INDIAN INSTITUTE OF INFORMATION TECHNOLOGY
SONEPAT

Design and Analysis of Algorithms Lab

(CSL 307)

Practical File of Design and Analysis of Algorithm

Submitted To:

Mr. Sandeep Singh

(Assistant Professor)
(Department of CSE)

Submitted By:

Aman Aditya

12111086
Branch: CSE
Semester: III
Session: 2021-25

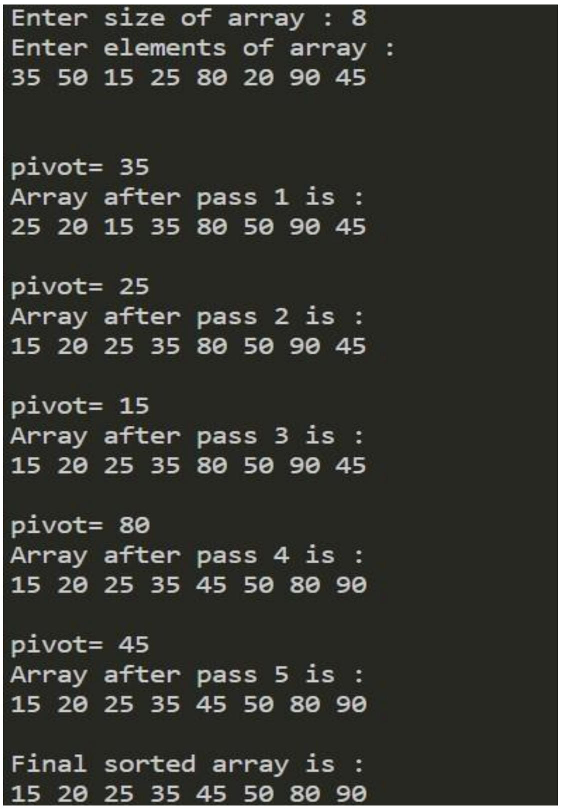
Index

Write a program for Quick Sort algorithm.

```
#include <bits/stdc++.h>
using namespace std;
int x = 1;
void printarray(int arr[], int n)
{
    x++;
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
}
int partition(int arr[], int p, int q)
{
    int pi = arr[p]; // initiliazing and defining the pivot.
    int i = p, j = q;
    // checking while i is less than j then everytime i<j we will swap the elements present at i
and j.
    while (i < j)
    {
        while (arr[i] <= arr[p])
        {
            i++;
        }
        while (arr[j] > arr[p])
        {
            j--;
        }
        if (i < j)
        {
            swap(arr[i], arr[j]);
        }
    }
    // and if i is greater than q than we will swap the element at index with pivot element.
    swap(arr[j], arr[p]);
    // returing the new index of the pivot to the quicksort function.
    return j;
}
void quicksort(int arr[], int p, int q, int n)
{
    if (p < q) // check if the array has more than one element.
    {
        int loc = partition(arr, p, q); // we will get the new index of the pivot after the partition
function is executed.
        // next two quicksort functions are applied for two subarrays formed on the left and right
of the pivot.
        cout << "\n\npivot= " << arr[loc] << endl;
        cout << "Array after pass " << x << " is :" << endl;
        printarray(arr, n);
        quicksort(arr, p, loc - 1, n);
        quicksort(arr, loc + 1, q, n);
    }
}
```

```
int main()
{
    int n;
    int arr[n];
    cout << "Enter size of array : ";
    cin >> n;
    cout << "Enter elements of array : " << endl;
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    quicksort(arr, 0, n - 1, n);
    cout << "\n\nFinal sorted array is : " << endl;
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    return 0;
}
```

OUTPUT :

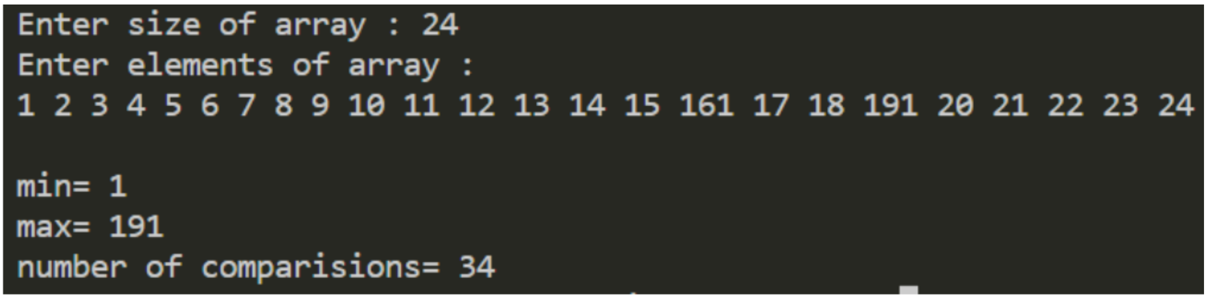


```
Enter size of array : 8
Enter elements of array :
35 50 15 25 80 20 90 45


pivot= 35
Array after pass 1 is :
25 20 15 35 80 50 90 45

pivot= 25
Array after pass 2 is :
15 20 25 35 80 50 90 45

pivot= 15
Array after pass 3 is :
15 20 25 35 80 50 90 45

pivot= 80
Array after pass 4 is :
15 20 25 35 45 50 80 90

pivot= 45
Array after pass 5 is :
15 20 25 35 45 50 80 90

Final sorted array is :
15 20 25 35 45 50 80 90
```

