

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

KRISHNA AJITH
(81323012)

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 **KRISHNA AJITH(81323012)**, 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

Sl.No	Title	Date	Page no
1	Linear Search	03/07/2024	1
2	Binary Search	03/07/2024	4
3	Bubble Sort	10/07/2024	6
4	Insertion Sort	10/07/2024	8
5	Selection Sort	17/07/2024	10
6	Merge Sort	17/07/2024	12
7	Quick Sort	24/07/2024	15
8	Heap Sort	24/07/2024	18
9	Graph (BFS & DFS traversal)	31/07/2024	21
10	Prim's Algorithm	07/08/2024	26
11	Kruskal's Algorithm	14/08/2024	29
12	Dijkstra's Algorithm	04/09/2024	32
13	Bellman-Ford Algorithm	09/09/2024	37
14	Floyd-Warshall Algorithm	09/09/2024	40
15	Dynamic Fibonacci Series	25/09/2024	43
16	Coin Row Problem	25/09/2024	45
17	Coin Change Making Problem	16/10/2024	48
18	Coin Collecting Problem	16/10/2024	51
19	Minimum Cost Path of a Matrix	30/10/2024	54
20	0/1 Knapsack Problem	30/10/2024	58
21	Longest Common Subsequence	06/11/2024	62

LINEAR SEARCH

DATE:03/07/2024

AIM

Implement Linear Search algorithm for a given array of integers.

ALGORITHM

```
int linearSearch (int arr[ ], int n, int key)
1.for i =0 upto n
2.    if ( arr [ i ] == key ) then
3.        return i ;
4.    end if
5.end for
6. return -1;
```

PROGRAM

```
#include<iostream>
using namespace std;
int LinearSearch(int n,int A[],int key){
    int ind=-1;
    for(int i=0;i<n;i++){
        if(A[i]==key){
            ind=i;
            return ind;
            break;
        }
    }
    return ind;
}
int main(){
    int n;
    cout<<"Enter the number of elements:";
    cin>>n;
    int A[n];
    cout<<"Enter the elements for array:";
    for(int i=0;i<n;i++){
        cin>>A[i];
    }
    int key;
```

```
    cout<<"Enter the key to search:";
    cin>>key;
    int ls=LinearSearch(n,A,key);
    if(ls!=-1){
        cout<<"The element "<<key<<" found at index "<<ls<<endl;
    }
    else{
        cout<<"The element not found"<<endl;
    }
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
Enter the number of elements:5
Enter the elements for array:1
5
7
44
3
Enter the key to search:44
The element 44 found at index 3
```

BINARY SEARCH

DATE:03/07/2024

AIM

Implement Binary Search algorithm for a given array of integers.

ALGORITHM

```
int BinarySearch(int ar[], int val, int LB, int UB)
1.   set index = -1;
2.   if (LB <= UB) then
3.       mid = (LB + UB) / 2;
4.       if (ar[mid] == val) then
5.           return mid;
6.       end if
7.       if (ar[mid] > val) then
8.           UB = mid - 1;
9.           return BinarySearch(ar, val, LB, UB)
10.      else if (ar[mid] < val) then
11.          LB = mid + 1;
12.          return BinarySearch(ar, val, LB, UB)
13.      end if
14.  end if
15.  return index;
```

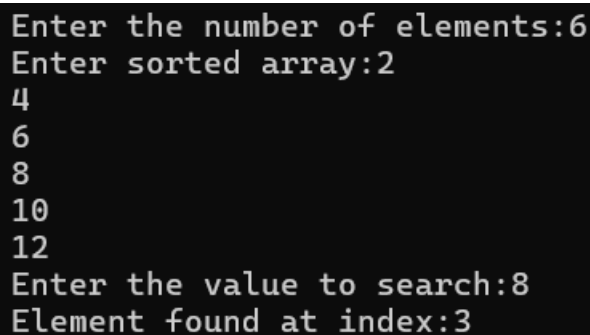
PROGRAM

```
#include <iostream>
using namespace std;
int BinarySearch(int ar[], int val, int LB, int UB) {
    int index = -1;
    if (LB <= UB) {
        int mid = (LB + UB) / 2;
        if (ar[mid] == val) {
            return mid;
        }
        if (ar[mid] > val) {
            UB = mid - 1;
            return BinarySearch(ar, val, LB, UB);
        } else if (ar[mid] < val) {
            LB = mid + 1;
        }
    }
}
```

```
        return BinarySearch(ar, val, LB, UB);
    }
}
return index;
}
int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    int ar[n];
    cout << "Enter sorted array: ";
    for (int i = 0; i < n; i++) {
        cin >> ar[i];
    }
    int key;
    cout << "Enter the value to search: ";
    cin >> key;
    int in = BinarySearch(ar, key, 0, n);
    if (in == -1) {
        cout << "Element not found" << endl;
    } else {
        cout << "Element found at index: " << in << endl;
    }

    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the number of elements:6
Enter sorted array:2
4
6
8
10
12
Enter the value to search:8
Element found at index:3
```

BUBBLE SORT

DATE:10/07/2024

AIM

Implement Bubble Sort algorithm for a given array of integers.

