DAY 7 LAB PROBLEMS

1. You are given the number of sides on a die (num_sides), the number of dice to throw (num_dice), and a target sum (target). Develop a program that utilizes dynamic programming to solve the Dice Throw Problem.

Test Cases:

1.Simple Case:

•Number of sides: 6

•Number of dice: 2

•Target sum: 7

2.More Complex Case:

•Number of sides: 4

•Number of dice: 3

•Target sum: 10

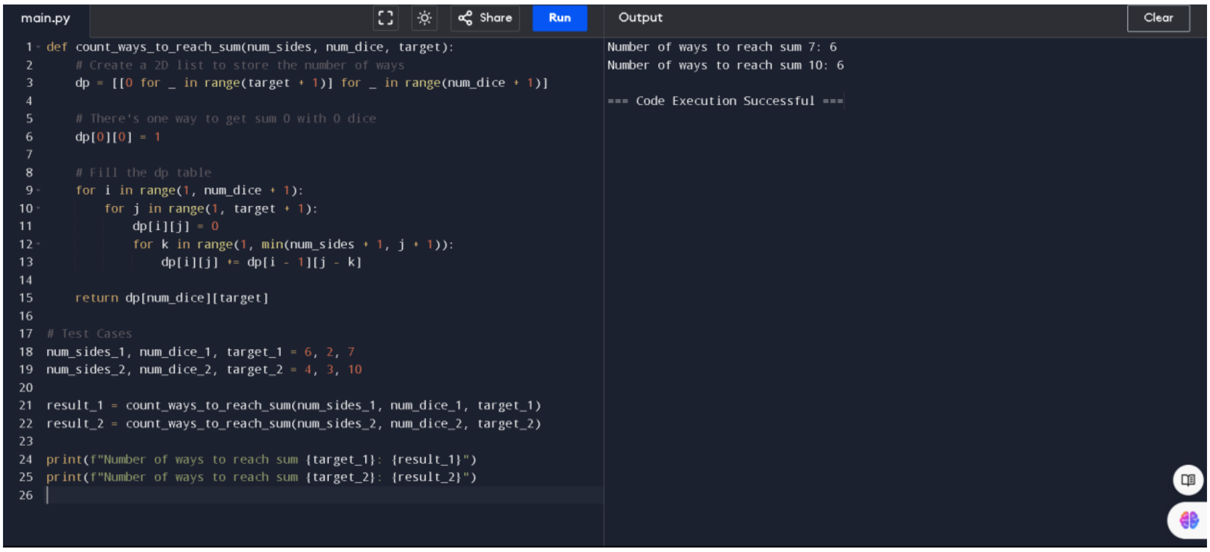
Output

Test Case 1:

Number of ways to reach sum 7: 6

Test Case 2:

Number of ways to reach sum 10: 27



2. In a factory, there are two assembly lines, each with n stations. Each station performs a specific task and takes a certain amount of time to complete. The task must go through each station in order, and there is also a transfer time for switching from one line to another. Given the time taken at each station on both lines and the transfer time between the lines, the goal is to find the minimum time required to process a product from start to end.

Input

n: Number of stations on each line.

a1[i]: Time taken at station i on assembly line 1.

a2[i]: Time taken at station i on assembly line 2.

t1[i]: Transfer time from assembly line 1 to assembly line 2 after station i.

t2[i]: Transfer time from assembly line 2 to assembly line 1 after station i.

e1: Entry time to assembly line 1.

e2: Entry time to assembly line 2.

x1: Exit time from assembly line 1.

x2: Exit time from assembly line 2.

Output

The minimum time required to process the product.

main.py

```python

    f1[0] = e1 + a1[0]
    f2[0] = e2 + a2[0]

    # Fill the DP tables for all stations
    for i in range(1, n):
        f1[i] = min(f1[i-1] + a1[i], f2[i-1] + t2[i-1] + a1[i])
        f2[i] = min(f2[i-1] + a2[i], f1[i-1] + t1[i-1] + a2[i])

    # Consider exit times
    min_time = min(f1[n-1] + x1, f2[n-1] + x2)

    return min_time

# Example usage
n = 4
a1 = [4, 5, 3, 2]
a2 = [2, 10, 1, 4]
t1 = [0, 7, 4, 5]
t2 = [0, 9, 2, 8]
e1 = 10
e2 = 12
x1 = 18
x2 = 7

min_time = min_processing_time(n, a1, a2, t1, t2, e1, e2, x1, x2)
print(f"The minimum time required to process the product is: {min_time}")
```

Output

```
The minimum time required to process the product is: 36

=== Code Execution Successful ===
```

