**Aim:** C++ ESE Practical Exam Submission

**IDE:** Visual Studio Code

**Programming Language:** C++

**Question-1(A)**

**Code :-**

#include <bits/stdc++.h>

using namespace std;

const int PRIME = 101;

const int MOD = 1e9 + 7;

long long calculateHash(const string &*str*, int *len*)

{

    long long hash = 0;

    long long power = 1; *// PRIME^0*

    for (int i = 0; i < *len*; ++i)

    {

        hash = (hash + *str*[i] \* power) % MOD;

        power = (power \* PRIME) % MOD;

    }

    return hash;

}

long long recalculateHash(const string &*text*, int *oldIndex*, int *newIndex*, long long *oldHash*, int *patternLen*, long long *primePower*)

{

*oldHash* = (*oldHash* - *text*[*oldIndex*] + MOD) % MOD;

*oldHash* = (*oldHash* / PRIME) % MOD;

*oldHash* = (*oldHash* + *text*[*newIndex*] \* *primePower*) % MOD;

    return *oldHash*;

}

vector<int> Rabin\_Karp(const string &*text*, const string &*pattern*)

{

    vector<int> result;

    int textLen = *text*.size();

    int patternLen = *pattern*.size();

    if (patternLen > textLen)

    {

        return result;

    }

    long long patternHash = calculateHash(*pattern*, patternLen);

    long long textHash = calculateHash(*text*, patternLen);

    long long primePower = 1;

    for (int i = 1; i < patternLen; ++i)

    {

        primePower = (primePower \* PRIME) % MOD;

    }

    for (int i = 0; i <= textLen - patternLen; ++i)

    {

        if (textHash == patternHash && *text*.substr(i, patternLen) == *pattern*)

        {

            result.push\_back(i);

        }

        if (i < textLen - patternLen)

        {

            textHash = recalculateHash(*text*, i, i + patternLen, textHash, patternLen, primePower);

        }

    }

    return result;

}

int main()

{

    string Reference\_string = "ATGC";

    string Input\_String = "";

    string pattern = "AGC";

    srand(time(0));

    for (int i = 0; i < 25; i++)

    {

        Input\_String += Reference\_string[rand() % 4];

    }

    cout << "Random String: " << Input\_String << endl;

    cout << "Pattern: " << pattern << endl;

    vector<int> occurrences = Rabin\_Karp(Input\_String, pattern);

    if (occurrences.size() == 0)

    {

        cout << "No occurrences found for the pattern." << endl;

    }

    else

    {

        cout << "Number of occurrences: " << occurrences.size() << endl;

        cout << "Occurrences Founded At Indices :- ";

        for (int ocr : occurrences)

        {

            cout << ocr << " ";

        }

        cout << endl;

    }

    return 0;

}

### Output :-

### 

**Question – 1(B)**

**Code :-**

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

### using namespace std;

### vector<string> Split\_String(string &*submitted\_doc*)

### {

### vector<string> input\_string;

### string s = "";

### for (char c : *submitted\_doc*)

### {

### if (c == ',' || c == '.')

### {

### continue;

### }

### if (c == ' ')

### {

### input\_string.push\_back(s);

### s = "";

### }

### else

### {

### s = s + c;

### }

### }

### return input\_string;

### }

### int main()

### {

### string submitted\_doc = "In this Paper, We propose a new algorithm for sorting.";

### string reference\_doc = "This paper introduces a new sorting algorithm";

### vector<string> input\_string = Split\_String(submitted\_doc);

### vector<string> reference\_string = Split\_String(reference\_doc);

### int n = input\_string.size();

### int m = reference\_string.size();

### int lcs[n + 1][m + 1];

### for (int i = 0; i <= n; i++)

### {

### for (int j = 0; j <= m; j++)

### {

### if (i == 0 || j == 0)

### lcs[i][j] = 0;

### else if (input\_string[i - 1] == reference\_string[j - 1])

### lcs[i][j] = lcs[i - 1][j - 1] + 1;

### else

### lcs[i][j] = max(lcs[i - 1][j], lcs[i][j - 1]);

### }

### }

### bool found = lcs[n + 1][m + 1] >= 4;

### cout << "Found: " << (found ? "Yes" : "No") << endl;

### return 0;

### }

### Output :-

### 

**Question – 2(A)**

**Code :-**

#include <bits/stdc++.h>

using namespace std;

int main()

{

    int denomination;

    cout << "Enter the Number of Denominations :- ";

    cin >> denomination;

    vector<pair<int, int>> storage;

    for (int i = 0; i < denomination; i++)

    {

        cout << "Enter The Details for (" << i + 1 << ") Coins :-" << endl;

        int val = 0;

        int quantity = 0;

        cout << "Enter the Value of Coin :- ";

        cin >> val;

        cout << "Enter the Quantity of Coins :- ";

        cin >> quantity;

        storage.push\_back({val, quantity});

    }

    sort(storage.begin(), storage.end(), [](const pair<int, int> &*a*, const pair<int, int> &*b*)

