

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

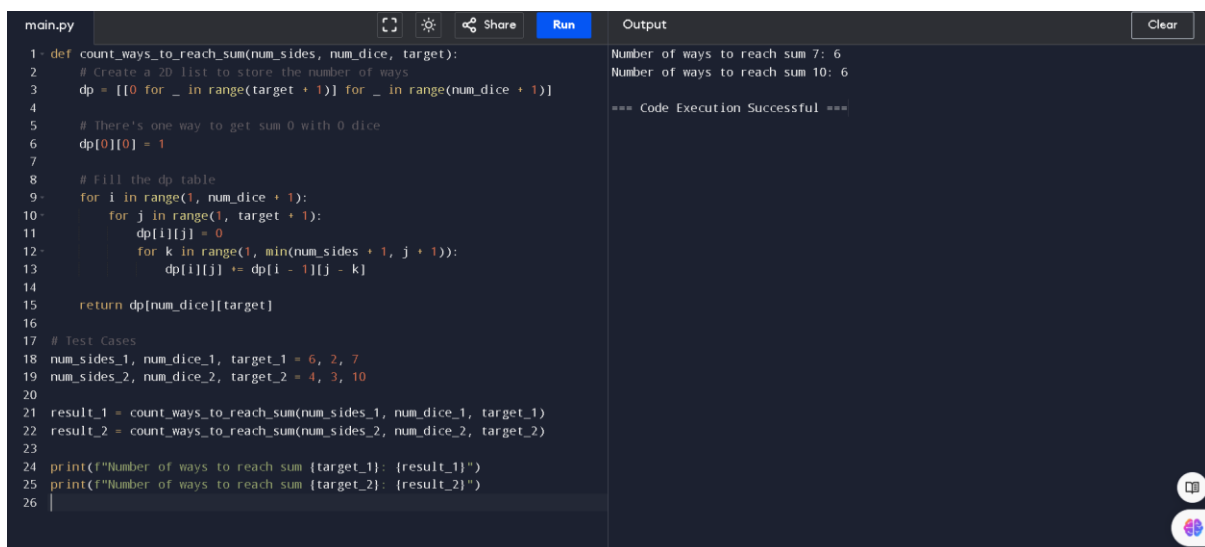
Output

Test Case 1:

Number of ways to reach sum 7: 6

Test Case 2:

Number of ways to reach sum 10: 27



```
main.py  Run  Output  Clear
1 def count_ways_to_reach_sum(num_sides, num_dice, target):
2     # Create a 2D list to store the number of ways
3     dp = [[0 for _ in range(target + 1)] for _ in range(num_dice + 1)]
4
5     # There's one way to get sum 0 with 0 dice
6     dp[0][0] = 1
7
8     # Fill the dp table
9     for i in range(1, num_dice + 1):
10        for j in range(1, target + 1):
11            dp[i][j] = 0
12            for k in range(1, min(num_sides + 1, j + 1)):
13                dp[i][j] += dp[i - 1][j - k]
14
15    return dp[num_dice][target]
16
17 # Test Cases
18 num_sides_1, num_dice_1, target_1 = 6, 2, 7
19 num_sides_2, num_dice_2, target_2 = 4, 3, 10
20
21 result_1 = count_ways_to_reach_sum(num_sides_1, num_dice_1, target_1)
22 result_2 = count_ways_to_reach_sum(num_sides_2, num_dice_2, target_2)
23
24 print(f"Number of ways to reach sum {target_1}: {result_1}")
25 print(f"Number of ways to reach sum {target_2}: {result_2}")
26
```

Number of ways to reach sum 7: 6
Number of ways to reach sum 10: 27
=== Code Execution Successful ===

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.

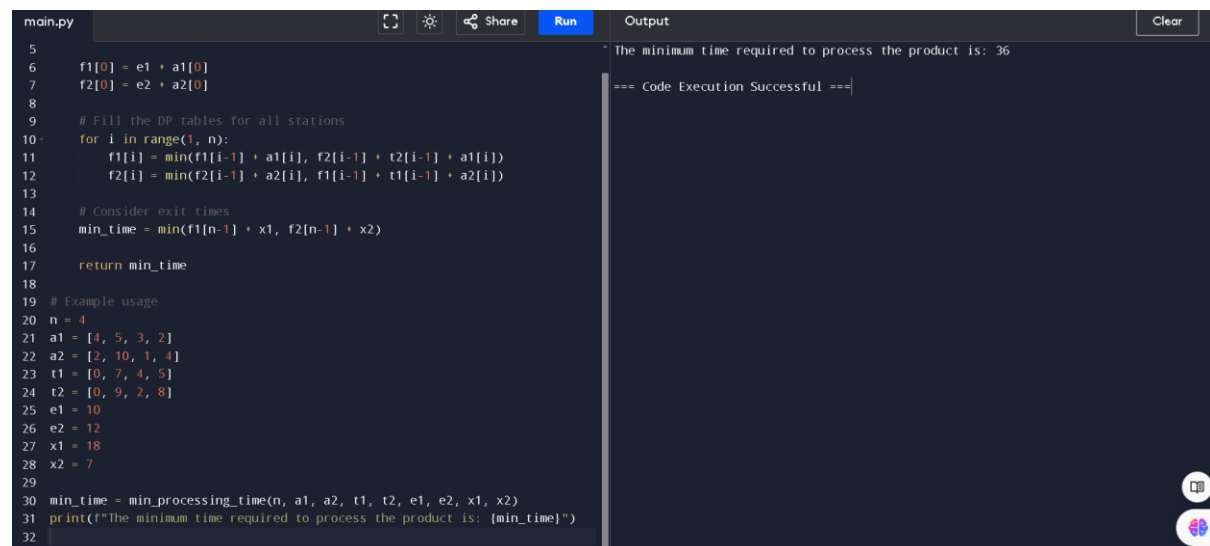
$e2$: Entry time to assembly line 2.

