WEEK-1

-->JUMP SEARCH

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

void jumpSearch(int arr[], int n, int key) {
   int step = sqrt(n);
   int prev = 0, comp = 0;
   while (arr[min(step, n)-1] < key) {
      prev = step;
      step += sqrt(n);
      if (prev >= n) {
         cout << "Not Present " << comp << endl;
         return;
      }
      comp++;
   }
   while (arr[prev] < key) {
      prev++;
      if (prev == min(step, n)) {
         cout << "Not Present " << comp << endl;
         return;
      }
      comp++;
   }
   if (arr[prev] == key) {
      cout << "Present " << comp << endl;
      return;
   }
```

```
    cout << "Not Present " << comp << endl;
}

int main() {
    int t;
    cin >> t;
    while (t--) {
        int n, key;
        cin >> n;
        int arr[n];
        for (int i = 0; i < n; i++)
            cin >> arr[i];
        cin >> key;
        jumpSearch(arr, n, key);
    }
    return 0;
}
```

WEEK-2

-->FIND TRIPLET'S

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

void findTriplets(int arr[], int n) {
    bool found = false;
```

```
    for(int i=0; i<n-2; i++) {
        for(int j=i+1; j<n-1; j++) {
            for(int k=j+1; k<n; k++) {
                if(arr[i] + arr[j] == arr[k]) {
                    cout << i+1 << ", " << j+1 << ", " << k+1 << endl;
                    found = true;
                }
            }
        }
    }

    if(!found)
        cout << "No sequence found." << endl;
}

int main() {
    int t, n;
    cin >> t;

    while(t--) {
        cin >> n;
        int arr[n];
        for(int i=0; i<n; i++)
            cin >> arr[i];
        findTriplets(arr, n);
    }

    return 0;
}
```

-->COUNT PAIR'S

```
#include <bits/stdc++.h>
using namespace std;

int countPairs(int arr[], int n, int k) {
    int count = 0;
    sort(arr, arr + n);
    int i = 0, j = 1;
    while (j < n) {
        int diff = arr[j] - arr[i];
        if (diff == k) {
            count++;
            i++;
            j++;
        } else if (diff > k) {
            i++;
        } else {
            j++;
        }
    }
    return count;
}

int main() {
    int t;
    cin >> t;
```

```
    while (t--) {
        int n, k;
        cin >> n;
        int arr[n];
        for (int i = 0; i < n; i++) {
            cin >> arr[i];
        }
        cin >> k;
        int count = countPairs(arr, n, k);
        cout << count << endl;
    }
    return 0;
}
```

WEEK-3

-->SELECTION SORT

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

void selectionSort(int arr[], int n, int& comparisons, int& swaps) {
    for (int i=0;i<n-1;i++)
    {
        int min_idx=i;
        for (int j=i+1;j<n;j++)
        {
```

```
            comparisons++;
            if (arr[j]<arr[min_idx])
            {
                min_idx=j;
            }
        }
        swaps++;
        swap(arr[min_idx],arr[i]);
    }
}

int main()
{
    int t;
    cin>>t;
    while(t--)
    {
        int n;
        cin>>n;
        int arr[n];
        for (int i=0;i<n;i++)
        {
            cin>>arr[i];
        }
        int comparisons=0,swaps=0;
        selectionSort(arr,n,comparisons,swaps);
        for (int i=0;i<n;i++)
        {
            cout<<arr[i]<<" ";
        }
        cout<<endl;
```

```
        cout<<"comparisons = "<<comparisons<<endl;
        cout<<"swaps = "<<swaps<<endl;
    }
    return 0;
}
```

-->INSERTION SORT

```
#include <iostream>
using namespace std;
void insertionSort(int arr[], int n, int &comparisons, int &shifts)
{
    for(int i=1;i<n;i++)
    {
        int key=arr[i];
        int j=i-1;
        while (j>=0&&arr[j]>key)
        {
            arr[j+1]=arr[j];
            j--;
            comparisons++;
            shifts++;
        }
        arr[j+1]=key;
        shifts++;
    }
}
```