Write a program for finding Min-Max element of given string by using DAC. Also find the no. of comparision.

```
#include <bits/stdc++.h>
using namespace std;
int cnt = 0; // Defining a global variable for counting the number of comparisions.
void minmax(int arr[], int l, int h, int &min, int &max)
{
    if (l == h) // if array has only one element.
    {
        if (min > arr[l])
            min = arr[l];
        if (max < arr[l])
            max = arr[l];
        return;
    }
    else if (h - l == 1) // if array has two elements.
    {
        if (arr[l] < arr[h])
        {
            if (arr[l] < min)
                min = arr[l];
            if (arr[h] > max)
                max = arr[h];
        }
        else
        {
            if (arr[h] < min)
                min = arr[h];
            if (arr[l] > max)
                max = arr[l];
        }
        cnt++; // if there are 2 elements in subarray then there will be only one comparision .
        return;
    }
    else // if array has more than two elements.
    {
        cnt += 2; // if more than two elements in array than there will be 2 comparisions.
        int mid = (l + h) / 2;
        if (mid % 2 == 0) // the division of subarray will be two parts each having even number
of elements.
            {
                mid++;
            }
        minmax(arr, l, mid, min, max);
        minmax(arr, mid + 1, h, min, max);
    }
}
int main()
{
    int n, min = INT_MAX, max = INT_MIN;
    int arr[n];
    cout << "Enter size of array : ";
    cin >> n;
    cout << "Enter elements of array : " << endl;
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    minmax(arr, 0, n - 1, min, max);
    cout << "\nmin= " << min << endl;
    cout << "max= " << max << endl;
    cout << "number of comparisions= " << cnt << endl;
    return 0;
}
```

OUTPUT :

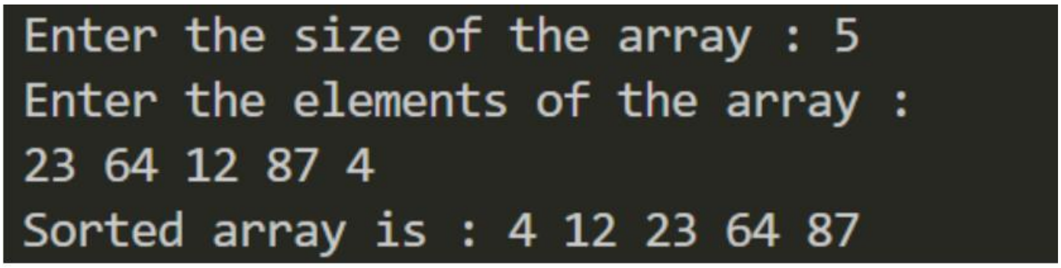


```
Enter size of array : 24
Enter elements of array :
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 161 17 18 191 20 21 22 23 24

min= 1
max= 191
number of comparisions= 34
```

Write a program for merge sort.

```
#include <bits/stdc++.h>
using namespace std;
void merge(int arr[], int l, int mid, int h)
{
    // In this function we are merging the sorted array we start with the arrays having one
element because they are already sorted. While merging the sorted arrays we must do
comparision in the given elements of the two arrays and then put them at the right index.
    int b[h - l + 1]; // We have created a new array to store the merged result of the two sorted
arrays.
    int i = l, j = mid + 1, k = 0, x = l;
    while (i <= mid && j <= h) // Here we are comaparing the elements of the two sorted
arrays.
    {
        if (arr[i] < arr[j])
        {
            b[k++] = arr[i++];
        }
        else
        {
            b[k++] = arr[j++];
        }
    }
    // Both of loops shown below are taken to confirm that if after comparison any of the
elements are remaining in any array they will be copied in the final new array.
    while (i <= mid)
    {
        b[k++] = arr[i++];
    }
    while (j <= h)
    {
        b[k++] = arr[j++];
    }
    // Here we are copying the data of the new formed array to the original array at the correct
index. That’s why we are intializing k=0 and x=l.
    for (k = 0, x = l; k < h - l + 1; k++, x++)
    {
        arr[x] = b[k];
    }
}
void mergesort(int arr[], int l, int h)
{
    if (l < h)
    {
        int mid = (l + h) / 2;
        mergesort(arr, l, mid);
        mergesort(arr, mid + 1, h);
        merge(arr, l, mid, h);
    }
}
int main()
{
    int n;
    int arr[n];
    cout << "Enter the size of the array : ";
    cin >> n;
    cout << "Enter the elements of the array : " << endl;
    for (int i = 0; i < n; i++)
    {
        cin >> arr[i];
    }
    mergesort(arr, 0, n - 1);
    cout << "Sorted array is : ";
    for (int i = 0; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    return 0;
}
```

OUTPUT :

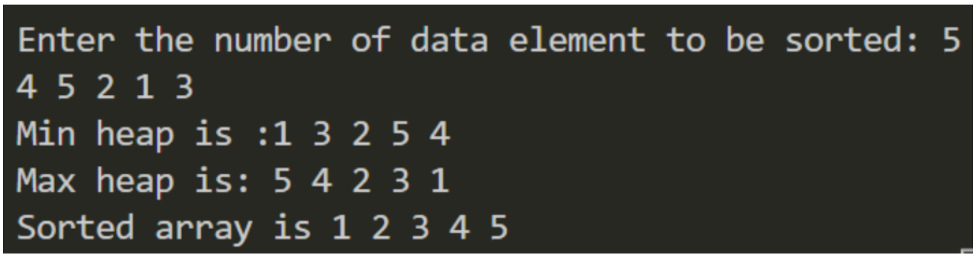


```
Enter the size of the array : 5
Enter the elements of the array :
23 64 12 87 4
Sorted array is : 4 12 23 64 87
```

