main.py

Share Run

```python
def full_justify(words, maxWidth):
    result = []
    current_line = []
    current_length = 0
    for word in words:
        if current_length + len(word) + len(current_line) > maxWidth:
            for i in range(maxWidth - current_length):
                current_line[i % (len(current_line) - 1 or 1)] += ' '
            result.append(''.join(current_line))
            current_line = []
            current_length = 0
        current_line.append(word)
        current_length += len(word)
    result.append(' '.join(current_line).ljust(maxWidth))
    return result
words1 = ["This", "is", "an", "example", "of", "text", "justification."]
maxWidth1 = 16
```

Output Clear

```
['This    is    an', 'example  of text', 'justification.  ']
['What   must   be', 'acknowledgment  ', 'shall be        ']

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

main.py

Share Run

```python
def word_break(s, wordDict):
    word_set = set(wordDict)
    dp = [False] * (len(s) + 1)
    dp[0] = True
    for i in range(1, len(s) + 1):
        for j in range(i):
            if dp[j] and s[j:i] in word_set:
                dp[i] = True
                break
    return dp[-1]
print(word_break("leetcode", ["leet", "code"]))
print(word_break("applepenapple", ["apple", "pen"]))
print(word_break("catsandog", ["cats", "dog", "sand", "and",
    "cat"]))
```

Output Clear

```
True
True
False

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

main.c

Share Run

Output Clear

```c
#include <stdio.h>
#include <limits.h>
#define N 5
int tsp(int mask, int pos, int dist[N][N], int dp[N][1 << N]) {
    if (mask == (1 << N) - 1) return dist[pos][0];
    if (dp[pos][mask] != -1) return dp[pos][mask];
    int ans = INT_MAX;
    for (int city = 0; city < N; city++) {
        if ((mask & (1 << city)) == 0) {
            int newAns = dist[pos][city] + tsp(mask | (1 <<
                city), city, dist, dp);
            ans = (ans < newAns) ? ans : newAns;
        }
    }
    return dp[pos][mask] = ans;
}
int main() {
    int dist[N][N] = {
        {0, 10, 15, 20, 25},
        {10, 0, 35, 25, 30},
```

```
/tmp/wSxbsz6O0B.o
Minimum distance: 85


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

main.c

Share Run

```c
        }
    }
    return dp[pos][mask] = ans;
}
int main() {
    int dist[N][N] = {
        {0, 10, 15, 20, 25},
        {10, 0, 35, 25, 30},
        {15, 35, 0, 30, 20},
        {20, 25, 30, 0, 15},
        {25, 30, 20, 15, 0}
    };
    int dp[N][1 << N];
    for (int j = 0; j < N; j++) for (int k = 0; k < (1 << N); k
        ++) dp[j][k] = -1;
    int min_distance = tsp(1, 0, dist, dp);
    printf("Minimum distance: %d\n", min_distance);
    return 0;
}
```

Output Clear

```
/tmp/wSxbsz6O0B.o
Minimum distance: 85


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

main.py

Share Run

```python
def dice_throw(num_sides, num_dice, target):
    dp = [[0 for _ in range(target + 1)] for _ in range(num_dice
        + 1)]
    dp[0][0] = 1
    for d in range(1, num_dice + 1):
        for s in range(1, target + 1):
            dp[d][s] = 0
            for k in range(1, num_sides + 1):
                if s - k >= 0:
                    dp[d][s] += dp[d-1][s-k]
    return dp[num_dice][target]
num_sides1 = 6
num_dice1 = 2
target1 = 7
print("Test Case 1: Number of ways to reach sum",target1,":",
    dice_throw(num_sides1, num_dice1, target1))
```

Output

Clear

```
Test Case 1: Number of ways to reach sum 7 : 6

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

main.c

Share Run

Output Clear

```c
#include <stdio.h>
#include <limits.h>
#define N 4
int tsp(int mask, int pos, int dist[N][N], int dp[N][1 << N]) {
    if (mask == (1 << N) - 1) return dist[pos][0];
    if (dp[pos][mask] != -1) return dp[pos][mask];
    int ans = INT_MAX;
    for (int city = 0; city < N; city++) {
        if ((mask & (1 << city)) == 0) {
            ans = (ans < dist[pos][city] + tsp(mask | (1 <<
                city), city, dist, dp)) ? ans :
                (dist[pos][city] + tsp(mask | (1 << city), city
                , dist, dp));
        }
    }
    return dp[pos][mask] = ans;
}
int main() {
    int distances[3][N][N] = {
        {{0, 10, 15, 20}, {10, 0, 35, 25}, {15, 35, 0, 30}, {20
```

```
/tmp/T7NeqzAEZn.o
Output for test case 1: 80
Output for test case 2: 40
Output for test case 3: 14


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

main.py

Share Run

Output Clear

```python
def word_segment(s, wordDict):
    word_set = set(wordDict)
    dp = [False] * (len(s) + 1)
    dp[0] = True
    for i in range(1, len(s) + 1):
        for j in range(i):
            if dp[j] and s[j:i] in word_set:
                dp[i] = True
                break
    return "Yes" if dp[-1] else "No"
dictionary = {"i", "like", "sam", "sung", "samsung", "mobile",
    "ice", "cream", "icecream", "man", "go", "mango"}
print(word_segment("ilike", dictionary))
print(word_segment("ilikesamsung", dictionary))
```

```
Yes
Yes

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

main.py

Share Run

Output Clear

