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

MARIYA JYOTHY
(81323014)

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* ***Mariya Jyothy(81323014)****, 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 third 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

1LINEAR SEARCH

03/07/2024

AIM

Implement Linear Search algorithm for a given array of integers.

ALGORITHM

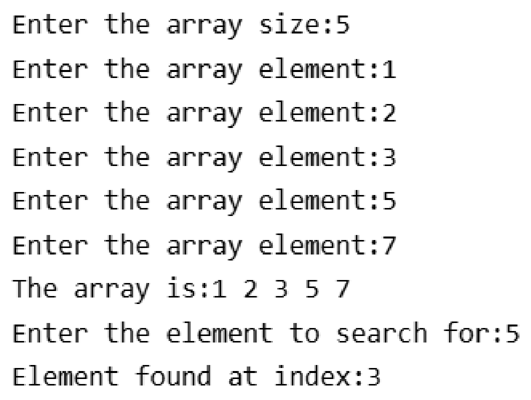
```
int linearSearch (int arr[ ], int target, int size)
     1 for i = 0 to size
     2 if (arr[i] == target)
     3   return i
     4 return-1;
```

PROGRAM

```
#include<iostream>

using namespace std;
void display(int A[], int UB) {
    cout << "The array is:";
    for (int i = 0; i < UB; i++) {
        cout << A[i] << " ";
    }
    cout << endl;
}

void LinearSearch(int A[], int x, int UB) {
    int ind = -1;
    for (int i = 0; i < UB; i++) {
        if (A[i] == x) {
            ind = i;
            cout << "Element found at index:" << i << endl;

        }
    }
    if (ind == -1) {
        cout << "Element not found" << endl;
    }

}
```

```
int main() {
    cout << "Enter the array size:";
    int n;
    cin >> n;
    int A[n];
    for (int i = 0; i < n; i++) {
        cout << "Enter the array element:";
        int el;
        cin >> el;
        A[i] = el;
    }
    display(A, n);
    cout << "Enter the element to search for:";
    int el;
    cin >> el;
    LinearSearch(A, el, n);
    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the array size:5
Enter the array element:1
Enter the array element:2
Enter the array element:3
Enter the array element:5
Enter the array element:7
The array is:1 2 3 5 7
Enter the element to search for:5
Element found at index:3
```

2.BINARY SEARCH

03/07/2024

AIM

Implement Binary Search algorithm for a given array of integers.

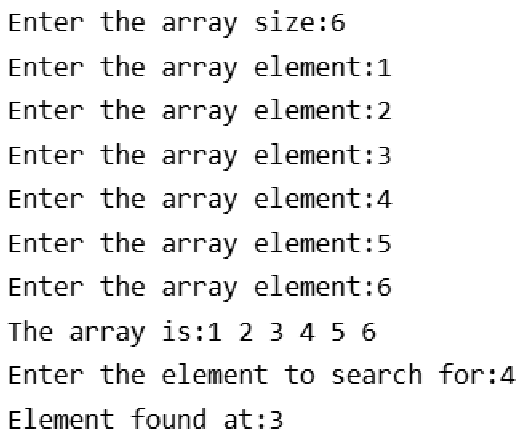
ALGORITHM

```
int binarySearch (int arr[ ], int target, int LB, int UB)
    1 left = LB
    2 right = UB - 1
    3 while left <= right:
    4    mid = left + (right - left)
    5       if arr[mid] == target:
    6            return mid
    7       if arr[mid] > target:
    8           right = mid - 1
    9        else:
    10            left = mid + 1
    11  return -1
```

PROGRAM

```
#include<iostream>

using namespace std;
void display(int A[], int UB) {
    cout << "The array is:";
    for (int i = 0; i < UB; i++) {
        cout << A[i] << " ";
    }
    cout << endl;
}
int binarySearch(int A[], int key, int lb, int ub) {
    int ind = -1;
    if (lb < ub) {
        int mid = (lb + ub) / 2;
        if (A[mid] == key) {
            ind = mid;
            return ind;
        }
        if (A[mid] > key) {
            ub = mid - 1;
            return binarySearch(A, key, lb, ub);
        } else if (A[mid] < key) {
            lb = mid + 1;
            return binarySearch(A, key, lb, ub);
        }
    }
    return ind;
    if (lb > ub) {
        cout << "NOT FOUND" << endl;
    }
}
int main() {
    cout << "Enter the array size:";
    int n;
    cin >> n;
    int A[n];
    for (int i = 0; i < n; i++) {
        cout << "Enter the array element:";
        int el;
        cin >> el;
        A[i] = el;
    }
    display(A, n);
    cout << "Enter the element to search for:";
    int el;
    cin >> el;
    cout << "Element found at:" << binarySearch(A, el, 0, n) << endl;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the array size:6
Enter the array element:1
Enter the array element:2
Enter the array element:3
Enter the array element:4
Enter the array element:5
Enter the array element:6
The array is:1 2 3 4 5 6
Enter the element to search for:4
Element found at:3
```

3.BUBBLE SORT

10/07/2024

AIM

Implement Bubble Sort to sort an array of natural numbers.

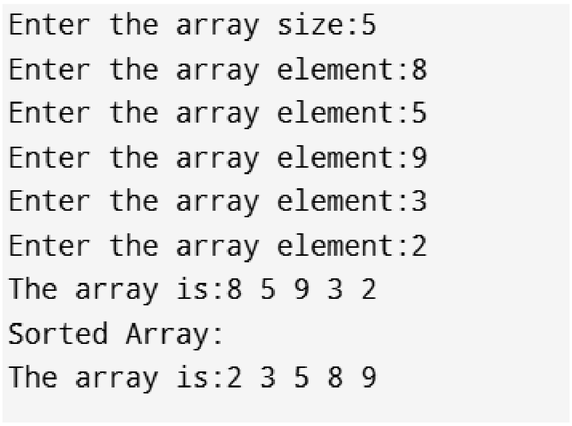
ALGORITHM

```
void bubbleSort(int A[],int len)
    1 for i from 0 to len - 2:
    2     for j from 0 to len - i - 2:
    3         if A[j+1] < A[j]:
    4             temp = A[j]
    5             A[j] = A[j+1]
    6             A[j+1] = temp
```

PROGRAM

```
#include<iostream>

using namespace std;
void display(int A[], int UB) {
    cout << "The array is:";
    for (int i = 0; i < UB; i++) {
        cout << A[i] << " ";
    }
    cout << endl;
}
void bubbleSort(int A[], int len) {
    for (int i = 0; i < len - 1; i++) {
        for (int j = 0; j < len - i - 1; j++) {
            if (A[j + 1] < A[j]) {
                int t = A[j];
                A[j] = A[j + 1];
                A[j + 1] = t;
            }
        }
    }
}
int main() {
    cout << "Enter the array size:";
    int n;
    cin >> n;
    int A[n];
    for (int i = 0; i < n; i++) {
        cout << "Enter the array element:";
        int el;
        cin >> el;
        A[i] = el;
    }
    display(A, n);
    bubbleSort(A, n);
    cout << "Sorted Array" << endl;
    display(A, n);
    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the array size:5
Enter the array element:8
Enter the array element:5
Enter the array element:9
Enter the array element:3
Enter the array element:2
The array is:8 5 9 3 2
Sorted Array:
The array is:2 3 5 8 9
```

4.INSERTION SORT

10/07/2024

AIM

Implement Insertion Sort to sort an array of natural numbers.

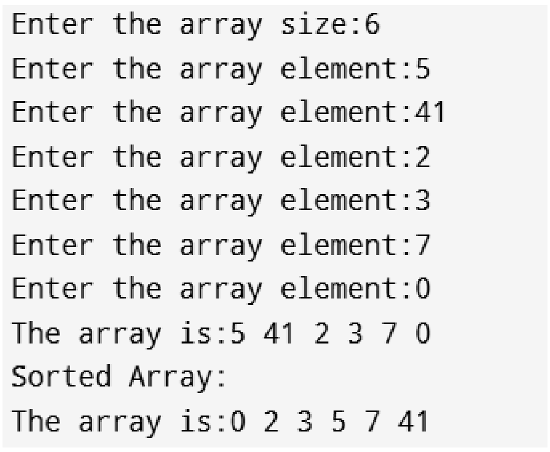
ALGORITHM

```
void insertionSort(int A[], int len):
    1  for i from 1 to len - 1:
    2      key = A[i]
    3      j = i - 1
    4      while j >= 0 and A[j] > key:
    5          A[j + 1] = A[j]
    6          j = j - 1
    7      A[j + 1] = key
```

PROGRAM

```
#include<iostream>

using namespace std;
void display(int A[], int UB) {
    cout << "The array is:";
    for (int i = 0; i < UB; i++) {
        cout << A[i] << " ";
    }
    cout << endl;
}
void insertion(int A[], int len) {
    int i = 1;
    while (i < len) {
        int j = i - 1;
        int key = A[i];
        while (j >= 0 and A[j] > key) {
            A[j + 1] = A[j];
            j--;
        }
        A[j + 1] = key;
        i++;
    }
}

int main() {
    cout << "Enter the array size:";
    int n;
    cin >> n;
    int A[n];
    for (int i = 0; i < n; i++) {
        cout << "Enter the array element:";
        int el;
        cin >> el;
        A[i] = el;
    }
    display(A, n);
    insertion(A, n);
    cout << "Sorted Array" << endl;
    display(A, n);
    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the array size:6
Enter the array element:5
Enter the array element:41
Enter the array element:2
Enter the array element:3
Enter the array element:7
Enter the array element:0
The array is:5 41 2 3 7 0
Sorted Array:
The array is:0 2 3 5 7 41
```

5.SELECTION SORT

17/07/2024

AIM

Implement Selection Sort to sort an array of natural numbers.