$x1$: Exit time from assembly line 1.

$x2$: Exit time from assembly line 2.

Output

The minimum time required to process the product.



```
main.py  Run  Output  Clear
5
6 f1[0] = e1 + a1[0]
7 f2[0] = e2 + a2[0]
8
9 # Fill the DP tables for all stations
10 for i in range(1, n):
11     f1[i] = min(f1[i-1] + a1[i], f2[i-1] + t2[i-1] + a1[i])
12     f2[i] = min(f2[i-1] + a2[i], f1[i-1] + t1[i-1] + a2[i])
13
14 # Consider exit times
15 min_time = min(f1[n-1] + x1, f2[n-1] + x2)
16
17 return min_time
18
19 # Example usage
20 n = 4
21 a1 = [4, 5, 3, 2]
22 a2 = [2, 10, 1, 4]
23 t1 = [0, 7, 4, 5]
24 t2 = [0, 9, 2, 8]
25 e1 = 10
26 e2 = 12
27 x1 = 18
28 x2 = 7
29
30 min_time = min_processing_time(n, a1, a2, t1, t2, e1, e2, x1, x2)
31 print(f"The minimum time required to process the product is: {min_time}")
32
```

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).

```
main.py  Run  Output  Clear  
36  
37 # Compute the minimum production time across all lines at the last  
   station  
38 min_time = float('inf')  
39 for line in range(num_lines):  
40     min_time = min(min_time, dp[line][-1])  
41  
42     return min_time  
43  
44 # Define the station times and transfer times  
45 station_times = [  
46     [5, 9, 3], # Line 1  
47     [6, 8, 4], # Line 2  
48     [7, 6, 5] # Line 3  
49 ]  
50  
51 transfer_times = [  
52     [0, 2, 3], # Transfer times from Line 1  
53     [2, 0, 4], # Transfer times from Line 2  
54     [3, 4, 0] # Transfer times from Line 3  
55 ]  
56  
57 dependencies = [(0, 1), (1, 2)] # Dependencies between stations  
58  
59 # Calculate minimum production time  
60 min_time = min_production_time(station_times, transfer_times, dependencies)  
61 print(f"Minimum production time: {min_time}")  
62  
Minimum production time: 17.0  
=== Code Execution Successful ===
```

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  Run  Output  Clear
1 #include <stdio.h>
2 #include <limits.h>
3
4 #define V 4 // Number of nodes
5
6 // Function to find the minimum path using TSP
7 int tsp(int graph[][V], int pos, int visited, int dp[][1 << V]) {
8     if (visited == (1 << V) - 1) {
9         return graph[pos][0]; // Return to the starting point
10    }
11
12    if (dp[pos][visited] != -1) {
13        return dp[pos][visited];
14    }
15
16    int minCost = INT_MAX;
17
18    for (int city = 0; city < V; city++) {
19        if ((visited & (1 << city)) == 0) {
20            int newCost = graph[pos][city] + tsp(graph, city, visited | (1 << city), dp);
21            minCost = newCost < minCost ? newCost : minCost;
22        }
23    }
24
25    return dp[pos][visited] = minCost;
26 }
27
28 int main() {
29     // Test Case 1
30     int graph1[4][4] = {
31         {0, 10, 15, 20},
32         {10, 0, 35, 25},
33         {15, 35, 0, 30},
34         {20, 25, 30, 0}
35     };
36     printf("Minimum path distance for graph1: %d\n", tsp(graph1, 0, 1, dp1));
37
38     // Test Case 2
39     int graph2[4][4] = {
40         {0, 10, 10, 10},
41         {10, 0, 10, 10},
42         {10, 10, 0, 10},
43         {10, 10, 10, 0}
44     };
45     printf("Minimum path distance for graph2: %d\n", tsp(graph2, 0, 1, dp2));
46
47     // Test Case 3
48     int graph3[4][4] = {
49         {0, 1, 2, 3},
50         {1, 0, 4, 5},
51         {2, 4, 0, 6},
52         {3, 5, 6, 0}
53     };
54     printf("Minimum path distance for graph3: %d\n", tsp(graph3, 0, 1, dp3));
55
56     return 0;
57 }
```

Output