Write a Program for min heap, max heap and max heap with apply heap sort.

```
#include <iostream>
using namespace std;
void min_heapify(int *a, int i, int n){
    int j, temp;
    temp = a[i];
    j = 2 * i;
    while (j <= n)
    {
        if (j < n && a[j + 1] < a[j])
            j = j + 1;
        if (temp < a[j])
            break;
        else if (temp >= a[j])
        {
            a[j / 2] = a[j];
            j = 2 * j;
        }
    }
    a[j / 2] = temp;
    return;
}
void build_minheap(int *a, int n){
    int i;
    for (i = n / 2; i >= 1; i--)
    {
        min_heapify(a, i, n);
    }
}
// A function to heapify the array.
void MaxHeapify(int a[], int i, int n){
    int j, temp;
    temp = a[i];
    j = 2 * i;
    while (j <= n){
        if (j < n && a[j + 1] > a[j])
            j = j + 1;
        // Break if parent value is already greater than child value.
        if (temp > a[j])
            break;
        // Switching value with the parent node if temp < a[j].
        else if (temp <= a[j])
        {
            a[j / 2] = a[j];
            j = 2 * j;
        }
    }
    a[j / 2] = temp;
    return;
}
void HeapSort(int a[], int n){
    int i, temp;
    for (i = n; i >= 2; i--){
        // Storing maximum value at the end.
        temp = a[i];
        a[i] = a[1];
        a[1] = temp;
        // Building max heap of remaining element.
        MaxHeapify(a, 1, i - 1);
    }
}
void Build_MaxHeap(int a[], int n){
    int i;
    for (i = n / 2; i >= 1; i--)
        MaxHeapify(a, i, n);
}
int main(){
    int n, i;
    cout << "\nEnter the number of data element to be sorted: ";
    cin >> n;
    n++;
    int arr[n];
    for (i = 1; i < n; i++){
        cin >> arr[i];
    }
    build_minheap(arr, n - 1);
    cout << "Min heap is :";
    for (i = 1; i < n; i++){
        cout << arr[i] << " ";
    }
    Build_MaxHeap(arr, n - 1);
    cout << "\nMax heap is: ";
    for (i = 1; i < n; i++) {
        cout << arr[i] << " ";
    }
    HeapSort(arr, n - 1);
    cout << "\nSorted array is ";
    for (i = 1; i < n; i++)
    {
        cout << arr[i] << " ";
    }
    return 0;
}
```

OUTPUT :



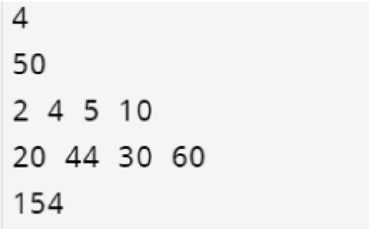
```
Enter the number of data element to be sorted: 5
4 5 2 1 3
Min heap is :1 3 2 5 4
Max heap is: 5 4 2 3 1
Sorted array is 1 2 3 4 5
```

Implement the greedy program.

1. Knapsack problem:

```
#include <bits/stdc++.h>
using namespace std;
int main(){
        int n;
        cin >> n;
        int capacity;
        cin >> capacity;
        vector<int> weight(n);
        for (int k = 0; k < n; k++){
            cin >> weight[k];
        }
        vector<int> value(n);
        for (int k = 0; k < n; k++){
            cin >> value[k];
        }
        vector<pair<double, pair<int, int>>> v;
        for (int k = 0; k < n; k++){
            double r = ((value[k]) * 1.0) / weight[k];
            int c = weight[k];
            int d =
                value[k];
            v.push_back({r, {c, d}});
        }
        sort(v.rbegin(), v.rend());
        double sum = 0;
        for (int k = 0; k < n; k++){
            if (capacity >= v[k].second.first && capacity > 0) {
                sum += v[k].second.second;
                capacity -= v[k].second.first;
            }
            else{
                sum += capacity * v[k].first;
                break;
            }
        }
        cout << sum << endl;
    return 0;
}
```

OUTPUT :

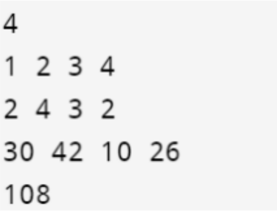


```
4
50
2 4 5 10
20 44 30 60
154
```

2. Job Sequencing:

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
        int n;
        cin >> n;
        vector<int> jobid(n);
        for (int k = 0; k < n; k++){
            cin >> jobid[k];
        }
        vector<int> deadline(n);
        for (int k = 0; k < n; k++){
            cin >> deadline[k];
        }
        vector<int> profit(n);
        for (int k = 0; k < n; k++){
            cin >> profit[k];
        }
        vector<pair<int, pair<int, int>>> v;
        for (int k = 0; k < n; k++){
            int a = jobid[k];
            int b = deadline[k];
            int c = profit[k];
            v.push_back({c, {b, a}});
        }
        sort(v.rbegin(), v.rend());
        vector<int> ans(n, -1);
        vector<int> fin;
        int pro = 0;
        int cnt = 0;
        for (int k = 0; k < n; k++) {
            int a = v[k].first;
            int b = v[k].second.first;
            int c = v[k].second.second;
            for (int j = b - 1; j >= 0; j--){
                if (ans[j] == -1){
                    pro += a;
                    ans[j] = c;
                    cnt++;
                    break;
                }
            }
        }
        cout << pro << endl;
    return 0;
}
```

OUTPUT :

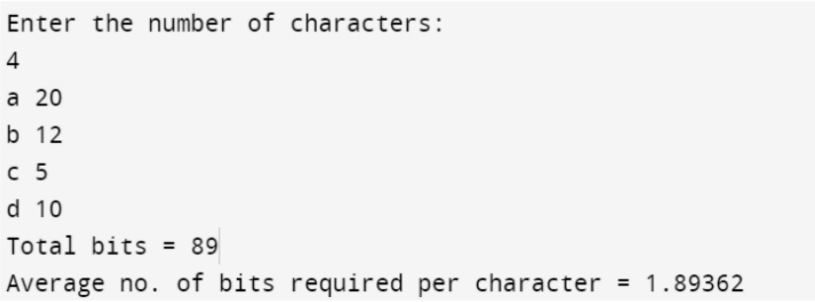


```
4
1  2  3  4
2  4  3  2
30  42  10  26
108
```

3. Huffman coding:

```
#include <bits/stdc++.h>
using namespace std;
template <typename X, typename Y, typename Z>
class
   triplet
{
public:
   X first;
   Y second;
   Z third;
};
template <typename X, typename Y, typename Z>
triplet<X, Y, Z> make_triplet(X
                          x,
                       Y y, Z z)
{
   triplet<X, Y, Z> t;
   t.first = x;
   t.second = y;
   t.third = z;
   return t;
}
class MinHeapNode
{
public:
   char data;
   int freq;
   MinHeapNode *left, *right;
   MinHeapNode(char data, int freq)
   {
      left = right = NULL;
      this->data = data;
      this->freq = freq;
   }
};
class compare
{
public:
   bool operator()(const MinHeapNode *left, const MinHeapNode
                                          *right)
   {
        return (left->freq > right->freq);
    }
};
float getTotalBits(vector<triplet<char, int, string>> &v_codes)
{
    float total = 0;
    for (auto i : v_codes)
    {
        total += ((i.second) * (i.third.length()));
    }
    return total;
}
void getCodes(MinHeapNode *root, string str, vector<triplet<char, int, string>> &v_codes)
{
    if (root)
    {
        if (root->data != '?')
        {
            v_codes.push_back(make_triplet(root->data, root->freq, str));
        }
        getCodes(root->left, str + "0", v_codes);
        getCodes(root->right, str + "1",
                 v_codes);
    }
}
vector<triplet<char, int, string>> createMinHeap(vector<pair<char, int>> &v)
{
    MinHeapNode *left, *right, *tmp;
    priority_queue<MinHeapNode *,
                   vector<MinHeapNode *>,
                   compare>
        minHeap;
    for (int i = 0; i < v.size(); i++)
    {
        minHeap.push(new MinHeapNode(v[i].first, v[i].second));
    }
    while (minHeap.size() != 1)
    {
        left = minHeap.top();
        minHeap.pop();
        right = minHeap.top();
        minHeap.pop();
        tmp = new MinHeapNode('?', left->freq + right->freq);
        tmp->left = left;
        tmp->right = right;
        minHeap.push(tmp);
    }
    vector<triplet<char, int, string>> v_codes;
    getCodes(minHeap.top(), "",
             v_codes);
    return v_codes;
}
int main()
{
    int n;
    cout << "Enter the number of characters:" << endl;
    cin >> n;
    vector<pair<char, int>> v;
    for (int i = 0; i < n; i++)
    {
        char data;
        int freq;
        cin >> data >> freq;
        v.push_back(make_pair(data, freq));
    }
    vector<triplet<char, int, string>> v_codes = createMinHeap(v);
    cout << "Total bits =" << getTotalBits(v_codes) << endl;
    int total_char = 0;
    for (auto pr : v)
    {
        total_char += pr.second;
    }
    cout << "Average no. of bits required per character = " << getTotalBits(v_codes) /
total_char;
    return 0;
}
```

OUTPUT :

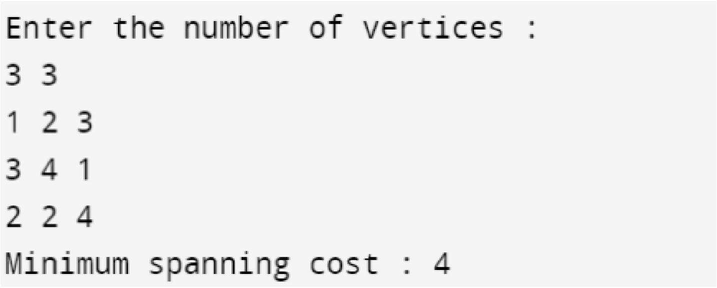


```
Enter the number of characters:
4
a 20
b 12
c 5
d 10
Total bits = 89
Average no. of bits required per character = 1.89362
```

4. Kruskal algorithm:

```
#include <bits/stdc++.h>
using namespace std;
class Edge
{
public:
    int source;
    int dest;
    int weight;
    Edge(int src, int dest, int wt){
        this->source = src;
        this->dest = dest;
        this->weight = wt;
    }
};
class Graph
{
private:
    vector<int> arr;
    int n;
    vector<int> parent;

public:
    Graph(int n)
{
        for (int i = 0; i < n; i++)
        {
            arr.push_back(0);
            parent.push_back(i);
        }
    }
    int findParent(int node){
        if (node == parent[node])
        {
            return node;
        }
        return parent[node] = findParent(parent[node]);
    }
    void unionFunc(int a, int b){
        a = findParent(a);
        b = findParent(b);
        if (arr[a] == arr[b])
        {
            parent[a] = b;
            arr[b]++;
        }
        else if (arr[a] > arr[b])
        {
            parent[b] = a;
        }
        else if (arr[a] < arr[b])
        {
            parent[a] = b;
        }
    }
};
int minimumSpanningTree(int v, vector<Edge> &edges)
{
    sort(edges.begin(), edges.end(),
         [](const Edge &e1, const Edge &e2) -> bool
         {
             return (e1.weight < e2.weight);
         });
    Graph g(v + 1);
    int weight = 0;
    int cnt = 0;
    for (int i = 0; i < edges.size();
         i++)
    {
        if (cnt == v - 1)
        {
            break;
        }
        else
        {
            if (g.findParent(edges[i].source) !=
                g.findParent(edges[i].dest))
            {
                weight += edges[i].weight;
                g.unionFunc(edges[i].source, edges[i].dest);
                cnt++;
            }
        }
    }
    return weight;
}
int main()
{
    vector<Edge> edges;
    cout << "Enter the number of vertices : \n";
    int v;
    cin >>
        v;
    int n;
    cin >> n;
    for (int i = 0; i < n; i++)
    {
        int src, dest, wt;
        cin >> src >> dest >> wt;
        Edge e(src, dest, wt);
        edges.push_back(e);
    }
    cout << "Minimum spanning cost : " << minimumSpanningTree(v, edges) << endl;
    return 0;
}
```

OUTPUT :

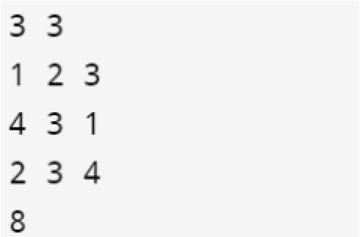


```
Enter the number of vertices :
3 3
1 2 3
3 4 1
2 2 4
Minimum spanning cost : 4
```

5. Prim's algorithm:

```
#include <bits/stdc++.h>
using namespace std;
const int N = 1e5 + 10;
int parent[N], sz[N];
void make(int v){
    parent[v] = v;
    sz[v] = 1;
}
int find(int v){
    if (parent[v] == v)
        return parent[v];
    return parent[v] = find(parent[v]);
}
void Union(int a, int b){
    a = find(a);
    b = find(b);
    if (a != b){
        if (sz[a] < sz[b])
            swap(a, b);
        parent[b] = a;
        sz[a] += sz[b];
    }
}
int minimumSpanningTree(int n, vector<pair<int, pair<int, int>>> edges){
    sort(edges.begin(), edges.end());
    for (int i = 1; i <= n; i++)
        make(i);
    int total_cost = 0;
    for (auto &edge : edges){
        int wt = edge.first;
        int u = edge.second.first;
        int v = edge.second.second;
        if (find(u) == find(v))
            continue;
        Union(u, v);
        total_cost += wt;
    }
    return total_cost;
}
int main(){
    int n, m;
    cin >> n >> m;
    vector<pair<int, pair<int, int>>> edges;
    for (int i = 0; i < m; i++){
        int u, v, wt;
        cin >> u >> v >> wt;
        edges.push_back({wt, {u, v}});
    }
    cout << minimumSpanningTree(n, edges) << endl;
}
```

OUTPUT :



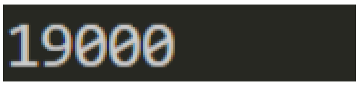
```
3 3
1 2 3
4 3 1
2 3 4
8
```

Dynamic Programming :

1. Matrix Chain Multiplication :

```
#include <bits/stdc++.h>
using namespace std;
#define ll long long
ll matrix_chain_mul(vector<ll> x)
{
    int n = x.size();
    vector<vector<ll>> m(n + 1, vector<ll>(n + 1, 0));
    for (int S = 2; S < n; S++)
    {
        for (int i = 1; i < n - S + 1; i++)
        {
            int j = i + S - 1;
            m[i][j] = INT_MAX;
            for (int k = i; k < j; k++)
            {
                m[i][j] = min(m[i][j], m[i][k] + m[k + 1][j] + x[i - 1] * x[k] * x[j]);
            }
        }
    }
    return m[1][n - 1];
}
int main()
{
    vector<ll> x = {10, 100, 20, 5, 80};
    cout << matrix_chain_mul(x);
    return 0;
}
```

OUTPUT :

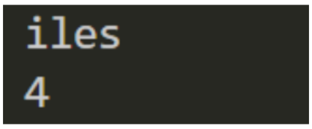


```
19000
```

2. Longest Common Subsequence :

```
#include <bits/stdc++.h>
using namespace std;
string subsequence(vector<vector<int>> LCS, string str1, string str2)
{
    int n = str1.length();
    int m = str2.length();
    int i = n;
    int j = m;
    string str = "";
    while (i > 0 && j > 0)
    {
        if (str1[i - 1] == str2[j - 1])
        {
            str += str1[i - 1];
            i--;
            j--;
        }
        else if (LCS[i - 1][j] > LCS[i][j - 1])
        {
            i--;
        }
        else
        {
            j--;
        }
    }
    reverse(str.begin(), str.end());
    return str;
}
int LCS_length(string str1, string str2)
{
    int n = str1.length();
    int m = str2.length();
    vector<vector<int>> LCS(n + 1, vector<int>(m + 1, 0));
    for (int i = 1; i <= n; i++)
    {
        for (int j = 1; j <= m; j++)
        {
            if (str1[i - 1] == str2[j - 1])
            {
                LCS[i][j] = 1 + LCS[i - 1][j - 1];
            }
            else
                LCS[i][j] = max(LCS[i - 1][j], LCS[i][j - 1]);
        }
    }
    cout << subsequence(LCS, str1, str2) << endl;
    return LCS[n][m];
}
int main()
{
    string str1 = "tiles";
    string str2 = "millets";
    cout << LCS_length(str1, str2);
    return 0;
}
```

OUTPUT :

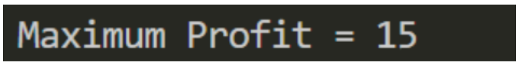


```
iles
4
```

3. 0/1 Knapsack Problem :

```
#include <bits/stdc++.h>
using namespace std;
int max_profit(vector<vector<int>> elements, int m)
{
    sort(elements.begin(), elements.end());
    vector<vector<int>> v(elements.size() + 1, vector<int>(m + 1, 0));
    for (int i = 1; i <= elements.size(); i++)
    {
        for (int w = 1; w <= m; w++)
        {
            if (w - elements[i - 1][1] >= 0)
            {
                v[i][w] = max(v[i - 1][w], v[i - 1][w - elements[i - 1][1]] + elements[i - 1][0]);
            }
            else
            {
                v[i][w] = v[i - 1][w];
            }
        }
    }
    return v[elements.size()][m];
}
int main()
{
    vector<vector<int>> elements = {
        // profit,weight
        {3, 2},
        {5, 3},
        {6, 4},
        {10, 5}};
    int m = 8;
    cout << "Maximum Profit = " << max_profit(elements, m);
    return 0;
}
```

OUTPUT :

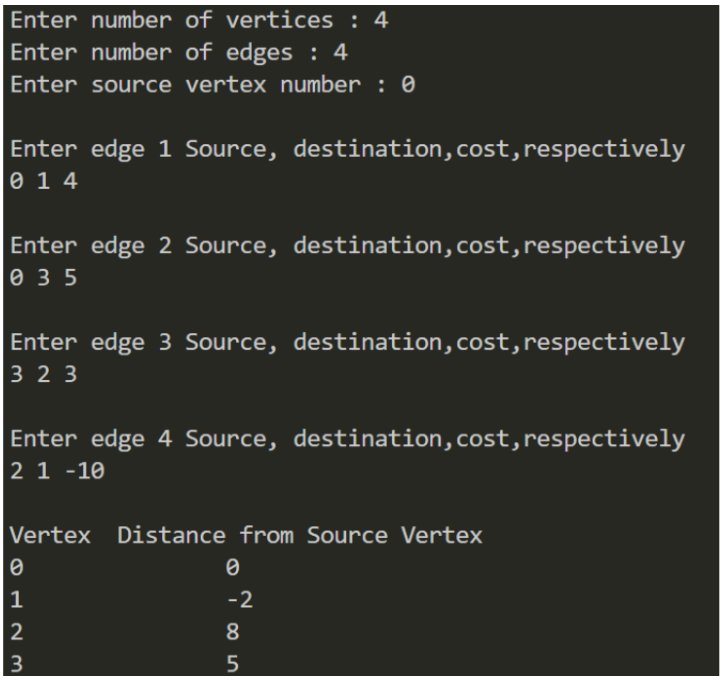