ALGORITHM

```
void BubbleSort (int arr[ ], int size)
1.for i = 0 upto size -1 do
2.    for j =0 upto size -i -1 do
3.        if ( arr [ j ] > arr [ j +1]) then
4.            swap ( arr [ j ] , arr [ j +1])
5.        end if
6.    end for
7.end for
```

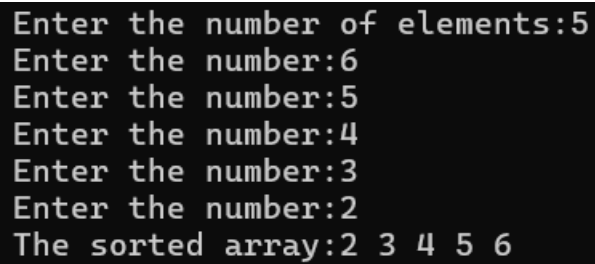
PROGRAM

```
#include <iostream>
using namespace std;
void BubbleSort(int n, int ar[]) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (ar[j] > ar[j + 1]) {
                int temp = ar[j];
                ar[j] = ar[j + 1];
                ar[j + 1] = temp;
            }
        }
    }
    cout << "The sorted array: ";
    for (int k = 0; k < n; k++) {
        cout << ar[k] << " ";
    }
    cout << endl;
}
int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
```



```
int ar[n];  
for (int m = 0; m < n; m++) {  
    cout << "Enter the number: ";  
    cin >> ar[m];  
}  
BubbleSort(n, ar);  
return 0;  
}
```

SAMPLE INPUT-OUTPUT

A screenshot of a terminal window with a black background and white text. It shows the input and output of a C++ program. The input consists of six lines: "Enter the number of elements:5", "Enter the number:6", "Enter the number:5", "Enter the number:4", "Enter the number:3", and "Enter the number:2". The output is a single line: "The sorted array:2 3 4 5 6".

```
Enter the number of elements:5  
Enter the number:6  
Enter the number:5  
Enter the number:4  
Enter the number:3  
Enter the number:2  
The sorted array:2 3 4 5 6
```

INSERTION SORT

DATE:10/07/2024

AIM

Implement Insertion Sort algorithm for a given array of integers.

ALGORITHM

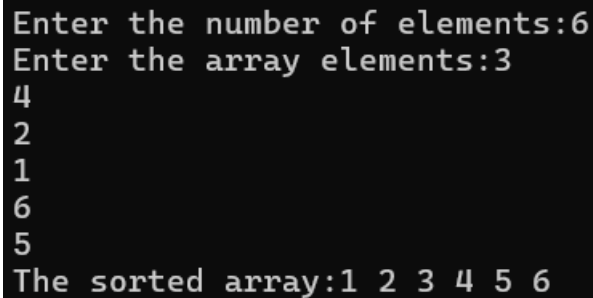
```
void InsertionSort(int ar[], int n)
1.for int i = 1 upto n do
2.    set key = ar[i]
3.    j = i - 1
4.    while (j >= 0 && ar[j] > key) do
5.        ar[j + 1] = ar[j]
6.        j--
7.    end while
8.    ar[j + 1] = key
9.end for
```

PROGRAM

```
#include <iostream>
using namespace std;
void InsertionSort(int ar[], int n) {
    for (int i = 1; i < n; i++) {
        int key = ar[i];
        int j = i - 1;
        while (j >= 0 && ar[j] > key) {
            ar[j + 1] = ar[j];
            j--;
        }
        ar[j + 1] = key;
    }
}
int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    int A[n];
    cout << "Enter the array elements: ";
```

```
    for (int i = 0; i < n; i++) {  
        cin >> A[i];  
    }  
    InsertionSort(A, n);  
    cout << "The sorted array: ";  
    for (int k = 0; k < n; k++) {  
        cout << A[k] << " ";  
    }  
    cout << endl;  
    return 0;  
}
```

SAMPLE INPUT-OUTPUT



```
Enter the number of elements:6  
Enter the array elements:3  
4  
2  
1  
6  
5  
The sorted array:1 2 3 4 5 6
```

SELECTION SORT

DATE:17/07/2024

AIM

Implement Selection Sort algorithm for a given array of integers.