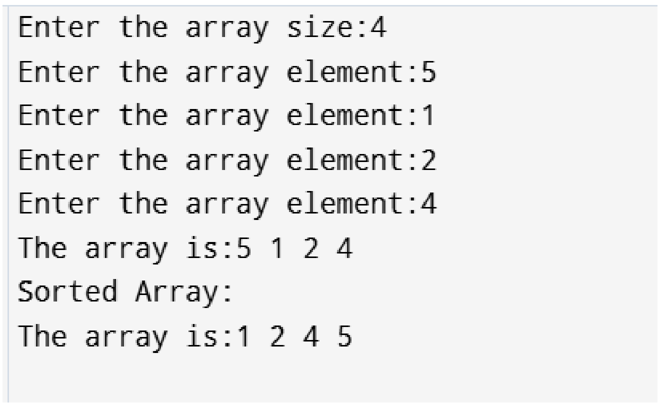
ALGORITHM

```
void selection(int A[], int len):
    1  for i from 0 to len - 1:
    2      min = i
    3      for j from i + 1 to len - 1:
    4          if A[min] > A[j]:
    5              temp = A[min]
    6              A[min] = A[j]
    7              A[j] = temp
    8      if min != i:
    9          u = A[min]
    10         A[min] = A[i]
    11         A[i] = u
```

PROGRAM

```
#include<iostream>

using namespace std;
void display(int A[], int UB) {
    cout << "The array is:";
    for (int i = 0; i < UB; i++) {
        cout << A[i] << " ";
    }
    cout << endl;
}
void selection(int A[], int len) {
    for (int i = 0; i < len; i++) {
        int min = i;
        for (int j = i + 1; j < len; j++) {
            if (A[min] > A[j]) {
                int t = A[min];
                A[min] = A[j];
                A[j] = t;
            }

        }
        if (min != i) {
            int u = A[min];
            A[min] = A[i];
            A[i] = A[min];
        }
    }
}
int main() {
    cout << "Enter the array size:";
    int n;
    cin >> n;
    int A[n];
    for (int i = 0; i < n; i++) {
        cout << "Enter the array element:";
        int el;
        cin >> el;
        A[i] = el;
    }
    display(A, n);
    selection(A, n);
    cout << "Sorted Array" << endl;
    display(A, n);
    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the array size:4
Enter the array element:5
Enter the array element:1
Enter the array element:2
Enter the array element:4
The array is:5 1 2 4
Sorted Array:
The array is:1 2 4 5
```

6.QUICK SORT

17/07/2024

AIM

Write a recursive algorithm for quicksort implementation to sort an N-element array.

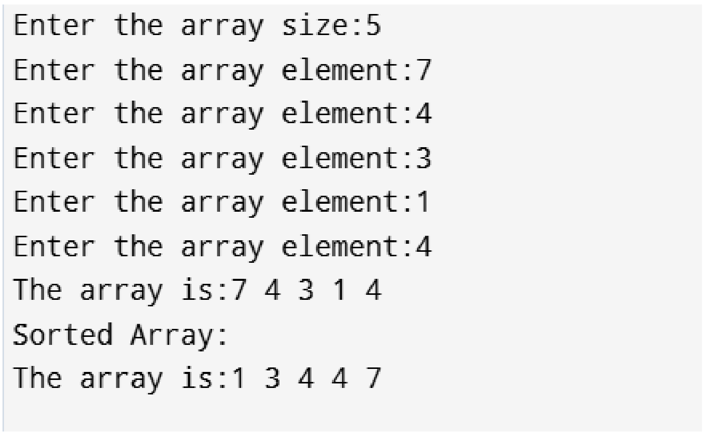
ALGORITHM

```
int partition(int A[], int LB, int len):
    1  j = LB - 1
    2  pivot = A[len - 1]
    3  i = LB
    4  while i < len:
    5      if A[i] < pivot:
    6          j = j + 1
    7          temp = A[i]
    8          A[i] = A[j]
    9          A[j] = temp
    10     i = i + 1
    11 u = A[j + 1]
    12 A[j + 1] = A[len - 1]
    13 A[len - 1] = u
    14 return j + 1

void quicksort(int LB, int len, int A[]):
    1  if LB < len:
    2      p = partition(A, LB, len)
    3      quicksort(LB, p, A)
    4      quicksort(p + 1, len, A)
```

PROGRAM

```
#include<iostream>

using namespace std;
void display(int A[], int UB) {
    cout << "The array is:";
    for (int i = 0; i < UB; i++) {
        cout << A[i] << " ";
    }
    cout << endl;
}

int partition(int A[], int LB, int len) {
    int j = LB - 1;
    int pivot = A[len - 1];
    int i = LB;
    while (i < len) {
        if (A[i] < pivot) {
            j = j + 1;
            int t = A[i];
            A[i] = A[j];
            A[j] = t;
        }
        i = i + 1;
    }
    int u = A[j + 1];
    A[j + 1] = A[len - 1];
    A[len - 1] = u;
    return j + 1;
}
void quicksort(int LB, int len, int A[]) {
    if (LB < len) {
        int p = partition(A, LB, len);
        quicksort(LB, p, A);
        quicksort(p + 1, len, A);
    }
}
int main() {
    cout << "Enter the array size:";
    int n;
    cin >> n;
    int A[n];
    for (int i = 0; i < n; i++) {
        cout << "Enter the array element:";
        int el;
        cin >> el;
        A[i] = el;
    }
    display(A, n);
    quicksort(0, n, A);
    cout << "Sorted Array" << endl;
    display(A, n);
    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the array size:5
Enter the array element:7
Enter the array element:4
Enter the array element:3
Enter the array element:1
Enter the array element:4
The array is:7 4 3 1 4
Sorted Array:
The array is:1 3 4 4 7
```

7.MERGE SORT

24/07/2024

AIM

Write a recursive algorithm for Merge Sort implementation to sort an N-element array.

ALGORITHM

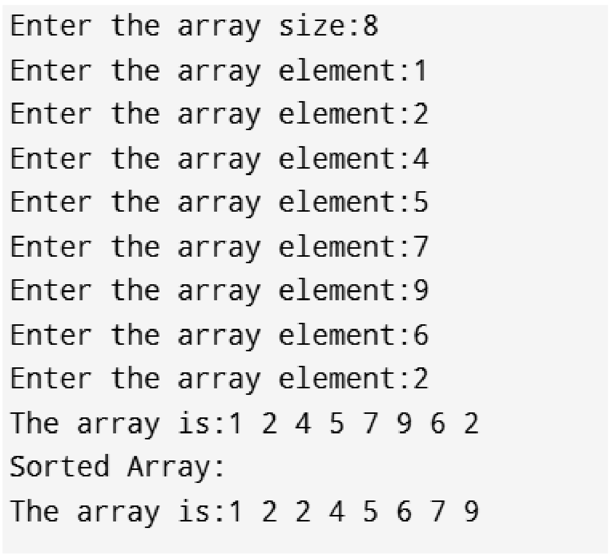
```
void  merge(int LB, int mid, int UB, int A[]):
    1  n1 = mid - LB
    2  n2 = UB - mid
    3  L[n1+1], R[n2+1]
    4  copy A[LB..mid] into L[0..n1-1]
    5  copy A[mid+1..UB] into R[0..n2-1]
    6  L[n1] =
    7  R[n2] =
    8  i = 0
    9  j = 0
    10 for k from LB to UB:
    11     if L[i] < R[j]:
    12         A[k] = L[i]
    13         i = i + 1
    14     else:
    15         A[k] = R[j]
    16         j = j + 1

void  merge_sort(int LB, int UB, int A[]):
    1  if LB < UB:
    2      M = (LB + UB) / 2
    3      merge_sort(LB, M, A)
    4      merge_sort(M + 1, UB, A)
    5      merge(LB, M, UB, A)
```

PROGRAM

```
#include<iostream>

#include<cstring>

#include<limits>
```

```
using namespace std;

void display(int A[], int UB) {
    cout << "The array is:";
    for (int i = 0; i < UB; i++) {
        cout << A[i] << " ";
    }
    cout << endl;
}

void merge(int LB, int mid, int UB, int A[]) {
    int n1 = mid - LB + 1;
    int n2 = UB - mid;
    int L[n1 + 1];
    int R[n2 + 1];
    memcpy(L, & A[LB], n1 * sizeof(int));
    memcpy(R, & A[mid + 1], n2 * sizeof(int));
    L[n1] = R[n2] = numeric_limits < int > ::max();
    int i = 0;
    int j = 0;
    for (int k = LB; k <= UB; k++) {
        if (L[i] < R[j]) {
            A[k] = L[i];
            i = i + 1;
        } else {
            A[k] = R[j];
            j = j + 1;
        }
    }
}

void merge_sort(int LB, int UB, int A[]) {
    if (LB < UB) {
        int M = (LB + UB) / 2;
        merge_sort(LB, M, A);
        merge_sort(M + 1, UB, A);
        merge(LB, M, UB, A);
    }
}

int main() {
    cout << "Enter the array size:";
    int n;
    cin >> n;
    int A[n];
    for (int i = 0; i < n; i++) {
        cout << "Enter the array element:";
        cin >> A[i];
    }
    display(A, n);
    merge_sort(0, n, A);
    cout << "Sorted Array" << endl;
    display(A, n);
    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the array size:8
Enter the array element:1
Enter the array element:2
Enter the array element:4
Enter the array element:5
Enter the array element:7
Enter the array element:9
Enter the array element:6
Enter the array element:2
The array is:1 2 4 5 7 9 6 2
Sorted Array:
The array is:1 2 2 4 5 6 7 9
```

8.HEAP SORT

24/07/2024

AIM

Implement Heap Sort to sort an array of natural numbers.

