.              IF arr[LB + j] > arr[LB + j + 1] THEN
6.                  SWAP arr[LB + j] AND arr[LB + j + 1]
7.              END IF
8.          END FOR
9.      END FOR
10. END FUNCTION
```

**PROGRAM**

```
#include<iostream>
using namespace std;

void Bubble_Sort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                swap(arr[j], arr[j + 1]);
            }
        }
    }
}

int main() {
    int n;
    cout << "Enter the number of elements you want to enter: ";
    cin >> n;

    int arr[n];
```



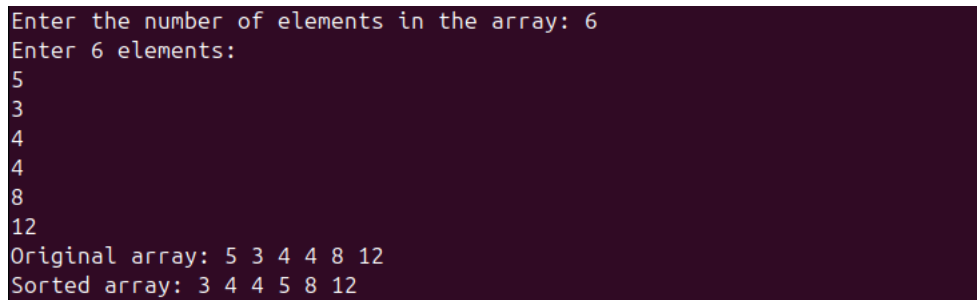
```
    cout << "Enter the elements one by one:" << endl;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    Bubble_Sort(arr, n);
    cout << "SORTED ARRAY \n";

    for (int i = 0; i < n; i++) {
        cout << arr[i] << ' ';
    }
    cout << endl;

    return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
Enter the number of elements in the array: 6
Enter 6 elements:
5
3
4
4
8
12
Original array: 5 3 4 4 8 12
Sorted array: 3 4 4 5 8 12
```

## 4.INSERTION SORT

**DATE:**

### ALGORITHM

```
1.  FUNCTION insertionSort(arr, LB, UB)
2.      FOR i = LB + 1 TO UB
3.          key = arr[i]
4.          j = i - 1
5.          WHILE j >= LB AND arr[j] > key
6.              arr[j + 1] = arr[j]
7.              j = j - 1
8.          END WHILE
9.          arr[j + 1] = key
10.     END FOR
11. END FUNCTION
```

### PROGRAM

```
#include <iostream>
using namespace std;

void insertion_Sort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int temp = arr[i];
        int j = i - 1;

        while (j >= 0 && arr[j] > temp) {
            arr[j + 1] = arr[j];
            j--;
        }

        arr[j + 1] = temp;
    }
}

int main() {
    int n;
    cout << "Enter the number of elements you want to enter: ";
```

```
    cin >> n;

    int arr[n];

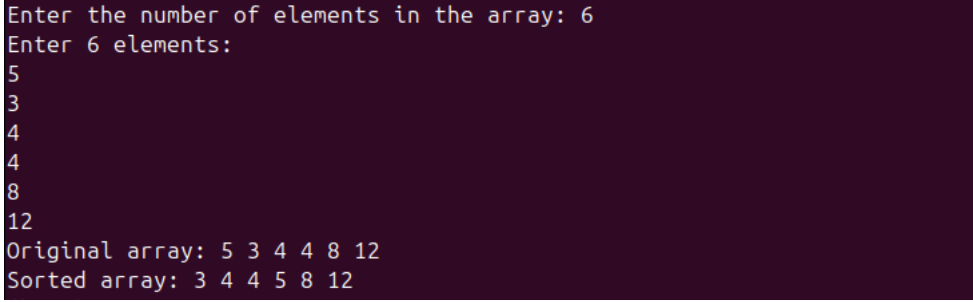
    cout << "Enter the elements one by one:" << endl;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    insertion_Sort(arr, n);
    cout << "SORTED ARRAY \n";

    for (int i = 0; i < n; i++) {
        cout << arr[i] << ' ';
    }
    cout << endl;

    return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
Enter the number of elements in the array: 6
Enter 6 elements:
5
3
4
4
8
12
Original array: 5 3 4 4 8 12
Sorted array: 3 4 4 5 8 12
```

## 5.SELECTION SORT

**DATE:**

### ALGORITHM

```
1.  FUNCTION selectionSort(arr, LB, UB)
2.      FOR i = LB TO UB - 1
3.          minIndex = i
4.          FOR j = i + 1 TO UB
5.              IF arr[j] < arr[minIndex] THEN
6.                  minIndex = j
7.              END IF
8.          END FOR
9.          IF minIndex != i THEN
10.             SWAP arr[i] AND arr[minIndex]
11.          END IF
12.      END FOR
13. END FUNCTION
```

### PROGRAM

```
#include <iostream>
using namespace std;

void Selection_Sort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int MinIndex = i;
        for (int j = i + 1; j < n; j++) {
            if (arr[j] < arr[MinIndex]) {
                MinIndex = j;
            }
        }
        swap(arr[i], arr[MinIndex]);
    }
}

int main() {
    int n;
    cout << "Enter the number of elements you want to enter: ";
    cin >> n;
```

```
int arr[n];

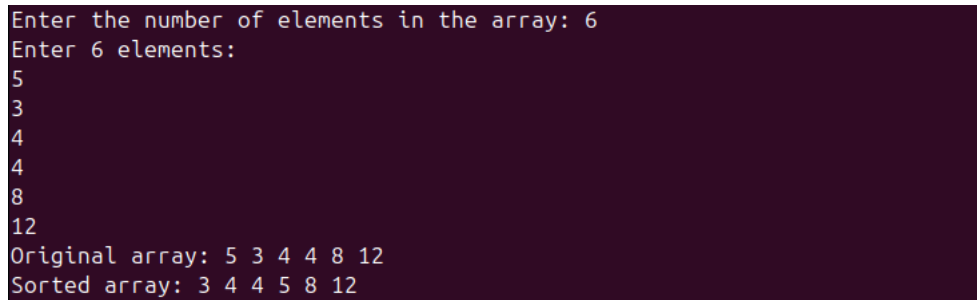
cout << "Enter the elements one by one:" << endl;
for (int i = 0; i < n; i++) {
    cin >> arr[i];
}

Selection_Sort(arr, n);
cout << "SORTED ARRAY \n";

for (int i = 0; i < n; i++) {
    cout << arr[i] << ' ';
}
cout << endl;

return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
Enter the number of elements in the array: 6
Enter 6 elements:
5
3
4
4
8
12
Original array: 5 3 4 4 8 12
Sorted array: 3 4 4 5 8 12
```

## 6.MERGE SORT

**DATE:**

### ALGORITHM

```
1.  FUNCTION mergeSort(arr, LB, UB)
2.      IF LB < UB THEN
3.          mid = LB + (UB - LB) / 2
4.          mergeSort(arr, LB, mid)
5.          mergeSort(arr, mid + 1, UB)
6.          merge(arr, LB, mid, UB)
7.      END IF
8.  END FUNCTION
9.
10. FUNCTION merge(arr, LB, mid, UB)
11.     n1 = mid - LB + 1
12.     n2 = UB - mid
13.
14.     CREATE leftArr[n1 + 1]
15.     CREATE rightArr[n2 + 1]
16.
17.     FOR i = 0 TO n1 - 1
18.         leftArr[i] = arr[LB + i]
19.     END FOR
20.     FOR j = 0 TO n2 - 1
21.         rightArr[j] = arr[mid + 1 + j]
22.     END FOR
23.
24.     leftArr[n1] = INFINITY
25.     rightArr[n2] = INFINITY
26.     i=0 , j=0 , k= LB
27.     FOR k = LB TO UB
28.         IF leftArr[i] <= rightArr[j] THEN
29.             arr[k] = leftArr[i]
30.             i = i + 1
31.         ELSE
32.             arr[k] = rightArr[j]
33.             j = j + 1
34.         END IF
35.     END FOR
36. END FUNCTION
```

**PROGRAM**

```
#include <iostream>
using namespace std;

void Merge(int arr[], int left, int mid, int right) {
    int n1 = mid - left + 1;
    int n2 = right - mid;

    int L[n1], R[n2];
    for (int i = 0; i < n1; i++) {
        L[i] = arr[left + i];
    }
    for (int j = 0; j < n2; j++) {
        R[j] = arr[mid + 1 + j];
    }
    int i = 0, j = 0, k = left;
    while (i < n1 && j < n2) {
        if (L[i] <= R[j]) {
            arr[k] = L[i];
            i++;
        } else {
            arr[k] = R[j];
            j++;
        }
        k++;
    }
    while (i < n1) {
        arr[k] = L[i];
        i++;
        k++;
    }
    while (j < n2) {
        arr[k] = R[j];
        j++;
        k++;
    }
}

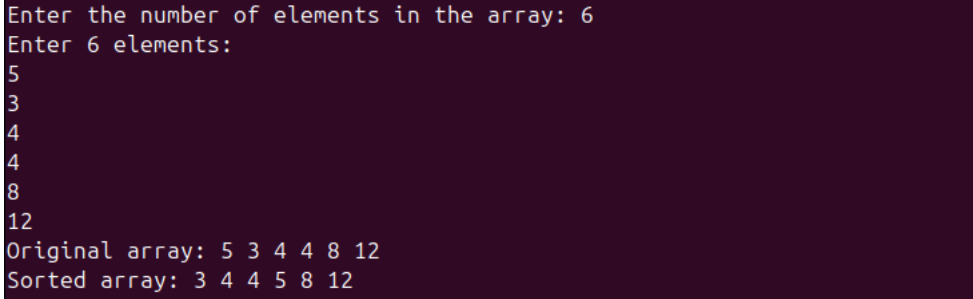
void Merge_sort(int arr[], int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        Merge_sort(arr, left, mid);
        Merge_sort(arr, mid + 1, right);
    }
}
```