```
Maximum Profit = 15
```

4. Bellman Ford Algorithm :

```
#include <bits/stdc++.h>
using namespace std;
class Edge
{
public:
    int source, destination, cost;
};
class Graph
{
public:
    int V, E;
    struct Edge *edge;
};
Graph *Graph_create(int V, int E)
{
    Graph *graph = new Graph;
    graph->V = V;
    graph->E = E;
    graph->edge = new Edge;
    return graph;
}
void output_final(int dist[], int n)
{
    cout << "\nVertex\tDistance from Source Vertex\n";
    for (int i = 0; i < n; ++i)
        cout << i << "\t\t" << dist[i] << "\n";
}
void Bellman_Ford(Graph *graph, int source)
{
    int V = graph->V;
    int E = graph->E;
    int Distance[V];
    for (int i = 0; i < V; i++)
        Distance[i] = INT_MAX;
    Distance[source] = 0;
    for (int i = 1; i <= V - 1; i++)
    {
        for (int j = 0; j < E; j++)
        {
            int u = graph->edge[j].source;
            int v = graph->edge[j].destination;
            int cost = graph->edge[j].cost;
            if (Distance[u] != INT_MAX && Distance[u] + cost < Distance[v])
                Distance[v] = Distance[u] + cost;
        }
    }
    for (int i = 0; i < E; i++)
    {
        int u = graph->edge[i].source;
        int v = graph->edge[i].destination;
        int cost = graph->edge[i].cost;
        if (Distance[u] != INT_MAX && Distance[u] + cost < Distance[v])
            cout << "\nThis graph contains negative edge cycle\n";
    }
    output_final(Distance, V);
}
int main()
{
    int V, E, S;
    cout << "Enter number of vertices : ";
    cin >> V;
    cout << "Enter number of edges : ";
    cin >> E;
    cout << "Enter source vertex number : ";
    cin >> S;
    Graph *graph = Graph_create(V, E);
    int i;
    for (i = 0; i < E; i++)
    {
        cout << "\nEnter edge " << i + 1 << " Source, destination, cost, respectively\n ";
        cin >> graph->edge[i].source;
        cin >> graph->edge[i].destination;
        cin >> graph->edge[i].cost;
    }
    Bellman_Ford(graph, S);
    return 0;
}
```

OUTPUT :

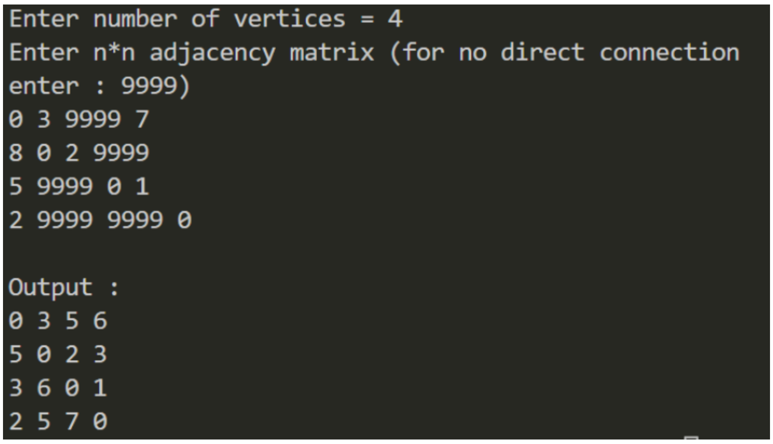


```
Enter number of vertices : 4
Enter number of edges : 4
Enter source vertex number : 0

Enter edge 1 Source, destination,cost,respectively
0 1 4

Enter edge 2 Source, destination,cost,respectively
0 3 5

Enter edge 3 Source, destination,cost,respectively
3 2 3

Enter edge 4 Source, destination,cost,respectively
2 1 -10

Vertex  Distance from Source Vertex
0               0
1               -2
2               8
3               5
```

5. Floyd Warshall Algorithm :

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int matrix[10][10], n, i, j, k;
    cout << "Enter number of vertices = ";
    cin >> n;
    cout << "Enter n*n adjacency matrix (for no direct connection enter : 9999)\n";
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            cin >> matrix[i][j];
        }
    }
    for (k = 0; k < n; k++)
    {
        for (i = 0; i < n; i++)
        {
            if (matrix[i][k] > 0)
            {
                for (j = 0; j < n; j++)
                {
                    if (matrix[k][j] > 0 && matrix[i][j] > (matrix[i][k] + matrix[k][j]))
                    {
                        matrix[i][j] = matrix[i][k] + matrix[k][j];
                    }
                }
            }
        }
    }
    cout << "\nOutput : \n";
    for (i = 0; i < n; i++)
    {
        for (j = 0; j < n; j++)
        {
            cout << matrix[i][j] << " ";
        }
        cout << "\n";
    }
    return 0;
}
```

OUTPUT :



```
Enter number of vertices = 4
Enter n*n adjacency matrix (for no direct connection
enter : 9999)
0 3 9999 7
8 0 2 9999
5 9999 0 1
2 9999 9999 0

Output :
0 3 5 6
5 0 2 3
3 6 0 1
2 5 7 0
```

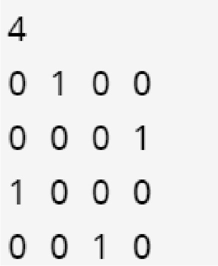
Backtracking :