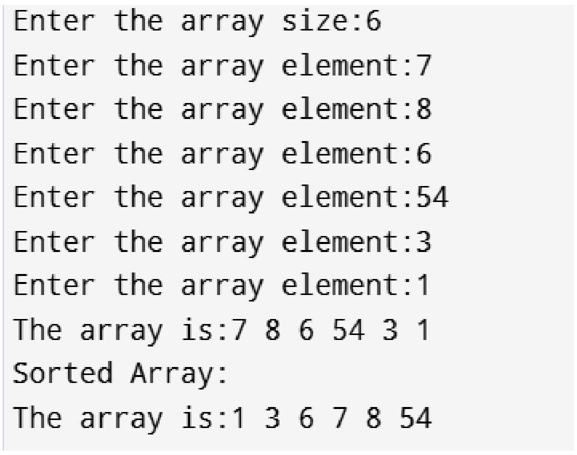
ALGORITHM

```
void heapify(int A[], int i, int n):
    1     largest = i
    2     l = 2 * i + 1
    3     r = 2 * i + 2
    4     if l < n and A[l] > A[largest]:
    5         largest = l
    6     if r < n and A[r] > A[largest]:
    7         largest = r
    8     if largest != i:
    9         swap(A, i, largest)
    10        heapify(A, largest, n)

void heapSort(int A[], int n):
    1     for i from (n - 2) / 2 down to 0:
    2         heapify(A, i, n)
    3     for i from n - 1 down to 0:
    4         swap(A, 0, i)
    5         heapify(A, 0, i)
```

PROGRAM

```
#include<iostream>

using namespace std;

void swap(int A[], int i, int j) {
    int temp = A[i];
    A[i] = A[j];
    A[j] = temp;
}

void heapify(int A[], int i, int n) {
    int largest = i;
    int l = 2 * i + 1;
    int r = 2 * i + 2;
    if (l < n && A[l] > A[largest]) {
        largest = l;
    }
    if (r < n && A[r] > A[largest]) {
        largest = r;
    }
    if (largest != i) {
        swap(A, i, largest);
        heapify(A, largest, n);
    }
}

void heapSort(int A[], int n) {
    for (int i = (n - 2) / 2; i >= 0; i--) {
        heapify(A, i, n);
    }
    for (int i = n - 1; i >= 0; i--) {
        swap(A, 0, i);
        heapify(A, 0, i);
    }
}

void display(int A[], int n) {
    cout << "The array is:";
    for (int i = 0; i < n; i++) {
        cout << A[i] << " ";
    }
    cout << endl;
}

int main() {
    cout << "Enter the array size:";
    int n;
    cin >> n;
    int A[n];
    for (int i = 0; i < n; i++) {
        cout << "Enter the array element:";
        cin >> A[i];
    }
    display(A, n);
    heapSort(A, n);
    cout << "Sorted Array:" << endl;
    display(A, n);
    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the array size:6
Enter the array element:7
Enter the array element:8
Enter the array element:6
Enter the array element:54
Enter the array element:3
Enter the array element:1
The array is:7 8 6 54 3 1
Sorted Array:
The array is:1 3 6 7 8 54
```

9.GRAPH (DFS & BFS TRAVERSAL)

31/07/2024

AIM

To implement DFS and BFS traversals on graphs.

ALGORITHM

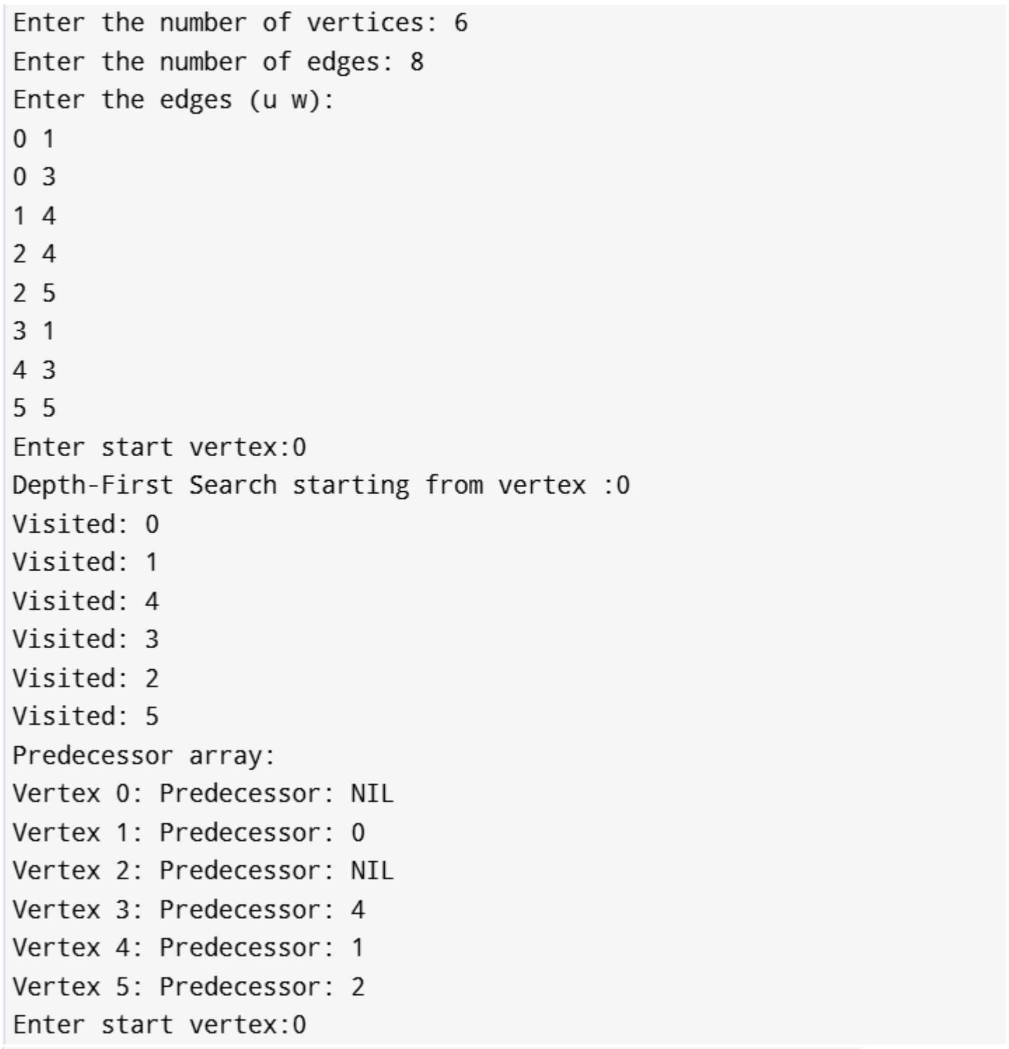
```
void DFS(u):
    1 print "Visited: " + u
    2 visited[u] = 1
    3 for v from 0 to n - 1:
    4   if adj[u][v] == 1 and visited[v] == 0:
    5       predecessor[v] = u
    6       DFS(v)
    7 for i from 0 to n - 1:
    8   if visited[i] != 1:
    9       DFS(i)



void BFS(start):
    1 create a queue Q
    2 Q.push(start)
    3 visited[start] = 1
    4 while Q is not empty:
    5   u = Q.front()
    6   Q.pop()
    7   print "Visited: " + u
    8   for v from 0 to n - 1:
    9       if adj[u][v] == 1 and visited[v] == 0:
    10      visited[v] = 1
    11      predecessor[v] = u
    12      Q.push(v)
    13 for i from 0 to n - 1:
    14 if visited[i] != 1:
    15 BFS(i)
```

PROGRAM

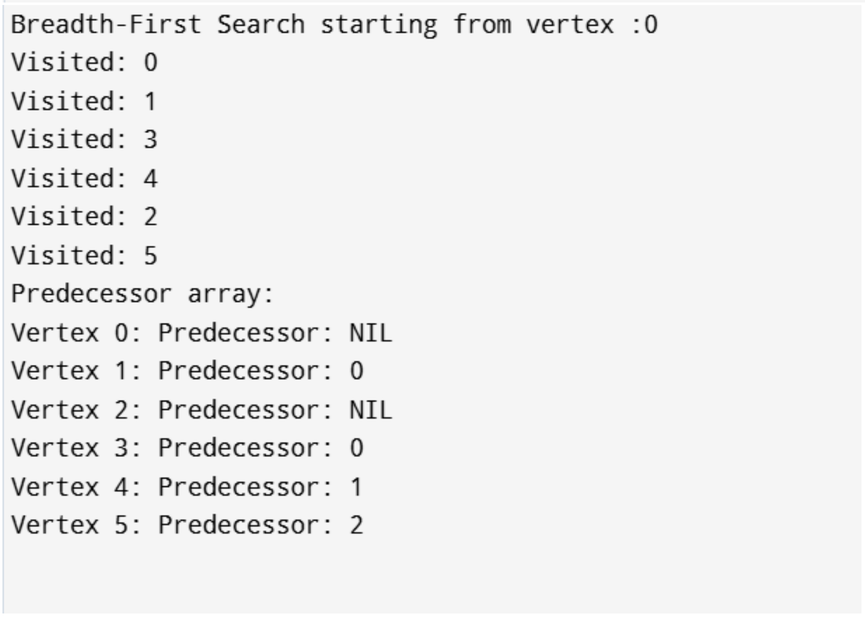
```
#include <iostream>
#include <queue>
```

```
using namespace std;

int adj[10][10]; // Adjacency matrix
int visited[10]; // Visited array
int predecessor[10]; // Predecessor array
int n; // Number of vertices

// DFS function
void DFS(int u) {
    cout << "Visited: " << u << endl;
    visited[u] = 1; // Mark the current vertex as visited
    for (int v = 0; v < n; v++) {
        if (adj[u][v] == 1 && visited[v] == 0) {
            predecessor[v] = u; // Set the predecessor of v
            DFS(v); // Recursive call to visit the connected vertex
        }
    }
}

// Clear arrays
void clear() {
    fill(visited, visited + n, 0); // Reset visited array
    fill(predecessor, predecessor + n, -1); // Reset predecessor array
}
// BFS function
void BFS(int start) {
    queue < int > Q;

    Q.push(start);
    visited[start] = 1;

    while (!Q.empty()) {
        int u = Q.front();
        Q.pop();
        cout << "Visited: " << u << endl;

        for (int v = 0; v < n; v++) {
            if (adj[u][v] == 1 && visited[v] == 0) {
                visited[v] = 1;
                predecessor[v] = u;
                Q.push(v);
            }
        }
    }
}

int main() {
    cout << "Enter the number of vertices: ";
    cin >> n; // Assign number of vertices to global variable n

    // Initialize adjacency matrix, visited array, and predecessor array
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            adj[i][j] = 0; // No edges initially
        }
        visited[i] = 0; // Not visited initially
        predecessor[i] = -1; // No predecessor initially
    }

    int edges, u, w;
    cout << "Enter the number of edges: ";
    cin >> edges;
    cout << "Enter the edges (u w):" << endl;
    for (int i = 0; i < edges; i++) {
        cin >> u >> w;
        adj[u][w] = 1; // Mark edge in adjacency matrix
    }

    // Perform DFS starting from vertex 0
    cout << "Enter start vertex:";
    int st;
    cin >> st;
    cout << "Depth-First Search starting from vertex :" << st << endl;
    DFS(st);

    // Print the predecessor array
    cout << "Predecessor array:" << endl;
    for (int i = 0; i < n; i++) {
        cout << "Vertex " << i << ":";
        if (predecessor[i] == -1) {
            cout << " Predecessor: NIL" << endl;
        } else {
            cout << " Predecessor: " << predecessor[i] << endl;
        }
    }
    clear();
    cout << "Enter start vertex:";
    int s;
    cin >> s;
    // Perform BFS starting from vertex 0
    cout << "Breadth-First Search starting from vertex :" << s << endl;
    BFS(s);
    // Print the predecessor array
    cout << "Predecessor array:" << endl;
    for (int i = 0; i < n; i++) {
        cout << "Vertex " << i << ":";
        if (predecessor[i] == -1) {
            cout << " Predecessor: NIL" << endl;
        } else {
            cout << " Predecessor: " << predecessor[i] << endl;
        }
    }
    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the number of vertices: 6
Enter the number of edges: 8
Enter the edges (u w):
0 1
0 3
1 4
2 4
2 5
3 1
4 3
5 5
Enter start vertex:0
Depth-First Search starting from vertex :0
Visited: 0
Visited: 1
Visited: 4
Visited: 3
Visited: 2
Visited: 5
Predecessor array:
Vertex 0: Predecessor: NIL
Vertex 1: Predecessor: 0
Vertex 2: Predecessor: NIL
Vertex 3: Predecessor: 4
Vertex 4: Predecessor: 1
Vertex 5: Predecessor: 2
Enter start vertex:0
```



```
Breadth-First Search starting from vertex :0
Visited: 0
Visited: 1
Visited: 3
Visited: 4
Visited: 2
Visited: 5
Predecessor array:
Vertex 0: Predecessor: NIL
Vertex 1: Predecessor: 0
Vertex 2: Predecessor: NIL
Vertex 3: Predecessor: 0
Vertex 4: Predecessor: 1
Vertex 5: Predecessor: 2
```

10.PRIMS'S ALGORITHM

07/08/2024

AIM

To find the minimum spanning tree of a graph using Prim's Algorithm.