```
/tmp/434XpS9Jmox.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

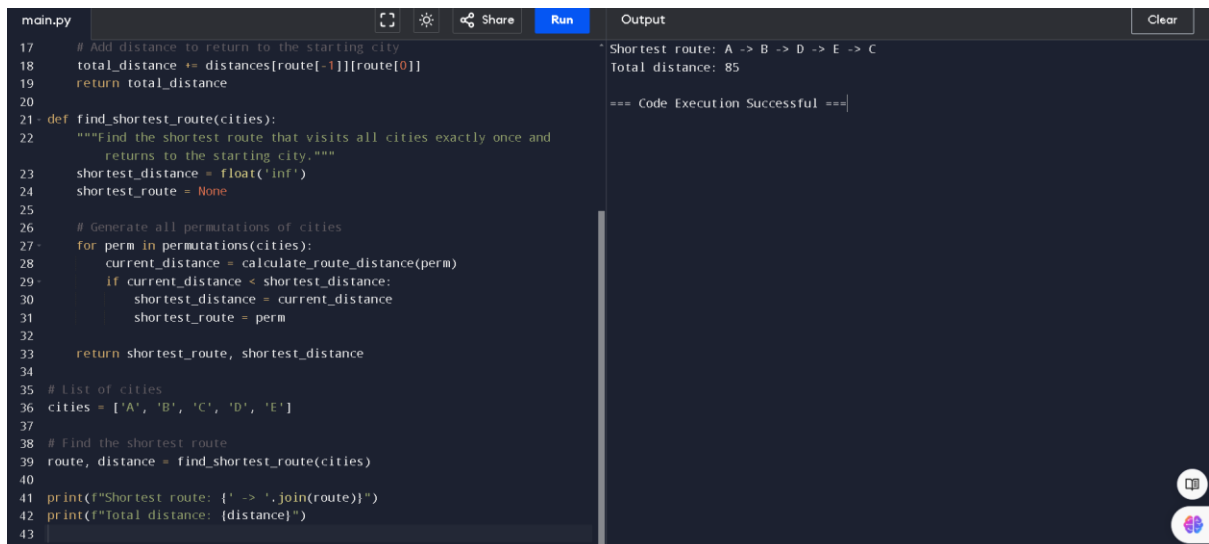
1. Symmetric Distances

- **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.\



```
main.py  Run  Output  Clear
17 # Add distance to return to the starting city
18 total_distance += distances[route[-1]][route[0]]
19 return total_distance
20
21 def find_shortest_route(cities):
22     """Find the shortest route that visits all cities exactly once and
23     returns to the starting city."""
24     shortest_distance = float('inf')
25     shortest_route = None
26
27     # Generate all permutations of cities
28     for perm in permutations(cities):
29         current_distance = calculate_route_distance(perm)
30         if current_distance < shortest_distance:
31             shortest_distance = current_distance
32             shortest_route = perm
33
34     return shortest_route, shortest_distance
35
36 # List of cities
37 cities = ['A', 'B', 'C', 'D', 'E']
38
39 # Find the shortest route
40 route, distance = find_shortest_route(cities)
41 print(f"Shortest route: {' -> '.join(route)}")
42 print(f"Total distance: {distance}")
43
```

Shortest route: A -> B -> D -> E -> C
Total distance: 85
=== Code Execution Successful ===

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* = "cbabd"

Output: "bb"

Constraints: • $1 \leq s.length \leq 1000$ • *s* consist of only digits and English letters.

```
main.py  Run  Output  Clear
2- if not s or len(s) == 1:
3-     return s
4-
5- start, end = 0, 0
6-
7- def expand_around_center(left: int, right: int) -> (int, int):
8-     while left >= 0 and right < len(s) and s[left] == s[right]:
9-         left -= 1
10-        right += 1
11-    return left + 1, right - 1
12-
13- for i in range(len(s)):
14-     # Expand around center i (odd length palindrome)
15-     l1, r1 = expand_around_center(i, i)
16-     # Expand around center i and i+1 (even length palindrome)
17-     l2, r2 = expand_around_center(i, i + 1)
18-
19-     if r1 - l1 > end - start:
20-         start, end = l1, r1
21-     if r2 - l2 > end - start:
22-         start, end = l2, r2
23-
24- return s[start:end + 1]
25-
26- # Examples
27- print(longest_palindromic_substring("babad")) # Output: "bab" or "aba"
28- print(longest_palindromic_substring("cbabd")) # Output: "bb"
29-
```

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

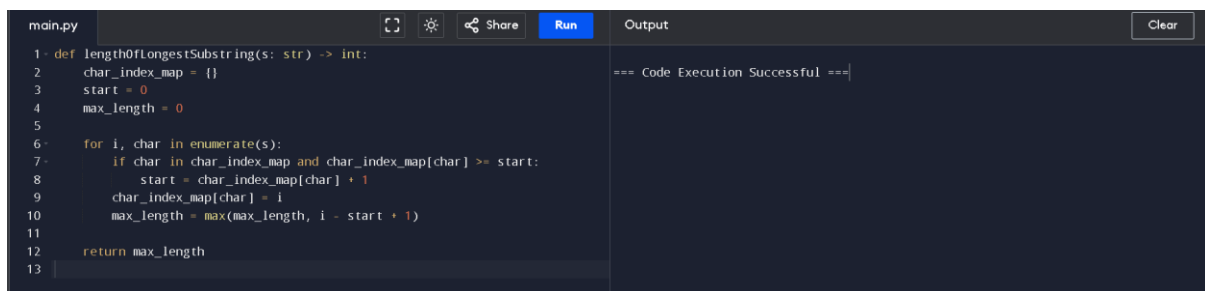
Explanation: The answer is "b", with the length of 1.

