Name - Abhineet Kumar Mishra
Registered Email ID - abhineetmishra769@gmail.com
Contact Number - 7250329544

a) String compression

Implement a method to perform string compression. E.g. 'aabcccccaaa' should be a2b1c5a3. The code to implement this is given in the link - https://www.educative.io/answers/string-compression-using-run-length-encoding

Think about memory occupied and how it can be improved.

Bonus 1:

The answer should be taken into second compressor and compress further.

E.g. a2b2c1a3c3 should become ab2c1ac3

Bonus 2: decompress2

ab2c1ac3 should return aabbcaaaccc.

Think about how you will test this code.

Ans - The provided code implements a basic string compression using run-length encoding. However, there's a small bug in the code. The check for the end of the string (i < num-1) should come before the comparison s[i] == s[i+1]. Otherwise, it might lead to accessing an invalid memory location.

Here's the corrected code:

```
#include<iostream>

#include <string>

using namespace std;


void compress(string s)

{

   int num = s.length(); //calculating length of the string

   int i = 0;

   while (i < num) {


       // Counting the repetitions of each character
```

```
        int repetition = 1;
        while (i < num-1 && s[i] == s[i+1]) {
            repetition++;
            i++;
        }

        // Print character and its count
        cout << s[i] << repetition;
        i++;
    }
}

int main()
{
    string str = "aaaabbcddddd";
    compress(str);
    return 0;
}
```

In terms of memory improvement, the current approach is already reasonably memory-efficient as it uses only a constant amount of extra memory. However, if we want to reduce memory usage further, we can avoid storing the compressed string and directly print the compressed version. This eliminates the need for extra memory to store the compressed string.This change doesn't use extra memory for storing the compressed string and prints the compressed version directly.

To achieve the bonus requirements, we can create two additional functions: compress2 to compress the already compressed string further, and decompress2 to decompress a doubly compressed string. Additionally, I'll provide a simple test function to test these functionalities.

Here's the updated code:

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

// Function to perform the first level of compression
string compress(string s)
```

```
{
    int num = s.length(); //calculating length of the string
    int i = 0;
    string compressedString = "";

    while (i < num) {
        // Counting the repetitions of each character
        int repetition = 1;
        while (i < num-1 && s[i] == s[i+1]) {
            repetition++;
            i++;
        }

        // Append character and its count to the compressed string
        compressedString += s[i] + to_string(repetition);
        i++;
    }

    return compressedString;
}

// Function to perform the second level of compression
string compress2(string s)
{
    return compress(compress(s));
}

// Function to decompress a doubly compressed string
string decompress2(string s)
{
    string decompressedString = "";
    int i = 0;
    while (i < s.length()) {
        // Extract character
        char character = s[i];
        i++;

        // Extract repetition count
        int repetition = 0;
        while (i < s.length() && isdigit(s[i])) {
            repetition = repetition * 10 + (s[i] - '0');
            i++;
        }

        // Append repeated characters to the decompressed string
        for (int j = 0; j < repetition; j++) {
            decompressedString += character;
        }
    }

    return decompressedString;
}

// Test function
void testCompression(string input)
{
    string compressed = compress2(input);
    cout << "Original: " << input << endl;
    cout << "Compressed: " << compressed << endl;
    cout << "Decompressed: " << decompress2(compressed) << endl;
    cout << endl;
}
```

```
int main()

{

    testCompression("aabcccccaaa");

    testCompression("aaaabbcdddddd");

    // Add more test cases as needed


    return 0;

