74.4

Muhammad Tanveer

Software Engineer

PDF generated at: 25 Jun 2024 02:27:11 UTC

View this report on HackerRank

Score

74.4% • 160 / 215

scored in CodePath TIP103: Unit 3 Assessment - Summer 2024 in 76 min 48 sec on 24 Jun 2024 18:08:36 PDT

Candidate Information

Email	tanveerm176@gmail.com
Test	CodePath TIP103: Unit 3 Assessment - Summer 2024
Candidate Packet	View
Taken on	24 Jun 2024 18:08:36 PDT
Time taken	76 min 48 sec/ 90 min
Invited by	CodePath

Skill Distribution

No.	Skill	Score
1	Problem Solving Basic	72%

Tags Distribution

Easy	72%	Problem Solving	72%
Algorithms	63%	Flood Fill	90%
Theme: E-commerce	45%	Interviewer Guidelines	45%
Arrays	50%	Strings	100%
Data Structures	100%		

Questions

Status	No.	Question	Time Taken	Skill	Score
✓	1	Bucket Fill Coding	25 min 39 sec	Problem Solving (Basic)	45/50
✗	2	Keyboard Coding	18 min 17 sec	Problem Solving (Basic)	0/50
✓	3	Last and Second-Last Coding	8 min 24 sec	Problem Solving (Basic)	50/50
✓	4	Can you store multiple keys with the same value in a hashmap? Multiple Choice	2 min 6 sec	-	5/5

✓	5	True or False: Hashmaps Multiple Choice	1 min 15 sec	-	5/5
✓	6	True or False: Heaps Multiple Choice	14 sec	-	5/5
✓	7	Rod Cutting Coding	20 min 22 sec	Problem Solving (Basic)	50/50

1. Bucket Fill

✓ Partially correct

Coding Easy Problem Solving Algorithms Flood Fill Theme: E-commerce Interviewer Guidelines

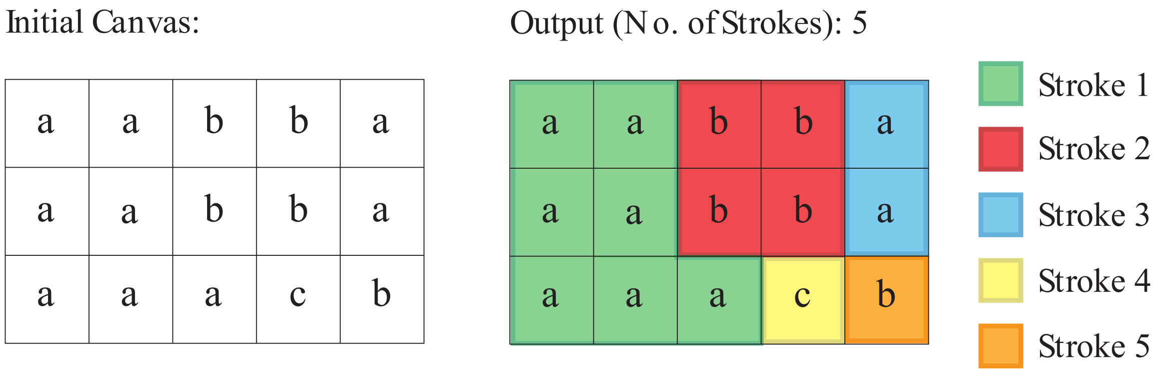
Question description

Digital graphics tools often make available a "bucket fill" tool that will only paint adjacent cells . In one *fill,* a modified bucket tool recolors adjacent cells (connected horizontally or vertically but not diagonally) that have the same color. Given a picture represented as a 2-dimensional array of letters representing colors, find the minimum number of fills to completely repaint the picture.

Example

picture= ["aabba", "aabba", "aaacb"]

Each string represents a row of the picture and each letter represents a cell's color. The diagram below shows the *5* fills needed to repaint the picture. It takes two fills each for *a* and *b*, and one for *c.* The array *picture* is shown below.



Initial Canvas:

a	a	b	b	a
a	a	b	b	a
a	a	a	c	b

Output (No. of Strokes): 5

🟩 a	🟩 a	🟥 b	🟥 b	🟦 a
🟩 a	🟩 a	🟥 b	🟥 b	🟦 a
🟩 a	🟩 a	🟩 a	🟨 c	🟧 b

Function Description

Complete the function *strokesRequired* in the editor below.

strokesRequired has the following parameter(s):

string picture[h]: an array of strings where each string represents one row of the picture to be painted

Output:

int: the minimum number of fills required to repaint the picture

Constraints

- *h* and *w* refer to height and width of the graph.
- $1 \le h \le 10^5$
- $1 \le w \le 10^5$
- $1 \le h*w \le 10^5$
- *length(picture[i]) = w (where $0 \le i < h$)*
- *picture[i][j]* is in the set *{'a', 'b', 'c'}* (where $0 \le i < h$ and $0 \le j < w$)

▼ INPUT FORMAT FOR CUSTOM TESTING

The first line contains an integer, *h*, that denotes the height of the picture and the number of elements in *picture*.