ALGORITHM

```
MST-Prim(G, w, r)
    1  for each u  G.V
    2      u.key = infinity
    3      u. = NIL
    4  r.key = 0
    5  Q = G.V
    6  while Q  phi
    7      u = EXTRACT-MIN(Q)
    8      for each v  G.Adj[u]
    9          if v  Q and w(u, v) < v.key
    10             v.pi = u
    11             v.key = w(u, v)
```

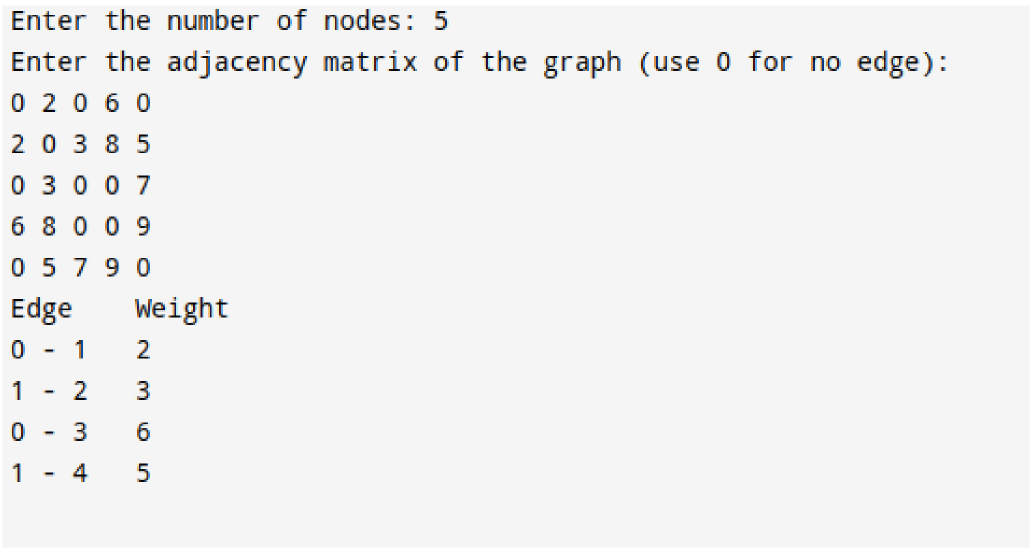
PROGRAM

```
#include <iostream>

#include <climits>

using namespace std;
// Function to find the vertex with the minimum key value
int extractMin(int key[], bool inMST[], int nodes) {
    int minKey = INT_MAX, minIndex = -1;

    for (int i = 0; i < nodes; i++) {
        if (!inMST[i] && key[i] < minKey) {
            minKey = key[i];
            minIndex = i;
        }
    }
    return minIndex;
}
```

```
void primMST(int graph[100][100], int nodes) {
    int parent[100];  // Array to store the MST
    int key[100];     // Key values to pick minimum weight edge
    bool inMST[100];  // To represent vertices included in MST

    //  Initialize all keys to infinity and inMST to false
    for (int i = 0; i < nodes; i++) {
        key[i] = INT_MAX;
        inMST[i] = false;
    }

    //  Start with the first node (arbitrary root)
    key[0] = 0;       // Key of the root node is 0
    parent[0] = -1;   // Root node has no parent

    //  Loop to construct the MST
    for (int count = 0; count < nodes - 1; count++) {
        // Pick the minimum key vertex not yet included in MST
        int u = extractMin(key, inMST, nodes);

        // Include the vertex in MST
        inMST[u] = true;

        // Update the key and parent of adjacent vertices
        for (int v = 0; v < nodes; v++) {
            // If the edge exists, and vertex v is not in MST,
            // and the weight is smaller than the current key
            if (graph[u][v] && !inMST[v] && graph[u][v] < key[v]) {
                parent[v] = u;
                key[v] = graph[u][v];
            }
        }
    }

    //  Print the MST
    cout << "Edge \tWeight\n";
    for (int i = 1; i < nodes; i++) {
        cout << parent[i] << " - " << i << " \t" << graph[i][parent[i]] << "\n";
    }
}
```

```
int main() {
    int nodes;
    cout << "Enter the number of nodes: ";
    cin >> nodes;

    int graph[100][100];

    cout << "Enter the adjacency matrix of the graph (use 0 for no edge):\n";
    for (int i = 0; i < nodes; i++) {
        for (int j = 0; j < nodes; j++) {
            cin >> graph[i][j];
        }
    }

    primMST(graph, nodes);

    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the number of nodes: 5
Enter the adjacency matrix of the graph (use 0 for no edge):
0 2 0 6 0
2 0 3 8 5
0 3 0 0 7
6 8 0 0 9
0 5 7 9 0
Edge    Weight
0 - 1   2
1 - 2   3
0 - 3   6
1 - 4   5
```

10.KRUSKAL'S ALGORITHM

14/08/2024

AIM

To find minimum spanning tree of a graph using Kruskal's Algorithm.

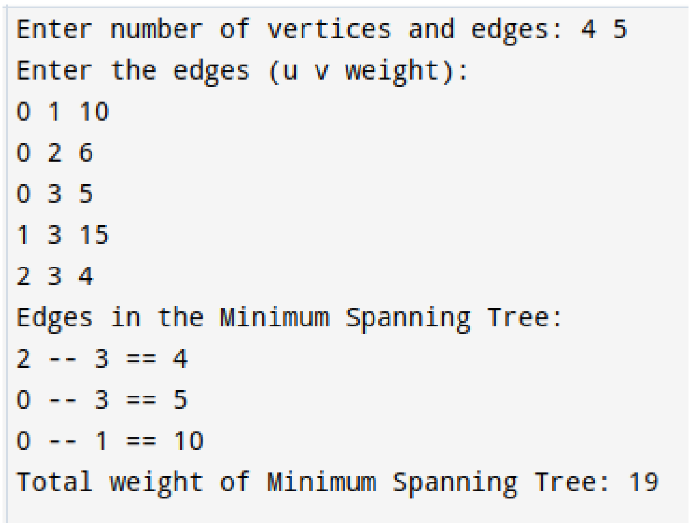
ALGORITHM

```
MST-KRUSKAL(G, w)
    1  A = phi
    2  for each vertex v in G.V
    3      MAKE-SET(v)
    4  sort the edges of G.E into nondecreasing order by weight w
    5  for each edge (u, v)  G.E, taken in nondecreasing order by weight
    6      if FIND-SET(u) != FIND-SET(v)
    7          A = A U {(u, v)}
    8          UNION(u, v)
    9  return A
```

PROGRAM

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

using namespace std;

// Structure to represent an edge in the graph
struct Edge {
    int u, v, weight;
};

// Function to find the parent of a vertex (using union-find)
int findSet(int parent[], int v) {
    if (parent[v] == v)
        return v;
    return findSet(parent, parent[v]);
}

// Function to perform union of two sets
void unionSet(int parent[], int rank[], int u, int v) {
    int rootU = findSet(parent, u);
    int rootV = findSet(parent, v);

    if (rank[rootU] > rank[rootV]) {
        parent[rootV] = rootU;
    } else if (rank[rootU] < rank[rootV]) {
        parent[rootU] = rootV;
    } else {
        parent[rootV] = rootU;
        rank[rootU]++;
    }
}

// Comparison function to sort edges by weight
bool compareEdges(Edge e1, Edge e2) {
    return e1.weight < e2.weight;
}

void kruskal(Edge edges[], int V, int E) {
    // Array to store the parent of each vertex
    int parent[V];
    int rank[V]; // Rank array for union by rank
    for (int i = 0; i < V; i++) {
        parent[i] = i;
        rank[i] = 0;
    }

    // Sort edges by weight
    sort(edges, edges + E, compareEdges);

    cout << "Edges in the Minimum Spanning Tree:" << endl;
    int mstWeight = 0;

    for (int i = 0; i < E; i++) {
        int u = edges[i].u;
        int v = edges[i].v;
        int weight = edges[i].weight;

        // Check if including this edge creates a cycle
        if (findSet(parent, u) != findSet(parent, v)) {
            cout << u << " -- " << v << " == " << weight << endl;
            mstWeight += weight;
            unionSet(parent, rank, u, v);
        }
    }

    cout << "Total weight of Minimum Spanning Tree: " << mstWeight << endl;
}

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

    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;
    }

    kruskal(edges, V, E);