}
```

In this code, compress2 calls compress on the input string and then calls compress again on the result. The decompress2 function is designed to decompress a string that has been compressed twice. The testCompression function is used to test the compression and decompression functionality.

b) Linked List - The link shows a program to find the nth element of a linked list.
https://www.geeksforgeeks.org/nth-node-from-the-end-of-a-linked-list/

Find a way to find the kth to the last element of linked list (assume length of linked list is not known)

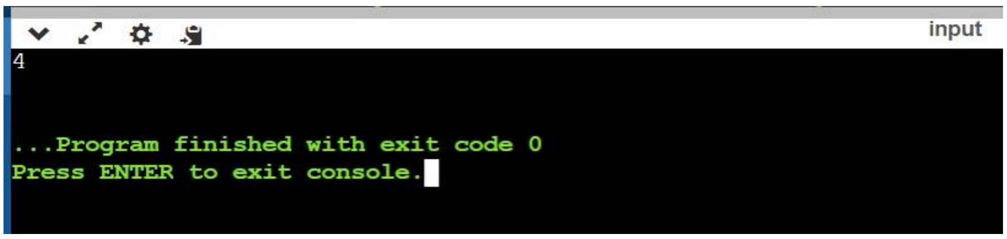
Bonus 1:

Can you minimize the number of times you run through the loop.

Ans - To find the kth to the last element of a linked list without knowing the length, we can use two pointers. Move one pointer (let's call it main_ptr) k nodes ahead, and then move both pointers simultaneously until the main_ptr reaches the end of the list. The second pointer (ref_ptr) will be at the kth to the last element.

Here's the modified code to achieve this:

```
class Node:

    def __init__(self, new_data):

        self.data = new_data

        self.next = None


class LinkedList:

    def __init__(self):

        self.head = None


    # CreateNode and make linked list

    def push(self, new_data):

        new_node = Node(new_data)

        new_node.next = self.head

        self.head = new_node


    # Function to get the kth to the last node of a linked list

    def printKthToLast(self, k):

        main_ptr = self.head

        ref_ptr = self.head
        # Move main_ptr k nodes ahead
        for i in range(k):
            if main_ptr is None:
                print(f"The linked list has fewer than {k} nodes.")
                return
            main_ptr = main_ptr.next

        # Move both pointers until main_ptr reaches the end
        while main_ptr is not None:
            main_ptr = main_ptr.next
            ref_ptr = ref_ptr.next

        # ref_ptr now points to the kth to the last node
        print(ref_ptr.data)

# Driver's Code
if __name__ == "__main__":
    llist = LinkedList()
    llist.push(20)
    llist.push(4)
    llist.push(15)
    llist.push(35)

    # Function call
    k = 2  # Example: Find the 2nd to the last element
    llist.printKthToLast(k)
```



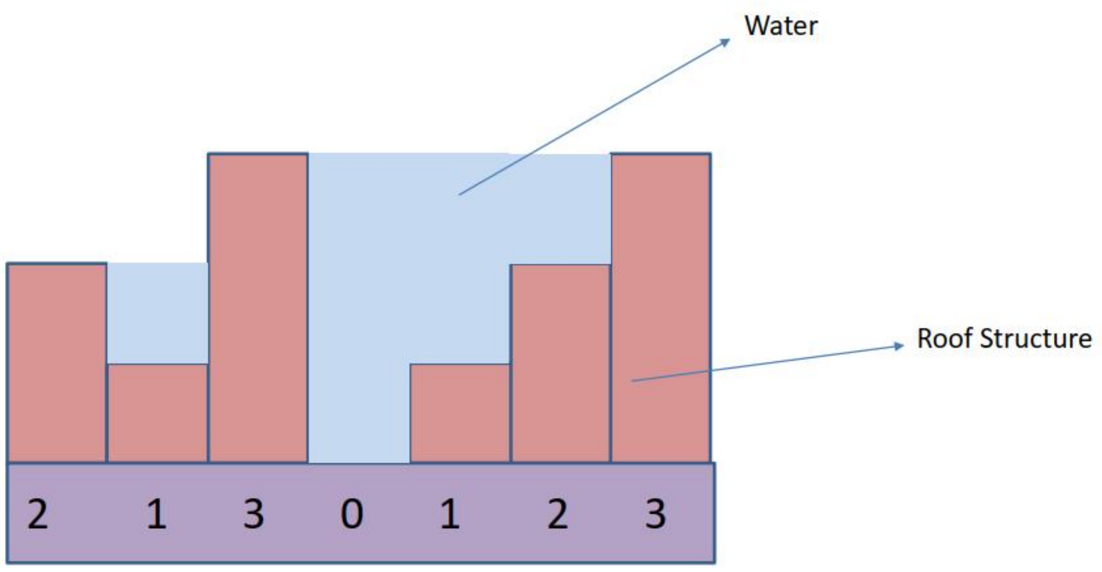
```
4