Each line *i* of the *h* subsequent lines (where $0 \le i < h$) contains a string that describes *picture[i]*.

▼ SAMPLE CASE 0

Sample Input For Custom Testing

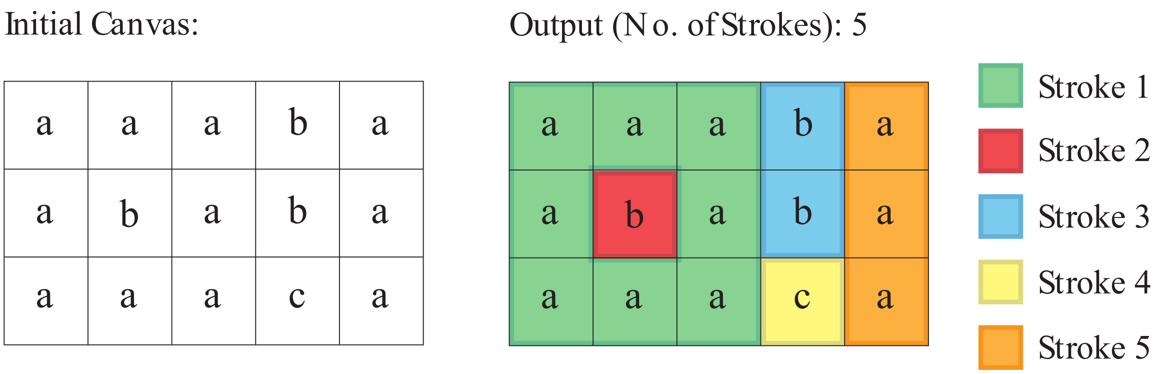
```
STDIN     Function
-----     --------
```

```
3     →   picture[] size h = 3
aaaba →   picture = [ "aaaba" , "ababa" , "aaaca" ]
ababa
aaaca
```

Sample Output

```
5
```

Explanation



Initial Canvas:

a	a	a	b	a
a	b	a	b	a
a	a	a	c	a

Output (N o. of Strokes): 5

a	a	a	b	a
a	b	a	b	a
a	a	a	c	a

Letter *a* takes *2* fills, *b* takes *2* fills and *c* takes *1* fill for a total of *5*.

▼ SAMPLE CASE 1

Sample Input For Custom Testing

```
STDIN     Function
-----     --------
4     →   picture[] size h = 4
bbba  →   picture = [ "bbba", "abba", "acaa" , "aaac" ]
abba
acaa
aaac
```

Sample Output

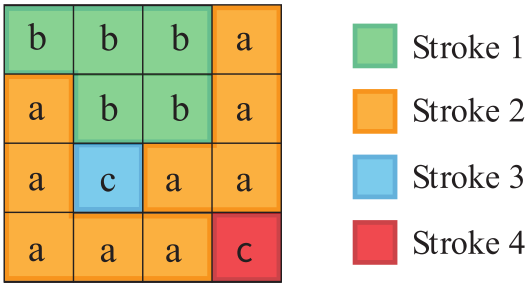
```
4
```

Explanation

Initial Canvas:

b	b	b	a
a	b	b	a
a	c	a	a
a	a	a	c

Output (No. of Strokes): 4



Letters *a* and *b* each take *1* fill and letter *c* takes *2* fills.

Interviewer guidelines

▼ SOLUTION

Skills: Problem solving, Depth-first search

Optimal Solution:

Create a bidirectional graph of connected cells. To do this, work from (0, 0) to the right and down, always checking connections with cells east and south. Once this is done, execute a depth-first search from a random location. When the search runs out, increment the group count. Pick one of the remaining unvisited cells and repeat the process. Do this until there are no cells left to explore.

```
def strokesRequired(picture):
    G = dict()
    rows, columns = len(picture), len(picture[0])
    # initialize a dictionary to hold arrays of tuples (row, column)
    for r in range(rows):
        for c in range(columns):
            G[(r, c)] = []
    # initialize a bidirectional graph of connected nodes
    for r in range(rows):
        for c in range(columns):
            if c+1 < columns:
                # check east
                if (picture[r][c+1] == picture[r][c]):
                    G[(r, c)].append((r, c+1))
                    G[(r, c+1)].append((r, c))
            if r+1 < rows:
                # check south
                if (picture[r+1][c] == picture[r][c]):
                    G[(r, c)].append((r+1, c))
                    G[(r+1, c)].append((r, c))
    # now that the graph is assembled, DFS to determine number
    # of groups, this one is iterative
    visited = set()
    groups = 0
    for k in G:
        if k in visited:
            continue
        Q = [k]
        while Q:
            node = Q.pop()
            if node in visited:
                continue
            visited.add(node)
            Q.extend(G[node])
        groups += 1
    return groups
```

▼ COMPLEXITY ANALYSIS

Time Complexity - O(n+e)

Each of the *n* cells is visited along each of the *e* edges.