    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter number of vertices and edges: 4 5
Enter the edges (u v weight):
0 1 10
0 2 6
0 3 5
1 3 15
2 3 4
Edges in the Minimum Spanning Tree:
2 -- 3 == 4
0 -- 3 == 5
0 -- 1 == 10
Total weight of Minimum Spanning Tree: 19
```

12.DIJKSTRA'S ALGORITHM

04/09/2024

AIM

To implement single source shortest path problems on a graph(with non negative edge weights) using Dijkstra's Algorithm.

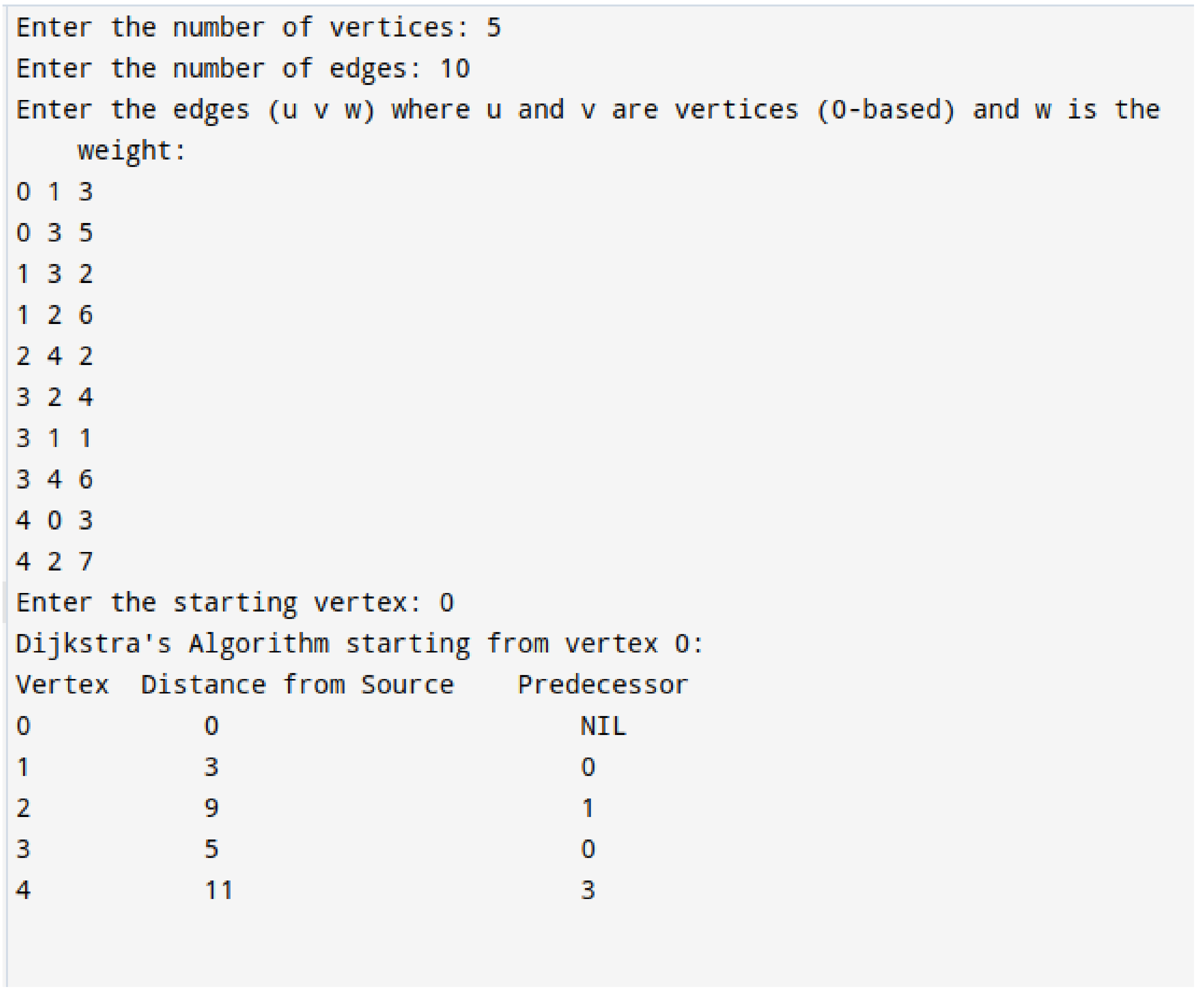
ALGORITHM

```
Initialize-Single-Source(G, s)
    1  for each vertex v  G.V
    2      v.d = infinity
    3      v.pi= NIL
    4  s.d = 0

Relax(u, v, w)
    1  if v.d > u.d + w(u, v)
    2      v.d = u.d + w(u, v)
    3      v.pi = u

Dijkstra(G, w, s)
    1  Initialize-Single-Source(G, s)
    2  S =phi
    3  Q = G.V
    4  while Q  0
    5      u = Extract-Min(Q)
    6      S = S  {u}
    7      for each vertex v  G.Adj[u]
    8          Relax(u, v, w)
```

PROGRAM

```
#include <iostream>
#include <climits> // For INT_MAX
using namespace std;

// Function to initialize the adjacency matrix
void initializeGraph(int adj[100][100], int V) {
    for (int i = 0; i < V; i++) {
        for (int j = 0; j < V; j++) {
            adj[i][j] = 0; // No edge initially
        }
    }
}

// Function to add an edge to the adjacency matrix
void addEdge(int adj[100][100], int u, int v, int w) {
    adj[u][v] = w; // Directed graph
}

// Function to find the vertex with the minimum distance
int findMinDistance(int distance[], bool visited[], int V) {
    int minDist = INT_MAX, minIndex = -1;

    for (int i = 0; i < V; i++) {
        if (!visited[i] && distance[i] < minDist) {
            minDist = distance[i];
            minIndex = i;
        }
    }
    return minIndex;
}

// Dijkstra's algorithm function
void dijkstra(int adj[100][100], int V, int src) {
    int distance[100];    // Array to store shortest distances from src
    bool visited[100];    // Array to keep track of visited vertices
    int predecessor[100]; // Array to store the predecessor of each vertex

    // Initialize all distances as infinite and visited array as false
    for (int i = 0; i < V; i++) {
        distance[i] = INT_MAX;
        visited[i] = false;
        predecessor[i] = -1; // No predecessor initially
    }

    distance[src] = 0; // Distance to source is 0

    // Main loop of Dijkstra's algorithm
    for (int count = 0; count < V - 1; count++) {
        // Find the unvisited vertex with the smallest distance
        int u = findMinDistance(distance, visited, V);
        if (u == -1) break; // If no valid vertex found, exit loop
        visited[u] = true; // Mark as visited

        // Update the distance and predecessor for adjacent vertices
        for (int v = 0; v < V; v++) {
            if (adj[u][v] != 0 && !visited[v] &&
            distance[u] + adj[u][v] < distance[v]) {
                distance[v] = distance[u] + adj[u][v];
                predecessor[v] = u;
            }
        }
    }

    // Output the distances and predecessors
    cout << "Vertex\tDistance from Source\tPredecessor\n";
    for (int i = 0; i < V; i++) {
        cout << i << "\t" << distance[i] << "\t\t\t";
        if (predecessor[i] == -1) {
            cout << "NIL" << endl;
        } else {
            cout << predecessor[i] << endl;
        }
    }
}

int main() {
    int V, E;
    cout << "Enter the number of vertices: ";
    cin >> V;
    if (V > 100) {
        cout << "Maximum number of vertices exceeded. Exiting...\n";
        return 1;
    }
    int adj[100][100];
    // Initialize the graph
    initializeGraph(adj, V);
    cout << "Enter the number of edges: ";
    cin >> E;
    // Input the edges and weights
    cout << "Enter the edges (u v w) where u and v are
    vertices (0-based) and w is the weight:\n";
    for (int i = 0; i < E; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        addEdge(adj, u, v, w);
    }

    int startVertex;
    cout << "Enter the starting vertex: ";
    cin >> startVertex;

    if (startVertex < 0 || startVertex >= V) {
        cout << "Invalid starting vertex. Exiting...\n";
        return 1;
    }

    cout << "Dijkstra's Algorithm starting from vertex " << startVertex << ":\n";
    dijkstra(adj, V, startVertex);

    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the number of vertices: 5
Enter the number of edges: 10
Enter the edges (u v w) where u and v are vertices (0-based) and w is the
    weight:
0 1 3
0 3 5
1 3 2
1 2 6
2 4 2
3 2 4
3 1 1
3 4 6
4 0 3
4 2 7
Enter the starting vertex: 0
Dijkstra's Algorithm starting from vertex 0:
Vertex  Distance from Source    Predecessor
0           0                       NIL
1           3                       0
2           9                       1
3           5                       0
4           11                      3
```

13.BELLMAN-FORD ALGORITHM

09/09/2024

AIM

To implement single source shortest path problem on a graph(with negative weighted edges) using Bellman-Ford Algorithm.

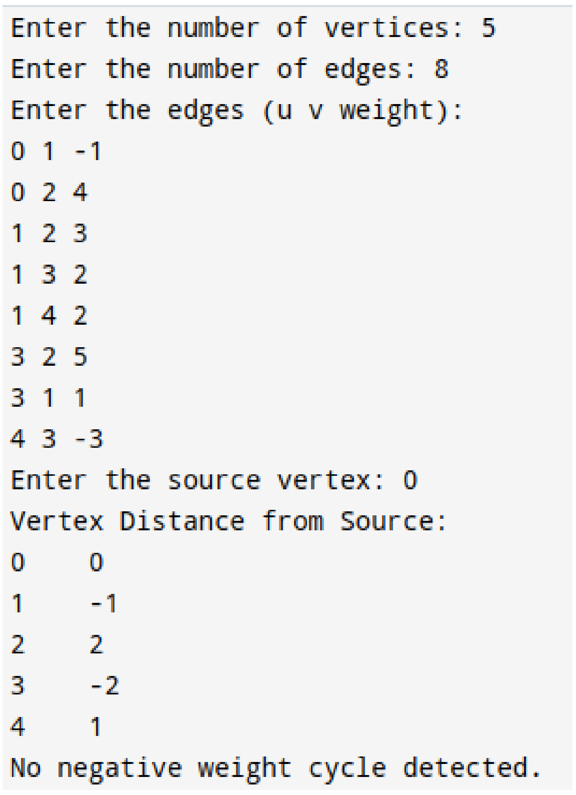
ALGORITHM

```
Bellman-Ford(G, w, s)
    1. Initialize-Single-Source(G, s)
    2. for i = 1 to |G.V| - 1 do
    3.     for each edge (u, v) in G.E do
    4.         Relax(u, v, w)
    5. for each edge (u, v) in G.E do
    6.     if v.d > u.d + w(u, v) then
    7.         return FALSE
    8. return TRUE
```

PROGRAM

```
#include <iostream>
#include <climits> // For INT_MAX

using namespace std;

// Function to implement Bellman-Ford algorithm
bool bellmanFord(int vertices, int edges, int edgeList[][3], int source) {
    //  Initialize distances to all vertices as infinite and the source distance to 0
    int distance[vertices];
    for (int i = 0; i < vertices; i++) {
        distance[i] = INT_MAX;
    }
    distance[source] = 0;

