PROJECT

Algorithm:

---

1. Problem Statement:

There are bricks of k different heights where, $1 < k < 15$. The task is to find out how many towers can be constructed of height N using these bricks. Also find out the time required to construct all the towers if each tower requires 2 minutes.

Inputs

- Height of the required tower
- Number of different heights available
- Value of heights

Output

- Time required to construct all possible towers

2. Data Structures

h[] : Array to store the different heights given in input.
f[] : Array to compute and store the vector V i.e to store the all the combinations possible for height fifteen 15 possible with the heights offered in the input.
M[15][15] : The base matrix used to compute the combinations for height greater than 15.

---

3. Algorithm

Accept the input from the user and store it in different data structures as mentioned.

```
For 0 to n      // Where n stands for the number of different heights
h[input height]=1  // set the value of input height to one in the array.

// for calculating the values of combinations possible with the given blocks till height 15
InitiateVector(15)
```

// for height greater than 15 the formula is given by. M the initial matrix and F is the computer vector.

R= $M^{(n-15)} \cdot F$

// the result of $M^{(n-15)}$ is obtained from the following function.

M=expMatrix(M,(N-5)) // N is the height of the required tower.

// the result is calculated in the given function.

calculateResult(M,F)

4. Analysis

Time Complexity

There are three main functions which impact the time complexity of the program namely

Let N denote the expected height.

- initiateVector(n)

  This value depends on the different blocks of heights given by the user. The recursive function is given by n(n-1)/2 therefore O(n^2). But considering the input and other functions the time taken by this function can be assumed to be a constant.

- expMatrix(M,N-15)

  expMatrix is a recursive function. It reduces the exponentiation factor each time by two and then calls itself again with the following parameters. expMatrix(M^2,(N-15)/2).
  The complexity is given by (complexity of matrix multiply)*log N.

- multiply(M,M)
  The complexity of this part of code depends on the type of multiplication we use. If we implement Strassen's it is $N^{2.8}$ or else it is N^3.

The time complexity of our code is either O(N^3.log N) or O($N^{2.8}$ log N).

Space Complexity

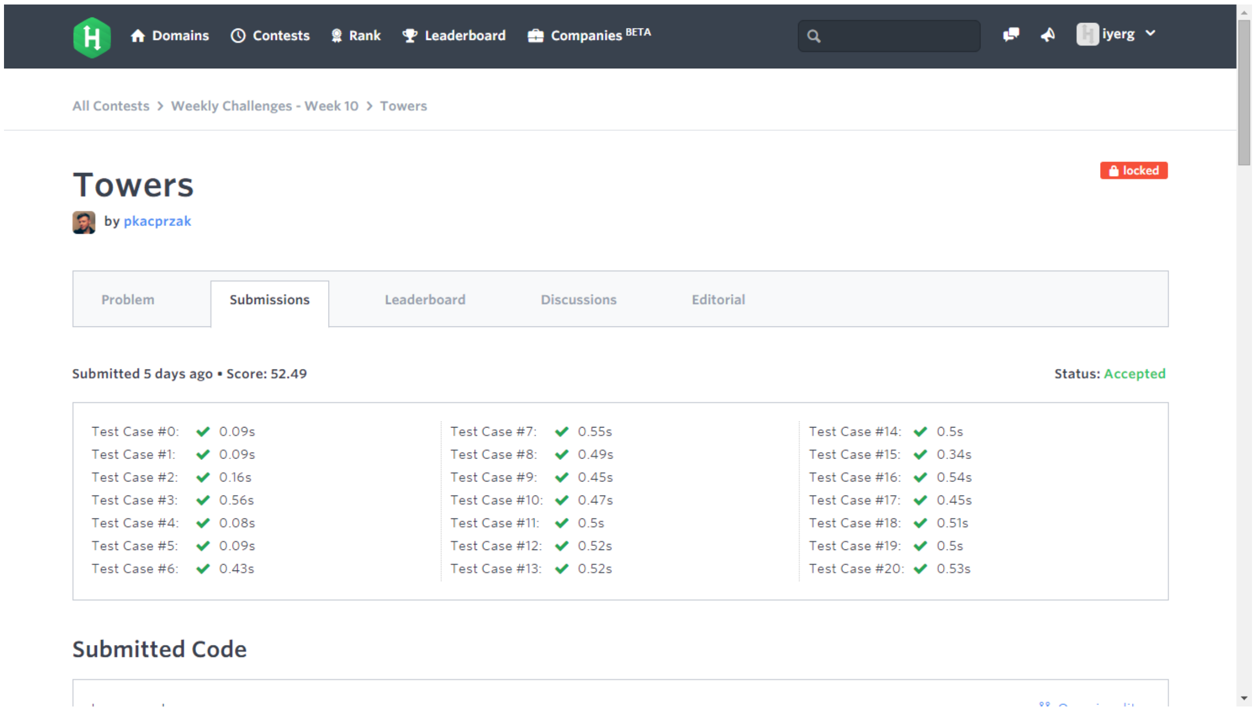
The space complexity of this code is O(1). We create a matrix M[][] which gets passed as value to all the functions. Intermediate functions have different local variables and thus are destroyed as soon as the program control is out of the function.

5. Implementation Tricks

- Since the value of the height is extremely large, we use BigInteger Object to store it.
- As the matrix exponentiation is called N-15 times the individual values in the matrix go beyond the capacity of long data type. As we can the mod the result by $10^9 + 7$, this value can be used to mod each individual element in the matrix. Thus instead of using a two dimensional array of BigInteger objects we can keep the matrix M of data type long.
- The intiateVector function uses a dictionary of the values computed in F[]. Thus it uses dynamic programming to save time and eliminate redundant calculations.
- Changed the dimensions of Matrix M to 16*16 to enable Strassen works faster.