```
int main()
{
   int t;
   cin>>t;
   while(t--)
   {
      int n;
      cin>>n;
      int arr[n];
      for(int i=0;i<n;i++)
      {
         cin>>arr[i];
      }
      int comparisons=0,shifts=0;
      insertionSort(arr,n,comparisons,shifts);

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

   return 0;
}
```

-->FIND DUPLICATE'S

```
#include<bits/stdc++.h>
using namespace std;
void duplicate(int arr[],int n){
   sort(arr,arr+n);
   for(int i=1;i<n;i++){
      if(arr[i]==arr[i-1]){
         cout<<"True";
         return ;
      }
   }
   cout<<"False";
}
int main()
{
   int n;
   cout<<"enter the size of array = ";
   cin>>n;
   int arr[n];
   cout<<"enter the elements of array"<<endl;
   for(int i=0;i<n;i++){
      cin>>arr[i];
   }
   duplicate(arr,n);
   return 0;
}
```

WEEK-4

-->MERGE SORT

```
#include <bits/stdc++.h>
using namespace std;

void merge(int arr[],int l,int m,int r,int &comp,int &inv) {
   int n1=m-l+1;
   int n2=r-m;
   int L[n1],R[n2];
   for (int i=0;i<n1;i++)
      L[i]=arr[l+i];
   for (int j=0;j<n2;j++)
      R[j]=arr[m+1+j];
   int i=0,j=0,k=l;
   while (i<n1&&j<n2) {
      comp++;
      if (L[i]<=R[j])
         arr[k++]=L[i++];
      else
      {
         arr[k++]=R[j++];
         inv+=n1-i;
      }
   }
   while(i<n1)
      arr[k++]=L[i++];
   while(j<n2)
```

```
        arr[k++]=R[j++];
}

void mergeSort(int arr[],int l,int r,int &comp,int &inv)
{
    if (l>=r)
        return;
    int m=l+(r-l)/2;
    mergeSort(arr,l,m,comp,inv);
    mergeSort(arr,m+1,r,comp,inv);
    merge(arr,l,m,r,comp,inv);
}

int main() {
    int t;
    cin>>t;
    while(t--)
    {
        int n;
        cin>>n;
        int arr[n];
        for(int i=0;i<n;i++)
            cin>>arr[i];
        int comp=0,inv=0;
        mergeSort(arr,0,n-1,comp,inv);
        for(int i=0;i<n;i++)
            cout<<arr[i]<<" ";
        cout<<"\ncomparisons = "<<comp<< "\ninversions = "<<inv<<"\n";
    }
    return 0;
}
```

-->QUICK SORT

```
#include<bits/stdc++.h>
using namespace std;
int partition(int arr[], int s, int e)
{

    int pivot = arr[s];
    int c = 0;
    for (int i = s + 1; i <= e; i++) {
        if (arr[i] <= pivot)
            c++;
    }
    int pivotIndex = s + c;
    swap(arr[pivotIndex], arr[s]);
    int i = s, j = e;

    while (i < pivotIndex && j > pivotIndex) {

        while (arr[i] <= pivot) i++;
        while (arr[j] > pivot)j--;
        if (i < pivotIndex && j > pivotIndex) {
            swap(arr[i++], arr[j--]);
        }
    }

    return pivotIndex;
```

```
}

void quickSort(int arr[], int start, int end)
{
   if (start >= end)
      return;
   int p = partition(arr, start, end);
   quickSort(arr, start, p - 1);
   quickSort(arr, p + 1, end);
}
int main()
{
   int n;
   cout<<"enter the size of array = ";
   cin>>n;
   int arr[n];
   cout<<"enter the elements of array"<<endl;
   for(int i=0;i<n;i++){
      cin>>arr[i];
   }
   quickSort(arr,0,n-1);
   for(auto i:arr) cout<<i<<" ";
   return 0;
}
```

-->COUNT SORT