ALGORITHM

```
void SelectionSort (int arr[ ], int n)
1. for i=0 upto n do
2.     min=i
3.     for j=i+1 upto n do
4.         if(ar[min]>ar[j]) then
5.             min=j
6.         end if
7.     end for
8.     if (min!= i) then
9.         swap arr [i] and arr [min]
10.    end if
11.end for
```

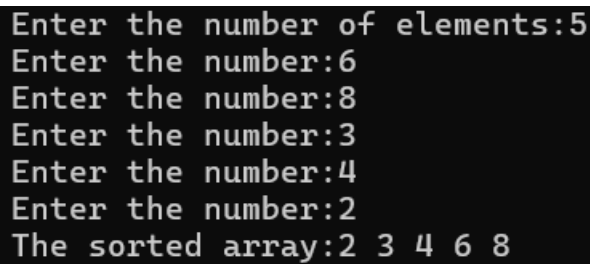
PROGRAM

```
#include<iostream>
using namespace std;
void selectionsort(int ar[], int n) {
    for (int i = 0; i < n; i++) {
        int min = i;
        for (int j = i + 1; j < n; j++) {
            if (ar[min] > ar[j]) {
                min = j;
            }
        }
        if (min != i) {
            int tem = ar[min];
            ar[min] = ar[i];
            ar[i] = tem;
        }
    }
    cout << "The sorted array: ";
```

```
        for (int k = 0; k < n; k++) {
            cout << ar[k] << " ";
        }
    }

int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    int ar[n];
    for (int m = 0; m < n; m++) {
        cout << "Enter the number: ";
        cin >> ar[m];
    }
    selectionsort(ar, n);
    cout << endl;
    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the number of elements:5
Enter the number:6
Enter the number:8
Enter the number:3
Enter the number:4
Enter the number:2
The sorted array:2 3 4 6 8
```

MERGE SORT

DATE:17/07/2024

AIM

Implement Merge Sort algorithm for a given array of integers.

ALGORITHM

```
void merge(int A[], int LB, int mid, int UB) {  
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] =infinity  
7.  R[n2] =infinity  
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  
17. end for
```

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

PROGRAM

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

```
void merge(int A[], int LB, int mid, int UB) {
    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, j = 0;
    for (int k = LB; k <= UB; k++) {
        if (L[i] < R[j]) {
            A[k] = L[i];
            i++;
        } else {
            A[k] = R[j];
            j++;
        }
    }
}

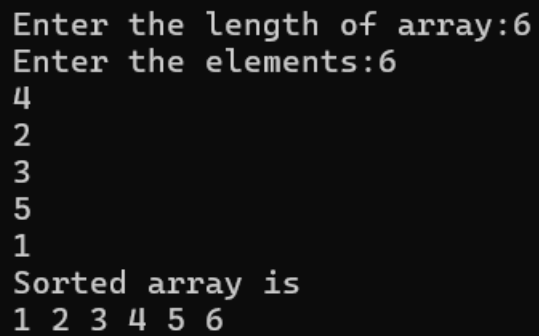
void mergesort(int ar[], int LB, int UB) {
    if (LB < UB) {
        int M = (LB + UB) / 2;
        mergesort(ar, LB, M);
        mergesort(ar, M + 1, UB);
        merge(ar, LB, M, UB);
    }
}

void display(int ar[], int n) {
    for (int i = 0; i < n; i++) {
        cout << ar[i] << " ";
    }
    cout << endl;
}

int main() {
    int len;
    cout << "Enter the length of array: ";
    cin >> len;
    int Arr[len];
    cout << "Enter the elements: ";
    for (int i = 0; i < len; i++) {
        cin >> Arr[i];
    }
}
```

```
    }  
    mergesort(Arr, 0, len - 1);  
    cout << "Sorted array is: ";  
    display(Arr, len);  
    return 0;  
}
```

SAMPLE INPUT-OUTPUT



```
Enter the length of array:6  
Enter the elements:6  
4  
2  
3  
5  
1  
Sorted array is  
1 2 3 4 5 6
```


QUICK SORT

DATE:24/07/2024

AIM

Implement Quick Sort algorithm for a given array of integers.