Space Complexity - O(n+e)

The tree is formed with *n* nodes (cells) and *e* connections. Store each node with its associated edges.

Candidate's Solution

Language used: **Python 3**

```
#!/bin/python3

import math
import os
import random
import re
import sys


#
```

```python
# Complete the 'strokesRequired' function below.
#
# The function is expected to return an INTEGER.
# The function accepts STRING_ARRAY picture as parameter.
#

def flood_fill(picture, visited, i, j, color):
    rows, cols = len(picture), len(picture[0])

    if i < 0 or i >= rows or j < 0 or j >= cols:
        return

    if visited[i][j] or picture[i][j] != color:
        return

    visited[i][j] = True

    flood_fill(picture, visited,i+1, j, color)
    flood_fill(picture, visited,i-1, j, color)
    flood_fill(picture, visited,i, j+1, color)
    flood_fill(picture, visited,i, j-1, color)

def strokesRequired(picture):
    # Write your code here
    if not picture or not picture[0]:
        return 0

    rows, cols = len(picture), len(picture[0])
    visited = [[False for _ in range(cols)] for _ in range(rows)]

    fill_count = 0

    for i in range(rows):
        for j in range(cols):
            if not visited[i][j]:
                fill_count += 1
                flood_fill(picture, visited, i, j, picture[i][j])

    return fill_count



if __name__ == '__main__':
    fptr = open(os.environ['OUTPUT_PATH'], 'w')

    picture_count = int(input().strip())

    picture = []

    for _ in range(picture_count):
        picture_item = input()
        picture.append(picture_item)

    result = strokesRequired(picture)

    fptr.write(str(result) + '\n')

    fptr.close()

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Test Case 0	Easy	Sample	Success	1	0.0327 sec	10.4 KB
Test Case 1	Easy	Sample	Success	1	0.0269 sec	10.3 KB
Test Case 2	Easy	Sample	Success	1	0.0338 sec	10.4 KB
Test Case 3	Easy	Sample	Success	4	0.0307 sec	10.3 KB
Test Case 4	Easy	Hidden	Success	4	0.0293 sec	10.4 KB
Test Case 5	Easy	Sample	Success	4	0.0351 sec	10.4 KB
Test Case 6	Easy	Hidden	Success	5	0.0365	10.4 KB

					sec	
Test Case 7	Easy	Hidden	Success	5	0.0411 sec	10.4 KB
Test Case 8	Easy	Hidden	Success	5	0.0749 sec	10.8 KB
Test Case 9	Easy	Hidden	Success	5	0.0423 sec	10.5 KB
Test Case 10	Easy	Hidden	Success	5	0.0452 sec	10.5 KB
Test Case 11	Easy	Hidden	Success	5	0.1041 sec	10.8 KB
Test Case 12	Easy	Hidden	Runtime Error	0	0.1067 sec	12 KB

ⓘ No comments.

2. Keyboard

⊗ Incorrect

Coding Easy Algorithms Problem Solving Theme: E-commerce Arrays Interviewer Guidelines

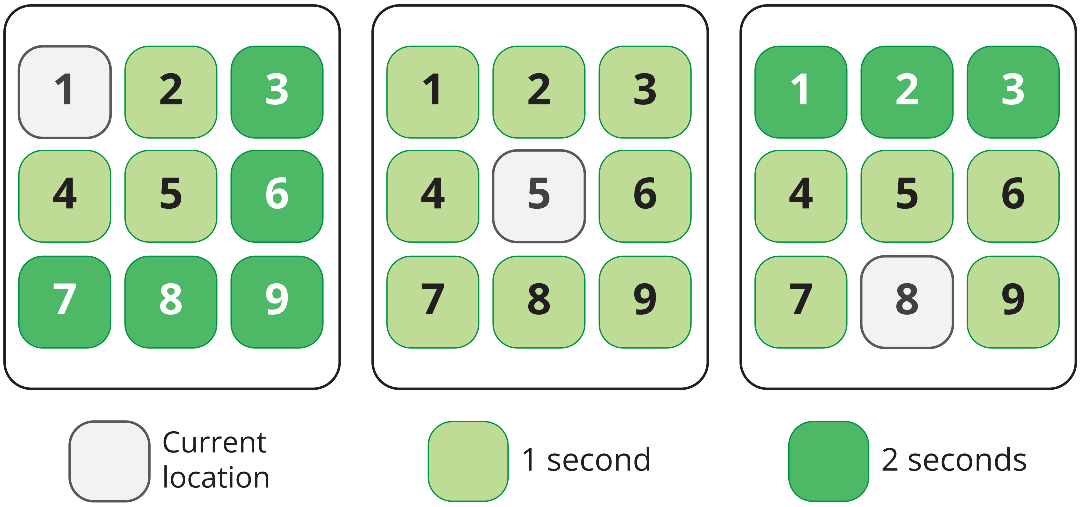
Question description

Each day, to enter their building, employees of an e-commerce company have to type a string of numbers into a console using a 3×3 numeric keypad. Every day, the numbers on the keypad are mixed up.

Use the following rules to calculate the total amount of time it takes to type a string:

- It takes *0* seconds to move their finger to the first key, and it takes *0* seconds to press the key where their finger is located any number of times.
- They can move their finger from one location to any adjacent key in one second. Adjacent keys include those on a diagonal.
- Moving to a non-adjacent key is done as a series of moves to adjacent keys.

Example



This diagram depicts the minimum amount of time it takes to move from the current location to all other locations on the keypad.

Function Description

Complete the function *entryTime* in the editor below.

entryTime has the following parameter(s):

string s: the string to type

string keypad: a string of *9* digits where each group of *3* digits represents a row on the keypad of the day, in order

Returns:

int : integer denoting the minimum amount of time it takes to type the string *s*.

Constraints

- $1 \le |s| \le 10^5$
- $|keypad| = 9$

- *keypad[i] ∈ [1-9]*

▼ INPUT FORMAT FOR CUSTOM TESTING

Input from stdin will be processed as follows and passed to the function.

The first line contains a string *s*.
The next line contains a string *keypad*.

▼ SAMPLE CASE 0

Sample Input 0

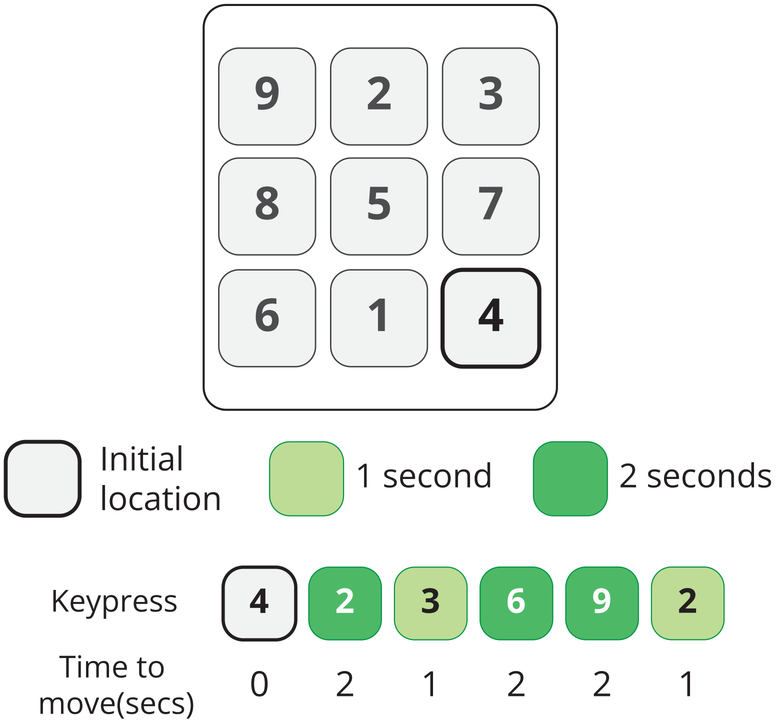
```
STDIN       Function Parameters
-----       -------------------
423692    →    string s = "423692"
923857614 →    string keypad = "923857614"
```

Sample Output 0

