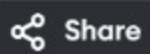




main.py



Run

Output

Clear

```
1 def full_justify(words, maxWidth):
2     result = []
3     current_line = []
4     current_length = 0
5     for word in words:
6         if current_length + len(word) + len(current_line) >
            maxWidth:
7             for i in range(maxWidth - current_length):
8                 current_line[i % (len(current_line) - 1 or 1)]
                    += ' '
9             result.append(''.join(current_line))
10            current_line = []
11            current_length = 0
12            current_line.append(word)
13            current_length += len(word)
14    result.append(' '.join(current_line).ljust(maxWidth))
15    return result
16 words1 = ["This", "is", "an", "example", "of", "text",
            "justification."]
17 maxWidth1 = 16
```

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

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



main.py



Share

Run

Output

Clear

```
1 def word_break(s, wordDict):
2     word_set = set(wordDict)
3     dp = [False] * (len(s) + 1)
4     dp[0] = True
5     for i in range(1, len(s) + 1):
6         for j in range(i):
7             if dp[j] and s[j:i] in word_set:
8                 dp[i] = True
9                 break
10    return dp[-1]
11 print(word_break("leetcode", ["leet", "code"]))
12 print(word_break("applepenapple", ["apple", "pen"]))
13 print(word_break("catsandog", ["cats", "dog", "sand", "and",
    "cat"]))
```

True

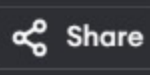
True

False

=== Code Execution Successful ===



main.c



Share

Run

Output

Clear

```
1  #include <stdio.h>
2  #include <limits.h>
3  #define N 5
4  int tsp(int mask, int pos, int dist[N][N], int dp[N][1 << N]) {
5      if (mask == (1 << N) - 1) return dist[pos][0];
6      if (dp[pos][mask] != -1) return dp[pos][mask];
7      int ans = INT_MAX;
8      for (int city = 0; city < N; city++) {
9          if ((mask & (1 << city)) == 0) {
10             int newAns = dist[pos][city] + tsp(mask | (1 <<
                city), city, dist, dp);
11             ans = (ans < newAns) ? ans : newAns;
12         }
13     }
14     return dp[pos][mask] = ans;
15 }
16 int main() {
17     int dist[N][N] = {
18         {0, 10, 15, 20, 25},
19         {10, 0, 35, 25, 30},
```

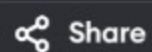
/tmp/wSxbsz600B.o

Minimum distance: 85

=== Code Execution Successful ===



main.c



Share

Run

Output

Clear



JS

GO

```
12     }
13 }
14 return dp[pos][mask] = ans;
15 }
16 int main() {
17     int dist[N][N] = {
18         {0, 10, 15, 20, 25},
19         {10, 0, 35, 25, 30},
20         {15, 35, 0, 30, 20},
21         {20, 25, 30, 0, 15},
22         {25, 30, 20, 15, 0}
23     };
24     int dp[N][1 << N];
25     for (int j = 0; j < N; j++) for (int k = 0; k < (1 << N); k
        ++) dp[j][k] = -1;
26     int min_distance = tsp(1, 0, dist, dp);
27     printf("Minimum distance: %d\n", min_distance);
28     return 0;
29 }
```

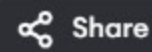
/tmp/wSxbsz600B.o

Minimum distance: 85

=== Code Execution Successful ===



main.py



Run

Output

Clear

```
1 def dice_throw(num_sides, num_dice, target):
2     dp = [[0 for _ in range(target + 1)] for _ in range(num_dice
3         + 1)]
4     dp[0][0] = 1
5     for d in range(1, num_dice + 1):
6         for s in range(1, target + 1):
7             dp[d][s] = 0
8             for k in range(1, num_sides + 1):
9                 if s - k >= 0:
10                     dp[d][s] += dp[d-1][s-k]
11     return dp[num_dice][target]
12 num_sides1 = 6
13 num_dice1 = 2
14 target1 = 7
15 print("Test Case 1: Number of ways to reach sum",target1,":",
16     dice_throw(num_sides1, num_dice1, target1))
```

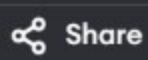
Test Case 1: Number of ways to reach sum 7 : 6

=== Code Execution Successful ===

JS



main.c



Run

Output

Clear



```
1  #include <stdio.h>
2  #include <limits.h>
3  #define N 4
4  int tsp(int mask, int pos, int dist[N][N], int dp[N][1 << N]) {
5      if (mask == (1 << N) - 1) return dist[pos][0];
6      if (dp[pos][mask] != -1) return dp[pos][mask];
7      int ans = INT_MAX;
8      for (int city = 0; city < N; city++) {
9          if ((mask & (1 << city)) == 0) {
10             ans = (ans < dist[pos][city] + tsp(mask | (1 <<
                city), city, dist, dp)) ? ans :
                (dist[pos][city] + tsp(mask | (1 << city), city
                , dist, dp));
11         }
12     }
13     return dp[pos][mask] = ans;
14 }
15 int main() {
16     int distances[3][N][N] = {
17         {{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