1. N Queen Problem:

```
#include <iostream>
using namespace std;
bool is_Safe(int **arr, int x, int y, int n){
    for (int row = 0; row < x; row++){
        if (arr[row][y] == 1) {
            return false;
        }
    }
    int row = x;
    int column = y;
    while (row >= 0 && column >= 0){
        if (arr[row][column] == 1){
            return false;
        }
        row--;
        column--;
    }
    row = x;
    column = y;
    while (row >= 0 && column < n){
        if (arr[row][column] == 1){
            return false;
        }
        row--;
        column++;
    }
    return true;
}
```

```
bool nQueen_Algorithm(int **arr, int x, int n){
    if (x >= n){
        return true;
    }
    for (int column = 0; column < n; column++){
        if (is_Safe(arr, x, column, n)){
            arr[x][column] = 1;
            if (nQueen_Algorithm(arr, x + 1, n)){
                return true;
            }
            arr[x][column] = 0; // backtracking
        }
    }
    return false;
}
int main(){
    int n;
    cin >> n;
    int **arr = new int *[n];
    for (int i = 0; i < n; i++){
        arr[i] = new int[n];
        for (int j = 0; j < n; j++){
            arr[i][j] = 0;
        }
    }
    if (nQueen_Algorithm(arr, 0, n)){
        for (int i = 0; i < n; i++){
            for (int j = 0; j < n; {
                cout << arr[i][j] << " ";
            }
            cout << endl;
        } }
    return 0;
}
```

OUTPUT:

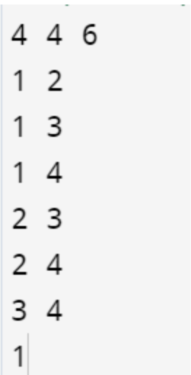


```
4
0 1 0 0
0 0 0 1
1 0 0 0
0 0 1 0
```

2. Vertex cover graph coloring:

```
#include <bits/stdc++.h>
using namespace std;
bool safetoassign(int i, int j, bool graph[101][101], int v, vector<int> &color){
    for (int k = 0; k < v; k++){
        if (graph[i][k] == 1 && color[k] == j)
            return false;
    }
    return true;
}
bool fun(bool graph[101][101], int m, int V, int i, vector<int> &color){
    if (i == V)
        return true;
    for (int j = 0; j < m; j++){
        if (safetoassign(i, j, graph, V, color)){
            color[i] = j;
            if (fun(graph, m, V, i + 1, color))
                return true;
            color[i] = -1;
        }
    }
    return false;
}
bool Graph_Vertex_Coloring(bool graph[101][101], int m, int V){
    vector<int> color(V, -1);
    return fun(graph, m, V, 0, color);
}
int main()
{
    bool graph[101][101];
    int m;
    cin >> m;
    int v;
    cin >> v;
    int edg;
    cin >> edg;
    while (edg--)
    {
        int x, y;
        cin >> x >> y;
        graph[x][y] = 1;
        graph[y][x] = 1;
    }
    cout << Graph_Vertex_Coloring(graph, m, v);
    return 0;
}
```

OUTPUT :



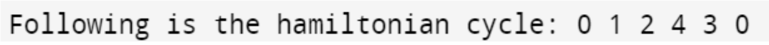
```
4 4 6
1 2
1 3
1 4
2 3
2 4
3 4
1
```

3. Hamiltonian cycles:

```
#include <iostream>
using namespace std;
#define NODE 5
int graph[NODE][NODE] = {
    {0, 1, 0, 1, 0},
    {1, 0, 1, 1, 1},
    {0, 1, 0, 0, 1},
    {1, 1, 0, 0, 1},
    {0, 1, 1, 1, 0},
};
int path[NODE];
void displayCycle(){
    cout << " Following is the hamiltonian cycle: ";
    for (int i = 0; i < NODE; i++){
        cout << path[i] << " ";
    }
    cout << path[0] << endl;
}
bool Valid(int v, int k){
    if (graph[path[k - 1]][v] == 0)
        return false;
    for (int i = 0; i < k; i++)    {
        if (path[i] == v)
```

```
            return false;
    }
    return true;
}
bool FoundCycle(int k){
    if (k == NODE){
        if (graph[path[k - 1]][path[0]] == 1)
            return true;
        else
            return false;
    }
    for (int v = 1; v < NODE; v++){
        if (Valid(v, k)){
            path[k] = v;
            if (FoundCycle(k + 1) == true){
                return true;
            }
            path[k] = -1;
        }
    }
    return false;
}
bool HamiltonianCycle(){
    for (int i = 0; i < NODE; i++){
        path[i] = -1;
    }
    path[0] = 0;
    if (FoundCycle(1) == false){
        cout << "No path possible" << endl;
        return false;
    }
    displayCycle();
    return true;
}
int main(){
    HamiltonianCycle();
}
```

OUTPUT :

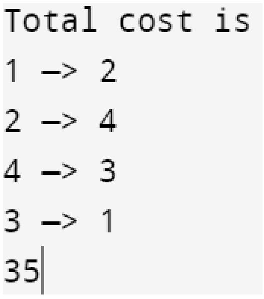


```
Following is the hamiltonian cycle: 0 1 2 4 3 0
```