```
8
```

Explanation 0

The keypad looks like this:



Time to type 423692 = 0+2+1+2+2+1 = 8 seconds

Calculate the time it takes to type *s = 423692* as follows:

- *4*: Start here, so it takes *0* seconds.
- *2*: It takes *2* seconds to move from *4 → 2*
- *3*: It takes *1* second to move from *2 → 3*
- *6*: It takes *2* seconds to move from *3 → 6*
- *9*: It takes *2* seconds to move from *6 → 9*
- *2*: It takes *1* second to move from *9 → 2*

The total time is *2 + 1 + 2 + 2 + 1 = 8*.

▼ SAMPLE CASE 1

Sample Input 1

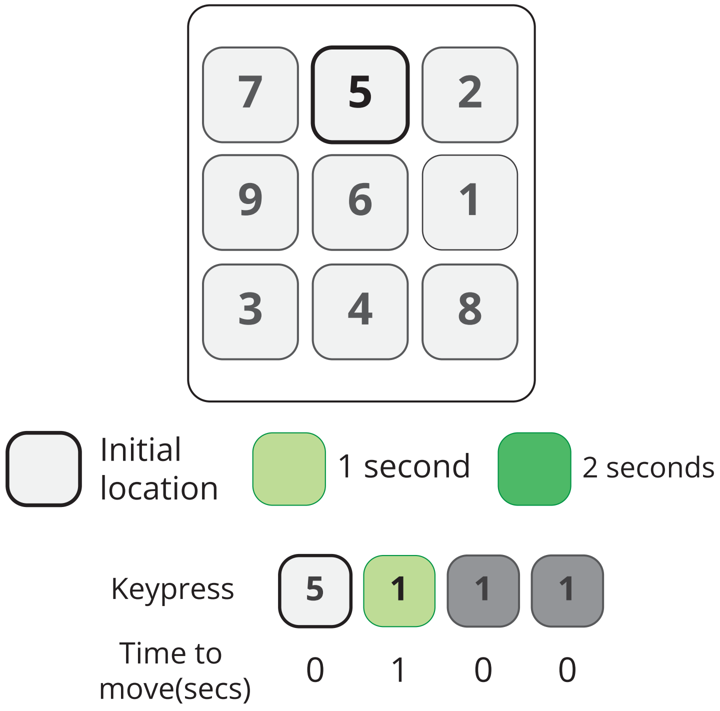
```
STDIN       Function Parameters
-----       -------------------
5111      →   string s = "5111"
752961348 →   string keypad = "752961348"
```

Sample Output 1