    // Relax all edges |V| - 1 times
    for (int i = 1; i <= vertices - 1; i++) {
        for (int j = 0; j < edges; j++) {
            int u = edgeList[j][0];
            int v = edgeList[j][1];
            int weight = edgeList[j][2];
            if (distance[u] != INT_MAX && distance[u] + weight < distance[v]) {
                distance[v] = distance[u] + weight;
            }
        }
    }

    //  Check for negative weight cycles
    for (int j = 0; j < edges; j++) {
        int u = edgeList[j][0];
        int v = edgeList[j][1];
        int weight = edgeList[j][2];

        if (distance[u] != INT_MAX && distance[u] + weight < distance[v]) {
            return false; // Negative weight cycle found
        }
    }

    // Print the results
    cout << "Vertex Distance from Source:" << endl;
    for (int i = 0; i < vertices; i++) {
        if (distance[i] == INT_MAX) {
            cout << i << " \t INF" << endl; // Replace "INF" directly in output
        } else {
            cout << i << " \t " << distance[i] << endl;
        }
    }
    return true; // No negative weight cycles
}

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

    int edgeList[edges][3]; // To store edges in the form {u, v, weight}
    cout << "Enter the edges (u v weight):" << endl;
    for (int i = 0; i < edges; i++) {
        cin >> edgeList[i][0] >> edgeList[i][1] >> edgeList[i][2];
    }

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

    if (bellmanFord(vertices, edges, edgeList, source)) {
        cout << "No negative weight cycle detected." << endl;
    } else {
        cout << "Negative weight cycle detected!" << endl;
    }

    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the number of vertices: 5
Enter the number of edges: 8
Enter the edges (u v weight):
0 1 -1
0 2 4
1 2 3
1 3 2
1 4 2
3 2 5
3 1 1
4 3 -3
Enter the source vertex: 0
Vertex Distance from Source:
0    0
1    -1
2    2
3    -2
4    1
No negative weight cycle detected.
```

14.FLOYD-WARSHALL ALGORITHM

09/09/2024

AIM

To implement all pair shortest path problem on graph using Floyd-Warshall Algorithm(Assuming there are no negative cycles).

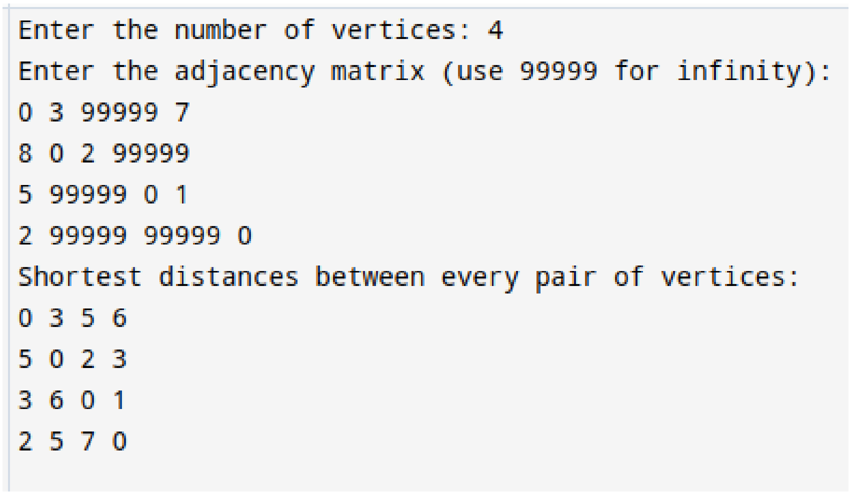
ALGORITHM

```
Floyd-Warshall(W)  //matrix containing edge weights
1. n = number of vertices in the graph
2. D = W  // Initialize D as the weight matrix W
3. for k = 1 to n do
4.     for i = 1 to n do
5.         for j = 1 to n do
6.             D[i][j] = min(D[i][j], D[i][k] + D[k][j])
7. return D
```

PROGRAM

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

#define INF 99999  // Define a large value to represent infinity (no direct connection)

void floydWarshall(int graph[][100], int n) {
    int dist[100][100];

    // Initialize the solution matrix same as the input graph matrix
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            dist[i][j] = graph[i][j];
        }
    }

    // Apply Floyd-Warshall algorithm
    for (int k = 0; k < n; k++) {          // Pick intermediate vertex k
        for (int i = 0; i < n; i++) {      // Pick source vertex
            for (int j = 0; j < n; j++) {  // Pick destination vertex
                // Update the distance if the path through k is shorter
                if (dist[i][k] != INF && dist[k][j] !=
                INF && dist[i][k] + dist[k][j] < dist[i][j]) {
                    dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
    }

    // Print the resulting shortest path matrix
    cout << "Shortest distances between every pair of vertices:\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            if (dist[i][j] == INF)
                cout << "INF ";
            else
                cout << dist[i][j] << " ";
        }
        cout << endl;
    }
}

int main() {
    int n;
    cout << "Enter the number of vertices: ";
    cin >> n;

    int graph[100][100];
    cout << "Enter the adjacency matrix (use " << INF << " for infinity):\n";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < n; j++) {
            cin >> graph[i][j];
        }
    }

    floydWarshall(graph, n);

    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the number of vertices: 4
Enter the adjacency matrix (use 99999 for infinity):
0 3 99999 7
8 0 2 99999
5 99999 0 1
2 99999 99999 0
Shortest distances between every pair of vertices:
0 3 5 6
5 0 2 3
3 6 0 1
2 5 7 0
```

15.DYNAMIC FIBANOCCI SERIES

25/09/2024

AIM

Implement a dynamic programming solution for calculating the nth term of the Fibonacci series.

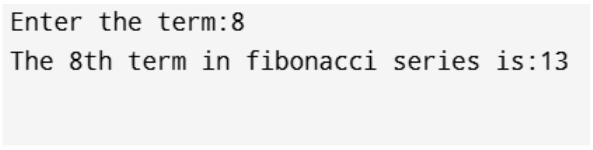
ALGORITHM

```
int Fibonacci(n)  \\n->number of terms
    1 set fib[0] = 0
    2 set fib[1] = 1
    3 for i from 2 to n
    4     fib[i] = fib[i-1] + fib[i-2]
    5 return fib[n]
```

PROGRAM

```
#include<iostream>

using namespace std;
int Fib(int n) {
    if (n <= 1) {
        return 0;
    }
    int fib[n + 1];
    fib[0] = 0;
    fib[1] = 1;
    for (int i = 2; i < n + 1; i++) {
        fib[i] = fib[i - 1] + fib[i - 2];
    }
    return fib[n - 1];
}
int main() {
    int n;
    cout << "Enter the term:";
    cin >> n;
    int F = Fib(n);
    cout << "The " << n << "th term in fibonacci series is:" << F << endl;
    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the term:8
The 8th term in fibonacci series is:13
```

16.COIN ROW PROBLEM

25/09/2024

AIM

Write a program to solve the following problem :Given a number of coins in unsorted order,find the maximum value we can achieve using the least amount of coins and no two adjacent coins are chosen during the process.

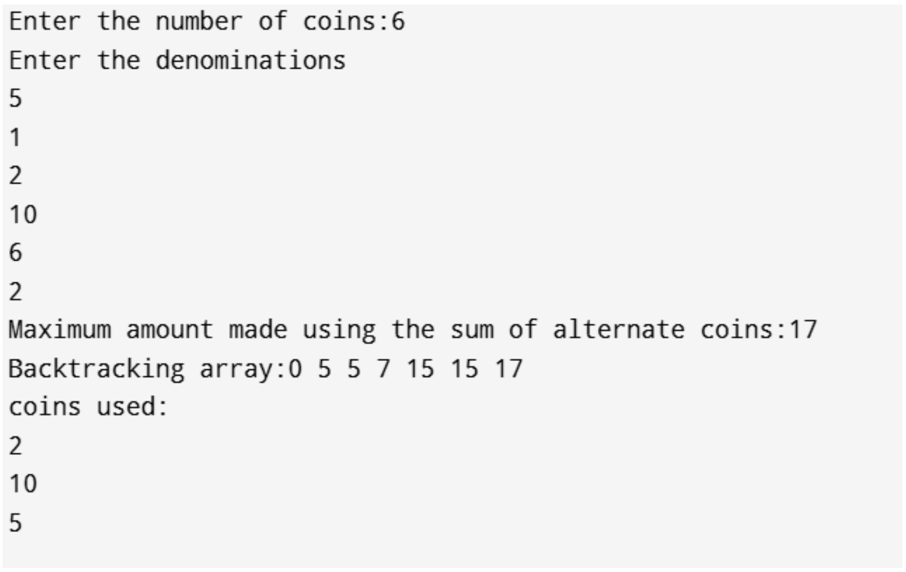
ALGORITHM

```
int  CoinRow(coins) \\coins->Array of coins
    1 Initialize an array F of size n + 1
    2 Set F[0] = 0
    3 Set F[1] = coins[0]
    4 For i from 2 to n
    5     F[i] = max(coins[i-1] + F[i-2], F[i-1])
    6 End For
    7 Return F[n]
    8 End Function
```

PROGRAM

```
#include<iostream>

using namespace std;
int max(int a, int b) {
    if (a >= b) {
        return a;
    } else {
        return b;
    }
}
int coinRow(int A[], int n) {
    int F[n + 1];
    F[0] = 0;
    F[1] = A[0];
    for (int i = 2; i <= n; i++) {
        F[i] = max(A[i - 1] + F[i - 2], F[i - 1]);

    }
    cout << "Maximum amount made using the sum of alternate coins:" << F[n] << endl;
    cout << "Backtracking array:";
    for (int i = 0; i <= n; i++) {
        cout << F[i] << " ";
    }
    cout << endl;
    cout << "coins used:" << endl;
    int i = n;
    while (i > 0) {
        if (i == 1 or F[i] != F[i - 1]) {
            cout << A[i - 1] << endl;
            i = i - 2;
        } else {
            i = i - 1;
        }
    }
    return F[n];
}
int main() {
    cout << "Enter the number of coins:";
    int n;
    cin >> n;
    int a[n];
    cout << "Enter the denominations" << endl;
    for (int i = 0; i < n; i++) {
        cin >> a[i];
    }
    coinRow(a, n);
}
```

SAMPLE INPUT-OUTPUT



```
Enter the number of coins:6
Enter the denominations
5
1
2
10
6
2
Maximum amount made using the sum of alternate coins:17
Backtracking array:0 5 5 7 15 15 17
coins used:
2
10
5
```

17.COIN CHANGE PROBLEM

16/10/2024

AIM

To determine the minimum number of coins required to make a given amount using specified denominations.