```
#include<bits/stdc++.h>
using namespace std;

void getMaxOccurringChar(char arr[], int n)
{
   int count[26]={0};

   for(int i=0;i<n;i++)
   {
      count[arr[i]-'a']++;
   }

   int max_count=0;
   char max_char=' ';
   for(int i=0;i<26;i++)
   {
      if(count[i]>max_count)
      {
         max_count=count[i];
         max_char=char(i + 'a');
      }
   }

   if(max_count > 1)
   {
      cout<<max_char<<" - "<<max_count<<endl;
   }
   else
```

```
    {
        cout<<"No Duplicates Present"<<endl;
    }
}

int main()
{
    int t;
    cin>>t;

    while(t--)
    {
        int n;
        cin>>n;

        char arr[n];
        for(int i=0;i<n;i++)
        {
            cin>>arr[i];
        }

        getMaxOccurringChar(arr, n);
    }
    return 0;
}
```

-->2 SUM

```
#include <iostream>
```

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

int main()
{
   int t;
   cin>>t;
   while(t--)
   {
      int n,key;
      cin>>n;
      int arr[n];
      for(int i=0;i<n;i++)
      {
         cin>>arr[i];
      }
      cin>>key;
      sort(arr,arr+n);
      int i=0,j=n-1;
      bool found=false;
      while(i<j)
      {
         if(arr[i]+arr[j]==key)
         {
            cout<<arr[i]<<" "<<arr[j]<<endl;
            found=true;
            break;
         }
         else if(arr[i]+arr[j]>key)
         {
            j--;
```

```
            }
            else
            {
                i++;
            }
        }
        if (!found)
        {
            cout<<"No Such Element Exist"<<endl;
        }
    }
    return 0;
}
```

-->FIND PAIR

```
#include<bits/stdc++.h>
using namespace std;
void count_sort(int arr[],int n){
    int maxi = 0;
    for(int i=0;i<n;i++){
        maxi = max(maxi,arr[i]);
    }
    int temp[maxi];
    for(int i=0;i<=maxi;i++){
        temp[i] = 0;
    }
    for(int i=0;i<n;i++){
```

```
        temp[arr[i]]++;
    }
    int j=0;
    for(int i=0;i<=maxi;i++){
        while(temp[i]!=0){
            arr[j++] = i;
            temp[i]--;
        }
    }
}
int main()
{
    int n;
    cout<<"enter the size of array = ";
    cin>>n;
    int arr[n];
    cout<<"enter the elements of array"<<endl;
    for(int i=0;i<n;i++){
        cin>>arr[i];
    }
    count_sort(arr,n);
    int k;
    cout<<"enter the kth element index =" ;
    cin>>k;
    cout<<"kth smallest element = "<<arr[k-1]<<endl;
    cout<<"kth largest element = "<<arr[n-k];
    return 0;
}
```

WEEK-6

-->DFS

```
#include <bits/stdc++.h>

using namespace std;

bool dfs(vector<vector<int>>& graph, vector<bool>& visited, int source, int destination) {
    visited[source] = true;
    if (source == destination)
        return true;

    for (int neighbor = 0; neighbor < graph.size(); ++neighbor) {
        if (graph[source][neighbor] == 1 && !visited[neighbor]) {
            if (dfs(graph, visited, neighbor, destination))
                return true;
        }
    }
    return false;
}

bool isPathExists(vector<vector<int>>& graph, int source, int destination) {
    vector<bool> visited(graph.size(), false);
    return dfs(graph, visited, source, destination);
}

int main() {
    int n;
    cin >> n;
```

```
    vector<vector<int>> graph(n, vector<int>(n));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> graph[i][j];
        }
    }

    int source, destination;
    cin >> source >> destination;

    if (isPathExists(graph, source, destination))
        cout << "YES PATH EXISTS" << endl;
    else
        cout << "NO SUCH PATH EXISTS" << endl;

    return 0;
}
```

-->BFS(bipartite graph)

```
#include <bits/stdc++.h>

using namespace std;
bool isBipartite(vector<vector<int>>& graph, int source) {
    int numVertices = graph.size();
```

```
    vector<int> color(numVertices, -1);

    color[source] = 0;

    queue<int> q;
    q.push(source);

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

        for (int neighbor : graph[currVertex]) {
            if (color[neighbor] == -1) {
                color[neighbor] = 1 - color[currVertex];
                q.push(neighbor);
            }
            else if (color[neighbor] == color[currVertex]) {
                return false;
            }
        }
    }

    return true;
}

int main() {
    int n;
    cin >> n;

    vector<vector<int>> graph(n, vector<int>(n));
    for (int i = 0; i < n; ++i) {
```

