Saved!

ALL

2

3

4

5

6

7

8

9

10

2. (Math Question) Largest 32-bit Unsigned Integer

What is the largest number representable by a 32-bit unsigned integer?

Pick **ONE** option

- (2^32) - 1
- (2^31) - 1
- 2^32
- 2^31

Clear Selection

ALL

3

4

5

6

7

8

9

3. (Math Question) Cycle-free Graph

A graph has N vertices. What is the maximum number of edges it can have so that the undirected graph is cycle-free.

Pick **ONE** option

- ○ N
- ○ N - 2
- ○ N - 1
- ○ N + 2

Clear Selection

Circular Track Intersection

ALL

4. (Math Question) Circular Track Intersection

Alex, Beth and Charlie start running simultaneously from point P, Q and R respectively on a circular track.

The distance between any two of the three points P, Q and R is L.

The ratio of the speeds of Alex, Beth and Charlie are 1:2:3.

If Alex and Beth run in opposite directions while Beth and Charlie run in the same direction, what is the distance run by Alex before Alex, Beth and Charlie meet for the 3rd time?

Pick **ONE** option

- 10L
- Alex, Beth and Charlie never meet
- 40/3 L
- 12L

Clear Selection

4
5
6
7
8
9
10

ALL

5. (Math Question) Tennis Tournament

512 players participated in a Men's Singles Tennis Knockout tournament. What is the total number of matches played in the tournament? (In a Knockout tournament there will be one winner in every match and loser is out of the tournament).

Pick **ONE** option

- 512
- 511
- 256
- 255
- 1023

Clear Selection

Continue

ALL

1

2

3

4

5

6

7

8

1. (Math Question) Probability for a Dice Roll

Two dice are thrown. What is the probability that the sum of the numbers appearing on the two dice is 8, if 3 appears on the first?

Pick **ONE** option

- ○ 1/36
- ○ 1/5
- ○ 1/6
- ○ 5/36

Clear Selection

2. (Math Question) Largest 32-bit Unsigned Integer

6. Fun Anagrams

Pranav is fond of words. He has two lists: a list of words called *words*, and a list of phrases called *phrases*.

From the list of words, he wants to figure out which words are anagrams (see note below).

Then, he wants to figure out how many different phrases he can make by replacing any anagram word with one of its corresponding anagrams in the list of phrases.

Note: In this context, an Anagram is a word formed by rearranging the letters of another word. Both such words are said to be anagrams of each other. E.g. "west" and "stew".

Solve question 6

... 'has', 'stew', 'good', 'it']
phrases = ['west has good stew', 'good stew']

From the list *words*, *west* is an anagram of *stew*. These two words can be replaced with their anagrams in the *phrases* list. The SIX phrases that can be created are:

- west has good stew
- stew has good west
- stew has good stew
- good stew
- good west

Language

```
32   *
33   *
34   * P
35   * l
36   *
37   *
38   *
39   *
40   *
41   * }
42   *
43   * long
44   *
45   *
46   *
47   *
48   *
49   *
50   *
51   *
52   *
53   * }
54   *
55   */
56   long* countA
     int* result_c
57
58   }
59
60 > int main()-
```

Test Results

Inbox - shrivastava.aj | MathWorks EDG Prog | HackerRank Questior | 3Sum - LeetCode

hackerrank.com/test/6ooobm77o63/questions/e8qa7i8cj0j

40m left

ALL

6

7

8

9

10

Each of the next *n* lines contains a string *words[i]*.

The next line contains an integer *m* denoting the number of elements in *phrases[]*.

Each of the next *m* lines contains a string *phrases[i]* made up of a number of words separated by spaces.

▼ Sample Case 0

Sample Input

```
STDIN       Function
-----       -----
6           → words[] size n = 6
the         → words = ['the', 'bats',
'tabs', 'in', 'cat', 'act']
bats
tabs
in
cat
act
3           → phrases[] size m = 3
cat the bats → phrases = ['cat the
bats', 'in the act', 'act tabs in']
in the act
act tabs in
```

Sample Output

```
4
2
4
```

Explanation

Phrase 1: For the phrase '*cat the bats*', the phrases that can be formed are:

- cat the bats
- act the bats

Language: Java 8

```
1 > import java.io.*;-
14  class Result {
15
16      /*
17       * Complete the 'co
18       *
19       * The function is
20       * The function acc
21       *  1. STRING_ARRAY
22       *  2. STRING_ARRAY
23       */
24
25      public static List<L
26
27      }
28
29  }
30 > public class Solution {-
```