```python
def length_of_longest_substring(s: str) -> int:
    char_index = {}
    max_length = start = 0
    for i, char in enumerate(s):
        if char in char_index and char_index[char] >= start:
            start = char_index[char] + 1
        char_index[char] = i
        max_length = max(max_length, i - start + 1)
    return max_length
print(length_of_longest_substring("abcabcbb"))
print(length_of_longest_substring("bbbbb"))
print(length_of_longest_substring("pwwkew"))
```

```
3
1
3

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

main.py

Share Run

```python
def automotive_production(station_times, transfer_times):
    T = [[0] * 3 for _ in range(3)]
    for j in range(3):
        T[0][j] = station_times[j][0]
    for i in range(1, 3):
        for j in range(3):
            T[i][j] = min(T[i-1][j] + station_times[j][i],
                          min(T[i-1][(j + k) % 3] +
                              transfer_times[(j + k) % 3][j] +
                              station_times[j][i] for k in [1, 2]
                              ))
    return min(T[2])
station_times = [[5, 9, 3], [6, 8, 4], [7, 6, 5]]
transfer_times = [[0, 2, 3], [2, 0, 4], [3, 4, 0]]
print("Minimum total production time:",automotive_production
    (station_times, transfer_times))
```

Output

Clear

```
Minimum total production time: 17

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

main.py

Share Run

```
            for i in range(maxWidth - current_length):
                current_line[i % (len(current_line) - 1 or 1)]
                    += ' '
            result.append(''.join(current_line))
            current_line = []
            current_length = 0
        current_line.append(word)
        current_length += len(word)
    result.append(' '.join(current_line).ljust(maxWidth))
    return result
words1 = ["This", "is", "an", "example", "of", "text",
    "justification."]
maxWidth1 = 16
output1 = full_justify(words1, maxWidth1)
print(output1)
words2 = ["What","must","be","acknowledgment","shall","be"]
maxWidth2 = 16
output2 = full_justify(words2, maxWidth2)
print(output2)
```

Output Clear

```
['This    is    an', 'example  of text', 'justification.  ']
['What   must   be', 'acknowledgment  ', 'shall be        ']

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

main.py

Share Run

Output Clear

```python
def longest_palindrome(s: str) -> str:
    start, max_len = 0, 0
    for i in range(len(s)):
        for j in (i, i + 1):
            left, right = i, j
            while left >= 0 and right < len(s) and s[left] ==
                s[right]:
                if right - left + 1 > max_len:
                    start, max_len = left, right - left + 1
                left -= 1
                right += 1
    return s[start:start + max_len]
print(longest_palindrome("babad"))
print(longest_palindrome("cbbd"))
```

```
bab
bb

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

main.py

Share Run

Output Clear

```python
class WordFilter:
    def f(words, pref, suff):
        max_index = -1
        for index, word in enumerate(words):
            if word.startswith(pref) and word.endswith(suff):
                max_index = index
        return max_index
words = ["apple"]
output = WordFilter.f(words, "a", "e")
print(output)

```

```
0

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

main.c

Share Run

```
}
int main() {
    int distances[3][N][N] = {
        {{0, 10, 15, 20}, {10, 0, 35, 25}, {15, 35, 0, 30}, {20
            , 25, 30, 0}},
        {{0, 10, 10, 10}, {10, 0, 10, 10}, {10, 10, 0, 10}, {10
            , 10, 10, 0}},
        {{0, 1, 2, 3}, {1, 0, 4, 5}, {2, 4, 0, 6}, {3, 5, 6,
            0}}
    };
    for (int i = 0; i < 3; i++) {
        int dp[N][1 << N] = {0};
        for (int j = 0; j < N; j++) for (int k = 0; k < (1 << N
            ); k++) dp[j][k] = -1;
        printf("Output for test case %d: %d\n", i + 1, tsp(1, 0
            , distances[i], dp));
    }
    return 0;
}
```

Output

Clear

```
/tmp/T7NeqzAEZn.o
Output for test case 1: 80
Output for test case 2: 40
Output for test case 3: 14


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

main.py

Share Run

```python
def assembly_line(n, a1, a2, t1, t2, e1, e2, x1, x2):
    T1 = [0] * n
    T2 = [0] * n
    T1[0] = e1 + a1[0]
    T2[0] = e2 + a2[0]
    for i in range(1, n):
        T1[i] = min(T1[i-1] + a1[i], T2[i-1] + t2[i-1] + a1[i])
        T2[i] = min(T2[i-1] + a2[i], T1[i-1] + t1[i-1] + a2[i])
    return min(T1[n-1] + x1, T2[n-1] + x2)
n = 4
a1 = [4, 5, 3, 2]
a2 = [2, 10, 1, 4]
t1 = [7, 4, 5]
t2 = [9, 2, 8]
e1 = 10
e2 = 12
x1 = 18
x2 = 7
print("Minimum time required to process the product:"
    ,assembly_line(n, a1, a2, t1, t2, e1, e2, x1, x2))
```

Output

Clear

```
Minimum time required to process the product: 35

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