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: • $0 \leq s.length \leq 5 * 10^4$ • `s` consists of English letters, digits, symbols and spaces.



```
main.py  [Icons]  Share  Run  Output  Clear

1 def lengthOfLongestSubstring(s: str) -> int:
2     char_index_map = {}
3     start = 0
4     max_length = 0
5
6     for i, char in enumerate(s):
7         if char in char_index_map and char_index_map[char] >= start:
8             start = char_index_map[char] + 1
9         char_index_map[char] = i
10        max_length = max(max_length, i - start + 1)
11
12    return max_length
13

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

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

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

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

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

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

Example 3:

Output: false

main.py

Share

Run

Output

Clear

```
1- def wordBreak(s, wordDict):
2     dp = [False] * (len(s) + 1)
3     dp[0] = True # empty string can be segmented
4
5     for i in range(1, len(s) + 1):
6         for j in range(i):
7             if dp[j] and s[j:i] in wordDict:
8                 dp[i] = True
9                 break
10
11     return dp[len(s)]
12
13- # Example usage:
14 s1 = "leetcode"
15 wordDict1 = ["leet", "code"]
16 print(wordBreak(s1, wordDict1)) # Output: True
17
18 s2 = "applepenapple"
19 wordDict2 = ["apple", "pen"]
20 print(wordBreak(s2, wordDict2)) # Output: True
21
```

True
True

=== Code Execution Successful ===

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  Run  Output  Clear
1- def word_break(s, word_dict):
2-     n = len(s)
3-     dp = [False] * (n + 1)
4-     dp[0] = True
5-
6-     for i in range(1, n + 1):
7-         for j in range(i):
8-             if dp[j] and s[j:i] in word_dict:
9-                 dp[i] = True
10-                break
11-
12-     return dp[n]
13-
14- # Test the function with the provided examples
15- word_dict = ["i", "like", "sam", "sung", "samsung", "mobile", "ice", "cream",
16-             "icecream", "man", "go", "mango"]
17-
18- # Example 1
19- s1 = "ilike"
20- print("Yes" if word_break(s1, word_dict) else "No") # Output: Yes
21-
22- # Example 2
23- s2 = "ilikesamsung"
24- print("Yes" if word_break(s2, word_dict) else "No") # Output: Yes
25-
26-
27-
28-
29-
30-
31-
32-
33-
34-
35-
36-
37-
38-
39-
40-
41-
42-
43-
44-
45-
46-
47-
48-
49-
50-
51-
52-
53-
54-
55-
56-
57-
58-
59-
60-
61-
62-
63-
64-
65-
66-
67-
68-
69-
70-
71-
72-
73-
74-
75-
76-
77-
78-
79-
80-
81-
82-
83-
84-
85-
86-
87-
88-
89-
90-
91-
92-
93-
94-
95-
96-
97-
98-
99-
100-
101-
102-
103-
104-
105-
106-
107-
108-
109-
110-
111-
112-
113-
114-
115-
116-
117-
118-
119-
120-
121-
122-
123-
124-
125-
126-
127-
128-
129-
130-
131-
132-
133-
134-
135-
136-
137-
138-
139-
140-
141-
142-
143-
144-
145-
146-
147-
148-
149-
150-
151-
152-
153-
154-
155-
156-
157-
158-
159-
160-
161-
162-
163-
164-
165-
166-
167-
168-
169-
170-
171-
172-
173-
174-
175-
176-
177-
178-
179-
180-
181-
182-
183-
184-
185-
186-
187-
188-
189-
190-
191-
192-
193-
194-
195-
196-
197-
198-
199-
200-
201-
202-
203-
204-
205-
206-
207-
208-
209-
210-
211-
212-
213-
214-
215-
216-
217-
218-
219-
220-
221-
222-
223-
224-
225-
226-
227-
228-
229-
230-
231-
232-
233-
234-
235-
236-
237-
238-
239-
240-
241-
242-
243-
244-
245-
246-
247-
248-
249-
250-
251-
252-
253-
254-
255-
256-
257-
258-
259-
260-
261-
262-
263-
264-
265-
266-
267-
268-
269-
270-
271-
272-
273-
274-
275-
276-
277-
278-
279-
280-
281-
282-
283-
284-
285-
286-
287-
288-
289-
290-
291-
292-
293-
294-
295-
296-
297-
298-
299-
300-
301-
302-
303-
304-
305-
306-
307-
308-
309-
310-
311-
312-
313-
314-
315-
316-
317-
318-
319-
320-
321-
322-
323-
324-
325-
326-
327-
328-
329-
330-
331-
332-
333-
334-
335-
336-
337-
338-
339-
340-
341-
342-
343-
344-
345-
346-
347-
348-
349-
350-
351-
352-
353-
354-
355-
356-
357-
358-
359-
360-
361-
362-
363-
364-
365-
366-
367-
368-
369-
370-
371-
372-
373-
374-
375-
376-
377-
378-
379-
380-
381-
382-
383-
384-
385-
386-
387-
388-
389-
390-
391-
392-
393-
394-
395-
396-
397-
398-
399-
400-
401-
402-
403-
404-
405-
406-
407-
408-
409-
410-
411-
412-
413-
414-
415-
416-
417-
418-
419-
420-
421-
422-
423-
424-
425-
426-
427-
428-
429-
430-
431-
432-
433-
434-
435-
436-
437-
438-
439-
440-
441-
442-
443-
444-
445-
446-
447-
448-
449-
450-
451-
452-
453-
454-
455-
456-
457-
458-
459-
460-
461-
462-
463-
464-
465-
466-
467-
468-
469-
470-
471-
472-
473-
474-
475-
476-
477-
478-
479-
480-
481-
482-
483-
484-
485-
486-
487-
488-
489-
490-
491-
492-
493-
494-
495-
496-
497-
498-
499-
500-
501-
502-
503-
504-
505-
506-
507-
508-
