DEPSTAR

Devang Patel Institute of Advance Technology and Research

(A Constitute Institute of CHARUSAT)

Certificate

This is to certify that

Mr./Mrs. Shah Dhruval M.

of 6CE-2 *Class,*

ID. No. 22DCE109 *has satisfactorily completed*

his/ her term work in CE397 Mastering Competitive Programming *for*

the ending in April 2024/2025

Date : 1/4/'25

Sign. of Faculty

Dgarg

Head of Department

CHAROTAR UNIVERSITY OF SCIENCE AND TECHNOLOGY
FACULTY OF TECHNOLOGY AND ENGINEERING (FTE)
Chandubhai S. Patel Institute of Technology (CSPIT) &
Devang Patel Institute of Advance Technology and Research (DEPSTAR)

ACADEMIC YEAR: 2024-25

Practical List

Subject: OCCE3004 BLOCKCHAIN AND ITS APPLICATIONS (PE-III) (6th Sem)

Exp. No.	Name of Experiment	Hours
1.	Study Blockchain Architecture and Block structure. Perform following tasks: I. Create Blockchain which consists of 5 blocks. [First Block is genesis block]. II. Validate the blocks by implementing mining logic [Cryptographically Mathematical Puzzles] III. Set the difficulty of blocks. IV. Keep timestamp, block version, data and previous hash as block parameters. V. Verify the blocks. VI. Access all transaction. VII. Modify the block data. Implement above all functionalities using Node js/Python/Java/C++/go lang.	2
2.	Install and configure the following development setup tools to implement Blockchain development. o Metamask (Wallet) o Ganache Local Private Blockchain Network o Go-Ethereum (Geth) Client o Truffle framework o Hardhat framework Study and configure all testnets available in Metamask and also setup custom network using Ganache.	2
3.	Configure Geth (Go-Ethereum) over Windows/Linux/Mac operating system. Perform the following tasks: o Build Your Own Ethereum Private Blockchain with two peer nodes locally. o Build Your Own Ethereum Private Blockchain with two peer nodes as geographically distributed. o Configure Ethereum testnet such as Sepolia, Goerli and Layer 2.	4

4.	Implement basic of Solidity programming: Syntax, Variables, Functions, mapping, access modifiers using Remix online IDE. Use online REMIX IDE by Ethereum to run solidity code.	6
5.	Write a Smart-contract of EVENT PARTICIPATION CERTIFICATE which includes the following things o Write code using 0.8.0 or higher version of solidity. o Define a struct to define different properties of certificate issues. o Define mapping function to struct o Create a constructor to initialize the first transaction using admin only. o Create function to issue a new certificate with certificate_id, student_name, branch, course_name and grade o Initialize proper datatypes of all variable. o Display the certificate details by certificate_id o Verify the certificate hash. Use online Ethereum REMIX IDE to compile and deploy the contract. Integrate the metamask wallet to perform the transaction of certificate.	2
6.	Write a Smartcontract to store details of PATIENTS's details which includes the following things o Write code using 0.8.0 or higher version of solidity. o Define struct to define different properties of adding patient's details. o Define mapping function to struct o Create a constructor to initialize first transaction using admin only. o Create function to add details of patient such as patient_id, patient_name, decease type, doctor_name and patient_contact. o Initialize proper datatypes of all variable. o Display the patient's details by patient_id o Display all patient's details Use online Ethereum REMIX IDE to compile and deploy the contract. Integrate the metamask wallet to perform the transaction of certificate.	2
7.	Refer to Practical 5: Use Truffle to compile and deploy the contract on Ganache and Integrate the Metamask to perform transactions. Test the smart contract using the Mocha and Chai framework before deployment.	2
8.	Refer to Practical 6: Use Hardhat to compile and deploy the contract using	2

	INFURA API and Integrate the Metamask to perform transactions. Test the smart contract using the Mocha and Chai framework before deployment.	
9.	Create Dapp (Decentralized Application) and link your client-side application with Blockchain network created in practical 5th and 6th .	2
10.	Install and configure the Hyperledger Fabric using the Linux platform. Create permissioned network and implement chain code to transfer the assets from owner to buyer.	2
11.	Create a CHARUSAT Crypto Coin and transfer the initial coin from Admin to different accounts.	2
12.	Create Non-Fungible-Token (NFT) for CHARUSAT Event. NFTs contain meta information such as date, original owner, and offers.	2

PRACTICAL: 1

AIM: Searching and sorting

1.1 Ferris wheel:

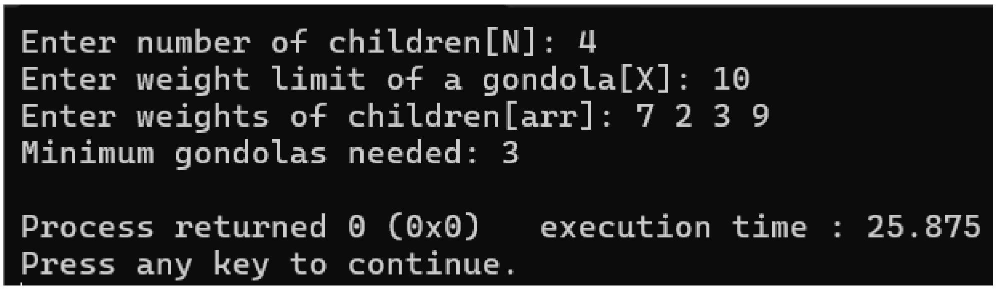
There are N children who want to go to a Ferris wheel in the form of an array arr[], and your task is to find a gondola for each child. Each gondola may have one or two children in it, and in addition, the total weight in a gondola may not exceed X. You know the weight of every child. What is the minimum number of gondolas needed for the children?
Input: N = 4, X = 10, arr[] = {7, 2, 3, 9}
Output: 3
Explanation: We need only 3 gondolas: {2, 3}, {7} and {9}.
Input: N = 4, X = 6, arr[] = {2, 3, 3, 4}
Output: 2
Explanation: We need only 2 gondolas: {2, 4} and {3, 3}

CODE:

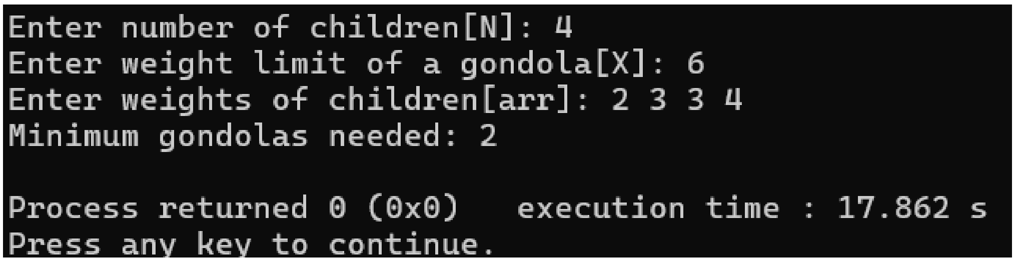
```
#include <bits/stdc++.h>
#define ll long long
using namespace std;

int solve(ll* arr, ll N, ll X) {
   sort(arr, arr + N); // Sort the array
   ll l = 0, h = N - 1, ans = 0;

   while (h >= l) {
      if (arr[l] + arr[h] <= X) l++; // Pair the lightest with heaviest
      h--; // Heaviest always gets a gondola
      ans++;
   }
   return ans;
}

int main() {
   ll N, X;
   cout << "Enter number of children[N]: ";
   cin >> N;

   ll arr[N];
   cout << "Enter weight limit of a gondola[X]: ";
   cin >> X;

   cout << "Enter weights of children[arr]: ";
```

```
    for (ll i = 0; i < N; i++) cin >> arr[i];

    cout << "Minimum gondolas needed: " << solve(arr, N, X) << endl;
    return 0;
}
```