ALGORITHM

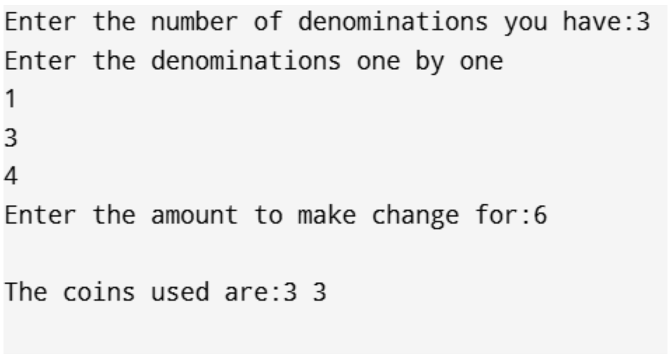
```
int coinchange(d[], n, m)  //d->denominations, n->amount, m->no of coins
    1 initialize arrays f[n+1] and b[n+1]
    2 set f[0] = 0 and b[0] = -1
    3 for i from 1 to n
    4     set temp = infinity, b = -1
    5     for j from 0 to m-1
    6         if i >= d[j] and temp > f[i - d[j]]
    7             temp = f[i - d[j]], b = j
    8     set f[i] = temp + 1, b[i] = b
    9 print array f
    10 print array b
    11 set s = f[n], create array arr of size s, k = 0
    12 while n > 0 and k < s
    13     coinindex = b[n]
    14     set c = d[coinindex], arr[k] = c
    15     n = n - c, k = k + 1
    16 print "the coins used are:" and array arr
```

PROGRAM

```
#include<iostream>

#include<limits>

using namespace std;
int min(int a, int b) {
    if (a < b) {
        return a;
    } else {
        return b;
    }
```

```
}
void display(int D[], int n) {
    for (int i = 0; i < n; i++) {
        cout << D[i] << " ";
    }
    cout << endl;
}
int coinChange(int D[], int n, int m) {
    int F[n + 1];
    int B[n + 1];
    F[0] = 0;
    B[0] = -1;
    for (int i = 1; i <= n; i++) {
        int temp = numeric_limits < int > ::max();
        int j = 0;
        int b = -1;
        while (j < m and i >= D[j]) {
            if (temp > F[i - D[j]]) {
                temp = F[i - D[j]];
                b = j;
            }
            //temp=min(F[i-D[j]],temp);
            j = j + 1;
        }
        F[i] = temp + 1;
        B[i] = b;

    }
    for (int i = 0; i <= n; i++) {
        cout << F[i] << " ";
    }
    cout << endl;
    for (int i = 0; i <= n; i++) {
        cout << B[i] << " ";
    }
    cout << endl;
    int s = F[n];
    int Arr[s];
    int k = 0;
    while (n > 0 and k < s) {
        int coinIndex = B[n];
        int c = D[coinIndex];
        Arr[k] = c;
        n = n - c;
        k++;
    }
    cout << "The coins used are:";
    for (int i = 0; i < s; i++) {
        cout << Arr[i] << " ";
    }
    cout << endl;
}
int main() {
    cout << "Enter the number of denominations you have:";
    int num;
    cin >> num;
    int D[num];
    cout << "Enter the denominations one by one" << endl;
    for (int i = 0; i < num; i++) {
        int den;
        cin >> den;
        D[i] = den;
    }
    cout << "Enter the amount to make change for:";
    int amt;
    cin >> amt;
    coinChange(D, amt, num);
}
```

SAMPLE INPUT-OUTPUT



```
Enter the number of denominations you have:3
Enter the denominations one by one
1
3
4
Enter the amount to make change for:6

The coins used are:3 3
```

18.COIN COLLECTING

16/10/2024

AIM

Several coins are placed in cells of an n × m board, no more than one coin per cell. A robot, located in the upper left cell of the board, needs to collect as many of the coins as possible and bring them to the bottom right cell. On each step, the robot can move either one cell to the right or one cell down from its current location. When the robot visits a cell with a coin, it always picks up that coin.

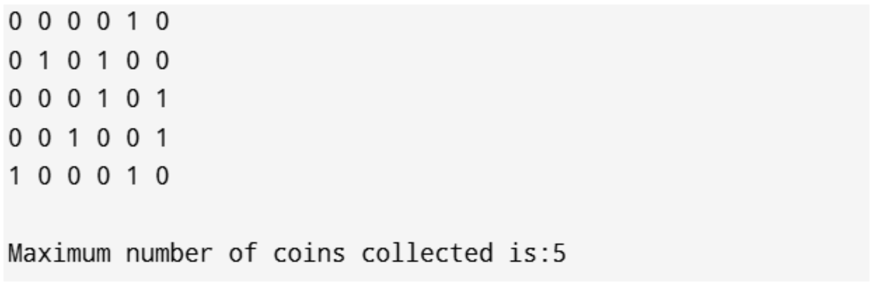
ALGORITHM

```
int coincollect(c[][]) //c->matrix of coins
    1 initialize f[5][6]
    2 for i from 0 to 4
    3     for j from 0 to 5
    4         set f[i][j] = c[i][j]
    5 for j from 1 to 5
    6     set f[0][j] = c[0][j] + f[0][j-1]
    7 for i from 1 to 4
    8     set f[i][0] = c[i][0] + f[i-1][0]
    9 for i from 1 to 4
    10    for j from 1 to 5
    11        set temp = max(f[i][j-1], f[i-1][j])
    12        set f[i][j] = temp + c[i][j]
    13 return f[4][5]
```

PROGRAM

```cpp
#include<iostream>

using namespace std;
void display(int mat[5][6]) {
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 6; j++) {
            cout << mat[i][j] << " ";
        }
        cout << endl;
    }
}
int coinCollect(int c[5][6]) {
    int f[5][6];
    for (int i = 0; i < 5; i++) {
        for (int j = 0; j < 6; j++) {
            f[i][j] = c[i][j];
        }
    }
    for (int j = 1; j < 6; j++) {
        f[0][j] = c[0][j] + f[0][j - 1];
    }
    for (int i = 1; i < 5; i++) {
        f[i][0] = c[i][0] + f[i - 1][0];
    }
    for (int i = 1; i < 5; i++) {
        for (int j = 1; j < 6; j++) {

            int temp = -1;
            if (f[i][j - 1] > f[i - 1][j]) {
                temp = f[i][j - 1];
            } else {
                temp = f[i - 1][j];
            }
            f[i][j] = temp + c[i][j];
        }
    }

    return f[4][5];
}
int main() {
    int inp[5][6] = {
        {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}
    };
    display(inp);
    cout << endl;
    cout << "Maximum number of coins collected is:" << coinCollect(inp);
}
```

SAMPLE INPUT-OUTPUT



```
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 number of coins collected is:5
```

19.MINIMUM COST PATH

30/10/2024

AIM

Given a cost matrix having a cost at each cell. Find the minimum cost it will take to reach cell (m, n) from top left corner cell (0, 0) if the only allowed directions to move from a cell are right, down and diagonally down.

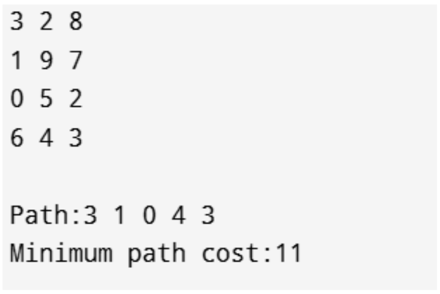
ALGORITHM

```
int mcp(mat, n, c)  //mat->matrix of path costs, n->rows, c->columns
    1 initialize f[n][c] by copying mat to f
    2 f[0][0] = mat[0][0]
    3 for i from 0 to n-1
    4     for j from 0 to c-1
    5         if i == 0 and j > 0
    6             f[i][j] = f[i][j-1] + mat[i][j]
    7         else if j == 0 and i > 0
    8             f[i][j] = f[i-1][j] + mat[i][j]
    9         else if i > 0 and j > 0
    10            f[i][j] = min(f[i-1][j], f[i][j-1], f[i-1][j-1]) + mat[i][j]
    11 i = n-1, j = c-1, b = empty array, ind = 0
    12 b[ind++] = mat[i][j]
    13 while i > 0 or j > 0
    14     if i > 0 and j > 0 and f[i][j] == f[i-1][j-1] + mat[i][j]
    15         i--, j--
    16     else if i > 0 and f[i][j] == f[i-1][j] + mat[i][j]
    17         i--
    18     else
    19         j--
    20 print "path:" and elements of b in reverse order
    21 return f[n-1][c-1]
```

PROGRAM

```
#include<iostream>

using namespace std;

void display(int mat[4][3]) {
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 3; j++) {
            cout << mat[i][j] << " ";
        }
        cout << endl;
    }
}
int min(int a, int b, int c) {
    if (a <= b and a <= c) {
        return a;
    } else if (b <= a and b <= c) {
        return b;
    } else {
        return c;
    }
}
int mcp(int mat[4][3], int n, int c) {
    int F[n][c];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < c; j++) {
            F[i][j] = mat[i][j];
        }
    }
    F[0][0] = mat[0][0];
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < c; j++) {
            if (i == 0 and j > 0) {
                F[i][j] = F[i][j - 1] + mat[i][j];
            }
            if (j == 0 and i > 0) {
                F[i][j] = F[i - 1][j] + mat[i][j];
            }
            if (i > 0 and j > 0) {
                F[i][j] = min(F[i - 1][j], F[i][j - 1], F[i - 1][j - 1]) + mat[i][j];
            }
        }
    }
    // backtracking
    int i = n - 1;
    int j = c - 1;
    int B[100];
    int ind = 0;
    B[ind++] = mat[i][j];
    while (i > 0 or j > 0) {
        if (i > 0 and j > 0 and F[i][j] == F[i - 1][j - 1] + mat[i][j]) {
            i--;
            j--;

        } else if (i > 0 and F[i][j] == F[i - 1][j] + mat[i][j]) {
            i--;
        } else {
            j--;
        }
        B[ind++] = mat[i][j];
    }
    cout << "Path:";
    for (int k = ind - 1; k >= 0; k--) {
        cout << B[k] << " ";
    }
    cout << endl;
    return F[n - 1][c - 1];
}
int main() {
    int inp[4][3]={{3, 2, 8},
    {1, 9, 7},
    {0, 5, 2},
    {6, 4, 3}};
    display(inp);
    cout << endl;
    cout << mcp(inp, 4, 3);
}
```

SAMPLE INPUT-OUTPUT



```
3 2 8
1 9 7
0 5 2
6 4 3

Path:3 1 0 4 3
Minimum path cost:11
```

20.0/1 KNAPSACK

30/10/2024

AIM

Given n items of known weights w1, . . . , wn and values v1, . . . , vn and a knapsack of capacity W, find the most valuable subset of the items that fit into the knapsack.