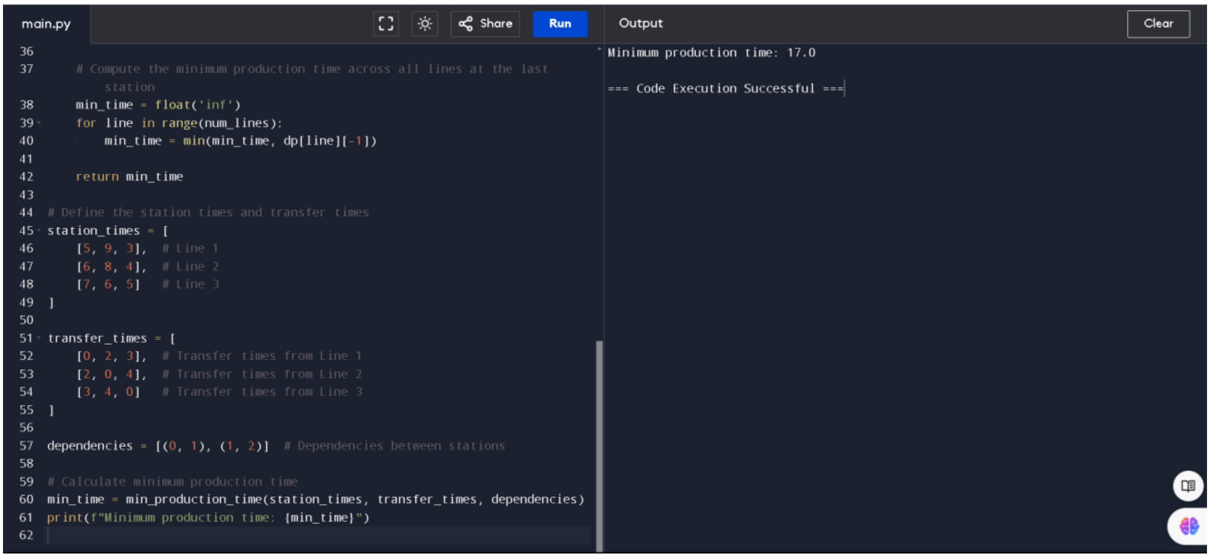
3. An automotive company has three assembly lines (Line 1, Line 2, Line 3) to produce different car models. Each line has a series of stations, and each station takes a certain amount of time to complete its task. Additionally, there are transfer times between lines, and certain dependencies must be respected due to the sequential nature of some tasks. Your goal is to minimize the total production time by determining the optimal scheduling of tasks across these lines, considering the transfer times and dependencies.

Number of stations: 3

- **Station times:**
- **Line 1: [5, 9, 3]**
- **Line 2: [6, 8, 4]**
- **Line 3: [7, 6, 5]**
- **Transfer times:**

[
[0, 2, 3],
[2, 0, 4],
[3, 4, 0]
]

Dependencies: [(0, 1), (1, 2)] (i.e., the output of the first station is needed for the second, and the second for the third, regardless of the line).



4. Write a c program to find the minimum path distance by using matrix form.

Test Cases:

1)

{0,10,15,20}

{10,0,35,25}

{15,35,0,30}

{20,25,30,0}

Output: 80

2)

{0,10,10,10}

{10,0,10,10}

{10,10,0,10}

{10,10,10,0}

Output: 40

3)

{0,1,2,3}

{1,0,4,5}

{2,4,0,6}

{3,5,6,0}

Output: 12

main.cpp

```
#include <stdio.h>
#include <limits.h>

#define V 4  // Number of nodes

// Function to find the minimum path using TSP
int tsp(int graph[][V], int pos, int visited, int dp[][1 << V]) {
    if (visited == (1 << V) - 1) {
        return graph[pos][0];  // Return to the starting point
    }

    if (dp[pos][visited] != -1) {
        return dp[pos][visited];
    }

    int minCost = INT_MAX;

    for (int city = 0; city < V; city++) {
        if ((visited & (1 << city)) == 0) {
            int newCost = graph[pos][city] + tsp(graph, city, visited | (1
                << city), dp);
            minCost = newCost < minCost ? newCost : minCost;
        }
    }

    return dp[pos][visited] = minCost;
}
```

Output

```
/tmp/434XpS9Jmx.o
Minimum path distance for graph1: 80
Minimum path distance for graph2: 40
Minimum path distance for graph3: 14


=== Code Execution Successful ===
```

5. Assume you are solving the Traveling Salesperson Problem for 4 cities (A, B, C, D) with known distances between each pair of cities. Now, you need to add a fifth city (E) to the problem.