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

    if (isBipartite(graph, 0))
        cout << "Yes Bipartite" << endl;
    else
        cout << "Not Bipartite" << endl;

    return 0;
}
```

-->cycle exists

```
#include <bits/stdc++.h>

using namespace std;

bool dfs(vector<vector<int>>& graph, vector<bool>& visited, vector<bool>& recursionStack, int vertex) {
    visited[vertex] = true;
    recursionStack[vertex] = true;

    for (int neighbor : graph[vertex]) {
        if (!visited[neighbor]) {
            if (dfs(graph, visited, recursionStack, neighbor))
                return true;
        }
```

```
        else if (recursionStack[neighbor]) {
            return true;
        }
    }

    recursionStack[vertex] = false;
    return false;
}

bool isCycleExists(vector<vector<int>>& graph) {
    int numVertices = graph.size();

    vector<bool> visited(numVertices, false);
    vector<bool> recursionStack(numVertices, false);

    for (int i = 0; i < numVertices; ++i) {
        if (!visited[i]) {
            if (dfs(graph, visited, recursionStack, i))
                return true;
        }
    }
    return false;
}

int main() {
    int n;
    cin >> n;
    vector<vector<int>> graph(n, vector<int>(n));
    for (int i = 0; i < n; ++i) {
        for (int j = 0; j < n; ++j) {
            cin >> graph[i][j];
```

```
        }
    }

    if (isCycleExists(graph))
        cout << "Yes Cycle Exists" << endl;
    else
        cout << "No Cycle Exists" << endl;

    return 0;
}
```

WEEK-7

-->BELLMAN FORD

```
#include <bits/stdc++.h>

using namespace std;

void printPath(const vector<int>& parent, int v) {
    if (parent[v] == -1) {
        cout << v;
        return;
    }
    printPath(parent, parent[v]);
    cout << " " << v;
```

```
}

void bellmanFord(vector<vector<pair<int, int>>>& graph, int V, int source) {
    vector<int> distance(V, numeric_limits<int>::max());
    vector<int> parent(V, -1);

    distance[source] = 0;
    for (int i = 0; i < V - 1; ++i) {
        for (int u = 0; u < V; ++u) {
            for (auto edge : graph[u]) {
                int v = edge.first;
                int weight = edge.second;

                if (distance[u] != numeric_limits<int>::max() && distance[u] + weight < distance[v]) {
                    distance[v] = distance[u] + weight;
                    parent[v] = u;
                }
            }
        }
    }

    for (int u = 0; u < V; ++u) {
        for (auto edge : graph[u]) {
            int v = edge.first;
            int weight = edge.second;

            if (distance[u] != numeric_limits<int>::max() && distance[u] + weight < distance[v]) {
                cout << "Negative weight cycle found!" << endl;
                return;
            }
        }
```

```
    }

    for (int v = 0; v < V; ++v) {
        cout << v + 1 << " : ";
        printPath(parent, v);
        cout << " : " << distance[v] << endl;
    }
}

int main() {
    int V;
    cin >> V;
    vector<vector<pair<int, int>>> graph(V);
    for (int u = 0; u < V; ++u) {
        for (int v = 0; v < V; ++v) {
            int weight;
            cin >> weight;
            if (weight != 0) {
                graph[u].push_back({v, weight});
            }
        }
    }

    int source;
    cin >> source;
    bellmanFord(graph, V, source - 1);

    return 0;
}
```

-->weighted path

```
#include <bits/stdc++.h>

using namespace std;
void shortestPathWithKEdges(vector<vector<int>>& graph, int V, int source, int destination, int k) {

   vector<vector<vector<int>>> distance(V, vector<vector<int>>(V, vector<int>(k + 1,
numeric_limits<int>::max())));