ALGORITHM

```
int partition(int LB, int n, int ar[])
1.  set j = LB - 1
2.  set pivot = ar[n - 1]
3.  set i = LB
4.  while (i < n) do
5.      if (ar[i] < pivot) then
6.          j = j + 1
7.          swap(ar[i],ar[j])
8.      end if
9.      i++
10. end while
11. swap(ar[j+1],pivot)
12. return j + 1

void quicksort(int LB, int n, int ar[])
1.  if (LB < n) then
2.      p = partition(LB, n, ar)
3.      quicksort(LB, p, ar)
4.      quicksort(p + 1, n, ar)
5.  end if
```

PROGRAM

```
#include <iostream>
using namespace std;
int partition(int LB, int n, int ar[]) {
    int j = LB - 1;
    int pivot = ar[n - 1];
    int i = LB;
    while (i < n) {
        if (ar[i] < pivot) {
            j = j + 1;
```

```
        int tem = ar[i];
        ar[i] = ar[j];
        ar[j] = tem;
    }
    i++;
}
int te = ar[j + 1];
ar[j + 1] = ar[n - 1];
ar[n - 1] = te;
return j + 1;
}
void quicksort(int LB, int n, int ar[]) {
    if (LB < n) {
        int p = partition(LB, n, ar);
        quicksort(LB, p, ar);
        quicksort(p + 1, n, ar);
    }
}
int main() {
    int n;
    cout << "Enter the number of elements: ";
    cin >> n;
    int ar[n];
    for (int m = 0; m < n; m++) {
        cout << "Enter the number: ";
        cin >> ar[m];
    }
    quicksort(0, n, ar);
    cout << "The sorted array: ";
    for (int k = 0; k < n; k++) {
        cout << ar[k] << " ";
    }
    cout << endl;
    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
Enter the number of elements:8
Enter the number:3
Enter the number:4
Enter the number:5
Enter the number:7
Enter the number:6
Enter the number:2
Enter the number:1
Enter the number:8
The sorted array:1 2 3 4 5 6 7 8
```

HEAP SORT

DATE:24/07/2024

AIM

Implement Heap Sort algorithm for a given array of integers.

ALGORITHM

```
void Heapify(int Arr[], int i, int n)
1.   largest = i
2.   l = 2 * i + 1
3.   r = 2 * i + 2
4.   if (l < n && Arr[largest] < Arr[l]) then
5.       largest = l
6.   end if
7.   if (r < n && Arr[largest] < Arr[r]) then
8.       largest = r
9.   end if
10.  if (largest != i) then
11.      Swap(i, largest, Arr)
12.      Heapify(Arr, largest, n)
13.  end if
```

```
void Heapsort(int Arr[], int n)
1.   for i = (n - 2) / 2 downto 0 do
2.       Heapify(Arr, i, n)
3.   end for
4.   for i = n - 1 downto 0 do
5.       Swap(0, i, Arr)
6.       Heapify(Arr, 0, i)
7.   end for
```

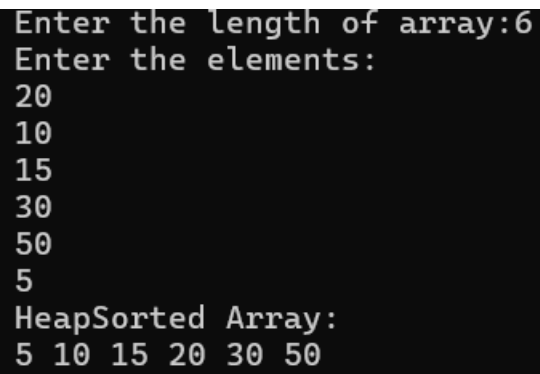
PROGRAM

```
#include <iostream>
using namespace std;
void Swap(int i, int j, int Arr[]) {
    int temp = Arr[i];
    Arr[i] = Arr[j];
    Arr[j] = temp;
```

```
}  
void Heapify(int Arr[], int i, int n) {  
    int largest = i;  
    int l = 2 * i + 1;  
    int r = 2 * i + 2;  
    if (l < n && Arr[largest] < Arr[l]) {  
        largest = l;  
    }  
    if (r < n && Arr[largest] < Arr[r]) {  
        largest = r;  
    }  
    if (largest != i) {  
        Swap(i, largest, Arr);  
        Heapify(Arr, largest, n);  
    }  
}  
  
void Heapsort(int Arr[], int n) {  
    for (int i = (n - 2) / 2; i >= 0; i--) {  
        Heapify(Arr, i, n);  
    }  
    for (int i = n - 1; i > 0; i--) {  
        Swap(0, i, Arr);  
        Heapify(Arr, 0, i);  
    }  
}  
  
int main() {  
    int len;  
    cout << "Enter the length of array: ";  
    cin >> len;  
    int Arr[len];  
    cout << "Enter the elements:" << endl;  
    for (int i = 0; i < len; i++) {  
        cin >> Arr[i];  
    }  
    Heapsort(Arr, len);  
    cout << "HeapSorted Array:" << endl;  
    for (int i = 0; i < len; i++) {  
        cout << Arr[i] << " ";  
    }  
}
```

```
    cout << endl;  
    return 0;  
}
```

SAMPLE INPUT-OUTPUT



```
Enter the length of array:6  
Enter the elements:  
20  
10  
15  
30  
50  
5  
HeapSorted Array:  
5 10 15 20 30 50
```

GRAPH BFS & DFS TRAVERSALS

DATE:31/07/2024

AIM

Implement Graph Data Structure by BFS and DFS algorithms.