...Program finished with exit code 0
Press ENTER to exit console.
```

This modification uses two pointers (main_ptr and ref_ptr) to traverse the linked list. The main_ptr moves k nodes ahead, and then both pointers move together until main_ptr reaches the end. The ref_ptr will be at the kth to the last node, and its data is printed.

c) Given an array of integers representing the elevation of a roof structure at various positions, each position is separated by a unit length, Write a program to determine the amount of water that will be trapped on the roof after heavy rainfall

Example:
input : [2 1 3 0 1 2 3]

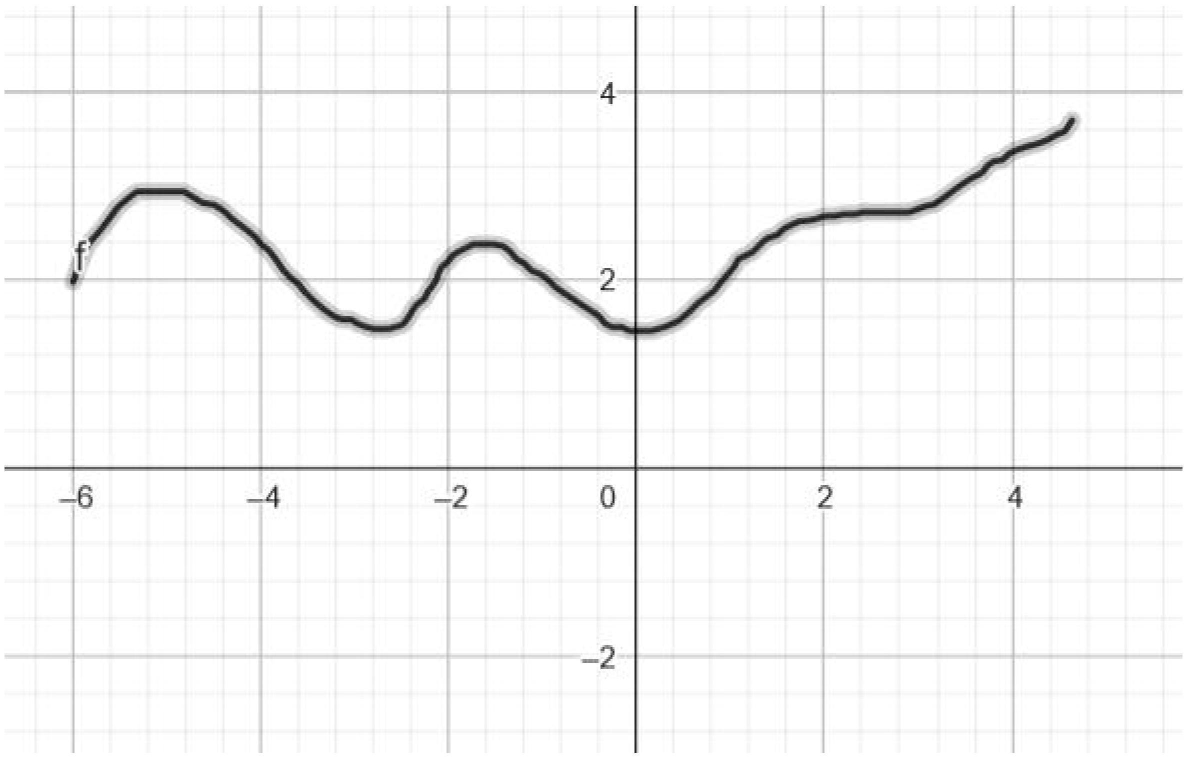
Ans : 7 units of water will be trapped



https://www.geeksforgeeks.org/trapping-rain-water/

Go through the above code for the solution.

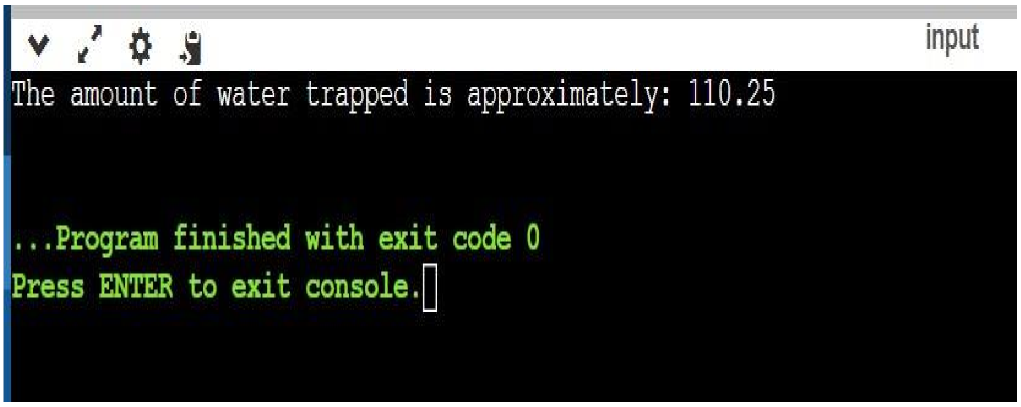
The next phase is that the values are now not discrete but analog. E.g. I give an equation of function that is bounded, can you predict how many units of water gets trapped.



Ans - from scipy import integrate