   for (int i = 0; i <= k; ++i) {
      distance[source][source][i] = 0;
   }

   for (int count = 0; count <= k; ++count) {
      for (int u = 0; u < V; ++u) {
         for (int v = 0; v < V; ++v) {
            if (graph[u][v] != 0) {
               for (int edgeCount = 0; edgeCount < k; ++edgeCount) {
                  if (distance[source][u][edgeCount] != numeric_limits<int>::max() &&
distance[source][u][edgeCount] + graph[u][v] < distance[source][v][edgeCount + 1]) {
                     distance[source][v][edgeCount + 1] = distance[source][u][edgeCount] + graph[u][v];
                  }
               }
            }
         }
      }
   }

   if (distance[source][destination][k] != numeric_limits<int>::max()) {
```

```
        cout << "Weight of shortest path from (" << source + 1 << "," << destination + 1 << ") with " << k
<< " edges : " << distance[source][destination][k] << endl;
    } else {
        cout << "No path of length " << k << " is available" << endl;
    }
}

int main() {
    int V;
    cin >> V;
    vector<vector<int>> graph(V, vector<int>(V));
    for (int i = 0; i < V; ++i) {
        for (int j = 0; j < V; ++j) {
            cin >> graph[i][j];
        }
    }

    int source, destination, k;
    cin >> source >> destination >> k;
    shortestPathWithKEdges(graph, V, source - 1, destination - 1, k);

    return 0;
}
```

WEEK-9

-->FLOYD WARSHAL

```
#include <bits/stdc++.h>
using namespace std;
#define INF INT_MAX
void printSolution(int V, vector<vector<int>> dist)
{
   for (int i = 0; i < V; i++)
   {
      for (int j = 0; j < V; j++)
      {
         if (dist[i][j] == INF)
            cout << "INF"
               << " ";
         else
            cout << dist[i][j] << " ";
      }
      cout << endl;
   }
}
void floydWarshall(int V, vector<vector<int>> graph)
{
   int i, j, k;
   vector<vector<int>> dist(V, vector<int>(V));
   for (i = 0; i < V; i++)
      for (j = 0; j < V; j++)
         dist[i][j] = graph[i][j];
   for (k = 0; k < V; k++)
   {
```

```
        for (i = 0; i < V; i++)
        {
            for (j = 0; j < V; j++)
            {
                if (dist[i][j] > (dist[i][k] + dist[k][j]) && (dist[k][j] != INF && dist[i][k] != INF))
                    dist[i][j] = dist[i][k] + dist[k][j];
            }
        }
    }
    cout << "Shortest Distance Matrix:" << endl;
    printSolution(V, dist);
}
int main()
{
    int V = 5;
    vector<vector<int>> graph{{0, 10, 5, 5, INF},
                              {INF, 0, 5, 5, 5},
                              {INF, INF, 0, INF, 10},
                              {INF, INF, INF, 0, 20},
                              {INF, INF, INF, 5, 0}};
    floydWarshall(V, graph);
    return 0;
}
```

-->FRACTIONAL KNAPSACK PROBLEM

```
#include <bits/stdc++.h>
using namespace std;
```

```
int main()
{
    int n, w, p;
    cin >> n;
    vector<int> weight;
    vector<int> price;
    for (int i = 0; i < n; i++)
    {
        cin >> w;
        weight.push_back(w);
    }
    for (int i = 0; i < n; i++)
    {
        cin >> p;
        price.push_back(p);
    }
    cin >> w;
    vector<pair<double, int>> perw;
    for (int i = 0; i < n; i++)
    {
        perw.push_back({((double)price[i]) / weight[i], i});
    }
    sort(perw.begin(), perw.end());
    vector<int> ans(n, 0);
    double temp = w;
    int i = n - 1;
    double res = 0;
    while (temp > 0)
    {
        if ((weight[perw[i].second]) <= temp)
        {
```