```
1
```

Explanation 1

The keypad looks like this:



Time to type 5111 = 0+1+0+0 = 1 second

Calculate the time it takes to type *s = 5111* as follows:

- *5*: Start here, so it takes *0* seconds and *totalTime = 0*.
- *1*: It takes *1* seconds to move from *5 → 1*
- *1*: It takes *0* seconds to move from *1 → 1*
- *1*: It takes *0* seconds to move from *1 → 1*

The total time is *1*.

▼ SAMPLE CASE 2

Sample Input 2

```
STDIN       Function Parameters
-----     -------------------
```

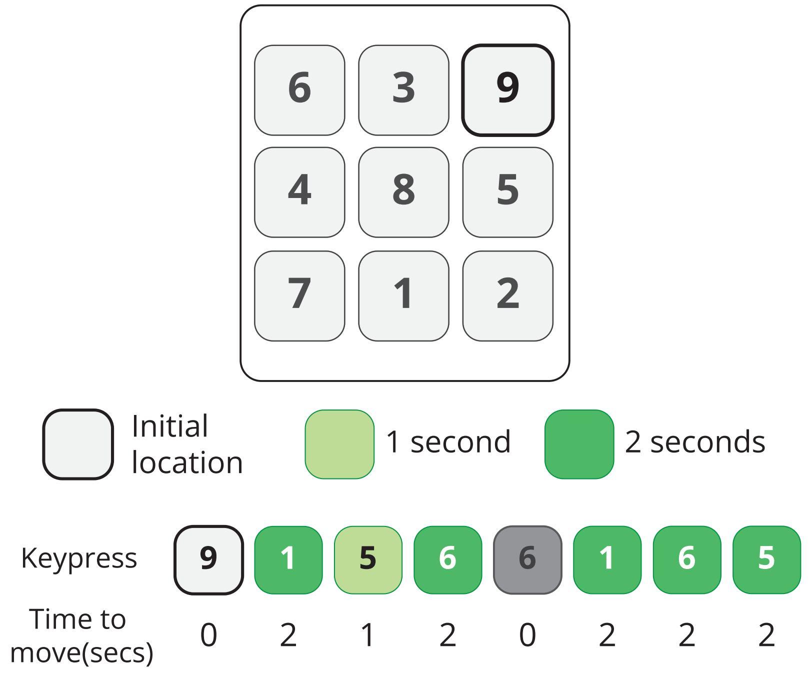
```
91566165  →   string s = "91566165"
639485712 →   string keypad = "639485712"
```

Sample Output 2