         { return *a*.first > *b*.first; });

    int amount;

    cout << "Enter the Amount :- ";

    cin >> amount;

    vector<int> dp(amount + 1, INT\_MAX);

    dp[0] = 0;

    for (int i = 0; i < denomination; i++)

    {

        for (int j = storage[i].first; j <= amount; j++)

        {

            if (dp[j - storage[i].first] != INT\_MAX && dp[j - storage[i].first] + 1 < dp[j])

            {

                dp[j] = dp[j - storage[i].first] + 1;

            }

        }

    }

    if (dp[amount] == INT\_MAX)

    {

        cout << "Minimum Number of Coins Required is Not Possible" << endl;

    }

    else

    {

        cout << "Minimum Number of Coins Required is " << dp[amount] << endl;

    }

    return 0;

}

### Output :-

**Question – 2(B)**

**Code :-**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int maxSegments(int *n*, int *x*, int *y*, int *z*)

{

    vector<int> dp(*n* + 1, -1);

    dp[0] = 0;

    for (int i = 1; i <= *n*; ++i)

    {

        if (i >= *x* && dp[i - *x*] != -1)

        {

            dp[i] = max(dp[i], dp[i - *x*] + 1);

        }

        if (i >= *y* && dp[i - *y*] != -1)

        {

            dp[i] = max(dp[i], dp[i - *y*] + 1);

        }

        if (i >= *z* && dp[i - *z*] != -1)

        {

            dp[i] = max(dp[i], dp[i - *z*] + 1);

        }

    }

    return dp[*n*] != -1 ? dp[*n*] : 0;

}

int main()

{

    int n, x, y, z;

    cout << "Enter the length of the line segment (n): ";

    cin >> n;

    cout << "Enter the first cut length (x): ";

    cin >> x;

    cout << "Enter the second cut length (y): ";

    cin >> y;

    cout << "Enter the third cut length (z): ";

    cin >> z;

    int result = maxSegments(n, x, y, z);

    cout << "The maximum number of segments that can be cut from length " << n

         << " using lengths " << x << ", " << y << ", and " << z << " is: " << result << endl;

    return 0;

}

### Output :-

**Question – 3(A)**

**Code :-**

#include <bits/stdc++.h>

using namespace std;

void findCriticalConnections(vector<vector<int>> &*graph*, int *node*)

{

    int min\_degree = INT\_MAX;

    int src = -1;

    int dest = -1;

    vector<pair<int, int>> criticalConnections;

    for (int i = 0; i < *node*; i++)

    {

        int curr\_degree = 0;

        for (int j = 0; j < *node*; j++)

        {

            if (*graph*[i][j] != 9999 && *graph*[i][j] != 0)

                curr\_degree++;

        }

        if (curr\_degree == 1)

        {

            for (int j = 0; j < *node*; j++)

            {

                if (*graph*[i][j] != 9999 && *graph*[i][j] != 0)

                {

                    src = i;

                    dest = j;

                    criticalConnections.push\_back({src, dest});

                }

            }

        }

    }

    for (auto connection : criticalConnections)

    {

        cout << "The Critical Connection is from " << connection.first << " to " << connection.second << endl;

    }

}

int main()

{

    int node;

    cout << "Enter The Number of Nodes :- ";

    cin >> node;

    vector<vector<int>> graph(node, vector<int>(node, 9999));

    cout << "Enter The Adjacency Matrix :- " << endl;

    for (int i = 0; i < node; i++)

    {

        for (int j = 0; j < node; j++)

        {

            cout << "Enter The Edge from " << i << " to " << j << " (9999 for no edge) :- ";

            cin >> graph[i][j];

        }

    }

    findCriticalConnections(graph, node);

    return 0;

}

### Output :-

**Question – 3(B)**

**Code :-**

#include <bits/stdc++.h>

using namespace std;

#define V 5

#define INF 99999

void printSolution(int *dist*[][V])

{

    cout << "The following matrix shows the shortest "

            "distances"

            " between every pair of vertices \n";

    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 *dist*[][V])

{

    int i, j, k;

    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];

            }

        }

    }

    printSolution(*dist*);

}

int main()

{

    int graph[V][V] = {{0, 3, INF, 7, 8},

                       {INF, 0, 1, 4, INF},

                       {INF, INF, 0, INF, INF},

                       {INF, INF, 2, 0, INF},

                       {INF, INF, INF, 3, 0}};

    floydWarshall(graph);

    return 0;

}

### Output :-

### 

**Question - 4**

**Code :-**

#include <bits/stdc++.h>

using namespace std;

void Prime\_Factorization(long long *n*)

{

    for (long long i = 2; i \* i <= *n*; i++)

    {

        while (*n* % i == 0)

        {

            cout << i << " X ";

*n* /= i;

        }

    }

    if (*n* > 1)

    {

        cout << *n*;

    }

}

int main()

{

    long long number;

    cout << "Enter A Number :- ";

    cin >> number;

    cout << "Prime Factorization of " << number << " is : ";

    Prime\_Factorization(number);

    return 0;

}

### Output :-

### 