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

// 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);
            }
        }
    }

    for (int i = 0; i < n; i++) { // Visit vertices with no predecessor
        if (visited[i] != 1) {
            BFS(i);
        }
    }
}

// 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) { // If there's an edge
                                                    // and v is not visited
            predecessor[v] = u; // Set the predecessor of v
        }
    }
}
```



```
        DFS(v); // Recursive call to visit the connected vertex
    }
}
for (int i = 0; i < n; i++) { // Visit vertices with no predecessor
    if (visited[i] != 1) {
        DFS(i);
    }
}
}

void clear(){
    fill(visited, visited + n, 0); // Reset visited array for BFS
    fill(predecessor, predecessor + n, -1); // Reset predecessor array for BFS
}

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
```

PRIM'S ALGORITHM

DATE:07/08/2024

AIM

Implement Minimum Spanning tree by using Prim's algorithm.

ALGORITHM

```
MST-Prim(G, w, r)
1  for each u ∈ G.V
2      u.key = inf
3      u.pi = NIL
4  r.key = 0
5  Q = G.V
6  while Q ≠ ∅
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

// Step 1: Initialize all keys to infinity and inMST to false
for (int i = 0; i < nodes; i++) {
    key[i] = INT_MAX;
    inMST[i] = false;
}

// Step 2: 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

// Step 3: 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];
        }
    }
}

// Step 4: 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
```

KRUSKAL'S ALGORITHM

DATE:14/08/2024

AIM

Implement Minimum Spanning tree by 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, 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
```

DIJKSTRA'S ALGORITHM

DATE:04/09/2024

AIM

Implement Graph Data Structure by using Dijkstra's algorithm.

ALGORITHM

Initialize-Single-Source(G, s)

```
1 for each vertex  $v \in G.V$ 
2      $v.d = \text{infinity}$ 
3      $v.pi = \text{NIL}$ 
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  $S = \emptyset$ 
3  $Q = G.V$ 
4 while  $Q \neq \emptyset$ 
5      $u = \text{Extract-Min}(Q)$ 
6      $S = S \cup \{u\}$ 
7     for each vertex  $v \in G.Adj[u]$ 
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
    // Uncomment the next line for an undirected graph:
    // adj[v][u] = w;
}

// 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
```

BELLMAN-FORD ALGORITHM

DATE:09/09/2024

AIM

Implement Graph Data Structure by 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) {
    // Step 1: 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;

    // Step 2: 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;
        }
    }
}

// Step 3: 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.
```

FLOYD-WARSHALL ALGORITHM

DATE:09/09/2024

AIM

Implement and understand the Floyd-Warshall algorithm on a graph to find all pair shortest path.

ALGORITHM

Floyd-Warshall(W)

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

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
```

FIBONACCI SERIES

DATE:25/09/2024

AIM

Implement Fibonacci Series using dynamic programming.

ALGORITHM

```
Fib(int n)
1.if(n<=1) then
2.    return 0
3. end if
4. set fib[n+1]
5. fib[0]=0
6. fib[1]=1
7. for i=2 upto n+1 do
8.    fib[i]=fib[i-1]+fib[i-2]
9. end for
10.return fib[n-1]
```

PROGRAM

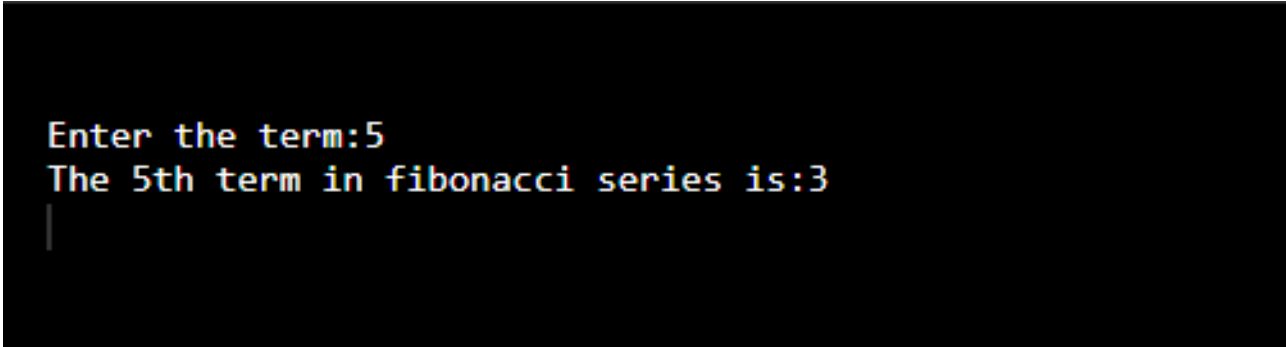
```
#include <iostream>
using namespace std;

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:5
The 5th term in fibonacci series is:3
|
```

COIN ROW PROBLEM

DATE:25/09/2024

AIM

Implement Coin Row problem using dynamic programming.