```
        Merge(arr, left, mid, right);
    }}

int main() {
    int n;
    cout << "Enter the number of elements you want to enter: ";
    cin >> n;
    int arr[n];
    cout << "Enter the elements one by one:" << endl;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    Merge_sort(arr, 0, n - 1);
    cout << "SORTED ARRAY \n";

    for (int i = 0; i < n; i++) {
        cout << arr[i] << ' ';
    }
    cout << endl;
    return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
Enter the number of elements in the array: 6
Enter 6 elements:
5
3
4
4
8
12
Original array: 5 3 4 4 8 12
Sorted array: 3 4 4 5 8 12
```



## 7. QUICK SORT

**DATE:**

### ALGORITHM

```
1. FUNCTION quickSort(arr, LB, UB)
2.     IF LB < UB THEN
3.         pivotIndex = partition(arr, LB, UB)
4.         quickSort(arr, LB, pivotIndex - 1)
5.         quickSort(arr, pivotIndex + 1, UB)
6.     END IF
7. END FUNCTION
8.
9. FUNCTION partition(arr, LB, UB)
10.    pivot = arr[UB]
11.    i = LB - 1
12.
13.    FOR j = LB TO UB - 1
14.        IF arr[j] <= pivot THEN
15.            i = i + 1
16.            SWAP arr[i] AND arr[j]
17.        END IF
18.    END FOR
19.
20.    SWAP arr[i + 1] AND arr[UB]
21.    RETURN i + 1
22. END FUNCTION
```

### PROGRAM

```
#include <iostream>
using namespace std;

int Partition(int arr[], int low, int high) {
    int PivotVal = arr[high];
    int i = low - 1;

    for (int j = low; j <= high - 1; j++) {
        if (arr[j] <= PivotVal) {
            i++;
        }
    }
}
```

```
        swap(arr[i], arr[j]);
    }
}

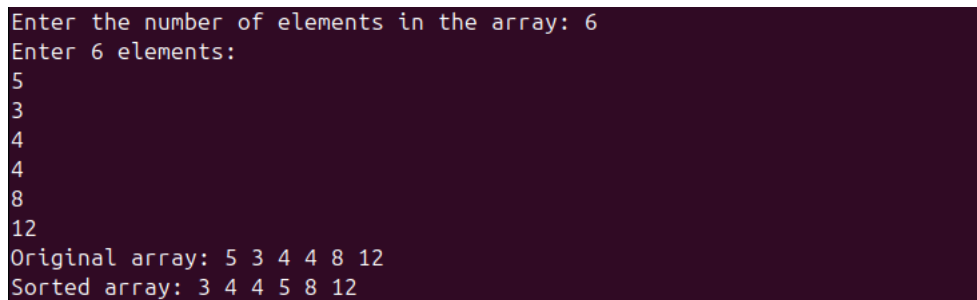
swap(arr[i + 1], arr[high]);
return (i + 1);
}

void Quick_sort(int arr[], int low, int high) {
    if (low < high) {
        int PivotIndex = Partition(arr, low, high);
        Quick_sort(arr, low, PivotIndex - 1);
        Quick_sort(arr, PivotIndex + 1, high);
    }
}

int main() {
    int n;
    cout << "Enter the number of elements you want to enter: ";
    cin >> n;

    int arr[n];
    cout << "Enter the elements one by one:" << endl;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }
    Quick_sort(arr, 0, n - 1);
    cout << "SORTED ARRAY \n";
    for (int i = 0; i < n; i++) {
        cout << arr[i] << ' ';
    }
    cout << endl;
    return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
Enter the number of elements in the array: 6
Enter 6 elements:
5
3
4
4
8
12
Original array: 5 3 4 4 8 12
Sorted array: 3 4 4 5 8 12
```

## 8. HEAP SORT

**DATE:**

### ALGORITHM

```
1.  FUNCTION heapSort(arr, LB, UB)
2.      n = UB - LB + 1
3.      buildHeap(arr, n, LB)
4.
5.      FOR i = UB TO LB + 1 STEP -1
6.          SWAP arr[LB] AND arr[i]
7.          heapify(arr, i - LB, LB)
8.      END FOR
9.  END FUNCTION
10.
11. FUNCTION buildHeap(arr, n, LB)
12.     FOR i = LB + (n // 2) - 1 TO LB STEP -1
13.         heapify(arr, n, i)
14.     END FOR
15. END FUNCTION
16.
17. FUNCTION heapify(arr, n, i)
18.     largest = i
19.     left = 2 * i + 1
20.     right = 2 * i + 2
21.
22.     IF left < n AND arr[left] > arr[largest] THEN
23.         largest = left
24.     END IF
25.
26.     IF right < n AND arr[right] > arr[largest] THEN
27.         largest = right
28.     END IF
29.
30.     IF largest != i THEN
31.         SWAP arr[i] AND arr[largest]
32.         heapify(arr, n, largest)
33.     END IF
34. END FUNCTION
```

### PROGRAM

```
#include <iostream>
using namespace std;

void Heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && arr[left] > arr[largest]) {
        largest = left;
    }

    if (right < n && arr[right] > arr[largest]) {
        largest = right;
    }

    if (largest != i) {
        swap(arr[i], arr[largest]);
        Heapify(arr, n, largest);
    }
}

void BuildHeap(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--) {
        Heapify(arr, n, i);
    }
}

void Heapsort(int arr[], int n) {
    BuildHeap(arr, n);

    for (int i = n - 1; i > 0; i--) {
        swap(arr[0], arr[i]);
        Heapify(arr, i, 0);
    }
}

int main() {
    int n;
    cout << "Enter the number of elements you want to enter: ";
    cin >> n;
```

```
int arr[n];

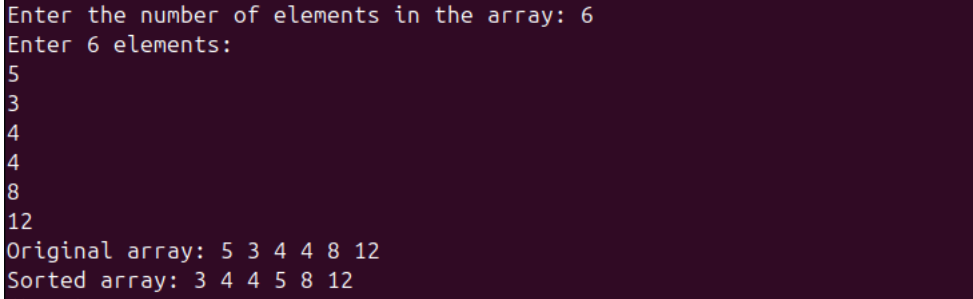
cout << "Enter the elements one by one:" << endl;
for (int i = 0; i < n; i++) {
    cin >> arr[i];
}

Heapsort(arr, n);
cout << "SORTED ARRAY \n";

for (int i = 0; i < n; i++) {
    cout << arr[i] << ' ';
}
cout << endl;

return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
Enter the number of elements in the array: 6
Enter 6 elements:
5
3
4
4
8
12
Original array: 5 3 4 4 8 12
Sorted array: 3 4 4 5 8 12
```

## 9. GRAPH BFS & DFS TRAVERSAL

DATE:

### ALGORITHM

```
1. FUNCTION BFS(matrix, startvertex)
2.     Create an array visited of size V, initialized to false
3.     Create an empty queue
4.     visited[startvertex] = true
5.     Enqueue startvertex into queue
6.     WHILE queue is not empty
7.         vertex = Dequeue from queue
8.         PRINT vertex
9.         FOR each i from 0 to V-1
10.            IF matrix[vertex][i] == 1 AND visited[i] == false
11.                visited[i] = true
12.                Enqueue i into queue
13.            END IF
14.        END FOR
15.    END WHILE
16. END FUNCTION
```

```
1. FUNCTION DFSu(matrix, vertex, visited) //dfs helper function
2.     visited[vertex] = true
3.     PRINT vertex
4.     FOR each i from 0 to V-1
5.         IF matrix[vertex][i] == 1 AND visited[i] == false
6.             CALL DFSu(matrix, i, visited)
7.         END IF
8.     END FOR
9. END FUNCTION
```

```
1. FUNCTION DFS(matrix, startvertex)
2.     Create an array visited of size V, initialized to false
3.     CALL DFSu(matrix, startvertex, visited)
4. END FUNCTION
```

**PROGRAM**

```
#include <iostream>
#include <vector>
#include <queue>
using namespace std;

// Add Edge as edges are undirectional
void addEdge(vector<vector<int>> &mat, int i ,int j){
    mat[i][j]=1;
    mat[j][i]=1;
}

void displayMatrix(vector<vector<int>> &mat){
    int V=mat.size();
    for (int i = 0; i < V; i++)
    {
        for (int j = 0; j < V; j++)
        {
            cout<<mat[i][j]<<" ";
        }
        cout<<endl;
    }
}

void BFS(vector<vector<int>> &mat,int startvertex){

    int V=mat.size();
    vector<bool> visited(V,false);
    queue<int> q;

    visited[startvertex]=true;
    q.push(startvertex);

    while(!q.empty()){
        int vertex=q.front();
        q.pop();

        cout<<vertex<<" ";