```
# Function representing the elevation
def elevation_function(x):
    # Replace this with your actual elevation function
    return 2 * x + 1

# Function to calculate trapped water using numerical integration
def calculate_trapped_water(start, end, num_points=1000):
    a, b = start[0], end[0]
    x_values = [a + (b - a) * i / (num_points - 1) for i in range(num_points)]
    y_values = [elevation_function(x) for x in x_values]

    max_elevation = max(y_values)
    trapped_water = integrate.trapz([max_elevation - y for y in y_values], x_values)

    return trapped_water
```

```
# Example usage
start_coordinate = [-6, 2]
end_coordinate = [4.5, 3.8]
result = calculate_trapped_water(start_coordinate, end_coordinate)
print(f"The amount of water trapped is approximately: {result}")
```



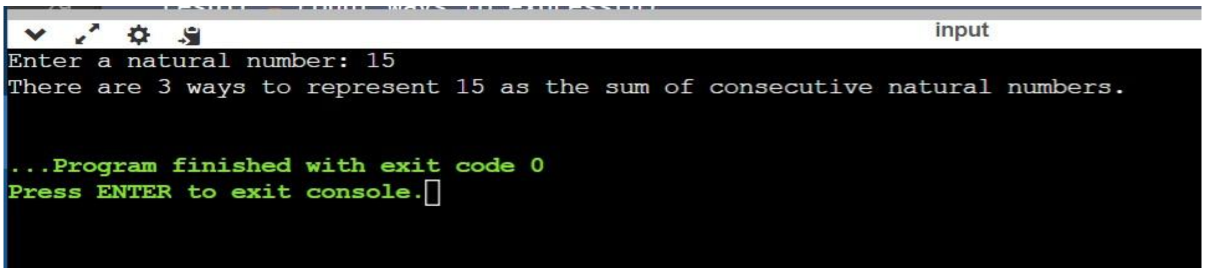
```
The amount of water trapped is approximately: 110.25


...Program finished with exit code 0
Press ENTER to exit console.
```

d) Count Ways to Express a Number as the Sum of Consecutive Natural Numbers
Given a natural number n, we are asked to find the total number of ways to express n as the sum of consecutive natural numbers.

Example 1: Input: 15
Output: 3
Explanation: There are 3 ways to represent 15 as sum of consecutive natural numbers as follows:
1 + 2 + 3 + 4 + 5 = 15
4 + 5 + 6 = 15
7 + 8 = 15

Ans -

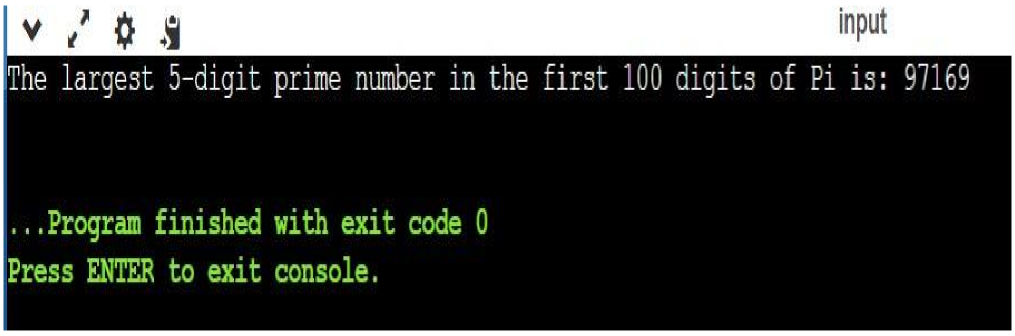
```
def count_ways_to_express(n):
    count = 0