Test Results | Custom Input

- each word ≤ 20
- $1 \leq m \leq 1000$
- $3 \leq$ words in a phrase ≤ 20

▶ **Input Format For Custom Testing**

▼ **Sample Case 0**

Sample Input

```
STDIN       Function
-----       -----
6       →   words[] size n = 6
the     →   words = ['the', 'bats',
'tabs', 'in', 'cat', 'act']
bats
tabs
in
cat
act
3       →   phrases[] size m = 3
cat the bats  →  phrases = ['cat the
bats', 'in the act', 'act tabs in']
in the act
act tabs in
```

Sample Output

```
4
2
4
```

Explanation

Phrase 1: For the phrase '*cat the bats*', the

Test Results

40m left

```
cat the bats → phrases = ['cat the
bats', 'in the act', 'act tabs in']
in the act
act tabs in
```

Sample Output

```
4
2
4
```

Explanation

Phrase 1: For the phrase '*cat the bats*', the phrases that can be formed are:

- cat the bats
- act the bats
- cat the tabs
- act the tabs

Phrase 2: For the phrase '*in the act*', the phrases that can be formed are:

- in the act
- in the cat

Phrase 3: For the phrase '*act tabs in*', the phrases that can be formed are:

- act tabs in
- cat tabs in
- act bats in
- cat bats in

Therefore, the integer array returned is [4, 2, 4].

Language: Java 8

```
import java.io.*;
class Result {

    /*
     * Complete the 'c
     *
     * The function is
     * The function acc
     *  1. STRING_ARRAY
     *  2. STRING_ARRAY
     */

    public static List<L

    }

}
public class Solution {
```

ALL

6 7 8 9 10

Test Results

Custom Input

59m left

ALL

7. Sum As You Go

You have to return the maximum sum that can be achieved as one goes from left to right in the array. You start with index 0 and the value at that index becomes your initial sum. Each iteration you can increment the index between *1* to *maxLength*. After each iteration, the new sum becomes the previous sum plus the value at the new index. It is important to note that the index increment can vary as you process the array (see Sample Cases for more clarity). You have to reach the end of the array (reaching the last element is mandatory) and return the maximum possible sum.

See Sample Cases for more clarity.

Function Description

Complete the function *maxSumLeftToRight* in the editor below. The function must return a long integer denoting the maximum attainable sum.

maxSumLeftToRight has the following parameter(s):

int path[n]: the array to traverse
int maxLength: the maximum index increment

Returns:

long: maximum attainable sum

Constraints

- $1 \le n \le 10^5$
- $0 \le |path[i]| \le 10^5$, where $0 \le i < n$ and $|x|$

Language C

```
1  > #include <assert.h> …
19
20   /*
21    * Complete the 'maxSumLeftToRight' function below.
22    *
23    * The function is expected to return a LONG_INTEGER.
24    * The function accepts following parameters:
25    *  1. INTEGER_ARRAY path
26    *  2. INTEGER maxLength
27    */
28
29   long maxSumLeftToRight(int path_count, int* path, int maxLength) {
30
31   }
32
33 > int main() …
```

Test Results Custom Input Run Code Run Tes

Inbox - shrivastava.aj | MathWorks EDG Prog | HackerRank Question | 3Sum

hackerrank.com/test/6ooobm77o63/questions/gbq8ro79ii0

23m left

ALL

Sample Case 0

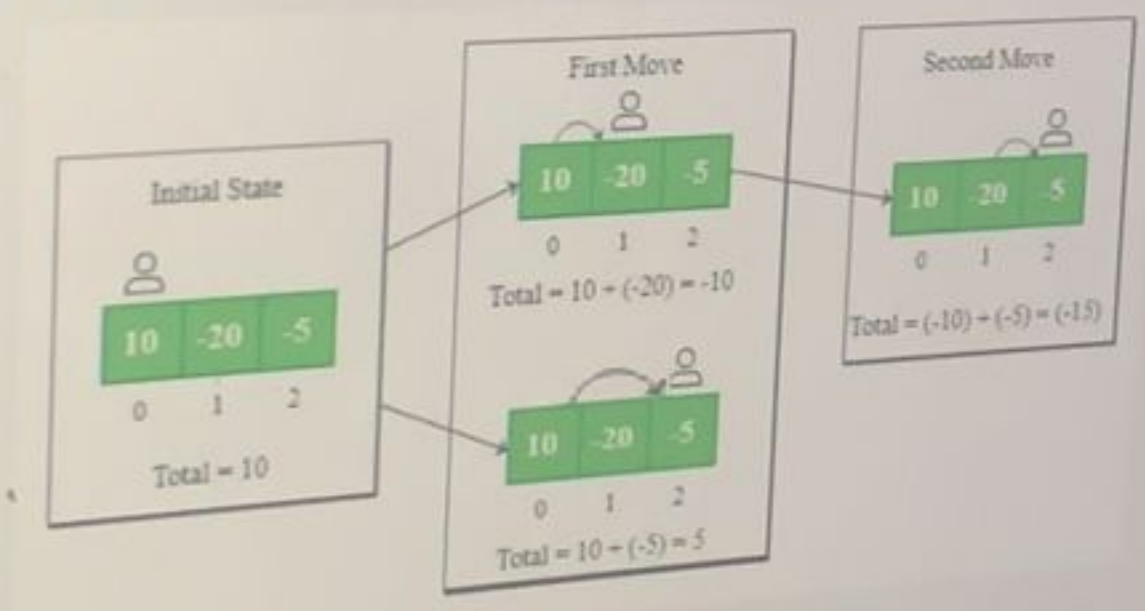
Sample Input 0

```
STDIN       Function
-----       --------
3       →   path[] size n = 3
10      →   path = [10, -20, -5]
-20
-5
2       →   maxLength = 2
```