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

```
        {
            if(mat[vertex][i]==1 && !visited[i]){
                visited[i]=true;
                q.push(i);
            }
        }
    }

void DFSu(vector<vector<int>> &mat, int vertex, vector<bool> &visited)
{
    visited[vertex]=true;
    cout<<vertex<<" ";
    for (int i = 0; i < mat.size(); i++)
    {
        if(mat[vertex][i]==1 && !visited[i]){
            DFSu(mat,i,visited);
        }
    }
}

void DFS(vector<vector<int>> &mat, int startvertex){
    int V =mat.size();
    vector<bool> visited(V,false);
    DFSu(mat,startvertex,visited);
}

int main(){

    int V, E;

    cout << "Enter the number of vertices: ";
    cin >> V;

    vector<vector<int>> mat(V, vector<int>(V, 0));

    cout << "Enter the number of edges: ";
    cin >> E;

    // Input each edge and add it to the graph
    for (int i = 0; i < E; i++) {
        int u, v;
        cout << "Enter edge " << i + 1 << " (u v): ";
        cin >> u >> v;
        addEdge(mat, u, v);
    }
}
```



```
displayMatrix(mat);

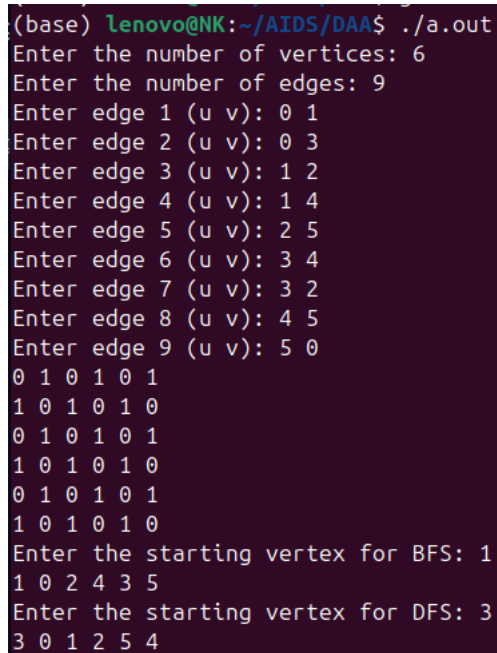
int startvertex;
cout << "Enter the starting vertex for BFS: ";
cin >> startvertex;
BFS(mat, startvertex);
cout << endl;

cout << "Enter the starting vertex for DFS: ";
cin >> startvertex;
DFS(mat, startvertex);
cout << endl;

return 0;

}
```

### SAMPLE INPUT-OUTPUT



```
(base) lenovo@NK:~/AIDS/DAA$ ./a.out
Enter the number of vertices: 6
Enter the number of edges: 9
Enter edge 1 (u v): 0 1
Enter edge 2 (u v): 0 3
Enter edge 3 (u v): 1 2
Enter edge 4 (u v): 1 4
Enter edge 5 (u v): 2 5
Enter edge 6 (u v): 3 4
Enter edge 7 (u v): 3 2
Enter edge 8 (u v): 4 5
Enter edge 9 (u v): 5 0
0 1 0 1 0 1
1 0 1 0 1 0
0 1 0 1 0 1
1 0 1 0 1 0
0 1 0 1 0 1
1 0 1 0 1 0
Enter the starting vertex for BFS: 1
1 0 2 4 3 5
Enter the starting vertex for DFS: 3
3 0 1 2 5 4
```

## 10. PRIM'S ALGORITHM

**DATE:**

### ALGORITHM

Prim's Algorithm to find the Minimum cost spanning tree

```
1.  FUNCTION minKey(key, mstSet, V)
2.      SET min = INFINITY, min_index
3.      FOR each vertex v from 0 to V-1
4.          IF mstSet[v] is FALSE AND key[v] < min
5.              SET min = key[v], min_index = v
6.      RETURN min_index

7.  FUNCTION printMST(parent, graph, V)
8.      PRINT "Edge    Weight"
9.      FOR each vertex i from 1 to V-1
10.         PRINT parent[i] - i and graph[i][parent[i]]

11. FUNCTION primMST(graph, V)
12.     INITIALIZE parent array of size V
13.     INITIALIZE key array of size V with INFINITY
14.     INITIALIZE mstSet array of size V with FALSE
15.     SET key[0] = 0
16.     SET parent[0] = -1
17.     FOR count from 0 to V-2
18.         SET u = minKey(key, mstSet, V)
19.         SET mstSet[u] = TRUE
20.         FOR each vertex v from 0 to V-1
21.             IF graph[u][v] > 0 AND mstSet[v] is FALSE AND graph[u][v] < key[v]
22.                 SET parent[v] = u
23.                 SET key[v] = graph[u][v]
24.     CALL printMST(parent, graph, V)
```

### PROGRAM

```
#include <iostream>
```

```
#include <vector>
#include <climits>
using namespace std;

int minKey(const vector<int>& key, const vector<bool>& mstSet, int V) {
    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(const vector<int>& parent, const vector<vector<int>>& graph, int V) {
    cout << "Edge \tWeight\n";
    for (int i = 1; i < V; i++)
        cout << parent[i] << " - " << i << " \t" << graph[i][parent[i]] << "\n";
}

void primMST(const vector<vector<int>>& graph, int V) {
    vector<int> parent(V);
    vector<int> key(V, INT_MAX);
    vector<bool> mstSet(V, false);

    key[0] = 0;
    parent[0] = -1;

    for (int count = 0; count < V - 1; count++) {
        int u = minKey(key, mstSet, V);
        mstSet[u] = true;

        for (int v = 0; v < V; v++) {
            if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }

    printMST(parent, graph, V);
}
```

```
int main() {
    int V, E;
    cout << "Enter the number of vertices: ";
    cin >> V;

    cout << "Enter the number of edges: ";
    cin >> E;

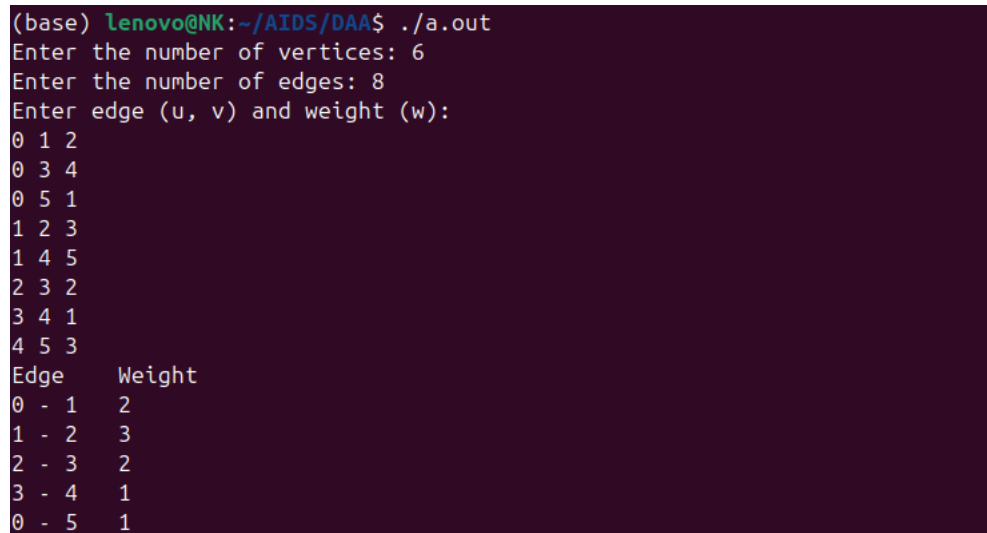
    vector<vector<int>> graph(V, vector<int>(V, 0));

    cout << "Enter edge (u, v) and weight (w):\n";
    for (int i = 0; i < E; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        graph[u][v] = w;
        graph[v][u] = w;
    }

    primMST(graph, V);

    return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
(base) lenovo@NK:~/AIDS/DAA$ ./a.out
Enter the number of vertices: 6
Enter the number of edges: 8
Enter edge (u, v) and weight (w):
0 1 2
0 3 4
0 5 1
1 2 3
1 4 5
2 3 2
3 4 1
4 5 3
Edge      Weight
0 - 1     2
1 - 2     3
2 - 3     2
3 - 4     1
0 - 5     1
```

## 11.KRUSKAL'S ALGORITHM

**DATE:**

### ALGORITHM

```
1.  CLASS DSU:
2.      FUNCTION DSU(n):
3.          INITIALIZE parent array of size n with -1
4.          INITIALIZE rank array of size n with 1

5.      FUNCTION find(i):
6.          IF parent[i] == -1:
7.              RETURN i
8.          RETURN parent[i] = find(parent[i])  // Path compression

9.      FUNCTION unite(x, y):
10.         SET s1 = find(x)
11.         SET s2 = find(y)
12.         IF s1 != s2:
13.             IF rank[s1] < rank[s2]:
14.                 SET parent[s1] = s2
15.             ELSE IF rank[s1] > rank[s2]:
16.                 SET parent[s2] = s1
17.             ELSE:
18.                 SET parent[s2] = s1
19.                 INCREMENT rank[s1]

20. CLASS Graph:
21.     FUNCTION Graph(V):
22.         INITIALIZE V (number of vertices)
23.         INITIALIZE edges as an empty list

24.     FUNCTION addEdge(u, v, w):
25.         ADD edge {w, u, v} to edges list

26.     FUNCTION kruskals_mst():
27.         SORT edges list based on weights
28.         INITIALIZE DSU s with V vertices
29.         SET mstWeight = 0
30.         PRINT "Following are the edges in the constructed MST:"
```

```
31.         FOR each edge in edges:
32.             SET w = edge[0], u = edge[1], v = edge[2]
33.             IF s.find(u) != s.find(v):
34.                 s.unite(u, v)
35.                 ADD w to mstWeight
36.                 PRINT u, "--", v, "=", w

37.         PRINT "Minimum Cost Spanning Tree: ", mstWeight
```