509-
510-
511-
512-
513-
514-
515-
516-
517-
518-
519-
520-
521-
522-
523-
524-
525-
526-
527-
528-
529-
530-
531-
532-
533-
534-
535-
536-
537-
538-
539-
540-
541-
542-
543-
544-
545-
546-
547-
548-
549-
550-
551-
552-
553-
554-
555-
556-
557-
558-
559-
560-
561-
562-
563-
564-
565-
566-
567-
568-
569-
570-
571-
572-
573-
574-
575-
576-
577-
578-
579-
580-
581-
582-
583-
584-
585-
586-
587-
588-
589-
590-
591-
592-
593-
594-
595-
596-
597-
598-
599-
600-
601-
602-
603-
604-
605-
606-
607-
608-
609-
610-
611-
612-
613-
614-
615-
616-
617-
618-
619-
620-
621-
622-
623-
624-
625-
626-
627-
628-
629-
630-
631-
632-
633-
634-
635-
636-
637-
638-
639-
640-
641-
642-
643-
644-
645-
646-
647-
648-
649-
650-
651-
652-
653-
654-
655-
656-
657-
658-
659-
660-
661-
662-
663-
664-
665-
666-
667-
668-
669-
670-
671-
672-
673-
674-
675-
676-
677-
678-
679-
680-
681-
682-
683-
684-
685-
686-
687-
688-
689-
690-
691-
692-
693-
694-
695-
696-
697-
698-
699-
700-
701-
702-
703-
704-
705-
706-
707-
708-
709-
710-
711-
712-
713-
714-
715-
716-
717-
718-
719-
720-
721-
722-
723-
724-
725-
726-
727-
728-
729-
730-
731-
732-
733-
734-
735-
736-
737-
738-
739-
740-
741-
742-
743-
744-
745-
746-
747-
748-
749-
750-
751-
752-
753-
754-
755-
756-
757-
758-
759-
760-
761-
762-
763-
764-
765-
766-
767-
768-
769-
770-
771-
772-
773-
774-
775-
776-
777-
778-
779-
780-
781-
782-
783-
784-
785-
786-
787-
788-
789-
790-
791-
792-
793-
794-
795-
796-
797-
798-
799-
800-
801-
802-
803-
804-
805-
806-
807-
808-
809-
810-
811-
812-
813-
814-
815-
816-
817-
818-
819-
820-
821-
822-
823-
824-
825-
826-
827-
828-
829-
830-
831-
832-
833-
834-
835-
836-
837-
838-
839-
840-
841-
842-
843-
844-
845-
846-
847-
848-
849-
850-
851-
852-
853-
854-
855-
856-
857-
858-
859-
860-
861-
862-
863-
864-
865-
866-
867-
868-
869-
870-
871-
872-
873-
874-
875-
876-
877-
878-
879-
880-
881-
882-
883-
884-
885-
886-
887-
888-
889-
890-
891-
892-
893-
894-
895-
896-
897-
898-
899-
900-
901-
902-
903-
904-
905-
906-
907-
908-
909-
910-
911-
912-
913-
914-
915-
916-
917-
918-
919-
920-
921-
922-
923-
924-
925-
926-
927-
928-
929-
930-
931-
932-
933-
934-
935-
936-
937-
938-
939-
940-
941-
942-
943-
944-
945-
946-
947-
948-
949-
950-
951-
952-
953-
954-
955-
956-
957-
958-
959-
960-
961-
962-
963-
964-
965-
966-
967-
968-
969-
970-
971-
972-
973-
974-
975-
976-
977-
978-
979-
980-
981-
982-
983-
984-
985-
986-
987-
988-
989-
990-
991-
992-
993-
994-
995-
996-
997-
998-
999-
1000-
1001-
1002-
1003-
1004-
1005-
1006-
1007-
1008-
1009-
1010-
1011-
1012-
1013-
1014-
1015-
1016-
1017-
1018-
1019-
1020-
1021-
1022-
1023-
1024-
1025-
1026-
1027-
1028-
1029-
1030-
1031-
1032-
1033-
1034-
1035-
1036-
1037-
1038-
1039-
1040-
1041-
1042-
1043-
1044-
1045-
1046-
1047-
1048-
1049-
1050-
1051-
1052-
1053-
1054-
1055-
1056-
1057-
1058-
1059-
1060-
1061-
1062-
1063-
1064-
1065-
1066-
1067-
1068-
1069-
1070-
1071-
1072-
1073-
1074-
1075-
1076-
1077-
1078-
1079-
1080-
1081-
1082-
1083-
1084-
1085-
1086-
1087-
1088-
1089-
1090-
1091-
1092-
1093-
1094-
1095-
1096-
1097-
1098-
1099-
1100-
1101-
1102-
1103-
1104-
1105-
1106-
1107-
1108-
1109-
1110-
1111-
1112-
1113-
1114-
1115-
1116-
1117-
1118-
1119-
1120-
1121-
1122-
1123-
1124-
1125-
1126-
1127-
1128-
1129-
1130-
1131-
1132-
1133-
1134-
1135-
1136-
1137-
1138-
1139-
1140-
1141-
1142-
1143-
1144-
1145-
1146-
1147-
1148-
1149-
1150-
1151-
1152-
1153-
1154-
1155-
1156-
1157-
1158-
1159-
1160-
1161-
1162-
1163-
1164-
1165-
1166-
1167-
1168-
1169-
1170-
1171-
1172-
1173-
1174-
1175-
1176-
1177-
1178-
1179-
1180-
1181-
1182-
1183-
1184-
1185-
1186-
1187-
1188-
1189-
1190-
1191-
1192-
1193-
1194-
1195-
1196-
1197-
1198-
1199-
1200-
1201-
1202-
1203-
1204-
1205-
1206-
1207-
1208-
1209-
1210-
1211-
1212-
1213-
1214-
1215-
1216-
1217-
1218-
1219-
1220-
1221-
1222-
1223-
1224-
1225-
1226-
1227-
1228-
1229-
1230-
1231-
1232-
1233-
1234-
1235-
1236-
1237-
1238-
1239-
1240-
1241-
1242-
1243-
1244-
1245-
1246-
1247-
1248-
1249-
1250-
1251-
1252-
1253-
1254-
1255-
1256-
1257-
1258-
1259-
1260-
1261-
1262-
1263-
1264-
1265-
1266-
1267-
1268-
1269-
1270-
1271-
1272-
1273-
1274-
1275-
1276-
1277-
1278-
1279-
1280-
1281-
1282-
1283-
1284-
1285-
1286-
1287-
1288-
1289-
1290-
1291-
1292-
1293-
1294-
1295-
1296-
1297-
1298-
1299-
1300-
1301-
1302-
1303-
1304-
1305-
1306-
1307-
1308-
1309-
1310-
1311-
1312-
1313-
1314-
1315-
1316-
1317-
1318-
1319-
1320-
1321-
1322-
1323-
1324-
1325-
1326-
1327-
1328-
1329-
1330-
1331-
1332-
1333-
1334-
1335-
1336-
1337-
1338-
1339-
1340-
1341-
1342-
1343-
1344-
1345-
1346-
1347-
1348-
1349-
1350-
1351-
1352-
1353-
1354-
1355-
1356-
1357-
1358-
1359-
1360-
1361-
1362-
1363-
1364-
1365-
1366-
1367-
1368-
1369-
1370-
1371-
1372-
1373-
1374-
1375-