4. Branch and bound technique:

```
#include <bits/stdc++.h>
using namespace std;
#define N 4
#define INF INT_MAX
struct Node{
    vector<pair<int, int>> path;
    int reducedMatrix[N][N];
    int cost;
    int vertex;
    int level;
};
Node *newNode(int parentMatrix[N][N], vector<pair<int, int>> const &path, int level, int i, int j){
    Node *node = new Node;
    node->path = path;
    if (level != 0)
        node->path.push_back(make_pair(i, j));
    memcpy(node->reducedMatrix, parentMatrix, sizeof node->reducedMatrix);
    for (int k = 0; level != 0 && k < N; k++){
        node->reducedMatrix[i][k] = INF;
        node->reducedMatrix[k][j] = INF;
    }
    node->reducedMatrix[j][0] = INF;
    node->level = level;
    node->vertex = j;
    return node;
}
int rowReduction(int reducedMatrix[N][N], int row[N]){
    fill_n(row, N, INF);
    for (int i = 0; i < N; i++){
        for (int j = 0; j < N; j++){
            if (reducedMatrix[i][j] < row[i]){
                row[i] = reducedMatrix[i][j];
            }
        }
    }
    for (int i = 0; i < N; i++){
        for (int j = 0; j < N; j++){
            if (reducedMatrix[i][j] != INF && row[i] != INF){
                reducedMatrix[i][j] -= row[i];
            }
        }
    }
}
int columnReduction(int reducedMatrix[N][N], int col[N])
{
    fill_n(col, N, INF);
    for (int i = 0; i < N; i++){
        for (int j = 0; j < N; j++){
            if (reducedMatrix[i][j] < col[j])
                col[j] = reducedMatrix[i][j];
        }
    }
    for (int i = 0; i < N; i++){
        for (int j = 0; j < N; j++){
            if (reducedMatrix[i][j] != INF && col[j] != INF)
                reducedMatrix[i][j] -= col[j];
        }
    }
}
int calculateCost(int reducedMatrix[N][N]){
    int cost = 0;
    int row[N];
    rowReduction(reducedMatrix, row);
    int col[N];
    columnReduction(reducedMatrix, col);
    for (int i = 0; i < N; i++){
        cost += (row[i] != INT_MAX) ? row[i] : 0,
        cost += (col[i] != INT_MAX) ? col[i] : 0;
    }
    return cost;
}
void printPath(vector<pair<int, int>> const &list){
    for (int i = 0; i < list.size(); i++){
        cout << list[i].first + 1 << " —> " << list[i].second + 1 << endl;
    }
}
struct comp{
    bool operator()(const Node *lhs, const Node *rhs) const{
        return lhs->cost > rhs->cost;
    }
};
int solve(int costMatrix[N][N]){
    priority_queue<Node *, vector<Node *>, comp> pq;
    vector<pair<int, int>> v;
    Node *root = newNode(costMatrix, v, 0, -1, 0);
    root->cost = calculateCost(root->reducedMatrix);
    pq.push(root);
    while (!pq.empty()){
        Node *min = pq.top();
        pq.pop();
        int i = min->vertex;
        if (min->level == N - 1){
            min->path.push_back(make_pair(i, 0));
            printPath(min->path);
            return min->cost;
        }
        for (int j = 0; j < N; j++){
            if (min->reducedMatrix[i][j] != INF){
                Node *child = newNode(min->reducedMatrix, min->path, min->level + 1, i, j);
                child->cost = min->cost + min->reducedMatrix[i][j] + calculateCost(child->reducedMatrix);
                pq.push(child);
            }
        }
        delete min;
    }
}
int main(){
    int costMatrix[N][N] = {
        {INF, 10, 15, 20},
        {5, INF, 9, 10},
        {6, 13, INF, 12},
        {8, 8, 9, INF}};
    cout << "Total cost is \n"<< solve(costMatrix);
    return 0;
}
```

OUTPUT :

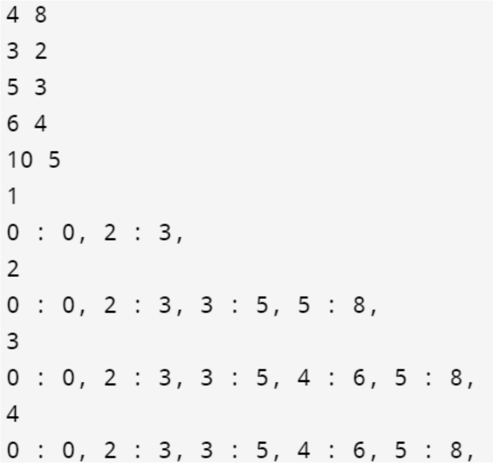


```
Total cost is
1 -> 2
2 -> 4
4 -> 3
3 -> 1
35
```

5. 0/1 Knapsack problem:

```
#include <bits/stdc++.h>
using namespace std;
void const print(map<int, int> ans){
    for (auto i = ans.begin(); i != ans.end(); i++){
        cout << (i->first) << " : " << (i->second) << ", ";
    }
    cout << endl;
}
int main(){
    map<int, int> ans;
    ans[0] = 0;
    int n, a, b, k = 1, m;
    cin >> n >> m;
    vector<pair<int, int>> profit_weight;
    for (int i = 0; i < n; i++){
        cin >> a >> b;
        pair<int, int> x(a, b);
        profit_weight.push_back(x);
    }
    for (auto i = profit_weight.begin(); i != profit_weight.end(); i++, ++k){
        auto x = *i;
        cout << k << endl;
        map<int, int> temp;
        for (auto j = ans.begin(); j != ans.end(); j++){
            if ((j->second + x.first) <= m)
                temp[(j->first) + (x.second)] = (j->second) + (x.first);
        }
        ans.insert(temp.begin(), temp.end());
        print(ans);
    }
    return 0;
}
```

OUTPUT :

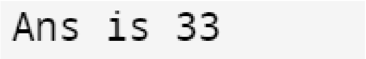


```
4 8
3 2
5 3
6 4
10 5
1
0 : 0, 2 : 3,
2
0 : 0, 2 : 3, 3 : 5, 5 : 8,
3
0 : 0, 2 : 3, 3 : 5, 4 : 6, 5 : 8,
4
0 : 0, 2 : 3, 3 : 5, 4 : 6, 5 : 8,
```

6. Traveling Salesman Problem:

```
#include <bits/stdc++.h >
using namespace std;
int n = 4;
int dp[16][4];
int dist[10][10] = {
    {0, 8, 15, 20},
    {5, 0, 9, 10},
    {6, 13, 0, 12},
    {8, 8, 9, 0}};
int VISIT_ALL = (1 << n) - 1;
int Travelling_salesman(int mask, int pos){
    if (mask == VISIT_ALL)
        return dist[pos][0];
    if (dp[mask][pos] != -1)
        return dp[mask][pos];
    int ans = INT_MAX;
    for (int city = 0; city < n; city++){
        if ((mask & (1 << city)) == 0){
            int newAns = dist[pos][city] + Travelling_salesman(mask | (1 << city), city);
            ans = min(ans, newAns);
        }
    }
    return dp[mask][pos] = ans;
}
int main(){
    for (int i = 0; i < (1 << n); i++){
        for (int j = 0; j < n; j++){
            dp[i][j] = -1;
        }
    }
    cout << "Ans is " << Travelling_salesman(1, 0) << endl;
    return 0;
}
```

OUTPUT :



```
Ans is 33
```