## PROGRAM

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

class DSU {
    int* parent;
    int* rank;

public:
    DSU(int n) {
        parent = new int[n];
        rank = new int[n];

        for (int i = 0; i < n; i++) {
            parent[i] = -1;
            rank[i] = 1;
        }
    }

    int find(int i) {
        if (parent[i] == -1)
            return i;
        return parent[i] = find(parent[i]);
    }

    void unite(int x, int y) {
        int s1 = find(x);
```

```
    int s2 = find(y);

    if (s1 != s2) {
        if (rank[s1] < rank[s2]) {
            parent[s1] = s2;
        } else if (rank[s1] > rank[s2]) {
            parent[s2] = s1;
        } else {
            parent[s2] = s1;
            rank[s1] += 1;
        }
    }
}

};

class Graph {
    vector<vector<int>>> edges;
    int V;

public:
    Graph(int V) {
        this->V = V;
    }

    void addEdge(int u, int v, int w) {
        edges.push_back({w, u, v});
    }

    void kruskals_mst() {
        sort(edges.begin(), edges.end());

        DSU s(V);
        int mstWeight = 0;
        cout << "Following are the edges in the constructed MST:" << endl;

        for (auto& edge : edges) {
            int w = edge[0];
            int u = edge[1];
            int v = edge[2];

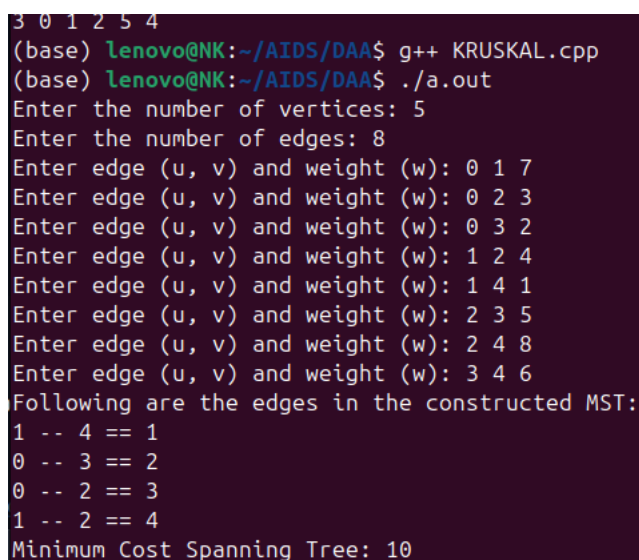
            if (s.find(u) != s.find(v)) {
```

```
        s.unite(u, v);
        mstWeight += w;
        cout << u << " -- " << v << " == " << w << endl;
    }}
    cout << "Minimum Cost Spanning Tree: " << mstWeight << endl;
    }

int main() {
    int V, E;
    cout << "Enter the number of vertices: ";
    cin >> V;
    cout << "Enter the number of edges: ";
    cin >> E;

    Graph g(V);
    for (int i = 0; i < E; i++) {
        int u, v, w;
        cout << "Enter edge (u, v) and weight (w): ";
        cin >> u >> v >> w;
        g.addEdge(u, v, w);
    }
    g.kruskals_mst();
    return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
3 0 1 2 5 4
(base) lenovo@NK:~/AIDS/DAA$ g++ KRUSKAL.cpp
(base) lenovo@NK:~/AIDS/DAA$ ./a.out
Enter the number of vertices: 5
Enter the number of edges: 8
Enter edge (u, v) and weight (w): 0 1 7
Enter edge (u, v) and weight (w): 0 2 3
Enter edge (u, v) and weight (w): 0 3 2
Enter edge (u, v) and weight (w): 1 2 4
Enter edge (u, v) and weight (w): 1 4 1
Enter edge (u, v) and weight (w): 2 3 5
Enter edge (u, v) and weight (w): 2 4 8
Enter edge (u, v) and weight (w): 3 4 6
Following are the edges in the constructed MST:
1 -- 4 == 1
0 -- 3 == 2
0 -- 2 == 3
1 -- 2 == 4
Minimum Cost Spanning Tree: 10
```



## 12.DIJKSTRA'S ALGORITHM

**DATE:**

### ALGORITHM

```
1.  FUNCTION minDistance(dist, sptSet, V)
2.      min ← INFINITY
3.      min_index ← -1
4.      FOR each vertex v from 0 to V-1 DO
5.          IF sptSet[v] is false AND dist[v] <= min THEN
6.              min ← dist[v]
7.              min_index ← v
8.      RETURN min_index
9.  END FUNCTION

10. FUNCTION printSolution(dist, V)
11.      PRINT "Vertex  Distance from Source"
12.      FOR i from 0 to V-1 DO
13.          PRINT i, dist[i]
14.  END FUNCTION

15. FUNCTION dijkstra(graph, src, V)
16.      dist ← array of size V initialized to INFINITY
17.      sptSet ← array of size V initialized to false
18.      dist[src] ← 0
19.      FOR count from 0 to V-1 DO
20.          u ← minDistance(dist, sptSet, V)
21.          sptSet[u] ← true
22.          FOR each vertex v from 0 to V-1 DO
23.              IF sptSet[v] is false AND graph[u][v] > 0 AND dist[u] < INFINITY AND (dist[u] + graph[u][v] < dist[v]) THEN
24.                  dist[v] ← dist[u] + graph[u][v]
25.      CALL printSolution(dist, V)
26.  END FUNCTION
```

### PROGRAM

```
#include <iostream>
#include <vector>
#include <limits.h>
```

```
using namespace std;

int minDistance(vector<int>& dist, vector<bool>& sptSet, int V) {
    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)
        if (sptSet[v] == false && dist[v] <= min)
            min = dist[v], min_index = v;

    return min_index;
}

void printSolution(vector<int>& dist, int V) {
    cout << "Vertex \t Distance from Source" << endl;
    for (int i = 0; i < V; i++)
        cout << i << " \t\t" << dist[i] << endl;
}

void dijkstra(vector<vector<int>>& graph, int src, int V) {
    vector<int> dist(V, INT_MAX);
    vector<bool> sptSet(V, false);

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {
        int u = minDistance(dist, sptSet, V);
        sptSet[u] = true;

        for (int v = 0; v < V; v++)
            if (!sptSet[v] && graph[u][v] && dist[u] != INT_MAX
                && dist[u] + graph[u][v] < dist[v])
                dist[v] = dist[u] + graph[u][v];
    }

    printSolution(dist, V);
}

int main() {
    int V, E;
    cout << "Enter the number of vertices: ";
    cin >> V;
```

```
    cout << "Enter the number of edges: ";
    cin >> E;

    vector<vector<int>> graph(V, vector<int>(V, 0));

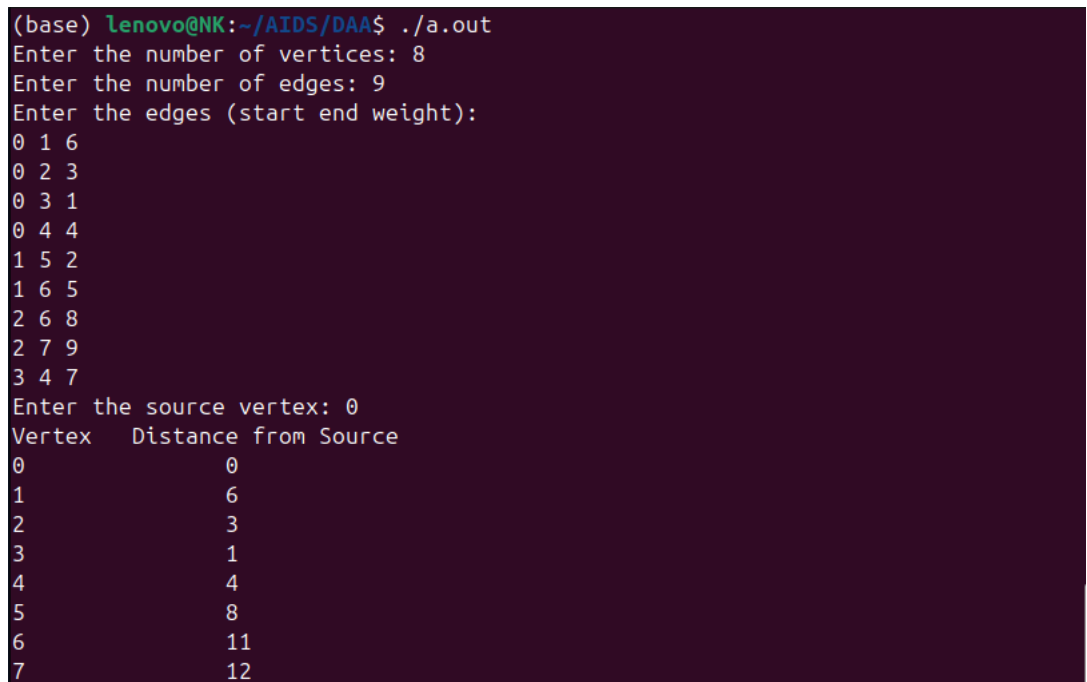
    cout << "Enter the edges (start end weight):" << endl;
    for (int i = 0; i < E; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        graph[u][v] = w;
        graph[v][u] = w;
    }

    int src;
    cout << "Enter the source vertex: ";
    cin >> src;

    dijkstra(graph, src, V);

    return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
(base) lenovo@NK:~/AIDS/DAA$ ./a.out
Enter the number of vertices: 8
Enter the number of edges: 9
Enter the edges (start end weight):
0 1 6
0 2 3
0 3 1
0 4 4
1 5 2
1 6 5
2 6 8
2 7 9
3 4 7
Enter the source vertex: 0
Vertex    Distance from Source
0          0
1          6
2          3
3          1
4          4
5          8
6         11
7         12
```

## 13.BELLMAN FORD ALGORITHM

**DATE:**

### ALGORITHM