Test Cases

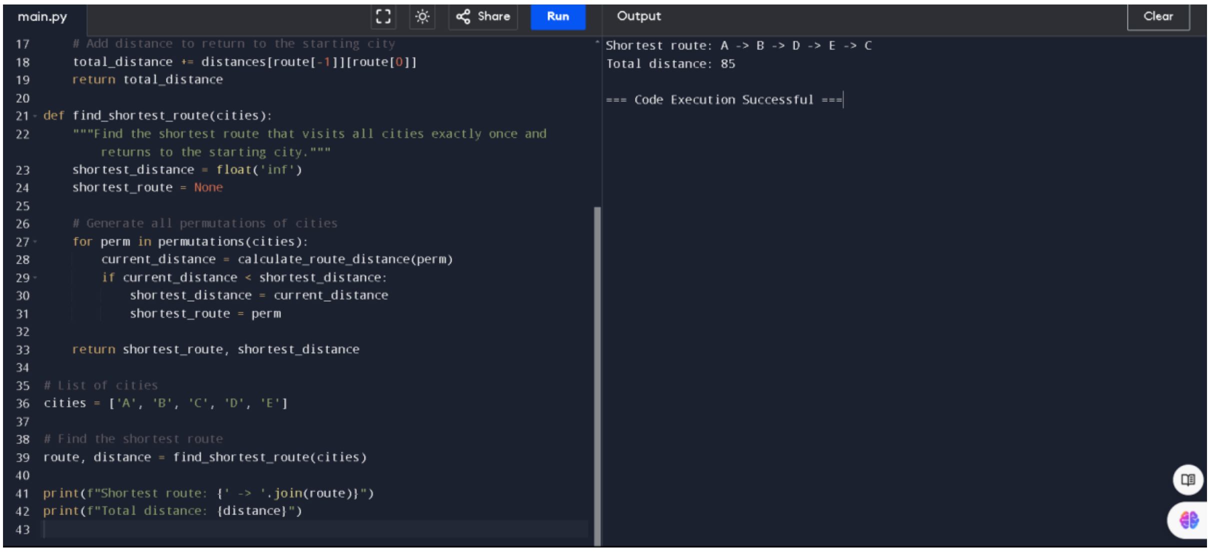
1. Symmetric Distances

• Description: All distances are symmetric (distance from A to B is the same as B to A).

Distances:

A-B: 10, A-C: 15, A-D: 20, A-E: 25 B-C: 35, B-D: 25, B-E: 30 C-D: 30, C-E: 20 D-E: 15

Expected Output: The shortest route and its total distance. For example, A -> B -> D -> E -> C -> A might be the shortest route depending on the given distances.



6. Given a string s, return the longest palindromic substring in S.

Example 1:

Input: s = "babad"

Output: "bab" Explanation: "aba" is also a valid answer.

Example 2:

Input: s = "cbbd"

Output: "bb"

Constraints: ● 1 <= s.length <= 1000 ● s consist of only digits and English letters.

main.py

```python
    if not s or len(s) == 1:
        return s

    start, end = 0, 0

    def expand_around_center(left: int, right: int) -> (int, int):
        while left >= 0 and right < len(s) and s[left] == s[right]:
            left -= 1
            right += 1
        return left + 1, right - 1

    for i in range(len(s)):
        # Expand around center i (odd length palindrome)
        l1, r1 = expand_around_center(i, i)
        # Expand around center i and i+1 (even length palindrome)
        l2, r2 = expand_around_center(i, i + 1)

        if r1 - l1 > end - start:
            start, end = l1, r1
        if r2 - l2 > end - start:
            start, end = l2, r2

    return s[start:end + 1]

# Examples
print(longest_palindromic_substring("babad"))  # Output: "bab" or "aba"
print(longest_palindromic_substring("cbbd"))   # Output: "bb"
```

Output

```
bab
bb

=== Code Execution Successful ===
```

7. Given a string s, find the length of the longest substring without repeating characters.

Example 1: Input: s = "abcabcbb" Output: 3

Explanation: The answer is "abc", with the length of 3.

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

Explanation: The answer is "b", with the length of 1.

Example 3: Input: s = "pwwkew" Output: 3

Explanation: The answer is "wke", with the length of 3.

Notice that the answer must be a substring, "pwke" is a subsequence and not a substring.