1376-
1377-
1378-
1379-
1380-
1381-
1382-
1383-
1384-
1385-
1386-
1387-
1388-
1389-
1390-
1391-
1392-
1393-
1394-
1395-
1396-
1397-
1398-
1399-
1400-
1401-
1402-
1403-
1404-
1405-
1406-
1407-
1408-
1409-
1410-
1411-
1412-
1413-
1414-
1415-
1416-
1417-
1418-
1419-
1420-
1421-
1422-
1423-
1424-
1425-
1426-
1427-
1428-
1429-
1430-
1431-
1432-
1433-
1434-
1435-
1436-
1437-
1438-
1439-
1440-
1441-
1442-
1443-
1444-
1445-
1446-
1447-
1448-
1449-
1450-
1451-
1452-
1453-
1454-
1455-
1456-
1457-
1458-
1459-
1460-
1461-
1462-
1463-
1464-
1465-
1466-
1467-
1468-
1469-
1470-
1471-
1472-
1473-
1474-
1475-
1476-
1477-
1478-
1479-
1480-
1481-
1482-
1483-
1484-
1485-
1486-
1487-
1488-
1489-
1490-
1491-
1492-
1493-
1494-
1495-
1496-
1497-
1498-
1499-
1500-
1501-
1502-
1503-
1504-
1505-
1506-
1507-
1508-
1509-
1510-
1511-
1512-
1513-
1514-
1515-
1516-
1517-
1518-
1519-
1520-
1521-
1522-
1523-
1524-
1525-
1526-
1527-
1528-
1529-
1530-
1531-
1532-
1533-
1534-
1535-
1536-
1537-
1538-
1539-
1540-
1541-
1542-
1543-
1544-
1545-
1546-
1547-
1548-
1549-
1550-
1551-
1552-
1553-
1554-
1555-
1556-
1557-
1558-
1559-
1560-
1561-
1562-
1563-
1564-
1565-
1566-
1567-
1568-
1569-
1570-
1571-
1572-
1573-
1574-
1575-
1576-
1577-
1578-
1579-
1580-
1581-
1582-
1583-
1584-
1585-
1586-
1587-
1588-
1589-
1590-
1591-
1592-
1593-
1594-
1595-
1596-
1597-
1598-
1599-
1600-
1601-
1602-
1603-
1604-
1605-
1606-
1607-
1608-
1609-
1610-
1611-
1612-
1613-
1614-
1615-
1616-
1617-
1618-
1619-
1620-
1621-
1622-
1623-
1624-
1625-
1626-
1627-
1628-
1629-
1630-
1631-
1632-
1633-
1634-
1635-
1636-
1637-
1638-
1639-
1640-
1641-
1642-
1643-
1644-
1645-
1646-
1647-
1648-
1649-
1650-
1651-
1652-
1653-
1654-
1655-
1656-
1657-
1658-
1659-
1660-
1661-
1662-
1663-
1664-
1665-
1666-
1667-
1668-
1669-
1670-
1671-
1672-
1673-
1674-
1675-
1676-
1677-
1678-
1679-
1680-
1681-
1682-
1683-
1684-
1685-
1686-
1687-
1688-
1689-
1690-
1691-
1692-
1693-
1694-
1695-
1696-
1697-
1698-
1699-
1700-
1701-
1702-
1703-
1704-
1705-
1706-
1707-
1708-
1709-
1710-
1711-
1712-
1713-
1714-
1715-
1716-
1717-
1718-
1719-
1720-
1721-
1722-
1723-
1724-
1725-
1726-
1727-
1728-
1729-
1730-
1731-
1732-
1733-
1734-
1735-
1736-
1737-
1738-
1739-
1740-
1741-
1742-
1743-
1744-
1745-
1746-
1747-
1748-
1749-
1750-
1751-
1752-
1753-
1754-
1755-
1756-
1757-
1758-
1759-
1760-
1761-
1762-
1763-
1764-
1765-
1766-
1767-
1768-
1769-
1770-
1771-
1772-
1773-
1774-
1775-
1776-
1777-
1778-
1779-
1780-
1781-
1782-
1783-
1784-
1785-
1786-
1787-
1788-
1789-
1790-
1791-
1792-
1793-
1794-
1795-
1796-
1797-
1798-
1799-
1800-
1801-
1802-
1803-
1804-
1805-
1806-
1807-
1808-
1809-
1810-
1811-
1812-
1813-
1814-
1815-
1816-
1817-
1818-
1819-
1820-
1821-
1822-
1823-
1824-
1825-
1826-
1827-
1828-
1829-
1830-
1831-
1832-
1833-
1834-
1835-
1836-
1837-
1838-
1839-
1840-
1841-
1842-
1843-
1844-
1845-
1846-
1847-
1848-
1849-
1850-
1851-
1852-
1853-
1854-
1855-
1856-
1857-
1858-
1859-
1860-
1861-
1862-
1863-
1864-
1865-
1866-
1867-
1868-
1869-
1870-
1871-
1872-
1873-
1874-
1875-
1876-
1877-
1878-
1879-
1880-
1881-
1882-
1883-
1884-
1885-
1886-
1887-
1888-
1889-
1890-
1891-
1892-
1893-
1894-
1895-
1896-
1897-
1898-
1899-
1900-
1901-
1902-
1903-
1904-
1905-
1906-
1907-
1908-
1909-
1910-
1911-
1912-
1913-
1914-
1915-
1916-
1917-
1918-
1919-
1920-
1921-
1922-
1923-
1924-
1925-
1926-
1927-
1928-
1929-
1930-
1931-
1932-
1933-
1934-
1935-
1936-
1937-
1938-
1939-
1940-
1941-
1942-
1943-
1944-
1945-
1946-
1947-
1948-
1949-
1950-
1951-
1952-
1953-
1954-
1955-
1956-
1957-
1958-
1959-
1960-
1961-
1962-
1963-
1964-
1965-
1966-
1967-
1968-
1969-
1970-
1971-
1972-
1973-
1974-
1975-
1976-
1977-
1978-
1979-
1980-
1981-
1982-
1983-
1984-
1985-
1986-
1987-
1988-
1989-
1990-
1991-
1992-
1993-
1994-
1995-
1996-
1997-
1998-
1999-
2000-
2001-
2002-
2003-
2004-
2005-
2006-
2007-
2008-
2009-
2010-
2011-
2012-
2013-
2014-
2015-
2016-
2017-
2018-
2019-
2020-
2021-
2022-
2023-
2024-
2025-
2026-
2027-
2028-
2029-
2030-
2031-
2032-
2033-
2034-
2035-
2036-
2037-
2038-
2039-
2040-
2041-
2042-
2043-
2044-
2045-
2046-
2047-
2048-
2049-
2050-
2051-
2052-
2053-
2054-
2055-
2056-
2057-
2058-
2059-
2060-
2061-
2062-
2063-
2064-
2065-
2066-
2067-
2068-
2069-
2070-
2071-
2072-
2073-
2074-
2075-
2076-
2077-
2078-
2079-
2080-
2081-
2082-
2083-
2084-
2085-
2086-
2087-
2088-
2089-
2090-
2091-
2092-
2093-
2094-
2095-
2096-
2097-
2098-
2099-
2100-
2101-
2102-
2103-
2104-
2105-
2106-
2107-
2108-
2109-
2110-
2111-
2112-
2113-
2114-
2115-
2116-
2117-
2118-
2119-
2120-
2121-
2122-
2123-
2124-
2125-
2126-
2127-
2128-
2129-
2130-
2131-
2132-
2133-
2134-
2135-
2136-
2137-
2138-
2139-
2140-
2141-
2142-
2143-
2144-
2145-
2146-
2147-
2148-
2149-
2150-
2151-
2152-
2153-
2154-
2155-
2156-
2157-
2158-
2159-
21
```