OUTPUT:



```
Enter number of children[N]: 4
Enter weight limit of a gondola[X]: 10
Enter weights of children[arr]: 7 2 3 9
Minimum gondolas needed: 3

Process returned 0 (0x0)   execution time : 25.875
Press any key to continue.
```



```
Enter number of children[N]: 4
Enter weight limit of a gondola[X]: 6
Enter weights of children[arr]: 2 3 3 4
Minimum gondolas needed: 2

Process returned 0 (0x0)   execution time : 17.862 s
Press any key to continue.
```

1.2 Sort Vowels in a String:

Given a 0-indexed string s, permute s to get a new string t suchthat: All consonants remain in their original places. More formally,if there is an index i with 0 <= i < s.length such that s[i] is a consonant, then t[i] = s[i].The vowels must be sorted in the nondecreasing order of their ASCII values. More formally, for pairs of indices i, j with 0

<=i < j < s.length such that s[i] and s[j] are vowels, then t[i] must not have a higher ASCII value than t[j]. Return the resulting string. The vowels are 'a', 'e', 'i', 'o', and 'u', and they can appear in lowercase or uppercase. Consonants comprise all letters that

arenot vowels.

Example 1:

Input: s ="lEetcOde"

Output:"lEOtcede"

Explanation: 'E', 'O', and 'e' are the vowels in s; 'l', 't', 'c', and 'd'are all consonants. The vowels are sorted according to their ASCII values, and the consonants remain in the same places.

Example 2:

Input: s ="lYmpH"

Output:"lYmpH"

Explanation: There are no vowels in s (all characters in s are consonants), so we return "lYmpH".

Constraints:

1 <= s.length <= 10^5

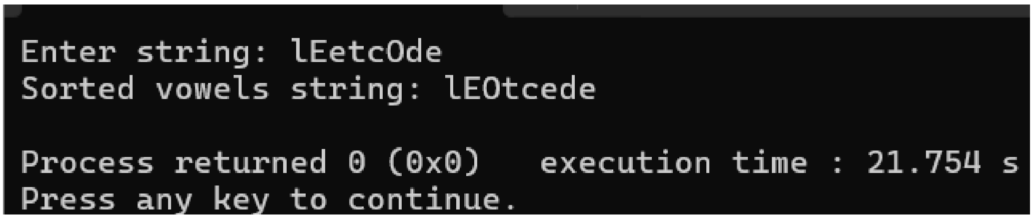
s consists only of letters of the English alphabet in uppercaseand

lowercase.

Code:

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

class Solution {
    static bool vowels(char ch) { // Make function static
        return (ch == 'a' || ch == 'e' || ch == 'i' || ch == 'o' || ch == 'u' ||
                ch == 'A' || ch == 'E' || ch == 'I' || ch == 'O' || ch == 'U');
    }
public:
    string sortVowels(string s) {
        vector<char> r;

        // Collect vowels
        for (char c : s) {
            if (vowels(c)) r.push_back(c);
        }
        // Sort vowels
        sort(r.begin(), r.end());

        // Replace vowels in original positions
        int i = 0;
        for (char &c : s) {
            if (vowels(c)) c = r[i++];
        }

        return s;
    }
};

int main() {
    Solution sol;
    string s;
    cout << "Enter string: ";
    cin >> s;
    cout << "Sorted vowels string: " << sol.sortVowels(s) << endl;
    return 0;
}
```

Output:



```
Enter string: lEetcOde
Sorted vowels string: lEOtcede

Process returned 0 (0x0)   execution time : 21.754 s
Press any key to continue.
```

PRACTICAL: 2

AIM: Algorithm design

2.1 Two pointers:

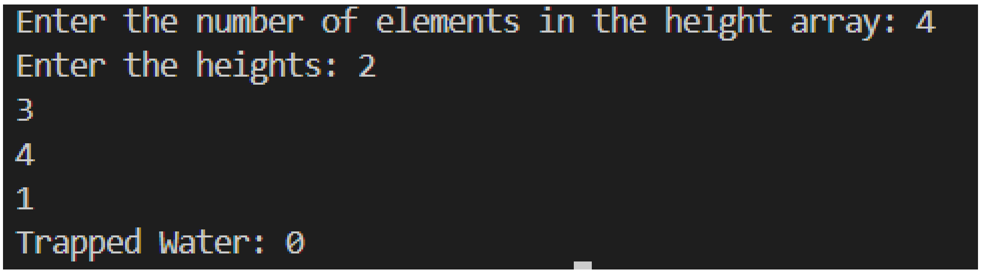
Given n non-negative integers representing an elevation map where the width of each bar is 1, compute how much water it can trap after raining.

Example 1:
Input: height = [0,1,0,2,1,0,1,3,2,1,2,1]
Output: 6

CODE:

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

int trap(vector<int>& height) {
    int left = 0, right = height.size() - 1;
    int leftMax = 0, rightMax = 0, trappedWater = 0;

    while (left < right) {
        if (height[left] < height[right]) {
            if (height[left] >= leftMax)
                leftMax = height[left];
            else
                trappedWater += leftMax - height[left];
            left++;
        } else {
            if (height[right] >= rightMax)
                rightMax = height[right];
            else
                trappedWater += rightMax - height[right];
            right--;
        }
    }
    return trappedWater;
}

int main() {
    int n;
    cout << "Enter the number of elements in the height array: ";
    cin >> n;
    vector<int> height(n);
    cout << "Enter the heights: ";
    for (int i = 0; i < n; i++) {
        cin >> height[i];
    }

    cout << "Trapped Water: " << trap(height) << endl;
    return 0;
}
```

OUTPUT:



```
Enter the number of elements in the height array: 4
Enter the heights: 2
3
4
1
Trapped Water: 0
```

2.2 Next greater/smaller element:

Given an array of integers temperatures represents the daily temperatures, return an array answer such that answer[i] is the number of days you have to wait after the ith day to get a warmer temperature. If there is no future day for which this ispossible, keep answer[i] == 0 instead.

Example 1:

Input: temperatures = [73,74,75,71,69,72,76,73]

Output:

[1,1,4,2,1,1,0,0]

Example 2:

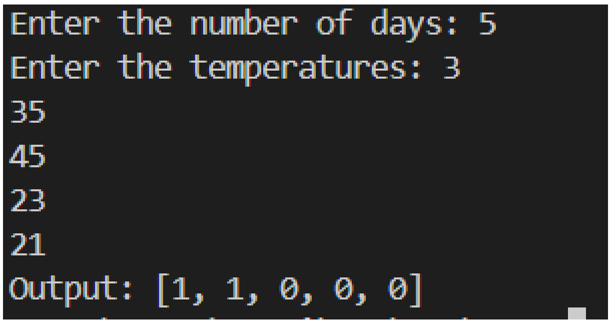
Input: temperatures =

[30,40,50,60]Output: [1,1,1,0]

Code:

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

vector<int> dailyTemperatures(vector<int>& temperatures) {
    int n = temperatures.size();
    vector<int> answer(n, 0);
    stack<int> st;

    for (int i = 0; i < n; i++) {
        while (!st.empty() && temperatures[i] > temperatures[st.top()]) {
            int prevIndex = st.top();
            st.pop();
            answer[prevIndex] = i - prevIndex;
        }
        st.push(i);
    }
    return answer;
}

int main() {
    int n;
    cout << "Enter the number of days: ";
    cin >> n;
    vector<int> temperatures(n);
    cout << "Enter the temperatures: ";
    for (int i = 0; i < n; i++) {
        cin >> temperatures[i];
    }

    vector<int> result = dailyTemperatures(temperatures);

    cout << "Output: [";
    for (int i = 0; i < n; i++) {
        cout << result[i];
        if (i < n - 1) cout << ", ";
    }
    cout << "]" << endl;

    return 0;
}
```

Output:



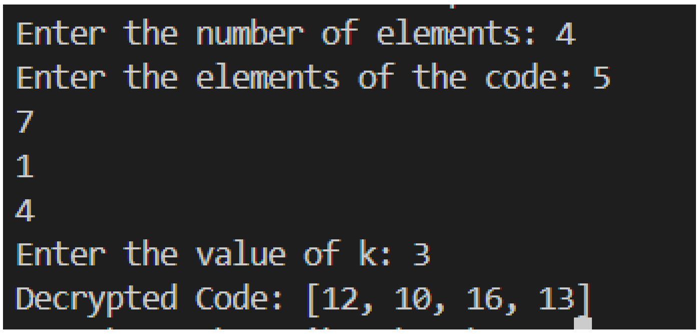
```
Enter the number of days: 5
Enter the temperatures: 3
35
45
23
21
Output: [1, 1, 0, 0, 0]
```

2.3Sliding window:

You have a bomb to defuse, and your time is running out! Your informer will provide you with a circular array code of length of n and a key k. To decrypt the code, you must replace every number. All the numbers are replaced simultaneously. If k > 0, replace the ith number with the sum of the next k numbers. If k < 0, replace the ith number with the sum of the previous k numbers. If k == 0, replace the ith number with 0. As code is circular, the next element of code[n-1] is code[0],and the previous element of code[0] is code[n-1]. Given the circular array code and an integer key k, return the decrypted code to defuse the bomb!
Example 1:
Input: code = [5,7,1,4], k = 3
Output: [12,10,16,13]
Explanation: Each number is replaced by the sum of the next
3numbers. The decrypted code is [7+1+4, 1+4+5, 4+5+7,
5+7+1]. Notice that the numbers wrap around.
Example 2:
Input: code = [1,2,3,4], k = 0
Output: [0,0,0,0]
Explanation: When k is zero, the numbers are replaced by 0.
Example 3:
Input: code = [2,4,9,3], k = -2
Output: [12,5,6,13]
Explanation: The decrypted code is [3+9, 2+3, 4+2, 9+4]. Notice
that the numbers wrap around again. If k is negative, the sum is of the previous numbers.
Constraints:
n ==
code.length1
<= n <= 100
1 <= code[i] <= 100
-(n - 1) <= k <= n – 1

CODE:

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

vector<int> decrypt(vector<int>& code, int k) {
    int n = code.size();
    vector<int> result(n, 0);

    if (k == 0) return result;

    int sum = 0, start, end;

    if (k > 0) {

        for (int i = 1; i <= k; i++)
            sum += code[i % n];

        for (int i = 0; i < n; i++) {
            result[i] = sum;
            sum -= code[(i + 1) % n];
            sum += code[(i + k + 1) % n];
        }
    }
    else {
        k = -k;
        for (int i = 1; i <= k; i++)
            sum += code[(n - i) % n];

        for (int i = 0; i < n; i++) {
            result[i] = sum;
            sum -= code[(n + i - k) % n];
            sum += code[i];
        }
    }
    return result;
}

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

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

    cout << "Enter the value of k: ";
    cin >> k;

    vector<int> decryptedCode = decrypt(code, k);

    cout << "Decrypted Code: [";
    for (int i = 0; i < n; i++) {
        cout << decryptedCode[i];
        if (i < n - 1) cout << ", ";
    }
    cout << "]" << endl;

    return 0;
}
```



```
Enter the number of elements: 4
Enter the elements of the code: 5
7
1
4
Enter the value of k: 3
Decrypted Code: [12, 10, 16, 13]
```

PRACTICAL: 3

AIM: String processing

3.1Wildcard matching:

Given an input string (s) and a pattern (p), implement Wildcard pattern matching with support for '?' and '*' where: '?' Matches any single character.'*' Matches any sequence of characters (including the empty sequence).The matching should cover the entire input string (not partial).

Example 1:

Input: s = "aa", p =

"a"Output: false

Explanation: "a" does not match the entire string "aa".

Example 2:

Input: s = "aa", p ="*"

Output: true

Explanation: '*' matches any sequence.

Example 3:

Input: s = "cb", p ="?a"

Output: false

Explanation: '?' matches 'c', but the second letter is 'a', Which does not match 'b'.

Constraints:

0 <= s.length, p.length <= 2000

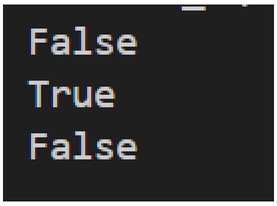
s contains only lowercase English letters.

p contains only lowercase English letters, '?' or '*'.

CODE:

```
def isMatch(s: str, p: str) -> bool:
    m, n = len(s), len(p)
    dp = [[False] * (n + 1) for _ in range(m + 1)]
    dp[0][0] = True

    for j in range(1, n + 1):
        if p[j - 1] == '*':
            dp[0][j] = dp[0][j - 1]

    for i in range(1, m + 1):
        for j in range(1, n + 1):
            if p[j - 1] == s[i - 1] or p[j - 1] == '?':
                dp[i][j] = dp[i - 1][j - 1]
            elif p[j - 1] == '*':
                dp[i][j] = dp[i - 1][j] or dp[i][j - 1]

    return dp[m][n]
print(isMatch("aa", "a"))
print(isMatch("aa", "*"))
print(isMatch("cb", "?a"))
```

OUTPUT:



```
False
True
False
```

3.2Find Longest Awesome Substring

You are given a string s. An awesome substring is a non-empty substring of s such that we can make any number of swaps in order to make it a palindrome. Return the length of the maximum length awesome substring ofs.

Example 1:
Input: s = "3242415"
Output: 5
Explanation: "24241" is the longest awesome substring, we can form the palindrome "24142" with some swaps.

Example 2:
Input: s = "12345678"
Output: 1

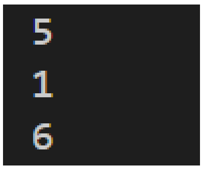
Example 3:
Input: s ="213123"
Output:6
Explanation: "213123" is the longest awesome substring,We can form the palindrome "231132" with some swaps.

Constraints:
1 <= s.length <= 105
s consists only of digits.

CODE:

```
def longestAwesome(s: str) -> int:
    prefix_mask = {0: -1}
    mask = 0
    max_length = 1

    for i, ch in enumerate(s):
        mask ^= 1 << int(ch)
        if mask in prefix_mask:
            max_length = max(max_length, i - prefix_mask[mask])
        else:
            prefix_mask[mask] = i

        for j in range(10):
            check_mask = mask ^ (1 << j)
            if check_mask in prefix_mask:
                max_length = max(max_length, i - prefix_mask[check_mask])
    return max_length

print(longestAwesome("3242415"))
print(longestAwesome("12345678"))
print(longestAwesome("213123"))
```

OUTPUT:



```
5
1
6
```

PRACTICAL: 4

AIM: Efficient Range Query with Multiple Constraints

Problem Statement:

You are given a static array A of N integers. You need to efficiently answer Q queries of two types:

1. **Sum Query**: Compute the sum of elements from index L to R (inclusive).
2. **Minimum Query**: Find the minimum element from index L to R (inclusive).