Constraints: ● 0 <= s.length <= 5 * 104 ● s consists of English letters, digits, symbols and spaces.

main.py

```python
def lengthOfLongestSubstring(s: str) -> int:
    char_index_map = {}
    start = 0
    max_length = 0

    for i, char in enumerate(s):
        if char in char_index_map and char_index_map[char] >= start:
            start = char_index_map[char] + 1
        char_index_map[char] = i
        max_length = max(max_length, i - start + 1)

    return max_length
```

Output

```
=== Code Execution Successful ===
```

8. Given a string s and a dictionary of strings wordDict, return true if s can be segmented into a space-separated sequence of one or more dictionary words.

Note that the same word in the dictionary may be reused multiple times in the segmentation.

Example 1:

Input: s = "leetcode", wordDict = ["leet","code"]

Output: true

Explanation: Return true because "leetcode" can be segmented as "leet code".

Example 2:

Input: s = "applepenapple", wordDict = ["apple","pen"]

Output: true

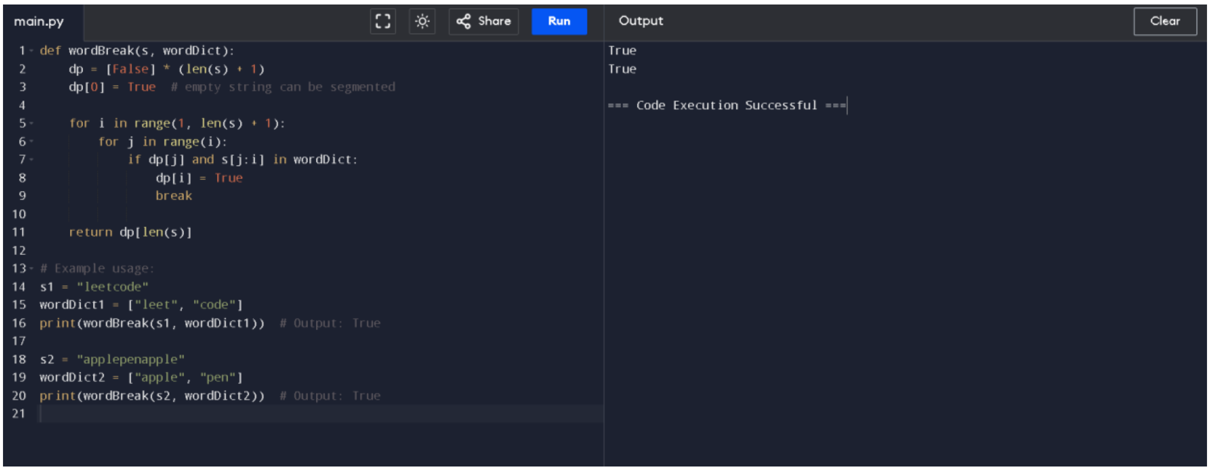
Explanation: Return true because "applepenapple" can be segmented as "apple pen apple".

Note that you are allowed to reuse a dictionary word.

Example 3:

Input: s = "catsandog", wordDict = ["cats","dog","sand","and","cat"]

Output: false



9. Given an input string and a dictionary of words, find out if the input string can be segmented into a space-separated sequence of dictionary words.Consider the following dictionary { i, like, sam, sung, samsung, mobile, ice, cream, icecream, man, go, mango}

Input: ilike

Output: Yes

The string can be segmented as "i like".

Input: ilikesamsung

Output: Yes The string can be segmented as "i like samsung" or "i like sam sung".

main.py

```python
def word_break(s, word_dict):
    n = len(s)
    dp = [False] * (n + 1)
    dp[0] = True

    for i in range(1, n + 1):
        for j in range(i):
            if dp[j] and s[j:i] in word_dict:
                dp[i] = True
                break

    return dp[n]

# Test the function with the provided examples
word_dict = {"i", "like", "sam", "sung", "samsung", "mobile", "ice", "cream",
    "icecream", "man", "go", "mango"}

# Example 1
s1 = "ilike"
print("Yes" if word_break(s1, word_dict) else "No")  # Output: Yes

# Example 2
s2 = "ilikesamsung"
print("Yes" if word_break(s2, word_dict) else "No")  # Output: Yes
```

Output

```
Yes
Yes

=== Code Execution Successful ===
```

10. Given an array of strings words and a width maxWidth, format the text such that each line has exactly maxWidth characters and is fully (left and right) justified. You should pack your words in a greedy approach; that is, pack as many words as you can in each line. Pad extra spaces ' ' when necessary so that each line has exactly maxWidth characters. Extra spaces between words should be distributed as evenly as possible. If the number of spaces on a line does not divide evenly between words, the empty slots on the left will be assigned more spaces than the slots on the right. For the last line of text, it should be left-justified, and no extra space is inserted between words. A word is defined as a character sequence consisting of non-space characters only. Each word's length is guaranteed to be greater than 0 and not exceed maxWidth. The input array words contains at least one word.