    # Iterate over possible lengths of consecutive sequences
    for length in range(1, n + 1):
        # Calculate the starting number using the formula
        x = (n - length * (length - 2) // 2) // length

        # Check if x is a positive integer
        if x > 0 and n == length * (2 * x + length - 1) // 2:
            count += 1

    return count

# Take user input
try:
    n = int(input("Enter a natural number: "))
    if n <= 0:
        raise ValueError("Please enter a positive natural number.")

    result = count_ways_to_express(n)
    print(f"There are {result} ways to represent {n} as the sum of consecutive natural numbers.")
except ValueError as e:
    print(f"Error: {e}")
```



```
Enter a natural number: 15
There are 3 ways to represent 15 as the sum of consecutive natural numbers.


...Program finished with exit code 0
Press ENTER to exit console.
```

e) Write a piece of code to find the largest 5 digit prime number in the first 100 digits of Pi?

Ans -
```
import mpmath
from sympy import isprime

# Set precision for pi
mpmath.mp.dps = 102  # We set precision to 102 to ensure we get at least 100 decimal places

# Get the first 100 digits of Pi
pi_digits = str(mpmath.mp.pi)[2:102]

# Find the largest 5-digit prime number
largest_prime = 0
for i in range(len(pi_digits) - 4):
    five_digit_number = int(pi_digits[i:i+5])
    if isprime(five_digit_number) and five_digit_number > largest_prime:
        largest_prime = five_digit_number

print("The largest 5-digit prime number in the first 100 digits of Pi is:", largest_prime)
```