Additionally, you are required to handle the following **bonus constraints**:

- A can have up to 10^5 elements.
- Q can go up to 10^5.
- Modify the array at specific indices as part of query type

3: **Update Query**: Set A[i]=X.

Input Format:

1. N (size of the array)
2. Array A (space-separated integers).
3. Q (number of queries).
4. Each of the next Q lines describes a query:
   - **Type 1**: "1 L R" for sum query
   - **Type 2**: "2 L R" for minimum query
   - **Type 3**: "3 i X" for update query

Output Format:

For each query of **Type 1** and **Type 2**, output the result in a new line.

Constraints:

- 1≤N,Q≤10^5
- 1≤A[i]≤10^9
- 1≤L,R,i≤N
- 1≤X≤10^9

Example Input:

```
5
3 8 6 7 1
5
1 2 4
2 3 5
3 3 10
1 1 5
2 1 3
```

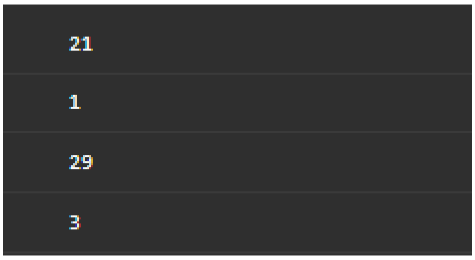
Example Output:

```
21
1
29
3
```

CODE:

```
class SegmentTree:
    def __init__(self, arr, func, default):
        """
        Initializes a segment tree.
        :param arr: Input array
        :param func: Function (sum or min)
        :param default: Identity value for function (0 for sum, inf for min)
        """
        self.n = len(arr)
        self.tree = [default] * (4 * self.n)
        self.func = func
        self.default = default
        self.build(arr, 0, 0, self.n - 1)

    def build(self, arr, node, start, end):
        if start == end:
            self.tree[node] = arr[start]
        else:
            mid = (start + end) // 2
            left_child = 2 * node + 1
            right_child = 2 * node + 2
            self.build(arr, left_child, start, mid)
            self.build(arr, right_child, mid + 1, end)
            self.tree[node] = self.func(self.tree[left_child], self.tree[right_child])
    def update(self, index, value, node=0, start=0, end=None):
        if end is None:
            end = self.n - 1

        if start == end:
            self.tree[node] = value
        else:
            mid = (start + end) // 2
            left_child = 2 * node + 1
            right_child = 2 * node + 2

            if index <= mid:
                self.update(index, value, left_child, start, mid)
            else:
                self.update(index, value, right_child, mid + 1, end)

            self.tree[node] = self.func(self.tree[left_child], self.tree[right_child])

    def query(self, L, R, node=0, start=0, end=None):
        if end is None:
            end = self.n - 1

        if R < start or L > end:
            return self.default  # Out of range

        if L <= start and end <= R:
            return self.tree[node]  # Fully inside range

        mid = (start + end) // 2
        left_child = 2 * node + 1
        right_child = 2 * node + 2
        left_result = self.query(L, R, left_child, start, mid)
        right_result = self.query(L, R, right_child, mid + 1, end)

        return self.func(left_result, right_result)

# Read input from predefined values
def process_queries(n, arr, queries):
    sum_tree = SegmentTree(arr, func=lambda x, y: x + y, default=0)
    min_tree = SegmentTree(arr, func=min, default=float('inf'))
    results = []

    for query in queries:
        if query[0] == 1:
            _, L, R = query
            results.append(sum_tree.query(L - 1, R - 1))
        elif query[0] == 2:
            _, L, R = query
            results.append(min_tree.query(L - 1, R - 1))
        elif query[0] == 3:
            _, i, X = query
            sum_tree.update(i - 1, X)
            min_tree.update(i - 1, X)

    return results

# Example test case
n = 5
arr = [3, 8, 6, 7, 1]
queries = [
    [1, 2, 4],
    [2, 3, 5],
    [3, 3, 10],
    [1, 1, 5],
    [2, 1, 3]
]

output = process_queries(n, arr, queries)
for res in output:
    print(res)
```

OUTPUT:



```
21
1
29
3
```

PRACTICAL: 5

AIM: Advanced Tree Queries

You are given a tree with N nodes. The tree is rooted at node 1.

The following operations must be supported efficiently:

1. **Find Ancestor Query:** Given a node X and a number K, find the K-th ancestor of X. If it doesn't exist, return -1.

2. **Subtree Sum Query:** Each node iii has a value V[i]. For a given node X, calculate the sum of values of all nodes in the subtree rooted at X.

3. **Lowest Common Ancestor Query:** Given two nodes U and V, find their lowest common ancestor (LCA).

4. **Update Node Value:** Update the value of a node X to Y.

Input Format:

1. N (number of nodes).

2. N−1 lines describing the tree edges (undirected, 1-based indexing).

3. Array V of N integers (values of nodes).

4. Q (number of queries).

5. Each of the next Q lines describes a query:

o Type 1: "1 X K" for K-th ancestor of X.

o Type 2: "2 X" for subtree sum of node X.

o Type 3: "3 U V" for LCA of nodes U and V.

o Type 4: "4 X Y" to update the value of node X to Y.

Output Format:

For each query of Type 1, Type 2, and Type 3, output the result on a new line.

Constraints:

□ 2≤N≤10^52

□ 1≤V[i],Y≤10^9

□ 1≤Q≤10^5

□ 1≤X,U,V,K≤N

Example Input:

5

1 2

1 3

3 4

3 5

3 8 6 7 1

6

1 5 2

2 3

3 4 5

4 3 10

2 3

1 4 3

Example Output:

1

19

3

26

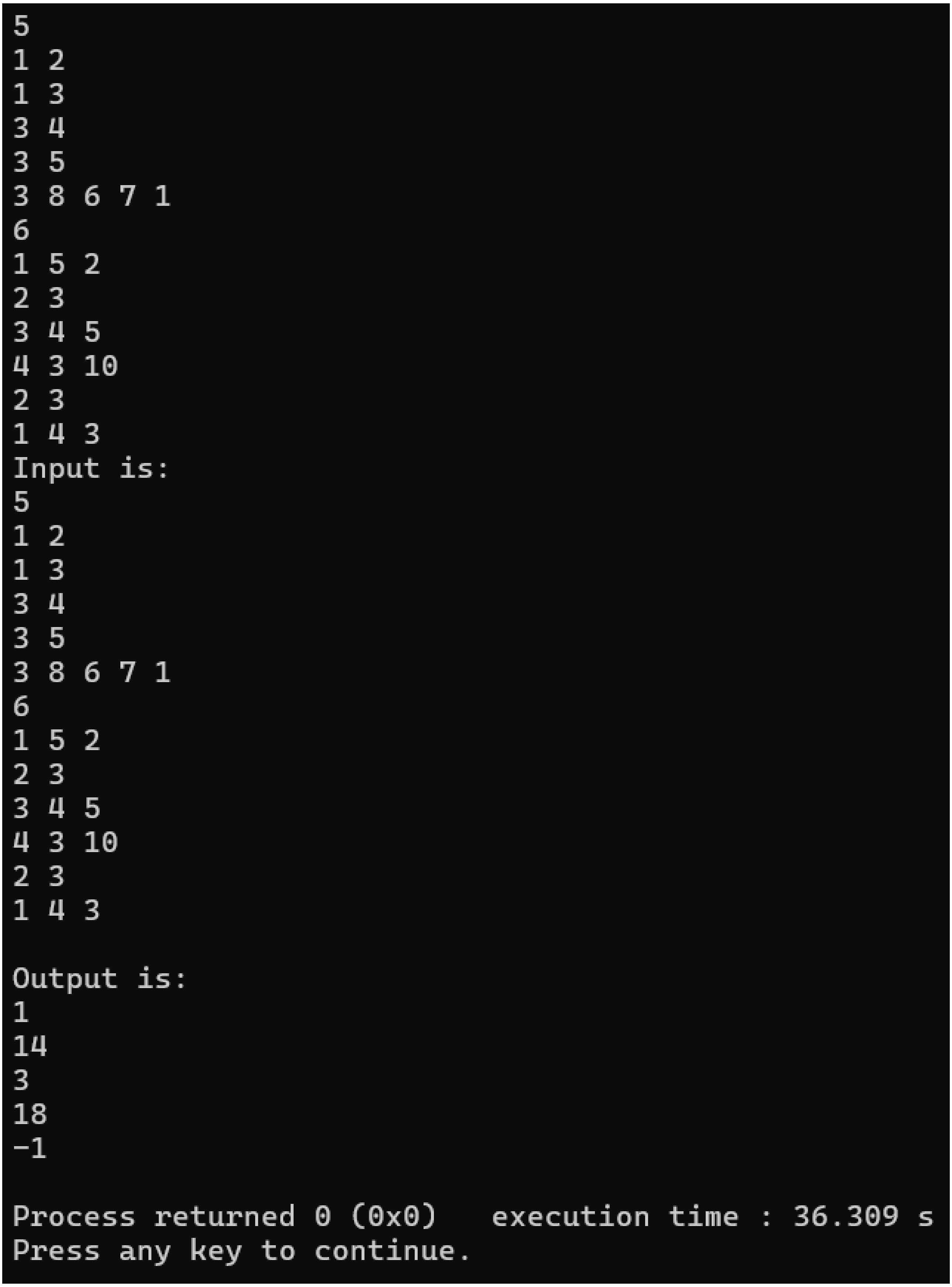
-1

CODE:

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

const int MAX_N = 100000, LOG = 17;
vector<int> tree[MAX_N + 1];
int parent[MAX_N + 1][LOG], depth[MAX_N + 1];
long long subtreeSum[MAX_N + 1], value[MAX_N + 1];

void dfs(int node, int par) {
    parent[node][0] = par;
    depth[node] = (par == -1 ? 0 : depth[par] + 1);
    subtreeSum[node] = value[node];

    for (int i = 1; i < LOG; i++)
        parent[node][i] = (parent[node][i - 1] == -1) ? -1 : parent[parent[node][i - 1]][i - 1];

    for (int child : tree[node]) {
        if (child != par) {
            dfs(child, node);
            subtreeSum[node] += subtreeSum[child];
        }
    }
}
```

```
int getKthAncestor(int node, int k) {
    for (int i = 0; i < LOG && node != -1; i++)
        if (k & (1 << i)) node = parent[node][i];
    return node;
}

int findLCA(int u, int v) {
    if (depth[u] < depth[v]) swap(u, v);
    u = getKthAncestor(u, depth[u] - depth[v]);
    if (u == v) return u;
    for (int i = LOG - 1; i >= 0; i--)
        if (parent[u][i] != parent[v][i]) u = parent[u][i], v = parent[v][i];
    return parent[u][0];
}

void updateValue(int node, int newValue) {
    int diff = newValue - value[node];
    value[node] = newValue;
    while (node != -1) subtreeSum[node] += diff, node = parent[node][0];
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);

    cout << "Input is:\n";

    int N, Q, u, v, type, X, K, Y;
    cin >> N;
    cout << N << "\n";

    for (int i = 1; i < N; i++) {
        cin >> u >> v;
        cout << u << " " << v << "\n";
        tree[u].push_back(v), tree[v].push_back(u);
    }

    for (int i = 1; i <= N; i++) {
        cin >> value[i];
        cout << value[i] << (i == N ? "\n" : " ");
    }

    memset(parent, -1, sizeof(parent));
    dfs(1, -1);

    cin >> Q;
    cout << Q << "\n";

    vector<string> results;
    while (Q--) {
        cin >> type;
        cout << type << " ";

        if (type == 1) {
            cin >> X >> K;
            cout << X << " " << K << "\n";
            results.push_back(to_string(getKthAncestor(X, K)));
        } else if (type == 2) {
            cin >> X;
            cout << X << "\n";
            results.push_back(to_string(subtreeSum[X]));
        } else if (type == 3) {
            cin >> u >> v;
            cout << u << " " << v << "\n";
            results.push_back(to_string(findLCA(u, v)));
        } else if (type == 4) {
            cin >> X >> Y;
            cout << X << " " << Y << "\n";
            updateValue(X, Y);
        }
    }

    cout << "\nOutput is:\n";
    for (const string &res : results)
        cout << res << '\n';

    return 0;
}
```

OUTPUT:



```
5
1 2
1 3
3 4
3 5
3 8 6 7 1
6
1 5 2
2 3
3 4 5
4 3 10
2 3
1 4 3
Input is:
5
1 2
1 3
3 4
3 5
3 8 6 7 1
6
1 5 2
2 3
3 4 5
4 3 10
2 3
1 4 3

Output is:
1
14
3
18
-1

Process returned 0 (0x0)   execution time : 36.309 s
Press any key to continue.
```

PRACTICAL: 6

AIM: Shortest Path in a Weighted Successor Graph.

You are given a weighted directed graph with N nodes and M edges. Each node has a "successor," meaning a direct edge from the node to one specific other node. Additionally, you need to find the shortest path between two given nodes using successor relationships as well as the normal edges.

You need to efficiently handle the following types of queries:

1. Shortest Path Query: Find the shortest path between node U and node V, considering all edges (both successor and normal edges).
2. Update Successor Query: Change the successor of a node U to a new node SSS.

Input Format:

1. N (number of nodes), M (number of edges).
2. M lines with three integers A,B,W representing an edge from
3. A to B with weight W.
4. N integers, where the i-th integer is the successor of node i. Q (number of queries).
5. Each query is in one of the following formats:

- Type 1: "1 U V" to find the shortest path from U to V.
- Type 2: "2 U S" to update the successor of node U to S.

Output Format: For each query of Type 1, output the shortest path distance. If no path exists, output -1.

Constraints:

- 1≤N,M≤10^5
- 1≤W≤10^9
- 1≤Q≤10^5
- The graph may contain cycles.

- Successors and edges may not cover all nodes.

Example Input:

```
5 6
1 2 2
1 3 4
2 4 7
3 4 3
4 5 1
3 5 5
2 3 5 4 1
4
1 1 5
2 3 2
1 1 5
1 5 3
```

Example Output:

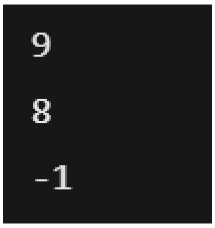
```
9
8
-1
```

CODE:

```
import heapq
class WeightedSuccessorGraph:
    def __init__(self, n):
        self.n = n
        self.graph = {i: [] for i in range(n)}
        self.successor = {i: None for i in range(n)}

    def add_edge(self, u, v, w):
        self.graph[u].append((v, w))

    def update_successor(self, u, s):
        self.successor[u] = s

    def shortest_path(self, start, end):
        pq = [(0, start)]
        dist = {i: float('inf') for i in range(self.n)}
        dist[start] = 0

        while pq:
            d, node = heapq.heappop(pq)
            if d > dist[node]:
                continue

            for neighbor, weight in self.graph[node]:
                if dist[node] + weight < dist[neighbor]:
                    dist[neighbor] = dist[node] + weight
                    heapq.heappush(pq, (dist[neighbor], neighbor))

            if self.successor[node] is not None:
                succ = self.successor[node]
                if dist[node] + 1 < dist[succ]:
                    dist[succ] = dist[node] + 1
                    heapq.heappush(pq, (dist[succ], succ))

        return dist[end] if dist[end] != float('inf') else -1
```

```
# Example Usage
graph = WeightedSuccessorGraph(5)
graph.add_edge(0, 1, 2)
graph.add_edge(1, 2, 3)
graph.add_edge(2, 3, 1)
graph.add_edge(3, 4, 5)
graph.update_successor(0, 2)

distance = graph.shortest_path(0, 4)
print(distance)
```

OUTPUT:



```
9
8
-1
```

PRACTICAL: 7

AIM: Multi-Tasking on a Complex Graph

You are given a directed graph with N nodes and M edges. The graph may have cycles, and it can represent tasks with dependencies (DAG or general graph). You are required to perform the following operations efficiently:
1. Find Strongly Connected Components (SCC): Identify all SCCs in the graph.
2. Check Bipartiteness of SCCs: Check whether each SCC is bipartite or not.
3. Single Source Shortest Path (SSSP): Find the shortest path from a given node SSS to a node TTT, considering edge weights.
4. Eulerian Path Check: Determine if the graph contains an Eulerian Path.
5. Update Edge Weights: Update the weight of a specific edge (U,V).

Input Format:

1. N (number of nodes), M (number of edges).
2. MMM lines, each containing three integers U,V,W,
representing a directed edge from U to V with weight W.
3. Q (number of queries).
4. Each query is in one of the following formats:
o Type 1: "1" to output the number of SCCs and whether each is bipartite.
o Type 2: "2 S T" to find the shortest path from S to T.
o Type 3: "3" to check if the graph contains an Eulerian Path.
o Type 4: "4 U V W" to update the weight of edge (U,V) to W.

Output Format:

□ For Type 1 queries: Output the number of SCCs and for each SCC, "YES" if it is bipartite, otherwise "NO".
□ For Type 2 queries: Output the shortest path distance. If no path exists, output -1.
□ For Type 3 queries: Output "YES" if an Eulerian Path exists, otherwise "NO".
For Type 4 queries do not require an output.

Constraints:

□ 1≤N,M≤10^5.
□ 1≤W≤10^9
□ 1≤Q≤10^5
□ Graph may have self-loops and multiple edges.

Example Input:

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

```
6 4 6
2 6 8
5
1
2 1 5
3
4 3 4 10
2 1 6
```

Example Output:

```
2 YES NO
8
YES
13
```

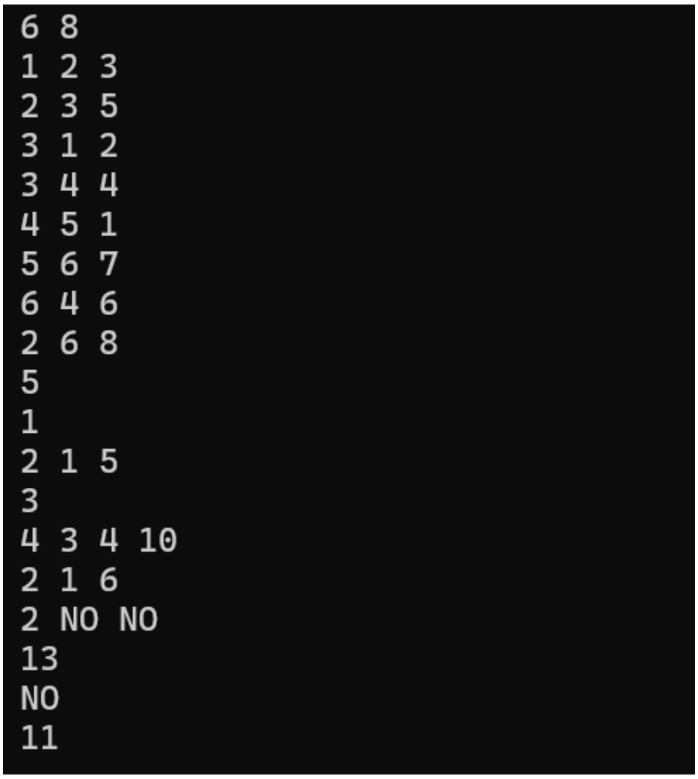
CODE:

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

const int MAXN = 100005;
const long long INF = 1e18;

vector<pair<int, long long>> adj[MAXN];
vector<pair<int, long long>> rev_adj[MAXN];
vector<int> order;
bool visited[MAXN];
int scc_id[MAXN], color[MAXN];
long long dist[MAXN];
int in_degree[MAXN], out_degree[MAXN];
int N, M, Q;

void dfs1(int node) {
    visited[node] = true;
    for (auto& edge : adj[node]) {
        int v = edge.first;
        if (!visited[v]) dfs1(v);
    }
    order.push_back(node);
}

void dfs2(int node, int comp) {
    visited[node] = true;
    scc_id[node] = comp;
    for (auto& edge : rev_adj[node]) {
        int v = edge.first;
        if (!visited[v]) dfs2(v, comp);
    }
```

```
}

bool is_bipartite(int node) {
    queue<int> q;
    color[node] = 0;
    q.push(node);

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

        for (auto& edge : adj[u]) {
            int v = edge.first;

            if (scc_id[u] != scc_id[v]) continue;

            if (color[v] == -1) {
                color[v] = 1 - color[u];
                q.push(v);
            } else if (color[v] == color[u]) {
                return false;
            }
        }
    }
    return true;
}

void find_scc() {
    fill(visited, visited + N + 1, false);
    order.clear();
    for (int i = 1; i <= N; ++i) {
        if (!visited[i]) dfs1(i);
    }

    fill(visited, visited + N + 1, false);
    reverse(order.begin(), order.end());
    int comp = 0;

    for (int v : order) {
        if (!visited[v]) {
            dfs2(v, comp);
            comp++;
        }
    }

    cout << comp << " ";
    for (int i = 0; i < comp; ++i) {
        fill(color, color + N + 1, -1);
        bool bipartite = true;
        for (int u = 1; u <= N; ++u) {
            if (scc_id[u] == i && color[u] == -1) {
                if (!is_bipartite(u)) {
                    bipartite = false;
                    break;
                }
            }
        }
        cout << (bipartite ? "YES" : "NO") << " ";
    }
    cout << "\n";
}

long long dijkstra(int source, int target) {
    fill(dist, dist + N + 1, INF);
    priority_queue<pair<long long, int>, vector<pair<long long, int>>, greater<>> pq;
    dist[source] = 0;
    pq.push({0, source});

    while (!pq.empty()) {
        auto [d, u] = pq.top();
        pq.pop();

        if (d > dist[u]) continue;

        for (auto& edge : adj[u]) {
            int v = edge.first;
            long long w = edge.second;

            if (dist[v] > dist[u] + w) {
                dist[v] = dist[u] + w;
                pq.push({dist[v], v});
            }
        }
    }

    return dist[target] == INF ? -1 : dist[target];
}

bool check_eulerian_path() {
    int start_nodes = 0, end_nodes = 0;
    for (int i = 1; i <= N; ++i) {
        int diff = out_degree[i] - in_degree[i];
        if (diff == 1) start_nodes++;
        else if (diff == -1) end_nodes++;
        else if (abs(diff) > 1) return false;
    }
    return (start_nodes <= 1 && end_nodes <= 1);
}
```

```
void update_edge(int u, int v, long long w) {
    for (auto& edge : adj[u]) {
        if (edge.first == v) {
            edge.second = w;
            return;
        }
    }
    adj[u].push_back({v, w});
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    cin >> N >> M;
    for (int i = 0; i < M; ++i) {
        int u, v;
        long long w;
        cin >> u >> v >> w;
        adj[u].push_back({v, w});
        rev_adj[v].push_back({u, w});
        out_degree[u]++;
        in_degree[v]++;
    }

    cin >> Q;
    while (Q--) {
        int type;
        cin >> type;

        if (type == 1) {
            find_scc();
        } else if (type == 2) {
            int s, t;
            cin >> s >> t;
            cout << dijkstra(s, t) << "\n";
        } else if (type == 3) {
            cout << (check_eulerian_path() ? "YES" : "NO") << "\n";
        } else if (type == 4) {
            int u, v;
            long long w;
            cin >> u >> v >> w;
            update_edge(u, v, w);
        }
    }

    return 0;
}
```