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 CR(int C[], int n) {
    int F[n + 1];
    F[0] = 0;
    F[1] = C[0];

    for (int i = 2; i < n + 1; i++) {
        F[i] = max(F[i - 1], F[i - 2] + C[i - 1]);
    }
}
```

```
    cout << "Coin index array: ";
    for (int i = 0; i < n + 1; i++) {
        cout << F[i] << " ";
    }
    cout << endl;

    cout << "Selected coins: ";
    int i = n;
    while (i > 0) {
        if (i == 1 || F[i] != F[i - 1]) {
            cout << C[i - 1] << " ";
            i -= 2;
        } else {
            i -= 1;
        }
    }
    cout << endl;

    return F[n];
}

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

    int C[n];
    cout << "Enter coins: ";
    for (int i = 0; i < n; i++) {
        cin >> C[i];
    }

    int c = CR(C, n);
    cout << "Max sum: " << c << endl;

    return 0;
}
```


SAMPLE INPUT-OUTPUT

```
Enter the number of coins:6
Enter coins:5
1
2
10
6
2
Coin index array:0 5 5 7 15 15 17
Selected coins:2 10 5

Max sum:17
|
```

COIN CHANGE MAKING PROBLEM

DATE:16/10/2024

AIM

Implement Coin change making problem using dynamic programming.

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 p, int q) {
    if (p < q) {
        return p;
    } else {
        return q;
    }
}

int coinchange(int Ac[], int amount, int nc) {
```

```
int F[amount + 1];
F[0] = 0;

for (int i = 1; i <= amount; i++) {
    int temp = numeric_limits<int>::max() - 1;
    int j = 0;
    while (j < nc && i >= Ac[j]) {
        temp = min(F[i - Ac[j]], temp);
        j++;
    }
    F[i] = temp + 1;
}

cout << "DP Array: ";
for (int k = 0; k <= amount; k++) {
    cout << F[k] << " ";
}
cout << endl;

int bal = amount;
int B[amount + 1];
int ind = 0;

while (bal > 0) {
    for (int j = 0; j < nc; j++) {
        if (bal >= Ac[j] && F[bal] == F[bal - Ac[j]] + 1) {
            B[ind++] = Ac[j];
            bal -= Ac[j];
        }
    }
}

cout << "Coins used for minimum change: ";
for (int i = 0; i < ind; i++) {
    cout << B[i] << " ";
}
cout << endl;

return F[amount];
}
```

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

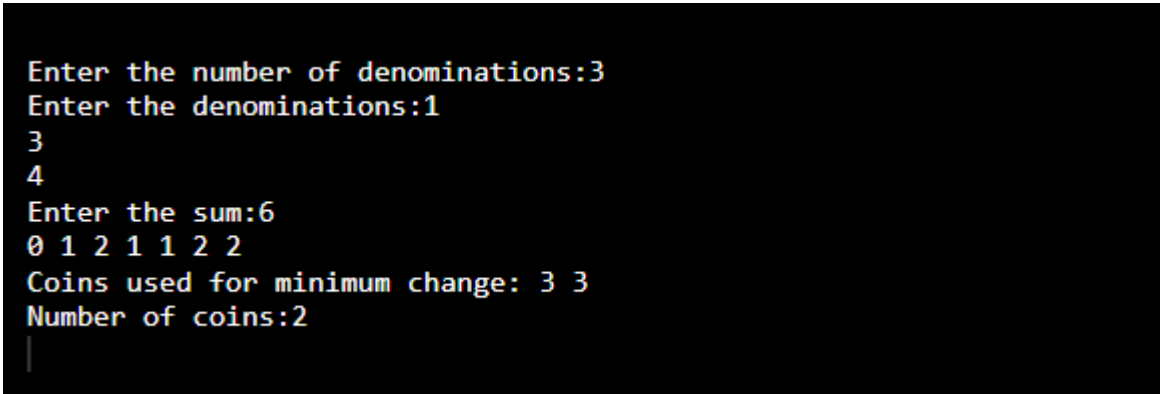
    int A[n];
    cout << "Enter the denominations: ";
    for (int i = 0; i < n; i++) {
        cin >> A[i];
    }

    int sum;
    cout << "Enter the sum: ";
    cin >> sum;

    int coins = coinchange(A, sum, n);
    cout << "Number of coins: " << coins << endl;