```
11
```

Explanation 2

The keypad looks like this:



Time to type 91566165 = 0+2+1+2+0+2+2+2 = 11 seconds

Calculate the time it takes to type *s = 91566165* as follows:

- *9*: Start here, so it takes *0* seconds.
- *1*: It takes *2* seconds to move from *9 → 1*
- *5*: It takes *1* second to move from *1 → 5*
- *6*: It takes *2* seconds to move from *5 → 6*
- *6*: It takes *0* seconds to move from *6 → 6*
- *1*: It takes *2* seconds to move from *6 → 1*
- *6*: It takes *2* seconds to move from *1 → 6*
- *5*: It takes *2* seconds to move from *6 → 5*,

The total time is *2 + 1 + 2 + 0 + 2 + 2 + 2 = 11*

Interviewer guidelines

▼ HINT 1

Notice that the minimum amount of time taken to type the string s is equal to the sum of minimum distances between adjacent characters of string s.

▼ HINT 2

For an easier implementation, you can derive the minimum distance between two cells at position (a,b) and (c,d) to be equal to max(|a-c|, |b-d|). For example, the minimum distance between the top right key (0,2) and bottom middle key (2,1) is max(|0-2|, |1-2|) = 2.

▼ SOLUTION

Concepts covered: Greedy, Strings

Optimal Solution:

We can preprocess the given string keypad to find the distance between each pair of keys. Since, we can move to an adjacent key (horizontal, vertical, diagonal) in one unit of time, the time taken to move from the key located at cell (a,b) and cell (c,d) is max(|a-c|, |b-d|). Store the distance between each pair of keys in a hash map. Now, since the time taken to press a key is 0 units, the total time taken to type the complete string is the sum of the distances of adjacent pairs of keys in the string s. To find this, we can query the hash map.

```
def entryTime(s, keypad):
    dis = {}            # dictionary to store distance between each pair of keys
    cells = [(i,j) for i in range(3) for j in range(3)]
    for u in cells:
        for v in cells:
            number1 = keypad[u[0] * 3 + u[1]]
            number2 = keypad[v[0] * 3 + v[1]]
            dis[(number1,number2)] = max(abs(u[0] - v[0]), abs(u[1] - v[1]))    # distance between two cells
    ans = 0
    for i in range(1, len(s)):      # iterating over adjacent keys
        ans += dis[(s[i],s[i-1])]
    return ans
```

Instead of a dictionary, a 9x9 array can store the distances. Each row indicates the key we're moving from, and the column indicates the key we're moving to.

```
def entryTime(s, keypad):
    # generate a list of cell coordinates
    cells = [(i,j) for i in range(3) for j in range(3)]
    # create a 9x9 array to hold distances (key from/to are coordinates)
    dis = [[0]*9for x in range(9)]
    # convert the elements to integer
    keypad = list(map(int, keypad))
    # calculate distances
    for u in cells:
        for v in cells:
            number1 = keypad[u[0] * 3 + u[1]]
            number2 = keypad[v[0] * 3 + v[1]]
            dis[number1-1][number2-1] = max(abs(u[0] - v[0]), abs(u[1] - v[1]))    # distance between two
cells
    # sum the results
    ans = 0
    for i in range(1, len(s)):
        ans += dis[int(s[i])-1][int(s[i-1])-1]
    return ans
```

Brute Force Approach:
A possible brute force solution could be to treat the keypads as a graph and use Flloyd Warshal Algorithm to find the distance between each pair of keys. This would be an overkill since the distance could be directly obtained through mathematical observation.
Error Handling: No such edge cases in the problem

▼ COMPLEXITY ANALYSIS

Time Complexity - O(n). where n is the length of string

Initial preprocessing of distances between each pair of keys takes a constant 81 passes since there are 9 cells. Next, to find the final answer, a single pass over the string s was required, hence O(n).

Space Complexity - O(1) - Constant extra space is required.

Since we are storing the distances between pair of keys in a dictionary, we will use additional space of 9 * 9 = 81 units. This is constant with respect to input.

▼ FOLLOW UP QUESTION

What if the keypad is huge? Let's suppose the keypad is a rectangular matrix of dimension n * m, and the number of keys is in the order of 10^5

We could use a hash table to store the coordinates of each key. Then the distance between any two keys can be calculated in O(1) time by using O(1) lookup table and using the formula of distance we derived above(max(|a-c|,|b-d|)).

Psuedo Code -

```
positions = []
for i in range(len(keypad)):
    positions[keypad[i]] = (i//m, i%m)
ans = 0
for i in range(1, len(s)):
    ans += max(abs(positions[s[i]][0] - positions[s[i-1]][0]), abs(positions[s[i]][1] - positions[s[i-1]][1]))
return ans
```

Candidate's Solution

Language used: **Python 3**

```python
#!/bin/python3

import math
import os
import random
import re
import sys


#
# Complete the 'entryTime' function below.
#
# The function is expected to return an INTEGER.
# The function accepts following parameters:
#  1. STRING s
#  2. STRING keypad
#

def entryTime(s, keypad):
    # Write your code here
    #map the keypad to a dictionary
    row_map = {}

    for i in range(len(keypad)):
        row = i // 3 + 1
        row_map[keypad[i]] = row

    # print(row_map)

    min_num_presses = 0
    for digit in s:
        min_num_presses = min_num_presses + row_map[digit]

    return min_num_presses
if __name__ == '__main__':
    fptr = open(os.environ['OUTPUT_PATH'], 'w')

    s = input()

    keypad = input()

    result = entryTime(s, keypad)

    fptr.write(str(result) + '\n')

    fptr.close()