```
1. function bellmanFord(V, E, edges, src)
2.     initialize dist as an array of size V with all values set to INF
3.     dist[src] ← 0 // Distance to the source vertex is set to 0

4.     for i ← 1 to V - 1 do
5.         for j ← 0 to E - 1 do
6.             u ← edges[j].u
7.             v ← edges[j].v
8.             weight ← edges[j].weight

9.             if dist[u] ≠ INF and dist[u] + weight < dist[v] then
10.                dist[v] ← dist[u] + weight
11.            end for
12.        end for

13.    for j ← 0 to E - 1 do
14.        u ← edges[j].u
15.        v ← edges[j].v
16.        weight ← edges[j].weight

17.        if dist[u] ≠ INF and dist[u] + weight < dist[v] then
18.            print "Graph contains a negative weight cycle"
19.            return
20.        end for

21.    print "Vertex\tDistance from Source"
22.    for i ← 0 to V - 1 do
23.        print i, dist[i]
24.    end for
25. end function
```

### PROGRAM

```
#include <iostream>
```

```
#include <vector>
#include <climits>
using namespace std;

struct Edge {
    int u, v, weight;
};

void bellmanFord(int V, int E, vector<Edge>& edges, int src) {
    vector<int> dist(V, INT_MAX); // Initialize distances with infinity
    dist[src] = 0; // Distance to source is 0

    // Relax all edges V-1 times
    for (int i = 1; i <= V - 1; i++) {
        for (int j = 0; j < E; j++) {
            int u = edges[j].u;
            int v = edges[j].v;
            int weight = edges[j].weight;

            if (dist[u] != INT_MAX && dist[u] + weight < dist[v]) {
                dist[v] = dist[u] + weight;
            }
        }
    }

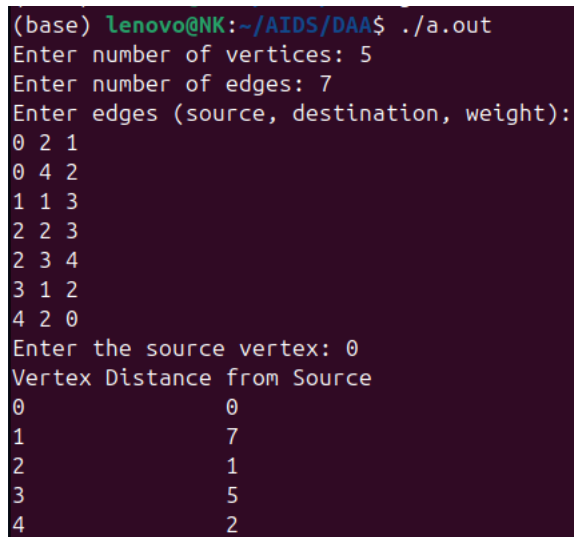
    // Check for negative-weight cycles
    for (int j = 0; j < E; j++) {
        int u = edges[j].u;
        int v = edges[j].v;
        int weight = edges[j].weight;

        if (dist[u] != INT_MAX && dist[u] + weight < dist[v]) {
            cout << "Graph contains a negative weight cycle" << endl;
            return;
        }
    }

    // Print the distances from the source vertex
    cout << "Vertex\tDistance from Source" << endl;
    for (int i = 0; i < V; i++) {
        cout << i << "\t\t" << dist[i] << endl;
    }
}
```

```
    }  
}  
  
int main() {  
    int V, E, src;  
    cout << "Enter the number of vertices: ";  
    cin >> V;  
    cout << "Enter the number of edges: ";  
    cin >> E;  
  
    vector<Edge> edges(E);  
    cout << "Enter the edges (u v weight):" << endl;  
    for (int i = 0; i < E; i++) {  
        cin >> edges[i].u >> edges[i].v >> edges[i].weight;  
    }  
  
    cout << "Enter the source vertex: ";  
    cin >> src;  
  
    bellmanFord(V, E, edges, src);  
  
    return 0;  
}
```

### SAMPLE INPUT-OUTPUT



```
(base) lenovo@NK:~/AIDS/DAA$ ./a.out  
Enter number of vertices: 5  
Enter number of edges: 7  
Enter edges (source, destination, weight):  
0 2 1  
0 4 2  
1 1 3  
2 2 3  
2 3 4  
3 1 2  
4 2 0  
Enter the source vertex: 0  
Vertex Distance from Source  
0          0  
1          7  
2          1  
3          5  
4          2
```

## 14. FLOYD WARSHALL ALGORITHM

**DATE:**

### ALGORITHM

```
1. function floydWarshall(dist, n)
2.     for k ← 0 to n - 1 do
3.         for i ← 0 to n - 1 do
4.             for j ← 0 to n - 1 do
5.                 if dist[i][k]  INF and dist[k][j]  INF then
6.                     dist[i][j] ← min(dist[i][j], dist[i][k] + dist[k][j])
7.             end for
8.         end function

9. function printMatrix(dist, n)
10.    for i ← 0 to n - 1 do
11.        for j ← 0 to n - 1 do
12.            if dist[i][j] = INF then
13.                print "INF"
14.            else
15.                print dist[i][j]
16.            end for
17.        print newline
18.    end for
19. end function

20. function main
21.    input V
22.    input E
23.    initialize dist as a VxV matrix filled with INF
24.    for each edge (u, v, w) from input do
25.        dist[u][v] ← w
26.    for i ← 0 to V - 1 do
27.        dist[i][i] ← 0
28.    floydWarshall(dist, V)
29.    print "Shortest distance matrix:"
30.    printMatrix(dist, V)
31. end function
```

**PROGRAM**

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

void floydWarshall(vector<vector<int>>& dist, int n) {
    for (int k = 0; k < n; k++) {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (dist[i][k] != INT_MAX && dist[k][j] != INT_MAX) {
                    dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
                }
            }
        }
    }
}

void printMatrix(const vector<vector<int>>& dist, int n) {
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (dist[i][j] == INT_MAX)
                cout << "INF" << "\t";
            else
                cout << dist[i][j] << "\t";
        }
        cout << endl;
    }
}

int main() {
    int V, E;
    cout << "Enter number of vertices: ";
    cin >> V;
    cout << "Enter number of edges: ";
    cin >> E;

    vector<vector<int>> dist(V, vector<int>(V, INT_MAX));

    cout << "Enter edges (source, destination, weight):" << endl;
```



```
for (int i = 0; i < E; i++) {
    int u, v, w;
    cin >> u >> v >> w;
    dist[u][v] = w;
}

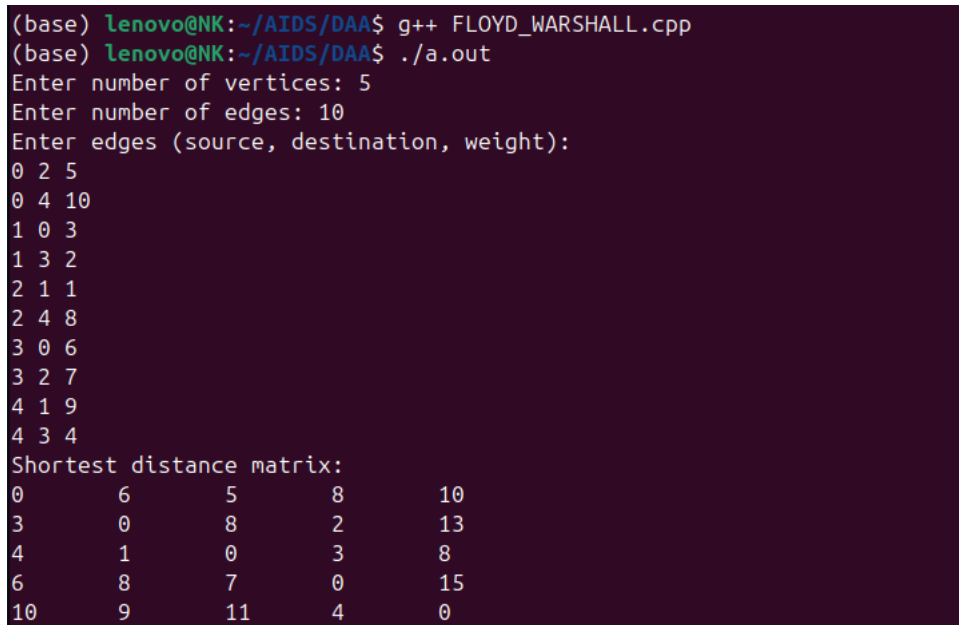
for (int i = 0; i < V; i++) {
    dist[i][i] = 0;
}

floydWarshall(dist, V);

cout << "Shortest distance matrix:" << endl;
printMatrix(dist, V);

return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
(base) lenovo@NK:~/AIDS/DAA$ g++ FLOYD_WARSHALL.cpp
(base) lenovo@NK:~/AIDS/DAA$ ./a.out
Enter number of vertices: 5
Enter number of edges: 10
Enter edges (source, destination, weight):
0 2 5
0 4 10
1 0 3
1 3 2
2 1 1
2 4 8
3 0 6
3 2 7
4 1 9
4 3 4
Shortest distance matrix:
0      6      5      8      10
3      0      8      2      13
4      1      0      3      8
6      8      7      0      15
10     9     11     4      0
```