    return 0;
}
```

SAMPLE INPUT-OUTPUT



```
Enter the number of denominations:3
Enter the denominations:1
3
4
Enter the sum:6
0 1 2 1 1 2 2
Coins used for minimum change: 3 3
Number of coins:2
|
```

COIN COLLECTING PROBLEM

DATE:16/10/2024

AIM

Implement Coin collecting problem using dynamic programming.

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

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

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

int coinCollection(int c[100][100], int m, int n) {
    int F[m + 1][n + 1] = {0};
    F[0][0] = c[0][0];
```

```
    for (int k = 1; k < m; k++) {
        F[k][0] = F[k - 1][0] + c[k][0];
    }

    for (int l = 1; l < n; l++) {
        F[0][l] = F[0][l - 1] + c[0][l];
    }

    for (int i = 1; i < m; i++) {
        for (int j = 1; j < n; j++) {
            F[i][j] = max(F[i][j - 1], F[i - 1][j]) + c[i][j];
        }
    }

    // Print the DP table (optional, for debugging or understanding)
    for (int a = 0; a < m; a++) {
        for (int b = 0; b < n; b++) {
            cout << F[a][b] << " ";
        }
        cout << endl;
    }

    return F[m - 1][n - 1];
}

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

    int c[100][100];
    cout << "Enter the values: ";
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            cin >> c[i][j];
        }
    }

    int coin = coinCollection(c, m, n);
```

```
    cout << "The maximum no of coins: " << coin << endl;

    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
Enter number of cols:6
Enter the values:0
0
0
0
1
0
0
1
0
1
0
0
0
0
0
1
0
1
0
1
0
1
0
1
0
1
1
0
0
0
1
0
0 0 0 0 1 1
0 1 1 2 2 2
0 1 1 3 3 4
0 1 2 3 3 5
1 1 2 3 4 5
The maximum no of coins:5
```

MINIMUM COST PATH PROBLEM

DATE:30/10/2024

AIM

Implement Minimum cost path problem using dynamic programming.

ALGORITHM

```
MinCostPath(n,m,mat[n][m]) {  
1.   F[n][m] = {0}  
2.   F[0][0] = mat[0][0];  
3.   for k = 1 upto n do  
4.       F[k][0] = F[k - 1][0] + mat[k][0]  
5.   for l = 1 upto m do  
6.       F[0][l] = F[0][l - 1] + mat[0][l]  
7.   for i = 1 upto n do  
8.       for j = 1 upto m do  
9.           F[i][j] = min(F[i - 1][j - 1], F[i - 1][j],  
                           F[i][j - 1]) + mat[i][j]
```

PROGRAM

```
#include <iostream>  
using namespace std;  
  
int min(int a, int b, int c) {  
    if (a < b && a < c) {  
        return a;  
    } else if (b < a && b < c) {  
        return b;  
    } else {  
        return c;  
    }  
}  
  
int MinCostPath(int n, int m, int mat[100][100]) {  
    int F[n][m] = {0};  
    F[0][0] = mat[0][0];  
  
    // Initialize the first column  
    for (int k = 1; k < n; k++) {
```



```
F[k][0] = F[k - 1][0] + mat[k][0];
}

// Initialize the first row
for (int l = 1; l < m; l++) {
    F[0][l] = F[0][l - 1] + mat[0][l];
}

// Fill the DP matrix
for (int i = 1; i < n; i++) {
    for (int j = 1; j < m; j++) {
        F[i][j] = min(F[i - 1][j - 1], F[i - 1][j],
                      F[i][j - 1]) + mat[i][j];
    }
}

// Print the DP matrix
cout << "The Matrix:" << endl;
for (int a = 0; a < n; a++) {
    for (int b = 0; b < m; b++) {
        cout << F[a][b] << " ";
    }
    cout << endl;
}

// Trace the minimum cost path
int B[6];
int i = n - 1, j = m - 1, ind = 0;
B[ind++] = mat[i][j];
while (i > 0 || j > 0) {
    if (i > 0 && j > 0 && F[i][j] == F[i - 1][j - 1] + mat[i][j]) {
        i--;
        j--;
    } else if (i > 0 && F[i][j] == F[i - 1][j] + mat[i][j]) {
        i--;
    } else {
        j--;
    }
    B[ind++] = mat[i][j];
}
```

```
// Print the path
cout << "The min cost path is: ";
for (int p = ind - 1; p >= 0; p--) {
    cout << B[p] << "-";
}
cout << endl;

return F[n - 1][m - 1];
}

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

    int mat[100][100];
    cout << "Enter the values: ";
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cin >> mat[i][j];
        }
    }

    int min = MinCostPath(n, m, mat);
    cout << "The minimum cost is: " << min << endl;

    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
Enter the number of rows:4
Enter the number of columns:3
Enter the values:3
2
8
1
9
7
0
5
2
6
4
3
The Matrix:
3 5 13
4 12 12
4 9 11
10 8 11
The min cost path is:3-1-0-4-3-
The minimum cost is:11
```

0/1 KNAPSACK PROBLEM

DATE:30/10/2024

AIM

Implement 0/1 Knapsack problem using dynamic programming.