Example 1:

Input: words = ["This", "is", "an", "example", "of", "text", "justification."],

maxWidth =

16

Output:

["This is an",

"example of text",

"justification. "

]

Example 2:

Input: words = ["What","must","be","acknowledgment","shall","be"], maxWidth = 16

Output:

[

"What must be",

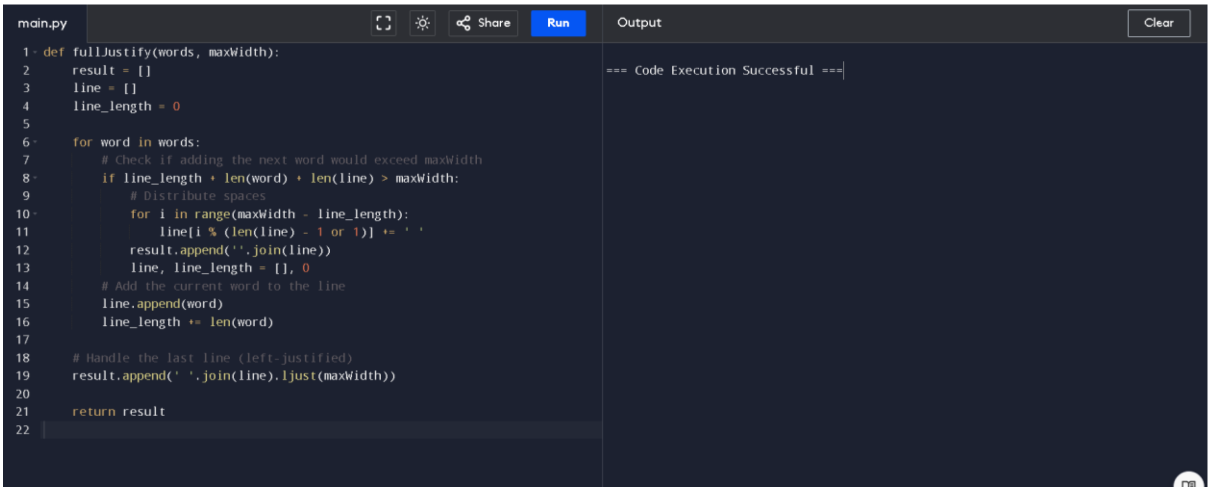
"acknowledgment ",

"shall be "

]

Explanation: Note that the last line is "shall be " instead of "shall be", because the last line must be left-justified instead of fully-justified.

Note that the second line is also left-justified because it contains only one word.



```python
def fullJustify(words, maxWidth):
    result = []
    line = []
    line_length = 0

    for word in words:
        # Check if adding the next word would exceed maxWidth
        if line_length + len(word) + len(line) > maxWidth:
            # Distribute spaces
            for i in range(maxWidth - line_length):
                line[i % (len(line) - 1 or 1)] += ' '
            result.append(''.join(line))
            line, line_length = [], 0
        # Add the current word to the line
        line.append(word)
        line_length += len(word)

    # Handle the last line (left-justified)
    result.append(' '.join(line).ljust(maxWidth))

    return result
```


11. Design a special dictionary that searches the words in it by a prefix and a suffix. Implement the WordFilter class: WordFilter(string[] words) Initializes the object with the words in the dictionary.f(string pref, string suff) Returns the index of the word in the dictionary, which has the prefix pref and the suffix suff. If there is more than one valid index, return the largest of them. If there is no such word in the dictionary, return -1.

Example 1:

Input

["WordFilter", "f"]

[[["apple"]], ["a", "e"]]

Output

[null, 0]

Explanation

WordFilter wordFilter = new WordFilter(["apple"]);

wordFilter.f("a", "e"); // return 0, because the word at index 0 has prefix = "a" and suffix

main.py

```python
class WordFilter:
    def __init__(self, words: list[str]):
        self.word_dict = {}
        self.words = words

        for index, word in enumerate(words):
            word_length = len(word)
            for i in range(word_length + 1):
                prefix = word[:i]
                for j in range(word_length + 1):
                    suffix = word[j:]
                    self.word_dict[(prefix, suffix)] = index

    def f(self, pref: str, suff: str) -> int:
        return self.word_dict.get((pref, suff), -1)

# Example usage:
word_filter = WordFilter(["apple"])
print(word_filter.f("a", "e"))  # Output: 0
```

Output

```
0

=== Code Execution Successful ===
```