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:

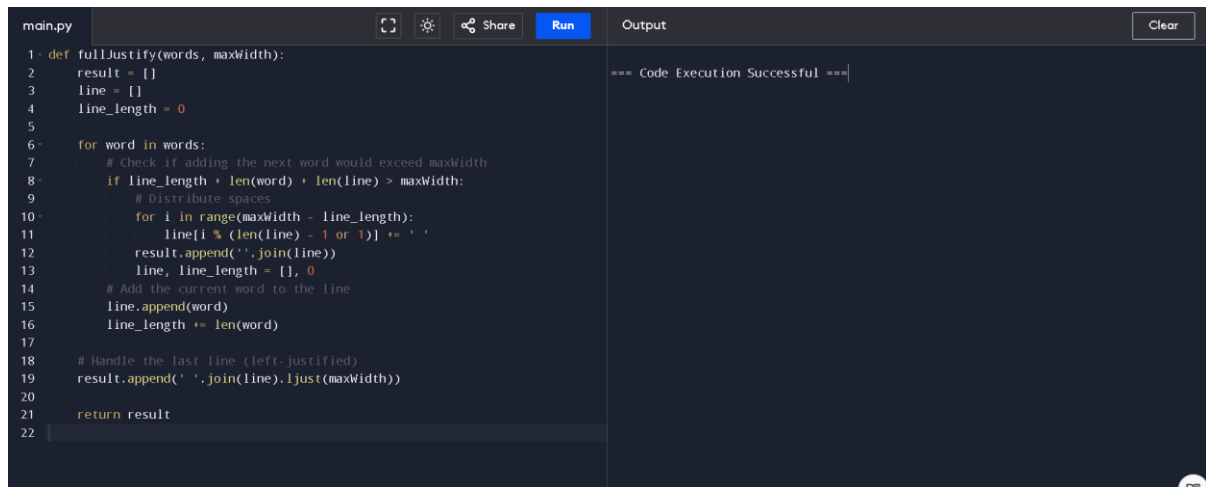
Input: `words = ["What","must","be","acknowledgment","shall","be"],` `maxWidth = 16`