```
            res += price[perw[i].second];
            temp -= weight[(perw[i].second)];
            ans[i] = weight[(perw[i].second)];
        }
        else
        {
            res += temp * perw[i].first;
            ans[i] = temp;
            temp -= temp;
        }
        i--;
    }
    cout << "Maximum value : " << res << endl;
    cout << "item-weight" << endl;
    for (int i = 0; i < n; i++)
    {
        cout << i + 1 << "-" << ans[i] << endl;
    }
    return 0;
}
```

WEEK-10

-->ACTIVITY SELECTION

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n, s, f;
    cin >> n;
    vector<int> start, finish;
    for (int i = 0; i < n; i++)
    {
        cin >> s;
        start.push_back(s);
    }
    for (int i = 0; i < n; i++)
    {
        cin >> f;
        finish.push_back(f);
    }
    vector<pair<int, int>> v;
    for (int i = 0; i < n; i++)
    {
        v.push_back({finish[i], i});
    }
    sort(v.begin(), v.end());
    int count = 1, i = v[0].second;
    vector<int> ans;
    ans.push_back(i + 1);
    for (int j = 1; j < n; j++)
    {
        if (start[v[j].second] >= finish[i])
        {
            count++;
```

```
            ans.push_back(v[j].second + 1);
            i = v[j].second;
        }
    }
    cout << "No. of non-conflicting activities: " << count << endl;
    cout << "List of selected activities: ";
    for (int i = 0; i < ans.size(); i++)
    {
        cout << ans[i] << " ";
    }
    return 0;
}
```

-->TASKS DONE

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
    int n, num;
    cin >> n;
    vector<int> task;
    vector<int> finish;
    for (int i = 0; i < n; i++)
    {
        cin >> num;
        task.push_back(num);
    }
```

```
for (int i = 0; i < n; i++)
{
   cin >> num;
   finish.push_back(num);
}
vector<pair<int, int>> v;
for (int i = 0; i < n; i++)
{
   v.push_back({finish[i], i});
}
sort(v.begin(), v.end());
int i = 0;
vector<int> res;
res.push_back(v[i].second + 1);
int temp = task[v[i].second];
for (i = 1; i < n; i++)
{
   if (finish[v[i].second] - temp >= task[v[i].second])
   {
      res.push_back(v[i].second + 1);
      temp += task[v[i].second];
   }
}
cout << "Max number of tasks =" << res.size() << endl;
cout << "Selected task numbers :";
sort(res.begin(), res.end());
for (int i = 0; i < res.size(); i++)
{
   cout << res[i] << " ";
}
return 0;
```

```
}
```

-->MAJORITY ELEMENT

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
   int n;
   cin >> n;
   int arr[n];
   for (int i = 0; i < n; i++)
   {
      cin >> arr[i];
   }
   sort(arr + 0, arr + n);
   if (((arr[0] == arr[n / 2]) || (arr[n / 2] == arr[n - 1])) && n % 2 != 0)
      cout << "Yes" << endl;
   else if (n % 2 == 0 && ((arr[0] == arr[n / 2]) || (arr[n / 2] == arr[n - 1] && arr[n / 2] == arr[n / 2 -1])))
      cout << "Yes" << endl;
   else
      cout << "No" << endl;
   if (n % 2 != 0)
      cout << arr[n / 2] << endl;
   else
   {
      cout << (arr[(n - 1) / 2] + arr[n / 2]) / 2.0 << endl;
   }
   return 0;
```

```
}
```

WEEK-11

-->SEQUENCE OF MATRIX

```
#include <bits/stdc++.h>
using namespace std;
int solve(vector<int> arr, int i, int j, vector<vector<int>> &dp)
{
    if (i >= j)
        return 0;
    if (dp[i][j] != -1)
        return dp[i][j];
    int mn = INT_MAX;
    for (int k = i; k < j; k++)
    {
        int tempans = solve(arr, i, k, dp) + solve(arr, k + 1, j, dp) + arr[i - 1] * arr[k] * arr[j];
        if (tempans < mn)
            mn = tempans;
    }
    return dp[i][j] = mn;
}
```

```
int main()
{
    int n, n1, n2;
    cin >> n;
    vector<int> arr;
    for (int i = 0; i < n; i++)
    {
        cin >> n1 >> n2;
        if (i == 0)
        {
            arr.push_back(n1);
            arr.push_back(n2);
        }
        else
            arr.push_back(n2);
    }
    n = arr.size();
    vector<vector<int>> dp(n + 1, vector<int>(n + 1, -1));
    cout << solve(arr, 1, n - 1, dp);
}
```