## 15.DYNAMIC FIBONACCI SERIES

**DATE:**

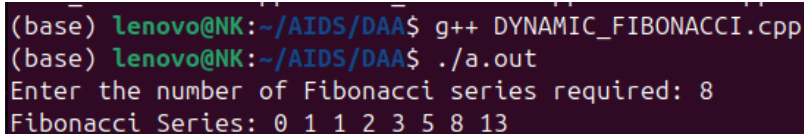
### ALGORITHM

1. INPUT: A non-negative integer n
2.  $F[0] = 0$
3.  $F[1] = 1$
4. for  $i = 2$  to  $n$  do
5.      $F[i] = F[i - 1] + F[i - 2]$
6. return  $F[n]$

### PROGRAM

```
#include <iostream>
#include <vector>
using namespace std;
void dynamic_fib(vector<int>& fibSeries, int n) {
    if (n > 0) fibSeries[0] = 0;
    if (n > 1) fibSeries[1] = 1;
    for (int i = 2; i < n; i++) {
        fibSeries[i] = fibSeries[i - 1] + fibSeries[i - 2];
    }
}
int main() {
    int n;
    cout << "Enter the number of Fibonacci series required: ";
    cin >> n;
    vector<int> fibSeries(n);
    dynamic_fib(fibSeries, n);
    cout << "Fibonacci Series: ";
    for (int i = 0; i < n; i++) {
        cout << fibSeries[i] << " ";
    }
    cout << "\n";
    return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
(base) lenovo@NK:~/AIDS/DAA$ g++ DYNAMIC_FIBONACCI.cpp
(base) lenovo@NK:~/AIDS/DAA$ ./a.out
Enter the number of Fibonacci series required: 8
Fibonacci Series: 0 1 1 2 3 5 8 13
```

## 16.COIN ROW PROBLEM

**DATE:**

### ALGORITHM

1. INPUT: Array C[1..n] of positive integers indicating the coin values
2. OUTPUT: The maximum amount of money that can be picked up
3. F[0] = 0
4. F[1] = C[1]
5. for i <- 2 to n do
6.     F[i] <- max(C[i] + F[i - 2], F[i - 1])
7. return F[n]

### PROGRAM

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int coinRowProblem(const vector<int>& coins, vector<int>& final) {
    int n = coins.size();
    if (n == 0) return 0;
    if (n == 1) {
        final.push_back(coins[0]);
        return coins[0];
    }

    final.resize(n);
    final[0] = coins[0];
    final[1] = max(coins[0], coins[1]);

    for (int i = 2; i < n; i++) {
        final[i] = max(final[i - 1], coins[i] + final[i - 2]);
    }
    return final[n - 1];
}

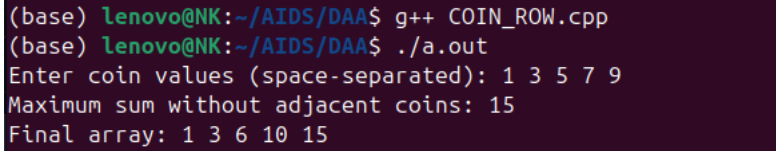
int main() {
    vector<int> coins;
```

```
int val;
cout << "Enter coin values (space-separated): ";
while (cin >> val) {
    coins.push_back(val);
    if (cin.peek() == '\n') break;
}

vector<int> final;
int maxSum = coinRowProblem(coins, final);
cout << "Maximum sum without adjacent coins: " << maxSum << endl;
cout << "Final array: ";
for (int value : final) {
    cout << value << " ";
}
cout << endl;

return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
(base) lenovo@NK:~/AIDS/DAA$ g++ COIN_ROW.cpp
(base) lenovo@NK:~/AIDS/DAA$ ./a.out
Enter coin values (space-separated): 1 3 5 7 9
Maximum sum without adjacent coins: 15
Final array: 1 3 6 10 15
```

## 17.COIN CHANGE-MAKING PROBLEM

**DATE:**

### ALGORITHM

1. INPUT: Positive integer n and array D[1..m] of increasing positive integers indicating
2. OUTPUT: The minimum number of coins that add up to n
3. F[0] = 0
4. for i <- 1 to n do
5.     temp <-     // Initialize temp to a large value
6.     j <- 1
7.     while j <= m and i >= D[j] do
8.         temp <- min(F[i - D[j]], temp)
9.         j <- j + 1
10.    F[i] <- temp + 1
11. return F[n]

### PROGRAM

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <climits>
using namespace std;

struct Coin {
    int denomination;
    int count;
};

vector<Coin> changeMaking(const vector<int>& D, int n) {
    vector<int> F(n + 1, INT_MAX);
    F[0] = 0;
    vector<int> lastCoin(n + 1, -1);

    for (int i = 1; i <= n; i++) {
        int temp = INT_MAX;
        int tempCoin = -1;
        for (int j = 0; j < D.size() && i >= D[j]; j++) {
            if (F[i - D[j]] != INT_MAX && F[i - D[j]] + 1 < temp) {
```

```
        temp = F[i - D[j]] + 1;
        tempCoin = D[j];
    }
}
if (temp != INT_MAX) {
    F[i] = temp;
    lastCoin[i] = tempCoin;
}
}

vector<Coin> coinsUsed;
int remainingAmount = n;
while (remainingAmount > 0 && lastCoin[remainingAmount] != -1) {
    Coin c;
    c.denomination = lastCoin[remainingAmount];
    c.count = 1;
    coinsUsed.push_back(c);

    remainingAmount -= lastCoin[remainingAmount];
    while (remainingAmount > 0 && lastCoin[remainingAmount] == c.denomination) {
        c.count++;
        remainingAmount -= c.denomination;
    }
}

return coinsUsed;
}

int main() {
    int n;
    cout << "Enter the amount: ";
    cin >> n;

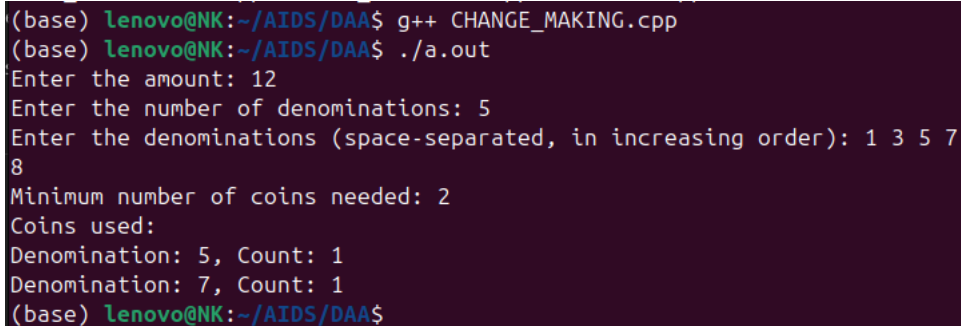
    vector<int> D;
    int m;
    cout << "Enter the number of denominations: ";
    cin >> m;
    cout << "Enter the denominations (space-separated, in increasing order): ";
    for (int i = 0; i < m; i++) {
        int denomination;
        cin >> denomination;
```

```
        D.push_back(denomination);
    }

    vector<Coin> result = changeMaking(D, n);
    if (!result.empty()) {
        cout << "Minimum number of coins needed: " << result.size() << endl;
        cout << "Coins used:" << endl;
        for (const auto& coin : result) {
            cout << "Denomination: " << coin.denomination << ", Count: " << coin.count << " ";
        }
    } else {
        cout << "It is not possible to make the given amount with the available denominations." << endl;
    }

    return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
(base) lenovo@NK:~/AIDS/DAA$ g++ CHANGE_MAKING.cpp
(base) lenovo@NK:~/AIDS/DAA$ ./a.out
Enter the amount: 12
Enter the number of denominations: 5
Enter the denominations (space-separated, in increasing order): 1 3 5 7 8
Minimum number of coins needed: 2
Coins used:
Denomination: 5, Count: 1
Denomination: 7, Count: 1
(base) lenovo@NK:~/AIDS/DAA$
```

## 18.COIN-COLLECTING PROBLEM

**DATE:**

### ALGORITHM

1. INPUT: Matrix  $C[1..n, 1..m]$  whose elements are equal to 1 (cells with coins) or 0 (cell without coins)
2. OUTPUT: Largest number of coins the robot can bring to cell  $(n, m)$
3.  $F[1, 1] \leftarrow C[1, 1]$
4. for  $j \leftarrow 2$  to  $m$  do
5.      $F[1, j] = F[1, j - 1] + C[1, j]$
6. for  $i \leftarrow 2$  to  $n$  do
7.      $F[i, 1] = F[i - 1, 1] + C[i, 1]$
8. for  $i \leftarrow 2$  to  $n$  do
9.     for  $j \leftarrow 2$  to  $m$  do
10.          $F[i, j] = \max(F[i - 1, j], F[i, j - 1]) + C[i, j]$
11. return  $F[n, m]$