```
The largest 5-digit prime number in the first 100 digits of Pi is: 97169


...Program finished with exit code 0
Press ENTER to exit console.
```

f) What is dot product and cross product? Explain use cases of where dot product is used and cross product is used in graphics environment. Add links to places where you studied this information and get back with the understanding.

Ans - **Dot Product**

The Dot Product is a vector operation that calculates the angle between two vectors. The dot product is calculated in two different ways.

Version 1

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \quad \vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

$$\vec{u} \cdot \vec{v} = u_1 v_1 + u_2 v_2 + u_3 v_3$$

In the above equation, information about the angle between the vectors is missing. However, the result from this equation can tell us the direction of each vector. For example, if the dot product is equal to 1, it means that both vectors have the same direction. If the dot product is 0, it means that both vectors are perpendicular on each other. Finally, if the dot product is -1, it means that both vectors are heading in opposite directions.

Version 2

If we are interested in finding the angle between two vectors, the dot-product equation below can be used.

$$\vec{u} \cdot \vec{v} = ||u|| \; ||v|| \; \cos\theta$$

Use Cases in Graphics:

Vector Projection: The dot product is used in computing the projection of one vector onto another. This is useful in applications such as shadow rendering or determining how much of one vector lies in the direction of another.

Illumination Models: In computer graphics, lighting calculations often involve the dot product between the direction of light and the normal vector at a point on a surface. This helps determine how much light is incident on that point.

Texture Mapping: The dot product is used in texture mapping to determine the intensity of texture colors based on the orientation of the surface with respect to the light source.

Cross Product

Two vectors produces a plane. A cross product operation produces a vector that is perpendicular to both vectors. The cross product of two vectors is calculated as follows:

$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} \quad \vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

$$\vec{u} \times \vec{v} = \begin{bmatrix} u_2v_3 - u_3v_2 \\ u_3v_1 - u_1v_3 \\ u_1v_2 - u_2v_1 \end{bmatrix}$$

It is important to remember that a cross product can only be calculated with vectors in three-dimensions. If vectors reside in two-dimensional space, and the cross product is required, the vectors must be converted to three-dimensional vectors.

Use Cases in Graphics:

Normal Vector Calculation: The cross product is frequently used to compute the normal vector of a surface. This is essential for rendering realistic lighting effects.

Rotation Axis: In computer graphics, the cross product is used to find the axis of rotation when applying rotations. This is particularly important in animation.

Area Calculations: The magnitude of the cross product of two vectors gives the area of the parallelogram spanned by those vectors. This is useful in various geometric calculations.

Collision Detection: Cross products can be used in collision detection algorithms, especially in determining the orientation of objects.

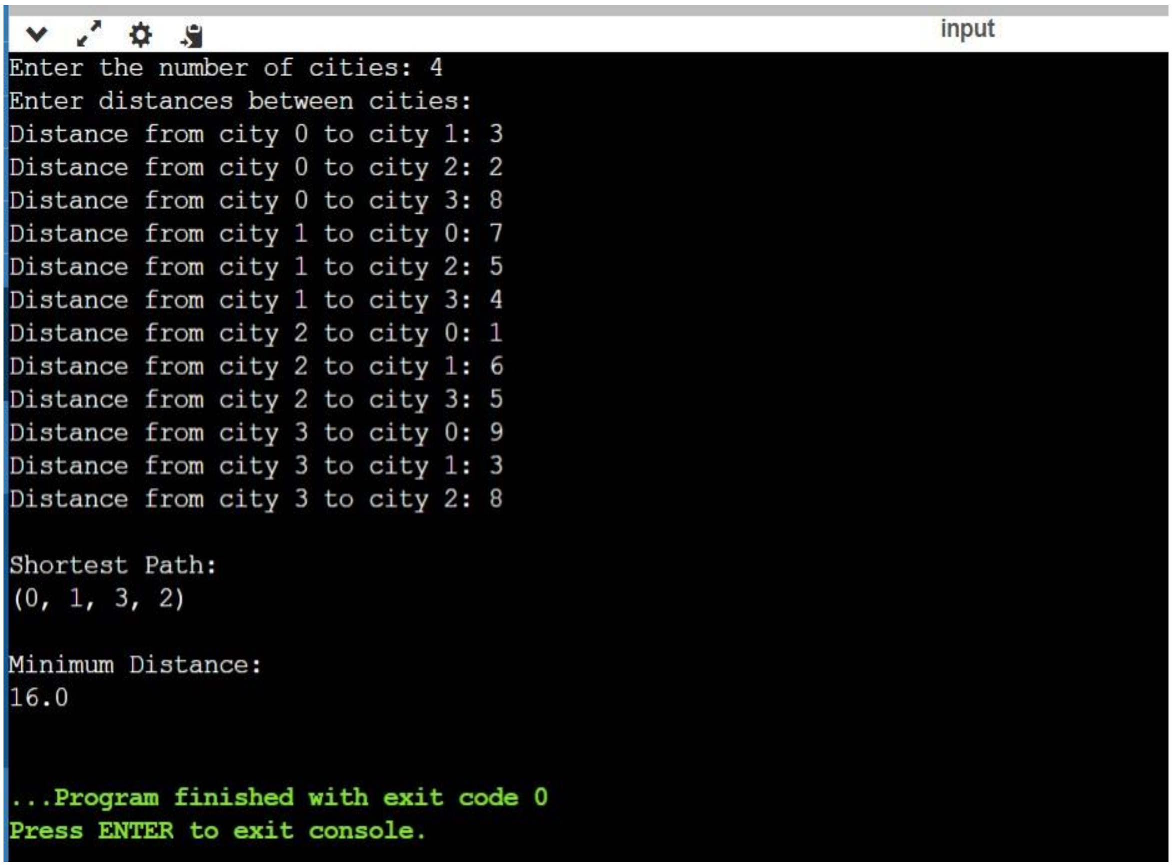
Links Used : - https://www.haroldserrano.com/blog/vectors-in-computer-graphics

https://www.mathworks.com/help/matlab/ref/dot.html

g) Explain a piece of code that you wrote which you are proud of? If you have not written any code, please write your favorite subject in engineering studies. We can go deep into that subject.

Ans - The Traveling Salesman Problem (TSP) is a classic optimization problem. Here's an example of solving the TSP in Python using user input and a simple approach called brute-force, which explores all possible permutations to find the shortest route:

```
import itertools

def calculate_total_distance(path, distances):
    total_distance = 0
    for i in range(len(path) - 1):
        total_distance += distances[path[i]][path[i + 1]]
    total_distance += distances[path[-1]][path[0]]  # Return to the starting point
    return total_distance

def traveling_salesman_problem(distances):
    cities = list(range(len(distances)))
    shortest_path = None
    min_distance = float('inf')

    for path in itertools.permutations(cities):
        total_distance = calculate_total_distance(path, distances)
        if total_distance < min_distance:
            min_distance = total_distance
            shortest_path = path

    return shortest_path, min_distance

def input_distances(num_cities):
    distances = []
    print("Enter distances between cities:")
    for i in range(num_cities):
        row = []
        for j in range(num_cities):
            if i == j:
                row.append(0)
            else:
                distance = float(input(f"Distance from city {i} to city {j}: "))
                row.append(distance)
        distances.append(row)
    return distances

if __name__ == "__main__":
    try:
        num_cities = int(input("Enter the number of cities: "))
        distances = input_distances(num_cities)

        shortest_path, min_distance = traveling_salesman_problem(distances)
        print("\nShortest Path:")
        print(shortest_path)

        print("\nMinimum Distance:")
        print(min_distance)

    except ValueError as e:
        print(f"Error: {e}")
```



```
Enter the number of cities: 4
Enter distances between cities:
Distance from city 0 to city 1: 3
Distance from city 0 to city 2: 2
Distance from city 0 to city 3: 8
Distance from city 1 to city 0: 7
Distance from city 1 to city 2: 5
Distance from city 1 to city 3: 4
Distance from city 2 to city 0: 1
Distance from city 2 to city 1: 6
Distance from city 2 to city 3: 5
Distance from city 3 to city 0: 9
Distance from city 3 to city 1: 3
Distance from city 3 to city 2: 8

Shortest Path:
(0, 1, 3, 2)

Minimum Distance:
16.0


...Program finished with exit code 0
Press ENTER to exit console.
```

h) Random crashes – you are given a source code to test and it randomly crashes and it never crashes in the same place (you have attached a debugger and you find this). Explain what all you would suspect and how would you go about with isolating the cause.

Ans - Random crashes in software can be challenging to diagnose, and finding the root cause may involve investigating various aspects of the code, environment, and dependencies. Here are some common suspects and steps you can take to isolate the cause of random crashes:

Suspects:

Memory Issues:
Memory Leaks: Unreleased memory can lead to crashes over time. Use tools like Valgrind or AddressSanitizer to detect memory leaks.
Dangling Pointers: Accessing memory after it has been freed can cause unpredictable behavior.

Thread Safety Issues:
Race Conditions: Concurrent access to shared data without proper synchronization can result in crashes. Use thread sanitizers and careful code review to identify race conditions.

Unhandled Exceptions:
Unhandled Exceptions: Unhandled exceptions can lead to unexpected program termination. Ensure that all potential exceptions are caught and properly handled.

Input Validation:
Invalid Input Handling: Improper validation of user inputs or external data can lead to buffer overflows or other security issues.

Resource Management:
File/Resource Handling: Incorrect file or resource handling may lead to crashes. Ensure that files are properly opened, closed, and resources are released.

Dependency Issues:
External Libraries: Incompatibility or incorrect usage of external libraries may cause crashes. Verify library versions and usage.

Compiler or Platform Specific Issues:
Compiler Flags: Incorrect compiler flags or optimizations might introduce issues. Review compiler settings.
Platform Dependencies: Ensure the code behaves consistently across different platforms.

Steps to Isolate the Cause:
Reproduce the Issue:
Try to reproduce the crash consistently. Identify specific inputs or actions that trigger the crash.

Debugger and Crash Dumps:
Use a debugger to analyze crash dumps. Examine stack traces, variable values, and the point of crash.

Logging:
Introduce logging at critical points in the code to gather information about the program's state before a crash.

Static Code Analysis:
Use static code analysis tools to identify potential issues before runtime. Tools like Clang Static Analyzer or PVS-Studio can be helpful.

Dynamic Analysis:
Employ dynamic analysis tools like Valgrind, AddressSanitizer, or similar tools to detect memory-related issues.

Code Review:
Conduct a thorough code review with an emphasis on potential suspects mentioned above.

Divide and Conquer:
Temporarily remove or isolate sections of code to narrow down the area causing the issue. Gradually reintroduce components until the crash reappears.

Version Control:
Check if the issue is related to recent changes. Use version control to identify the last working version and analyze changes since then.

Environment Variability:
Investigate if the crashes are dependent on specific runtime conditions, such as system load, network activity, or hardware variations.

Documentation and Community:
Review documentation for third-party libraries and seek community support. Others may have faced similar issues.

I declare that I have done the above work by myself and not worked with anyone or got help from any individual on the internet.