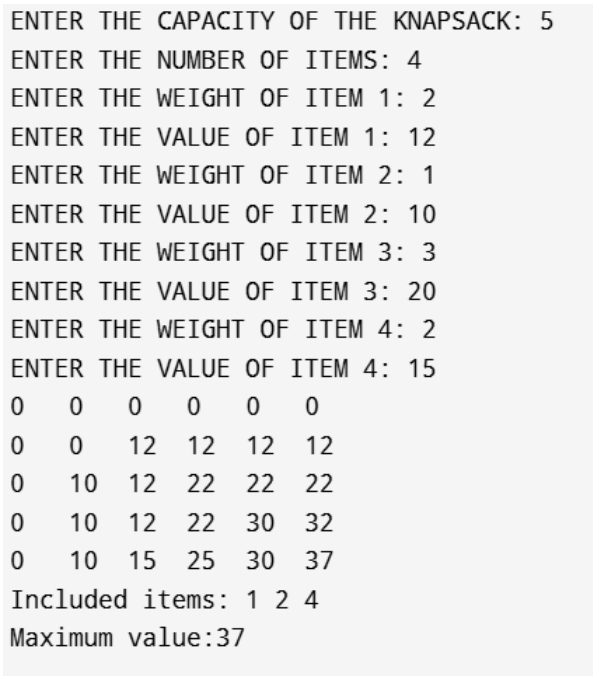
ALGORITHM

```
int knapsack(n, c, w[], v[])\\c->capacity, n->number of items, w->weights, v->values
    1 mat ← create 2d array of size (n+1) x (c+1)
    2 for i from 0 to n:
    3     mat[i][0] = 0
    4 for i from 0 to c:
    5     mat[0][i] = 0
    6 for i from 1 to n:
    7     for j from 1 to c:
    8         if j - w[i - 1] >= 0:
    9             mat[i][j] = max(mat[i - 1][j], v[i - 1] + mat[i - 1][j - w[i - 1]])
    10         else:
    11             mat[i][j] = mat[i - 1][j]
    12 include_items ← array of size n
    13 index ← 0
    14 cap ← c
    15 for i from n down to 1:
    16     if mat[i][cap] != mat[i - 1][cap]:
    17         include_items[index++] = i - 1
    18         cap = cap - w[i - 1]
    19 return mat[n][c]
```

PROGRAM

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

int max(int a, int b) {
    if (a>b){
        return a;
    }
    else{
        return b;
    }
}

int knapsack(int n, int c, int w[], int v[]) {
    int mat[n + 1][c + 1];

    // Initialize base cases
    for (int i = 0; i <= n; i++) {
        mat[i][0] = 0;
    }
    for (int i = 0; i <= c; i++) {
        mat[0][i] = 0;
    }

    // Fill the DP table
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= c; j++) {
            if (j - w[i - 1] >= 0) {
                mat[i][j] = max(mat[i - 1][j], v[i - 1] + mat[i - 1][j - w[i - 1]]);
            } else {
                mat[i][j] = mat[i - 1][j];
            }
        }
    }

    // Print DP table (optional)
    for (int i = 0; i <= n; i++) {
        for (int j = 0; j <= c; j++) {
            cout << mat[i][j] << "\t";
        }
        cout << endl;
    }

    int val = mat[n][c];

    // Traceback to find included items
    int includeItems[n]; // array to store included items
    int index = 0;
    int cap = c; // capacity tracker
    for (int i = n; i > 0; i--) {
        if (mat[i][cap] != mat[i - 1][cap]) {
            includeItems[index++] = i - 1;
            cap -= w[i - 1];
        }
    }

    // Print included items
    cout << "Included items: ";
    for (int i = index - 1; i >= 0; i--) {
        cout << includeItems[i] << " ";
    }
    cout << endl;
    cout<<"Maximum value:"<<mat[n][c]<<endl;
    return val;
}

int main() {
    cout << "ENTER THE CAPACITY OF THE KNAPSACK: ";
    int cap;
    cin >> cap;

    cout << "ENTER THE NUMBER OF ITEMS: ";
    int n;
    cin >> n;

    int W[n], V[n];
    for (int i = 0; i < n; i++) {
        cout << "ENTER THE WEIGHT OF ITEM " << i + 1 << ": ";
        cin >> W[i];
        cout << "ENTER THE VALUE OF ITEM " << i + 1 << ": ";
        cin >> V[i];
    }

     knapsack(n, cap, W, V) ;
}
```

SAMPLE INPUT-OUTPUT



```
ENTER THE CAPACITY OF THE KNAPSACK: 5
ENTER THE NUMBER OF ITEMS: 4
ENTER THE WEIGHT OF ITEM 1: 2
ENTER THE VALUE OF ITEM 1: 12
ENTER THE WEIGHT OF ITEM 2: 1
ENTER THE VALUE OF ITEM 2: 10
ENTER THE WEIGHT OF ITEM 3: 3
ENTER THE VALUE OF ITEM 3: 20
ENTER THE WEIGHT OF ITEM 4: 2
ENTER THE VALUE OF ITEM 4: 15
0   0   0   0   0   0
0   0   12  12  12  12
0   10  12  22  22  22
0   10  12  22  30  32
0   10  15  25  30  37
Included items: 1 2 4
Maximum value:37
```

21.LONGEST COMMON SUBSEQUENCE

6/11/2024

AIM

Formally, given a sequence X = (x1, x2 , .., xm), another sequence Z =(Z1,Z2,. . . .,Zk)is a subsequence of X if there exists a strictly increasing sequence (i1, i2, , ik) of indices of X such that for all j = 1, 2,.., k, we have xij = Zj.

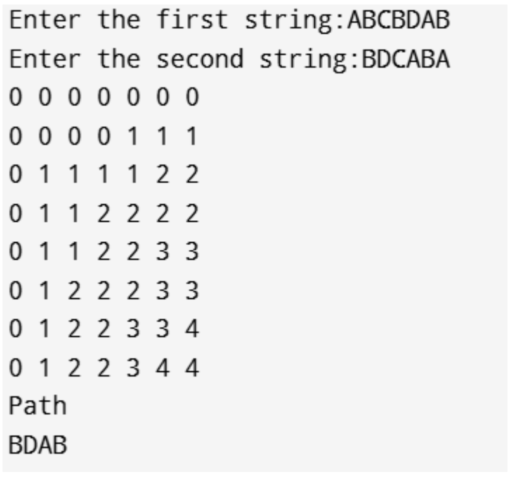
ALGORITHM

```
void lcs(s1, s2) //strings
    1. rows ← length of s1 + 1
    2. cols ← length of s2 + 1
    3. mat ← create 2d array of size rows x cols
    4. for i from 0 to rows:
    5.     mat[i][0] ← 0
    6. for i from 0 to cols:
    7.     mat[0][i] ← 0
    8. for i from 1 to rows - 1:
    9.     for j from 1 to cols - 1:
    10.        if s1[i - 1] == s2[j - 1]:
    11.            mat[i][j] ← mat[i - 1][j - 1] + 1
    12.        else:
    13.            mat[i][j] ← max(mat[i - 1][j], mat[i][j - 1])
    14. s ← mat[rows - 1][cols - 1]
    15. arr ← array of size s + 1
    16. r ← rows - 1
    17. c ← cols - 1
    18. d ← s - 1
    19. arr[s] ← '\0'
    20. while r > 0 and c > 0:
    21.     if s1[r - 1] == s2[c - 1]:
    22.         arr[d--] ← s2[c - 1]
    23.         r ← r - 1
    24.         c ← c - 1
    25.     else:
    26.         if mat[r - 1][c] > mat[r][c - 1]:
    27.             r ← r - 1
    28.         else:
    29.             c ← c - 1
    30. print "path"
    31. print arr as a string
    32. return mat[rows - 1][cols - 1]
```

PROGRAM

```
#include<iostream>

using namespace std;
int max(int a, int b) {
    if (a > b) {
        return a;
    } else {
        return b;
    }
}
int lcs(string s1, string s2) {
    int rows = s1.length() + 1;
    int cols = s2.length() + 1;
    int mat[rows][cols];
    for (int i = 0; i < rows; i++) {
        mat[i][0] = 0;
    }
    for (int i = 0; i < cols; i++) {
        mat[0][i] = 0;
    }
    for (int i = 1; i < rows; i++) {
        for (int j = 1; j < cols; j++) {
            if (s1[i - 1] == s2[j - 1]) {
                mat[i][j] = mat[i - 1][j - 1] + 1;
            } else {
                mat[i][j] = max(mat[i - 1][j], mat[i][j - 1]);
            }
        }
    }
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cout << mat[i][j] << " ";
        }
        cout << endl;
    }
    int s = mat[rows - 1][cols - 1];
    char arr[s + 1];
    int r = rows - 1;
    int c = cols - 1;
    int d = s - 1;
    arr[s] = '\0';
    while (r > 0 and c > 0) {
        if (s1[r - 1] == s2[c - 1]) {
            arr[d--] = s2[c - 1];
            r--;
            c--;
        } else {
            if (mat[r - 1][c] > mat[r][c - 1]) {
                r--;
            } else {
                c--;
            }
        }
    }
    cout << "Path" << endl;
    cout << arr << endl;
    cout << endl;
    return mat[rows - 1][cols - 1];

}
int main() {
    cout << "Enter the first string:";
    string s1;
    cin >> s1;
    cout << "Enter the second string:";
    string s2;
    cin >> s2;
    lcs(s1, s2);
}
```

SAMPLE INPUT-OUTPUT



```
Enter the first string:ABCBDAB
Enter the second string:BDCABA
0 0 0 0 0 0 0
0 0 0 0 1 1 1
0 1 1 1 1 2 2
0 1 1 2 2 2 2
0 1 1 2 2 3 3
0 1 2 2 2 3 3
0 1 2 2 3 3 4
0 1 2 2 3 4 4
Path
BDAB
```