WEEK-12

-->LONGEST COMMON SUBSEQUENCE

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
   string str1, str2;
   cout << "Sequence1: ";
   cin >> str1;
   cout << "Sequence2: ";
   cin >> str2;
   int n = str1.size(), m = str2.size();
   int t[n + 1][m + 1];
   for (int i = 0; i <= n; i++)
   {
      for (int j = 0; j <= m; j++)
      {
         if (i == 0 || j == 0)
         {
            t[i][j] = 0;
         }
      }
   }
   for (int i = 1; i <= n; i++)
   {
      for (int j = 1; j <= m; j++)
      {
         if (str1[i - 1] == str2[j - 1])
         {
            t[i][j] = 1 + t[i - 1][j - 1];
         }
         else
         {
```

```
            t[i][j] = max(t[i][j - 1], t[i - 1][j]);
        }
      }
    }
    string str = "";
    int i = n, j = m;
    while (i > 0 && j > 0)
    {
      if (str1[i - 1] == str2[j - 1])
      {
        str += str1[i - 1];
        i--;
        j--;
      }
      else
      {
        if (t[i][j - 1] > t[i - 1][j])
          j--;
        else
          i--;
      }
    }
    reverse(str.begin(), str.end());
    cout << "Longest Common Subsequence: " << str << endl;
    cout << "Length = " << t[n][m] << endl;
}
```

-->0/1 KNAPSACK PROBLEM

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
   int n, w, p;
   cin >> n;
   vector<int> wgt;
   vector<int> price;
   for (int i = 0; i < n; i++)
   {
      cin >> w;
      wgt.push_back(w);
   }
   for (int i = 0; i < n; i++)
   {
      cin >> p;
      price.push_back(p);
   }
   cin >> w;
   int dp[n + 1][w + 1];
   for (int i = 0; i <= n; i++)
   {
      for (int j = 0; j <= w; j++)
      {
         if (i == 0 || j == 0)
            dp[i][j] = 0;
      }
   }
   vector<int> v1, v2;
   for (int i = 1; i <= n; i++)
   {
```

```
    for (int j = 1; j <= w; j++)
    {
        if (wgt[i - 1] <= j)
        {
            dp[i][j] = max(price[i - 1] + dp[i - 1][j - wgt[i - 1]], dp[i - 1][j]);
        }
        else
        {
            dp[i][j] = dp[i - 1][j];
        }
    }
}
int res = dp[n][w];
for (int i = n; i > 0 && res > 0; i--)
{
    if (res == dp[i - 1][w])
        continue;
    else
    {
        v1.push_back(i - 1);
        res = res - price[i - 1];
        w = w - wgt[i - 1];
    }
}
cout << "Value = " << dp[n][w] << endl;
cout << "Weights selected : ";
for (int i = 0; i < v1.size(); i++)
{
    cout << wgt[v1[i]] << " ";
}
cout << endl
```

```
        << "Weights selected : ";
    for (int i = 0; i < v1.size(); i++)
    {
        cout << price[v1[i]] << " ";
    }
    return 0;
}
```

WEEK-13

-->FREQUENCE OF WORDS

```
#include <bits/stdc++.h>

using namespace std;

void findDistinctCharacters(const vector<char>& arr) {
    unordered_map<char, int> freqMap;

    for (char ch : arr) {
        freqMap[ch]++;
    }
```

```
    vector<char> distinctChars;
    for (auto it = freqMap.begin(); it != freqMap.end(); ++it) {
        distinctChars.push_back(it->first);
    }
    sort(distinctChars.begin(), distinctChars.end());

    for (char ch : distinctChars) {
        cout << ch << " " << freqMap[ch] << endl;
    }
}

int main() {
    int n;
    cin >> n;
    vector<char> arr(n);
    for (int i = 0; i < n; ++i) {
        cin >> arr[i];
    }
    findDistinctCharacters(arr);

    return 0;
}
```