```
1 def word_segment(s, wordDict):
2     word_set = set(wordDict)
3     dp = [False] * (len(s) + 1)
4     dp[0] = True
5     for i in range(1, len(s) + 1):
6         for j in range(i):
7             if dp[j] and s[j:i] in word_set:
8                 dp[i] = True
9                 break
10    return "Yes" if dp[-1] else "No"
11    dictionary = {"i", "like", "sam", "sung", "samsung", "mobile",
12                 "ice", "cream", "icecream", "man", "go", "mango"}
13    print(word_segment("ilike", dictionary))
14    print(word_segment("ilikesamsung", dictionary))
```

Yes

Yes

=== Code Execution Successful ===

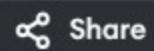


JS





main.py



Run

Output

Clear

```
1 def length_of_longest_substring(s: str) -> int:
2     char_index = {}
3     max_length = start = 0
4     for i, char in enumerate(s):
5         if char in char_index and char_index[char] >= start:
6             start = char_index[char] + 1
7             char_index[char] = i
8             max_length = max(max_length, i - start + 1)
9     return max_length
10 print(length_of_longest_substring("abcabcbb"))
11 print(length_of_longest_substring("bbbbbb"))
12 print(length_of_longest_substring("pwwkew"))
```

3

1

3

=== Code Execution Successful ===





main.py



Share

Run

Output

Clear

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

Minimum total production time: 17

=== Code Execution Successful ===

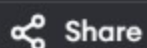


JS





main.py



Share

Run

Output

Clear



```
7   for i in range(maxWidth - current_length):
8       current_line[i % (len(current_line) - 1 or 1)]
          += ' '
9       result.append(' '.join(current_line))
10      current_line = []
11      current_length = 0
12      current_line.append(word)
13      current_length += len(word)
14      result.append(' '.join(current_line).ljust(maxWidth))
15      return result
16 words1 = ["This", "is", "an", "example", "of", "text",
           "justification."]
17 maxWidth1 = 16
18 output1 = full_justify(words1, maxWidth1)
19 print(output1)
20 words2 = ["What", "must", "be", "acknowledgment", "shall", "be"]
21 maxWidth2 = 16
22 output2 = full_justify(words2, maxWidth2)
23 print(output2)
```

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

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



main.py



Share

Run

Output

Clear



```
1 def longest_palindrome(s: str) -> str:
2     start, max_len = 0, 0
3     for i in range(len(s)):
4         for j in (i, i + 1):
5             left, right = i, j
6             while left >= 0 and right < len(s) and s[left] ==
              s[right]:
7                 if right - left + 1 > max_len:
8                     start, max_len = left, right - left + 1
9                 left -= 1
10                right += 1
11     return s[start:start + max_len]
12 print(longest_palindrome("babad"))
13 print(longest_palindrome("cbbsd"))
```

bab

bb

=== Code Execution Successful ===

JS

GO



main.py



Share

Run

Output

Clear



JS



```
1 class WordFilter:
2     def f(words, pref, suff):
3         max_index = -1
4         for index, word in enumerate(words):
5             if word.startswith(pref) and word.endswith(suff):
6                 max_index = index
7         return max_index
8 words = ["apple"]
9 output = WordFilter.f(words, "a", "e")
10 print(output)
11
```

0

=== Code Execution Successful ===



main.c



Share

Run

Output

Clear

```
14 }
15 int main() {
16     int distances[3][N][N] = {
17         {{0, 10, 15, 20}, {10, 0, 35, 25}, {15, 35, 0, 30}, {20
            , 25, 30, 0}},
18         {{0, 10, 10, 10}, {10, 0, 10, 10}, {10, 10, 0, 10}, {10
            , 10, 10, 0}},
19         {{0, 1, 2, 3}, {1, 0, 4, 5}, {2, 4, 0, 6}, {3, 5, 6,
            0}}
20     };
21     for (int i = 0; i < 3; i++) {
22         int dp[N][1 << N] = {0};
23         for (int j = 0; j < N; j++) for (int k = 0; k < (1 << N
            ); k++) dp[j][k] = -1;
24         printf("Output for test case %d: %d\n", i + 1, tsp(1, 0
            , distances[i], dp));
25     }
26     return 0;
27 }
```

^ /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

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

Minimum time required to process the product: 35

=== Code Execution Successful ===