### PROGRAM

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

pair<int, vector<pair<int, int>>> robotCoinCollection(const vector<vector<int>>& C) {
    int n = C.size();
    int m = C[0].size();

    // DP matrix to store maximum coins up to each cell
    vector<vector<int>> F(n, vector<int>(m, 0));
    // Direction matrix to reconstruct the path
    vector<vector<pair<int, int>>> direction(n, vector<pair<int, int>>(m, {-1, -1}));

    // Initialize starting point
    F[0][0] = C[0][0];

    // Fill first row (only right moves are possible)
    for (int j = 1; j < m; j++) {
        F[0][j] = F[0][j - 1] + C[0][j];
        direction[0][j] = {0, j - 1}; // came from the left
    }
}
```



```
}

// Fill first column (only down moves are possible)
for (int i = 1; i < n; i++) {
    F[i][0] = F[i - 1][0] + C[i][0];
    direction[i][0] = {i - 1, 0}; // came from above
}

// Fill the rest of the matrix
for (int i = 1; i < n; i++) {
    for (int j = 1; j < m; j++) {
        if (F[i - 1][j] > F[i][j - 1]) {
            F[i][j] = F[i - 1][j] + C[i][j];
            direction[i][j] = {i - 1, j}; // came from above
        } else {
            F[i][j] = F[i][j - 1] + C[i][j];
            direction[i][j] = {i, j - 1}; // came from the left
        }
    }
}

// Reconstruct the path
vector<pair<int, int>> path;
int i = n - 1, j = m - 1;
while (i >= 0 && j >= 0) {
    path.push_back({i + 1, j + 1}); // Store 1-based index
    auto prev = direction[i][j];
    i = prev.first;
    j = prev.second;
}

reverse(path.begin(), path.end()); // Reverse to get the path from start to end
return {F[n - 1][m - 1], path}; // Return the maximum coins and the path
}

int main() {
    int n, m;
    cout << "Enter the number of rows: ";
    cin >> n;
    cout << "Enter the number of columns: ";
    cin >> m;
```

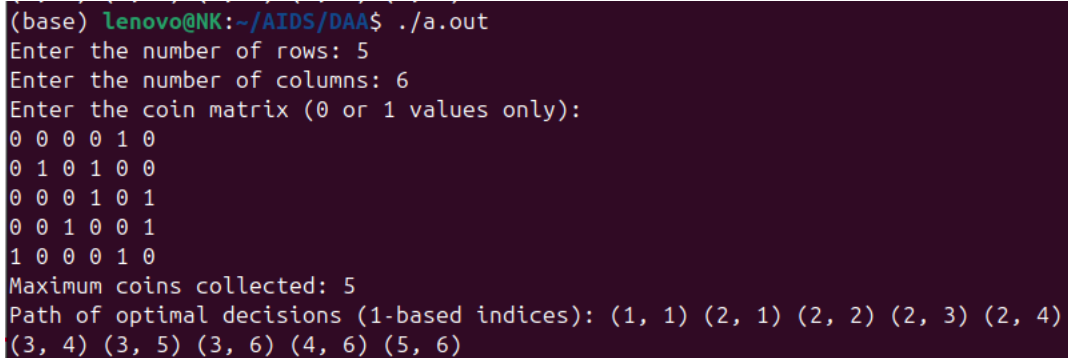
```
vector<vector<int>> C(n, vector<int>(m));
cout << "Enter the coin matrix (0 or 1 values only): " << endl;
for (int i = 0; i < n; i++) {
    for (int j = 0; j < m; j++) {
        cin >> C[i][j];
    }
}

auto result = robotCoinCollection(C);
int maxCoins = result.first;
vector<pair<int, int>> path = result.second;

cout << "Maximum coins collected: " << maxCoins << endl;
cout << "Path of optimal decisions (1-based indices): ";
for (const auto& p : path) {
    cout << "(" << p.first << ", " << p.second << ") ";
}
cout << endl;

return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
(base) lenovo@NK:~/AIDS/DAA$ ./a.out
Enter the number of rows: 5
Enter the number of columns: 6
Enter the coin matrix (0 or 1 values only):
0 0 0 0 1 0
0 1 0 1 0 0
0 0 0 1 0 1
0 0 1 0 0 1
1 0 0 0 1 0
Maximum coins collected: 5
Path of optimal decisions (1-based indices): (1, 1) (2, 1) (2, 2) (2, 3) (2, 4)
(3, 4) (3, 5) (3, 6) (4, 6) (5, 6)
```

## 19.MINIMUM COST PATH IN A MATRIX

**DATE:**

### ALGORITHM

1. INPUT: Cost matrix  $C[1..m, 1..n]$  having a cost at each cell
2. OUTPUT: The minimum cost to reach cell  $(m, n)$  from  $(0, 0)$
3.  $F[1, 1] \leftarrow C[1, 1]$
4. for  $j \leftarrow 2$  to  $n$  do
5.      $F[1, j] \leftarrow F[1, j - 1] + C[1, j]$
6. for  $i \leftarrow 2$  to  $m$  do
7.      $F[i, 1] \leftarrow F[i - 1, 1] + C[i, 1]$
8. for  $i \leftarrow 2$  to  $m$  do
9.     for  $j \leftarrow 2$  to  $n$  do
10.          $F[i, j] \leftarrow \min(F[i - 1, j], F[i, j - 1], F[i - 1, j - 1]) + C[i, j]$
11. return  $F[m, n]$

### PROGRAM

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

pair<int, vector<pair<int, int>>> minCostPath(const vector<vector<int>>& C) {
    int n = C.size();          // Number of rows
    int m = C[0].size();       // Number of columns

    vector<vector<int>> F(n, vector<int>(m, 0));
    vector<vector<pair<int, int>>> direction(n, vector<pair<int, int>>(m, {-1, -1}));

    F[0][0] = C[0][0];

    for (int j = 1; j < m; j++) {
        F[0][j] = F[0][j - 1] + C[0][j];
        direction[0][j] = {0, j - 1};
    }

    for (int i = 1; i < n; i++) {
        F[i][0] = F[i - 1][0] + C[i][0];
```

```
        direction[i][0] = {i - 1, 0};
    }

    for (int i = 1; i < n; i++) {
        for (int j = 1; j < m; j++) {
            if (F[i - 1][j] <= F[i][j - 1] && F[i - 1][j] <= F[i - 1][j - 1]) {
                F[i][j] = F[i - 1][j] + C[i][j];
                direction[i][j] = {i - 1, j};
            } else if (F[i][j - 1] <= F[i - 1][j] && F[i][j - 1] <= F[i - 1][j - 1]) {
                F[i][j] = F[i][j - 1] + C[i][j];
                direction[i][j] = {i, j - 1};
            } else {
                F[i][j] = F[i - 1][j - 1] + C[i][j];
                direction[i][j] = {i - 1, j - 1};
            }
        }
    }

    vector<pair<int, int>> path;
    int i = n - 1, j = m - 1;
    while (i >= 0 && j >= 0) {
        path.push_back({i + 1, j + 1});
        auto prev = direction[i][j];
        i = prev.first;
        j = prev.second;
    }

    reverse(path.begin(), path.end());
    return {F[n - 1][m - 1], path};
}

int main() {
    int n, m;
    cout << "Enter the number of rows: ";
    cin >> n;
    cout << "Enter the number of columns: ";
    cin >> m;

    vector<vector<int>> C(n, vector<int>(m));
    cout << "Enter the cost matrix: " << endl;
    for (int i = 0; i < n; i++) {
```

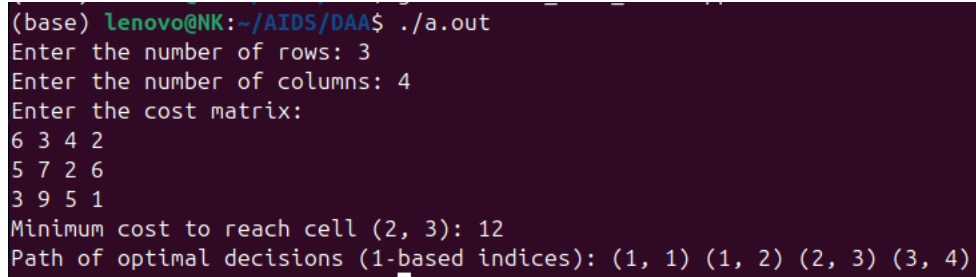
```
        for (int j = 0; j < m; j++) {
            cin >> C[i][j];
        }
    }

    auto result = minCostPath(C);
    int minCost = result.first;
    vector<pair<int, int>> path = result.second;

    cout << "Minimum cost to reach cell (" << n - 1 << ", " << m - 1 << "): " << minCost << endl;
    cout << "Path of optimal decisions (1-based indices): ";
    for (const auto& p : path) {
        cout << "(" << p.first << ", " << p.second << ") ";
    }
    cout << endl;

    return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
(base) lenovo@NK:~/AIDS/DAA$ ./a.out
Enter the number of rows: 3
Enter the number of columns: 4
Enter the cost matrix:
6 3 4 2
5 7 2 6
3 9 5 1
Minimum cost to reach cell (2, 3): 12
Path of optimal decisions (1-based indices): (1, 1) (1, 2) (2, 3) (3, 4)
```