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample	Wrong Answer	0	0.0348 sec	10.2 KB
TestCase 1	Easy	Sample	Wrong Answer	0	0.0359 sec	10.3 KB
TestCase 2	Easy	Sample	Wrong Answer	0	0.0456 sec	10.2 KB
TestCase 3	Medium	Hidden	Wrong Answer	0	0.0307 sec	10.4 KB

TestCase 4	Medium	Hidden	Wrong Answer	0	0.0378 sec	10.2 KB
TestCase 5	Medium	Sample	Wrong Answer	0	0.034 sec	10.2 KB
TestCase 6	Medium	Hidden	Wrong Answer	0	0.027 sec	10.4 KB
TestCase 7	Hard	Sample	Wrong Answer	0	0.039 sec	10.2 KB
TestCase 8	Hard	Hidden	Wrong Answer	0	0.0431 sec	10.4 KB
TestCase 9	Hard	Hidden	Wrong Answer	0	0.044 sec	10.4 KB
TestCase 10	Hard	Hidden	Wrong Answer	0	0.0298 sec	10.3 KB
TestCase 11	Hard	Hidden	Wrong Answer	0	0.0343 sec	10.4 KB

No comments.

3. Last and Second-Last

✓ Correct

Coding Easy Problem Solving Strings

Question description

Given a string, create a new string made up of its last two letters, reversed and separated by a space.

Example

Given the word '*bat*', return '*t a*'.

Function Description

Complete the function *lastLetters* in the editor below.

lastLetters has the following parameter(s):
string word: a string to process

Returns:
string: a string of two space-separated characters

Constraint

- *2 ≤* length of *word ≤ 100*

▼ INPUT FORMAT FOR CUSTOM TESTING

Input from stdin will be processed as follows and passed to the function.

The line contains a string, *word*.

▼ SAMPLE CASE 0

Sample Input

```
STDIN    Function
-----    -----
APPLE  →  word = 'APPLE'
```

Sample Output

```
E L
```

Explanation

The last letter in '*APPLE*' is *E* and the second-to-last letter is *L*, so return *E L*.

Candidate's Solution

Language used: **Python 3**

```python
#!/bin/python3

import math
import os
import random
import re
import sys


#
# Complete the 'lastLetters' function below.
#
# The function is expected to return a STRING.
# The function accepts STRING word as parameter.
#

def lastLetters(word):
    # Write your code here
    last_2_chars = word[-2:]
    reversed_2_chars = last_2_chars[::-1]

    result = ' '.join(reversed_2_chars)
    return result
if __name__ == '__main__':
    fptr = open(os.environ['OUTPUT_PATH'], 'w')

    word = input()

    result = lastLetters(word)

    fptr.write(result + '\n')

    fptr.close()

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED

Testcase 0	Easy	Sample	Success	1	0.0277 sec	10.2 KB
Testcase 1	Easy	Hidden	Success	7	0.0491 sec	10.4 KB
Testcase 2	Easy	Hidden	Success	8	0.042 sec	10.2 KB
Testcase 3	Easy	Hidden	Success	8	0.0337 sec	10.2 KB
Testcase 4	Medium	Hidden	Success	11	0.0318 sec	10.3 KB
Testcase 5	Hard	Hidden	Success	14	0.0359 sec	10.1 KB
Testcase 6	Easy	Sample	Success	1	0.0305 sec	10.3 KB

No comments.

4. Can you store multiple keys with the same value in a hashmap?

✓ Correct

Multiple Choice

Question description

Can you store multiple keys with the same value in a hashmap?

Candidate's Solution

Options: (Expected answer indicated with a tick)

● No, multiple keys with the same value can't be stored in a hashmap. When you try to insert a new key that is already present in the hashmap, then overriding will happen and the old key will be replaced with the new key and a new value. ✓

○ Yes, multiple keys with the same value can be stored in a hashmap. When you try to insert a new key that is already present in the hashmap, then overriding will happen and the old key will be replaced with the new key and a new value.

ⓘ No comments.

5. True or False: Hashmaps

✓ Correct

Multiple Choice

Question description

There is no particular order for storing keys in a hashmap. All the keys are stored in an unsorted order.

Candidate's Solution

Options: (Expected answer indicated with a tick)

● True ✓

○ False

ⓘ No comments.

6. True or False: Heaps

✓ Correct

Multiple Choice

Question description

Heap exhibits the property of a binary tree.

Candidate's Solution

Options: (Expected answer indicated with a tick)

● True ✓

○ False

ⓘ No comments.

7. Rod Cutting

✓ Correct

Coding Easy Data Structures Algorithms Arrays Problem Solving

Question description

Given an array with the lengths of various metal rods, repeatedly perform the following:

1. Count the number of rods.
2. Find the rod(s) with the shortest length.
3. Discard any rod of that length.

4. Cut that shortest length from each of the longer rods. These are *offcuts.*
5. Discard all offcuts.
6. Repeat until there are no more rods.

Maintain an array of the numbers of rods at the beginning of each round of actions and return that array.

Example

n = 4

lengths = [1, 1, 3, 4]

- The shortest rods are *1* unit long, so discard them and record their length.
- Remove their length, *1* unit, from the longer rods and discard the offcuts.
- Now, there are *2* rods, *lengths = [2, 3]*. Discard the rod of length *2*.
- Cut *2* from the rod length *3,* and discard the offcut.
- Now there is only one rod of length *1*. It is the shortest, so discard it.

Return an array with the number of rods at the start of each turn: *[4, 2, 1]*.