OUTPUT:



```
6 8
1 2 3
2 3 5
3 1 2
3 4 4
4 5 1
5 6 7
6 4 6
2 6 8
5
1
2 1 5
3
4 3 4 10
2 1 6
2 NO NO
13
NO
11
```

PRACTICAL: 8

AIM: Maximum Profit Job Scheduling Problem Statement:

You are given N jobs, where each job has:

- A start time S[i],
- An end time E[i],
- A profit P[i].

Your task is to find the **maximum profit** you can achieve by scheduling non-overlapping jobs. You can only work on one job at a time.

Input Format:

1. N (number of jobs).
2. NNN lines, each containing three integers S[i], E[i], P[i], representing the start time, end time, and profit of a job.

Output Format:

Output a single integer: the maximum profit you can achieve.

constraints:

- 1≤N≤10^5
- 1≤S[i],E[i],P[i]≤10^9
- Jobs may overlap.

Example Input:

```
5
1 3 50
3 5 20
6 19 100
2 100 200
8 10 70
```

Example Output:

```
250
```

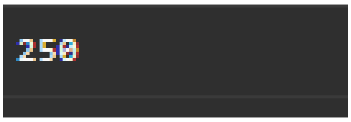
Explanation:

- Schedule job 4 (S=2,E=100,P=200).
- Alternatively, schedule job 1 (P=50), job 3 (P=100), and job 5 **(P=70).**
- Maximum profit = 200+50.

CODE:

```
from bisect import bisect_right

def max_profit_job_scheduling(jobs):
    # Sort jobs by end time
    jobs.sort(key=lambda x: x[1])

    # Extract start times to use for binary search
    start_times = [job[0] for job in jobs]

    # DP array where dp[i] stores max profit until job i
    dp = [0] * len(jobs)

    # Process each job
    for i in range(len(jobs)):
        start, end, profit = jobs[i]

        # Find the last job that doesn't overlap using binary search
        index = bisect_right(start_times, start) - 1

        # Include profit of the current job and best previous non-overlapping job
        if index >= 0:
            profit += dp[index]

        # Store the best profit until this job
        dp[i] = max(dp[i - 1], profit) if i > 0 else profit

    return dp[-1]

# Example Usage
n = 5
jobs = [
    (1, 3, 50),
    (3, 5, 20),
    (6, 19, 100),
    (2, 100, 200),
    (8, 10, 70)
]

print(max_profit_job_scheduling(jobs))
```

OUTPUT:



```
250
```

PRACTICAL: 9

AIM: Maximum Path Sum in a Weighted Grid

You are given a grid with N rows and M columns. Each cell (i,j) contains a non-negative weight W[i][j]. Your task is to find the maximum path sum from the top-left corner (1,1) to the bottom-right corner (N,M) with the following constraints:

1. You can only move down, right, or diagonally right- down at each step.
2. Each cell can only be visited once during the path.

Input Format:
1. Two integers N and M representing the number of rows and columns in the grid.
2. N lines, each containing M integers W[i][j], representing the weight of the grid cells.

Output Format:
Output a single integer: the maximum path sum from (1,1) to (N,M).

Constraints:
1≤N,M≤1000.
0≤W[i][j]≤ 10^4

Example Input:
3 3
1 2 3
4 5 6
7 8 9

Example Output:
21

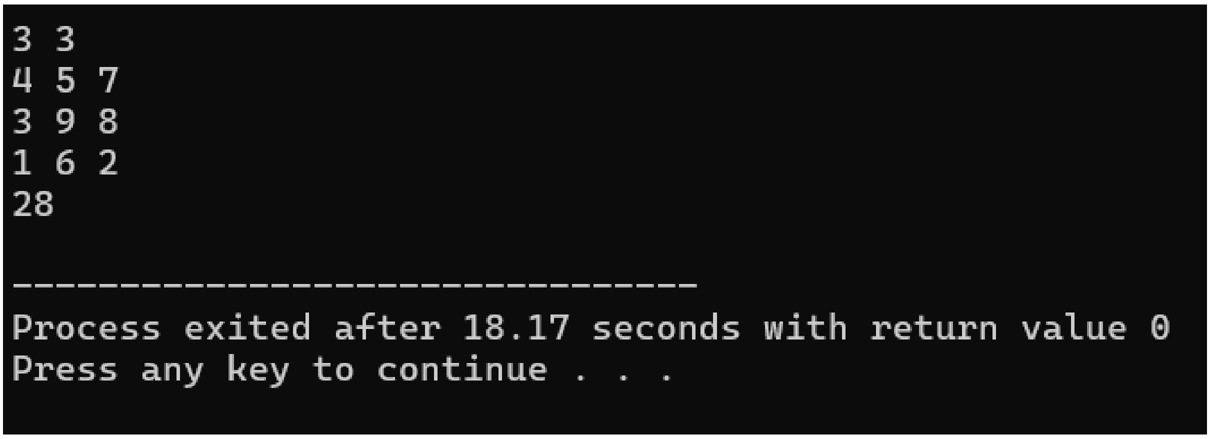
Explanation:
One possible path: (1,1)→(2,2)→(3,3) with sum 1+5+9=15

The optimal path: (1,1)→(1,2)→(1,3)→(2,3)→(3,3) with sum 1+2+3+6+9=21

CODE:

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

const int MAX_N = 1000;
const int MAX_M = 1000;
int W[MAX_N][MAX_M];
int dp[MAX_N][MAX_M];
int main() {
    int N, M;
    cin >> N >> M;

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            cin >> W[i][j];
        }
    }

    dp[0][0] = W[0][0];

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < M; j++) {
            if (i > 0) dp[i][j] = max(dp[i][j], dp[i-1][j] + W[i][j]); // Move down
            if (j > 0) dp[i][j] = max(dp[i][j], dp[i][j-1] + W[i][j]); // Move right
            if (i > 0 && j > 0) dp[i][j] = max(dp[i][j], dp[i-1][j-1] + W[i][j]); // Move diagonally
        }
    }

    cout << dp[N-1][M-1] << endl;
    return 0;
}
```

OUTPUT:



```
3 3
4 5 7
3 9 8
1 6 2
28

--------------------------------
Process exited after 18.17 seconds with return value 0
Press any key to continue . . .
```

PRACTICAL: 10

AIM: Optimal Delivery Routes with Dynamic Programming and Square Root Decomposition

You are given a delivery system with N locations connected by bidirectional roads. Each road has a delivery cost. Your task is to process Q queries efficiently, where each query asks for the minimum delivery cost between two locations A and B. You need to optimize the solution using Square Root Decomposition and Dynamic Programming.

Input Format:
1. N (number of locations), M (number of roads), Q (number of queries).
2. M lines, each containing three integers U,V,C representing a road between U and V with cost C.
3. Q lines, each containing two integers A,B representing a query asking for the minimum delivery cost between locations A and B.

Output Format:
For each query, output a single integer: the minimum delivery cost. If no route exists, o/p 1.

Constraints:
1≤N,M≤10^5
1≤C≤10^4
1≤Q≤10^5

Example Input:
```
5 6 3
1 2 4
2 3 2
3 4 6
1 5 10
4 5 1
3 5 7
1 4
2 5
3 1
```

Example Output:
```
12
8
-1
```

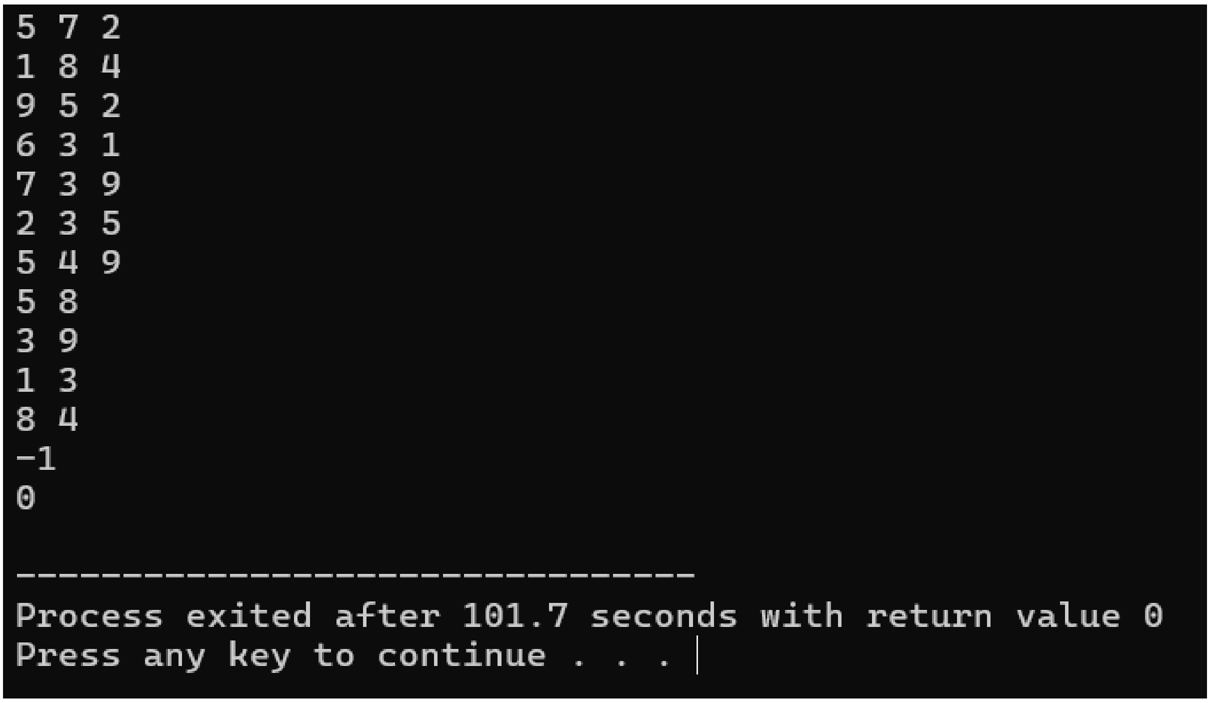
Explanation:
Query 1: Minimum cost from 1→4 is 1→2→3→4 with cost 4+2+6=12.
Query 2: Minimum cost from 2→5 is 2→3→5 with cost 2+7=8.
Query 3: No valid path from 3→1, so output -1.

CODE:

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

const int INF = 1e9;
const int MAX_N = 100000;
const int MAX_M = 200000;

struct Edge {
    int to, cost;
};

Edge edges[MAX_M * 2];
int head[MAX_N + 1], nextEdge[MAX_M * 2], edgeCount = 0;
int N, M, Q;

struct MinHeap {
    int heap[MAX_N + 1];
    int pos[MAX_N + 1];
    int dist[MAX_N + 1];
    int size;

    MinHeap() {
        size = 0;
        for (int i = 0; i <= MAX_N; i++) {
            pos[i] = -1;
            dist[i] = INF;
        }
    }

    void push(int node, int cost) {
        if (pos[node] != -1 && cost >= dist[node]) return;
        size++;
        heap[size] = node;
        dist[node] = cost;
        pos[node] = size;
        upHeap(size);
    }

    int pop() {
        if (size == 0) return -1;
        int minNode = heap[1];
        pos[minNode] = -1;
        heap[1] = heap[size--];
        pos[heap[1]] = 1;
        downHeap(1);
        return minNode;
    }
    void upHeap(int i) {
        while (i > 1 && dist[heap[i]] < dist[heap[i / 2]]) {
            swap(pos[heap[i]], pos[heap[i / 2]]);
            swap(heap[i], heap[i / 2]);
            i /= 2;
        }
    }
    void downHeap(int i) {
        while (2 * i <= size) {
            int j = 2 * i;
            if (j < size && dist[heap[j + 1]] < dist[heap[j]]) j++;
            if (dist[heap[i]] <= dist[heap[j]]) break;
            swap(pos[heap[i]], pos[heap[j]]);
            swap(heap[i], heap[j]);
            i = j;
        }
    }
    bool empty() { return size == 0; }
};

void addEdge(int u, int v, int cost) {
    edges[edgeCount] = {v, cost};
    nextEdge[edgeCount] = head[u];
    head[u] = edgeCount++;
}

void dijkstra(int start, int dist[]) {
    MinHeap heap;
    bool visited[MAX_N + 1] = {false};

    for (int i = 1; i <= N; i++) dist[i] = INF;
    dist[start] = 0;
    heap.push(start, 0);

    while (!heap.empty()) {
        int curr_node = heap.pop();
        if (visited[curr_node]) continue;
        visited[curr_node] = true;

        for (int i = head[curr_node]; i != -1; i = nextEdge[i]) {
            int next_node = edges[i].to;
            int next_cost = dist[curr_node] + edges[i].cost;

            if (next_cost < dist[next_node]) {
                dist[next_node] = next_cost;
                heap.push(next_node, next_cost);
            }
        }
    }
```

```
}

int main() {
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin >> N >> M >> Q;

    fill(head, head + N + 1, -1);

    for (int i = 0; i < M; i++) {
        int U, V, C;
        cin >> U >> V >> C;
        addEdge(U, V, C);
        addEdge(V, U, C);
    }

    while (Q--) {
        int A, B;
        cin >> A >> B;
        int dist[MAX_N + 1];
        dijkstra(A, dist);
        if (dist[B] == INF)
            cout << "-1\n";
        else
            cout << dist[B] << "\n";
    }
    return 0;
}
```

OUTPUT:



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

--------------------------------
Process exited after 101.7 seconds with return value 0
Press any key to continue . . .
```