Screenshots

1. Code accepted without Strassens.



Towers

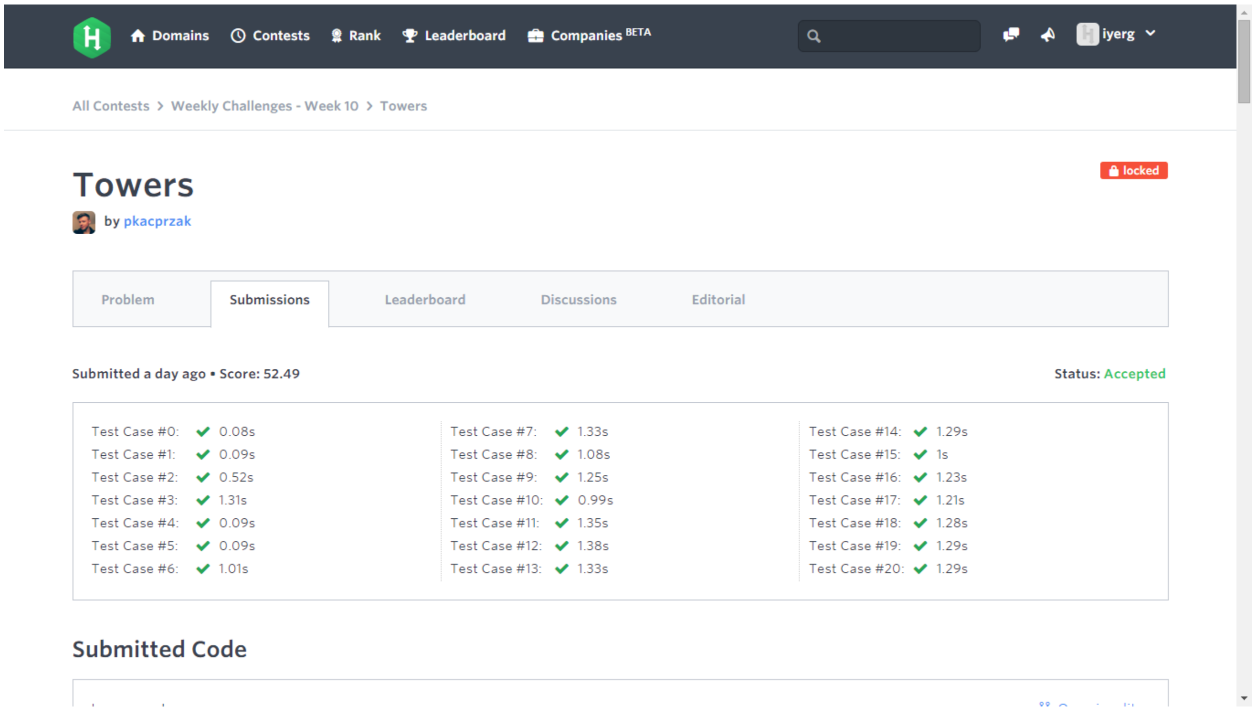
by pkacprzak

Submitted 5 days ago • Score: 52.49 Status: Accepted

Test Case #0:	✓ 0.09s	Test Case #7:	✓ 0.55s	Test Case #14:	✓ 0.5s
Test Case #1:	✓ 0.09s	Test Case #8:	✓ 0.49s	Test Case #15:	✓ 0.34s
Test Case #2:	✓ 0.16s	Test Case #9:	✓ 0.45s	Test Case #16:	✓ 0.54s
Test Case #3:	✓ 0.56s	Test Case #10:	✓ 0.47s	Test Case #17:	✓ 0.45s
Test Case #4:	✓ 0.08s	Test Case #11:	✓ 0.5s	Test Case #18:	✓ 0.51s
Test Case #5:	✓ 0.09s	Test Case #12:	✓ 0.52s	Test Case #19:	✓ 0.5s
Test Case #6:	✓ 0.43s	Test Case #13:	✓ 0.52s	Test Case #20:	✓ 0.53s

Submitted Code

2. Code accepted with Strassens.



Towers

by pkacprzak

Submitted a day ago • Score: 52.49 Status: Accepted

Test Case #0:	✓ 0.08s	Test Case #7:	✓ 1.33s	Test Case #14:	✓ 1.29s
Test Case #1:	✓ 0.09s	Test Case #8:	✓ 1.08s	Test Case #15:	✓ 1s
Test Case #2:	✓ 0.52s	Test Case #9:	✓ 1.25s	Test Case #16:	✓ 1.23s
Test Case #3:	✓ 1.31s	Test Case #10:	✓ 0.99s	Test Case #17:	✓ 1.21s
Test Case #4:	✓ 0.09s	Test Case #11:	✓ 1.35s	Test Case #18:	✓ 1.28s
Test Case #5:	✓ 0.09s	Test Case #12:	✓ 1.38s	Test Case #19:	✓ 1.29s
Test Case #6:	✓ 1.01s	Test Case #13:	✓ 1.33s	Test Case #20:	✓ 1.29s

Submitted Code

Strassens vs Normal Matrix Multiplication:

Though Strassens matrix multiplication is more efficient as compared to normal matrix multiplication, in our case the results of normal matrix multiplication are better. This is because of the fact that are matrices are very small, thus for such dimensions of matrix normal matrix multiplication seems more effective.