```
lengths    cut length    rods
1 1 3 4        1           4
_ _ 2 3        2           2
_ _ _ 1        1           1
_ _ _ _       DONE        DONE
```

Function Description

Complete the function *rodOffcut* in the editor below.

rodOffcut has the following parameter(s):

int lengths[n]: the starting lengths of each rod

Returns:

int[]: the number of rods at the start of each turn

Constraints

- $1 \le n \le 10^3$

- $1 \le lengths[i] \le 10^3$, where $0 \le i < n$

▼ INPUT FORMAT FOR CUSTOM TESTING

Input from stdin will be processed as follows and passed to the function.

The first line contains an integer, *n*, the number elements in *lengths[n]*.
The next *n* lines each contain an integer, *lengths[i] (0 ≤ i < n)*.

▼ SAMPLE CASE 0

Sample Input

```
STDIN     Function
-----     -----
6      →  lengths[] size n = 6
5      →  lengths = [5, 4, 4, 2, 2, 8]
4
4
2
2
8
```

Sample Output

```
6
4
2
1
```

Explanation

```
  lengths     cut length     rods
5 4 4 2 2 8        2           6
3 2 2 _ _ 6        2           4
1 _ _ _ _ 4        1           2
_ _ _ _ _ 3        3           1
_ _ _ _ _ _      DONE        DONE
```

Return the array of the number of rods at the start of each turn: *[6, 4, 2, 1].*

▼ SAMPLE CASE 1

Sample Input

```
STDIN     Function
-----     -----
8      →  lengths[] size n = 8
1      →  lengths = [1, 2, 3, 4, 3, 3, 2, 1]
2
3
4
3
3
2
1
```

Sample Output

```
8
6
4
1
```

Explanation

```
   lengths       cut length   rods
1 2 3 4 3 3 2 1       1          8
_ 1 2 3 2 2 1 _       1          6
_ _ 1 2 1 1 _ _       1          4
_ _ _ 1 _ _ _ _       1          1
_ _ _ _ _ _ _ _     DONE       DONE
```

Return the array of the number of rods at the start of each turn: *[8, 6, 4, 1]*.

Candidate's Solution Language used: **Python 3**

```python
#!/bin/python3

import math
import os
import random
import re
import sys



#
# Complete the 'rodOffcut' function below.
#
# The function is expected to return an INTEGER_ARRAY.
# The function accepts INTEGER_ARRAY lengths as parameter.
#
def rodOffcut(lengths):
    result = []
    while lengths:
        result.append(len(lengths))
        min_length = min(lengths)

        lengths = [(length - min_length) for length in lengths if length != min_length]
        lengths = [length for length in lengths if length > 0]

    return result



if __name__ == '__main__':
    fptr = open(os.environ['OUTPUT_PATH'], 'w')

    lengths_count = int(input().strip())

    lengths = []

    for _ in range(lengths_count):
        lengths_item = int(input().strip())
        lengths.append(lengths_item)

    result = rodOffcut(lengths)

    fptr.write('\n'.join(map(str, result)))
    fptr.write('\n')

    fptr.close()

```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME	MEMORY

					TAKEN	USED
TestCase 0	Easy	Sample	Success	1	0.0309 sec	10.3 KB
TestCase 1	Easy	Sample	Success	1	0.0389 sec	10.3 KB
TestCase 2	Medium	Sample	Success	1	0.0296 sec	10.4 KB
TestCase 3	Medium	Hidden	Success	7	0.0367 sec	10.3 KB
TestCase 4	Medium	Hidden	Success	7	0.0305 sec	10.3 KB
TestCase 5	Hard	Hidden	Success	8	0.0314 sec	10.3 KB
TestCase 6	Hard	Hidden	Success	8	0.0506 sec	10.4 KB
TestCase 7	Hard	Hidden	Success	8	0.0321 sec	10.3 KB
TestCase 8	Easy	Hidden	Success	3	0.0381 sec	10.3 KB
Testcase 9	Easy	Hidden	Success	3	0.0284 sec	10.3 KB
Testcase 10	Easy	Hidden	Success	3	0.0359 sec	10.3 KB

No comments.