Sample Output 0

```
5
```

Explanation 0



The initial value of *total* is *10*, and you can move forward a maximum of *maxLength = 2* cells in any movement. The two possible journeys through the array are shown above.

Sample Case 1

Sample Input 1

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

Language: Java 8

```
class Resu

/*
 * Comp
 *
 * The
 * The
 *  1. I
 *  2. I
 */

public st
    if (pa
    int cu

    while
        cur
        lef
    }

    left--;
    maxSum =
    int n = A
    int right

    while (ri
        currSu
        left--
        currSu
```

Test Results

Custo

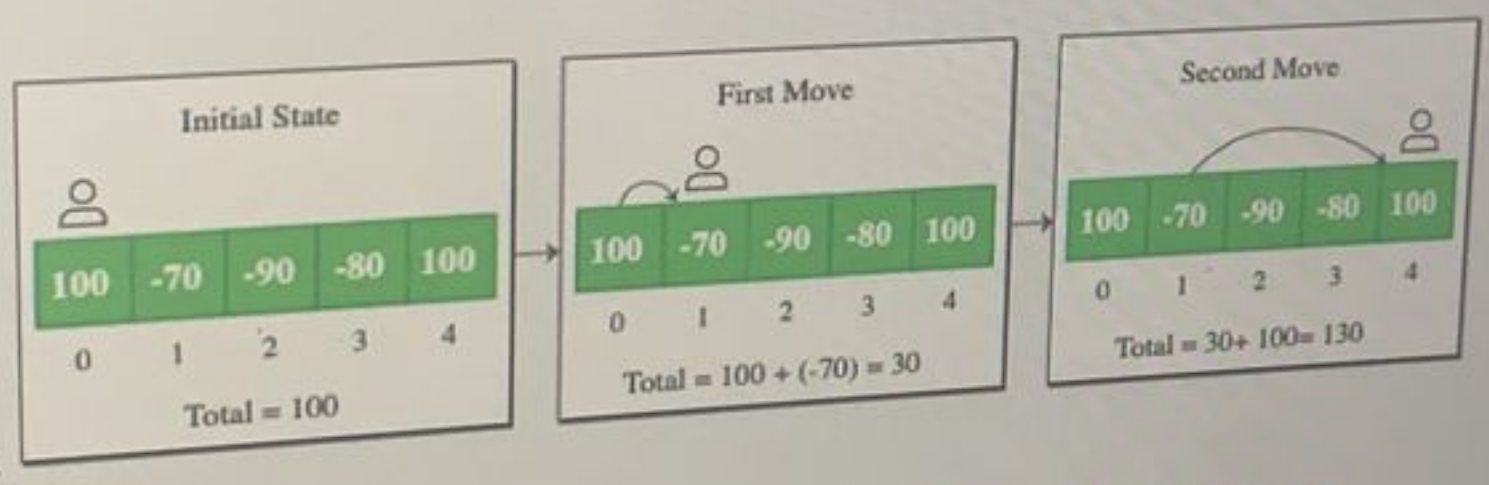
Sample Input 1

```
STDIN       Function
-----       --------
5       →   path[] size n = 5
100     →   path = [100, -70, -90, -80,
100]
-70
-90
-80
100
3       →   maxLength = 3
```

Sample Output 1

```
130
```

Explanation 1



The initial value of *total* is *100*, and you can move forward a maximum of *maxLength = 3* cells in any movement. The optimal journey is shown above.

The maximum possible value of *total = path[0] + path[1] + path[4] = 100 + -70 + 100 = 130.*

▶ Sample Case 2

Test Results