ALGORITHM

Knapsack(c,n,v[],w[])

```
1.   F[n + 1][c + 1];
2.   for i = 0 upto n+1 do
3.       F[i][0] = 0
4.   for j = 0 upto c+1 do
5.       F[0][j] = 0;
6.   for i = 1 upto n+1 do
7.       for j = 1 upto c+1 do
8.           if (j - w[i - 1] >= 0)
9.               F[i][j] = max(F[i - 1][j], v[i - 1] +
10.                  F[i - 1][j - w[i - 1]])
11.           else
12.               F[i][j] = F[i - 1][j]
```

PROGRAM

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

int max(int a, int b) {
    return (a > b) ? a : b;
}

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

    // Initialize the DP matrix
    for (int i = 0; i < n + 1; i++) {
        F[i][0] = 0;
    }
    for (int j = 0; j < c + 1; j++) {
        F[0][j] = 0;
    }
}
```

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

// Print the DP matrix
cout << "Knapsack matrix:" << endl;
for (int i = 0; i < n + 1; i++) {
    for (int j = 0; j < c + 1; j++) {
        cout << F[i][j] << " ";
    }
    cout << endl;
}

// Determine the items included in the knapsack
int items[n];
int ind = 0;
int cap = c;
for (int i = n; i > 0; i--) {
    if (F[i][cap] != F[i - 1][cap]) {
        items[ind++] = i - 1;
        cap -= w[i - 1];
    }
}

// Print included items' values and weights
cout << "Included Items (price and weight):" << endl;
for (int i = ind - 1; i >= 0; i--) {
    cout << v[items[i]] << " " << w[items[i]] << endl;
}

return F[n][c];
}
```

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

    int c;
    cout << "Enter the capacity: ";
    cin >> c;

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

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

    int kp = knapsack(c, n, v, w);
    cout << "Max capacity: " << kp << endl;

    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
Enter the number of items:4
Enter the capacity:5
Enter the values:12
10
20
15
Enter the weights:2
1
3
2
Knapsack matrix:
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 price and weight:
12 2 10 1 15 2
Max capacity:37
|
```

LONGEST COMMON SUBSEQUENCE PROBLEM

DATE:06/11/2024

AIM

Implement Longest Common Subsequence problem using dynamic programming.

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;

// Function to find the maximum of two integers
int max(int a, int b) {
    return (a > b) ? a : b;
}

// Function to compute the Longest Common Subsequence (LCS)
string LCS(string X, string Y) {
    int m = X.length();
    int n = Y.length();
    int F[m + 1][n + 1];

    // Initialize the DP matrix
    for (int i = 0; i <= m; i++) {
        F[i][0] = 0;
    }
    for (int j = 0; j <= n; j++) {
        F[0][j] = 0;
    }

    // Fill the DP matrix
    for (int i = 1; i <= m; i++) {
        for (int j = 1; j <= n; j++) {
            if (X[i - 1] == Y[j - 1]) {
                F[i][j] = F[i - 1][j - 1] + 1;
            } else {
                F[i][j] = max(F[i - 1][j], F[i][j - 1]);
            }
        }
    }

    // Print the LCS matrix
    cout << "The Longest Common Subsequence matrix:" << endl;
    for (int i = 0; i <= m; i++) {
        for (int j = 0; j <= n; j++) {
            cout << F[i][j] << " ";
        }
    }
```

```
        cout << endl;
    }

    // Trace back to construct the LCS string
    string lcs = "";
    int k = m, l = n;
    while (k > 0 && l > 0) {
        if (X[k - 1] == Y[l - 1]) {
            lcs = X[k - 1] + lcs;
            k--;
            l--;
        } else if (F[k - 1][l] > F[k][l - 1]) {
            k--;
        } else {
            l--;
        }
    }

    cout << "The length of the subsequence: " << F[m][n] << endl;
    return lcs;
}

// Main function
int main() {
    string A, B;
    cout << "Enter the first string: ";
    cin >> A;
    cout << "Enter the second string: ";
    cin >> B;

    string lcs = LCS(A, B);
    cout << "The Longest Common Subsequence is: " << lcs << endl;

    return 0;
}
```

SAMPLE INPUT-OUTPUT

```
Enter the first string:ABCBDAAB
Enter the second string:BDCABA
The Longest Common Subsequence matrix:
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
The length of the subsequence:4
The Longest Common Subsequence is:BDAB
```