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.



The screenshot shows a code editor with a file named 'main.py'. The code defines a function 'fullJustify(words, maxWidth)' that takes a list of words and a maximum width. It iterates through the words, building lines that are fully justified (spaces distributed evenly) until the next word would exceed the maximum width. The last line is left-justified. The output panel shows '=== Code Execution Successful ==='.

```
1 def fullJustify(words, maxWidth):
2     result = []
3     line = []
4     line_length = 0
5
6     for word in words:
7         # Check if adding the next word would exceed maxWidth
8         if line_length + len(word) + len(line) > maxWidth:
9             # Distribute spaces
10            for i in range(maxWidth - line_length):
11                line[i % (len(line) - 1 or 1)] += ' '
12            result.append(' '.join(line))
13            line, line_length = [], 0
14            # Add the current word to the line
15            line.append(word)
16            line_length += len(word)
17
18    # Handle the last line (left-justified)
19    result.append(' '.join(line).ljust(maxWidth))
20
21    return result
22
```

Output: === Code Execution Successful ===

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"]]
```

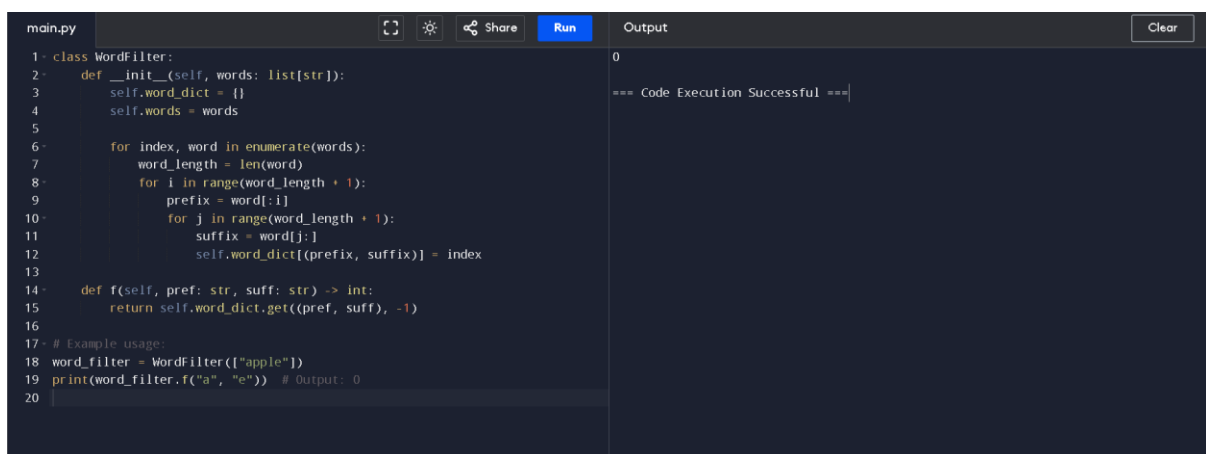
Output

```
[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  Run  Output  Clear
1 class WordFilter:
2     def __init__(self, words: list[str]):
3         self.word_dict = {}
4         self.words = words
5
6         for index, word in enumerate(words):
7             word_length = len(word)
8             for i in range(word_length + 1):
9                 prefix = word[:i]
10                for j in range(word_length + 1):
11                    suffix = word[j:]
12                    self.word_dict[(prefix, suffix)] = index
13
14    def f(self, pref: str, suff: str) -> int:
15        return self.word_dict.get((pref, suff), -1)
16
17 # Example usage:
18 word_filter = WordFilter(["apple"])
19 print(word_filter.f("a", "e")) # Output: 0
20
```

0

=== Code Execution Successful ===