## 20.0/1 KNAPSACK PROBLEM

**DATE:**

### ALGORITHM

```
1. INPUT:
2.     - Values array V of size n representing the values of items
3.     - Weights array W of size n representing the weights of items
4.     - Capacity C representing the maximum capacity of the knapsack
5. OUTPUT:
6.     - Maximum value that can be obtained by filling the knapsack
7. dp[0][w] = 0 for all w in 0 to C
8. for i <- 1 to n do
9.     for w <- 1 to C do
10.        if W[i - 1] > w then
11.            dp[i][w] = dp[i - 1][w]
12.        else
13.            dp[i][w] = max(dp[i - 1][w], V[i - 1] + dp[i - 1][w - W[i - 1]])
14. selectedCoins = []
15. selectedCount = 0
16. i = n
17. j = C
18. while i > 0 and j > 0 do
19.     if dp[i][j] != dp[i - 1][j] then
20.         add (i - 1) to selectedCoins
21.         j = j - W[i - 1]
22.         i = i - 1
23.     else
24.         i = i - 1
25. output "Selected weights:"
26. for k <- selectedCount - 1 to 0 do
27.     output selectedCoins[k] + 1
```

### PROGRAM

```
#include <iostream>
#include <vector>
#include <algorithm>

using namespace std;
```

```
int knapsack(const vector<int>& values, const vector<int>& weights, int capacity) {
    int n = values.size();
    vector<vector<int>> dp(n + 1, vector<int>(capacity + 1, 0));

    // Fill the dp table
    for (int i = 1; i <= n; i++) {
        for (int w = 1; w <= capacity; w++) {
            if (weights[i - 1] > w) {
                dp[i][w] = dp[i - 1][w];
            } else {
                dp[i][w] = max(dp[i - 1][w], values[i - 1] + dp[i - 1][w - weights[i - 1]]);
            }
        }
    }

    // Backtrack to find the selected items
    vector<int> selectedItems;
    int i = n;
    int j = capacity;

    while (i > 0 && j > 0) {
        if (dp[i][j] != dp[i - 1][j]) {
            selectedItems.push_back(i - 1); // Store the index of the selected item
            j -= weights[i - 1]; // Reduce the capacity
        }
        i--;
    }

    // Output selected items
    cout << "Selected weights: ";
    for (int k = selectedItems.size() - 1; k >= 0; k--) {
        cout << selectedItems[k] + 1 << " "; // Convert to 1-based index
    }
    cout << endl;

    return dp[n][capacity]; // Return the maximum value
}

int main() {
    int n, capacity;
```

```
    cout << "Enter the number of items: ";
    cin >> n;
    vector<int> values(n), weights(n);

    cout << "Enter the values of items: ";
    for (int i = 0; i < n; i++) {
        cin >> values[i];
    }

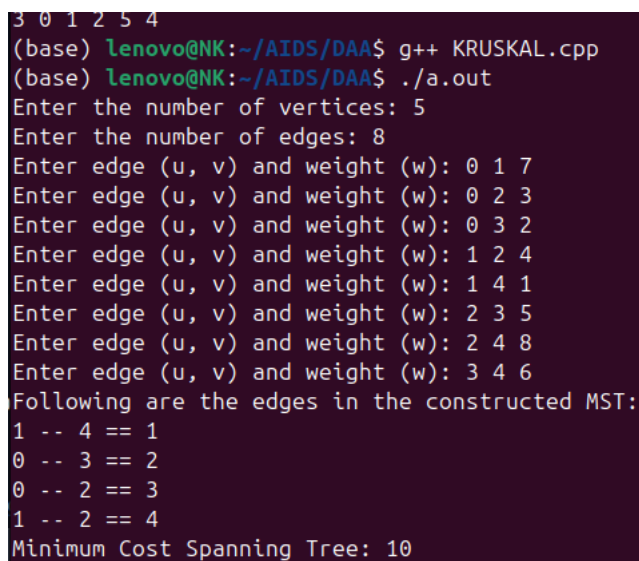
    cout << "Enter the weights of items: ";
    for (int i = 0; i < n; i++) {
        cin >> weights[i];
    }

    cout << "Enter the maximum capacity of the knapsack: ";
    cin >> capacity;

    int maxValue = knapsack(values, weights, capacity);
    cout << "Maximum value that can be obtained: " << maxValue << endl;

    return 0;
}
```

### SAMPLE INPUT-OUTPUT



```
3 0 1 2 5 4
(base) lenovo@NK:~/AIDS/DAA$ g++ KRUSKAL.cpp
(base) lenovo@NK:~/AIDS/DAA$ ./a.out
Enter the number of vertices: 5
Enter the number of edges: 8
Enter edge (u, v) and weight (w): 0 1 7
Enter edge (u, v) and weight (w): 0 2 3
Enter edge (u, v) and weight (w): 0 3 2
Enter edge (u, v) and weight (w): 1 2 4
Enter edge (u, v) and weight (w): 1 4 1
Enter edge (u, v) and weight (w): 2 3 5
Enter edge (u, v) and weight (w): 2 4 8
Enter edge (u, v) and weight (w): 3 4 6
Following are the edges in the constructed MST:
1 -- 4 == 1
0 -- 3 == 2
0 -- 2 == 3
1 -- 2 == 4
Minimum Cost Spanning Tree: 10
```



## 21.LONGEST COMMON SUBSEQUENCE

**DATE:**

### ALGORITHM

```
1. INPUT: Two sequences X and Y
2. OUTPUT: The longest subsequence that is common to both sequences
3. function LCS(X, Y)
4.     m = length(X)
5.     n = length(Y)
6.     let b[1..m, 1..n] and mat[0..m, 0..n] be new tables
7.     for i = 1 to m do
8.         mat[i, 0] = 0
9.     for j = 1 to n do
10.        mat[0, j] = 0
11.    for i = 1 to m do
12.        for j = 1 to n do
13.            if X[i] == Y[j] then
14.                mat[i, j] = mat[i - 1, j - 1] + 1
15.                b[i, j] = "diagonal" // indicates a match
16.            else if mat[i - 1, j] >= mat[i, j - 1] then
17.                mat[i, j] = mat[i - 1, j]
18.                b[i, j] = "up" // indicates coming from above
19.            else
20.                mat[i, j] = mat[i, j - 1]
21.                b[i, j] = "left" // indicates coming from the left
22.    return b and mat
```

### PROGRAM

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>

using namespace std;

pair<vector<vector<int>>, vector<vector<int>>>> LCS(const string& X, const string& Y) {
    int m = X.length();
    int n = Y.length();
```

```
vector<vector<int>> mat(m + 1, vector<int>(n + 1, 0));
vector<vector<int>> b(m + 1, vector<int>(n + 1, 0));

for (int i = 1; i <= m; i++) {
    for (int j = 1; j <= n; j++) {
        if (X[i - 1] == Y[j - 1]) {
            mat[i][j] = mat[i - 1][j - 1] + 1;
            b[i][j] = 1; // Marked for diagonal move
        } else if (mat[i - 1][j] >= mat[i][j - 1]) {
            mat[i][j] = mat[i - 1][j];
            b[i][j] = 2; // Marked for move up
        } else {
            mat[i][j] = mat[i][j - 1];
            b[i][j] = 3; // Marked for move left
        }
    }
}

return make_pair(b, mat);
}

string constructLCS(const string& X, const vector<vector<int>>& b) {
    int i = X.length();
    int j = b[0].size() - 1; // Size of Y
    string lcs;

    while (i > 0 && j > 0) {
        if (b[i][j] == 1) { // Diagonal
            lcs.push_back(X[i - 1]); // Add current character to LCS
            i--;
            j--;
        } else if (b[i][j] == 2) { // Move up
            i--;
        } else { // Move left
            j--;
        }
    }

    reverse(lcs.begin(), lcs.end()); // Reverse the LCS string
    return lcs;
}
```

```
}

int main() {
    string X, Y;

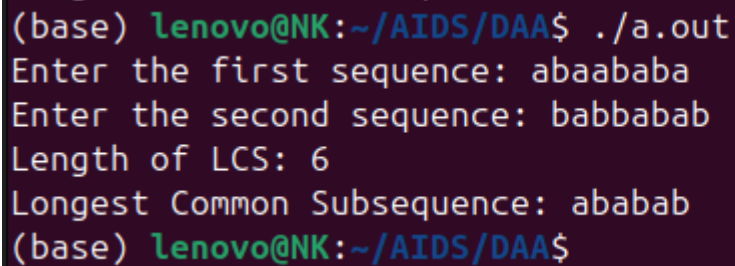
    cout << "Enter the first sequence: ";
    cin >> X;
    cout << "Enter the second sequence: ";
    cin >> Y;

    auto [b, mat] = LCS(X, Y);
    string lcs = constructLCS(X, b);

    cout << "Length of LCS: " << mat[X.length()][Y.length()] << endl;
    cout << "Longest Common Subsequence: " << lcs << endl;

    return 0;
}
```

#### SAMPLE INPUT-OUTPUT



```
(base) lenovo@NK:~/AIDS/DAA$ ./a.out
Enter the first sequence: abaababa
Enter the second sequence: babbabab
Length of LCS: 6
Longest Common Subsequence: ababab
(base) lenovo@NK:~/AIDS/DAA$
```