651. 4 Keys Keyboard

Dec. 10, 2017 | 13.9K views

Average Rating: 1.80 (78 votes)

Imagine you have a special keyboard with the following keys:

`Key 1: (A)` : Print one 'A' on screen.

`Key 2: (Ctrl-A)` : Select the whole screen.

`Key 3: (Ctrl-C)` : Copy selection to buffer.

`Key 4: (Ctrl-V)` : Print buffer on screen appending it after what has already been printed.

Now, you can only press the keyboard for **N** times (with the above four keys), find out the maximum numbers of 'A' you can print on screen.

Example 1:

```
Input: N = 3
Output: 3
Explanation:
We can at most get 3 A's on screen by pressing following key sequence:
A, A, A
```

Example 2:

```
Input: N = 7
Output: 9
Explanation:
We can at most get 9 A's on screen by pressing following key sequence:
A, A, A, Ctrl A, Ctrl C, Ctrl V, Ctrl V
```

Note:

1. 1 <= N <= 50
2. Answers will be in the range of 32-bit signed integer.

Approach Framework

Explanation

We either press 'A', or press 'CTRL+A', 'CTRL+C', and some number of 'CTRL+V's. Thus, in the context of making `N` keypresses to write the letter 'A' `M` times, there are only two types of moves:

- Add (`1` keypress): Add `1` to `M`.
- Multiply (`k+1` keypresses): Multiply `M` by `k`, where `k >= 2`.

In the following explanations, we will reference these as moves.

Approach #1: Dynamic Programming [Accepted]

Intuition and Algorithm

Say `best[k]` is the largest number of written 'A's possible after `k` keypresses.

If the last move in some optimal solution of `k` keypresses was adding, then `best[k] = best[k-1] + 1`.

Otherwise, if the last move was multiplying, then we multiplied by `x`, and `best[k-(x+1)] = best[k-(x+1)] * x` for some `x < k-1`.

Taking the best of these candidates lets us find `best[k]` in terms of previous `best[j]`, when `j < k`.

Java | Python — Copy

```
class Solution(object):
    def maxA(self, N):
        best = [0, 1]
        for k in xrange(2, N+1):
            best.append(max(best[x] * (k-x-1) for x in xrange(k-1)))
            best[-1] = max(best[-1], best[-2] + 1) #addition
        return best[N]
```

Complexity Analysis

- Time Complexity: $O(N^2)$. We have two nested for-loops, each of which do $O(N)$ work.
- Space Complexity: $O(N)$, the size of `best`.

Approach #2: Optimized Dynamic Programming [Accepted]

Intuition

If we multiply by `2N`, paying a cost of `2N+1`, we could instead multiply by `N` then `2`, paying `N+4`. When `N >= 3`, we don't pay more by doing it the second way.

Similarly, if we are to multiply by `2N+1` paying `2N+2`, we could instead multiply by `N+1` then `2`, paying `N+5`. Again, when `N >= 3`, we don't pay more doing it the second way.

Thus, we never multiply by more than `5`.

Algorithm

Our approach is the same as *Approach #1*, except we do not consider multiplying by more than 5 in our inner loop. For brevity, we have omitted this solution.

Complexity Analysis

- Time Complexity: $O(N)$. We have two nested for-loops, but the inner loop does $O(1)$ work.
- Space Complexity: $O(N)$, the size of `best`.

Approach #3: Mathematical [Accepted]

Explanation

As in *Approach #2*, we never multiply by more than 5.

When `N` is arbitrarily large, the long run behavior of multiplying by `k` repeatedly is to get to the value $k^{\frac{N}{k+1}}$. Analyzing the function $k^{\frac{N}{k+1}}$ at values $k = 2, 3, 4, 5$, it attains a peak at $k = 4$. Thus, we should expect that *eventually*, `best[K] = best[K-5] * 4`.

Now, we need to make a few more deductions.

- We never add after multiplying: if we add `c` after multiplying by `k`, we should instead multiply by `k+c`.
- We never add after 5: If we add `1` then multiply by `k` to get to `(x+1) * k = xk + k`, we could instead multiply by `k+1` to get to `xk + x`. Since `k <= 5`, we must have `x <= 5` for our additions to not be dominated.
- The number of multiplications by 2, 3, or 5 is bounded.
- Every time we've multiplied by 2 two times, we prefer to multiply by 4 once for less cost. (4^1 for a cost of 5, vs 2^2 for a cost of 6.)
- Every time we've multiplied by 3 five times, we prefer to multiply by 4 four times for the same cost but a larger result. (4^4 > 3^5, and cost is 20.)
- Every time we've multiplied by 5 five times, we prefer to multiply by 4 six times for the same cost but a larger result. (4^6 > 5^5, and cost is 30.)

Together, this shows there are at most 5 additions and 9 multiplications by a number that isn't 4.

We can find the first 14 operations on 1 by hand: `1, 2, 3, 4, 5, 6, 9, 12, 16, 20, 27, 36, 48, 64, 81`. After that, every subsequent number is achieved by multiplying by 4: ie. `best[K] = best[K-5] * 4`.

Java | Python — Copy

```
class Solution(object):
    def maxA(self, N):
        best = [0, 1, 2, 3, 4, 5, 6, 9, 12,
                16, 20, 27, 36, 48, 64, 81]
        q = (N - 11) / 5 if N > 15 else 0
        return best[N - 5*q] * 4**q
```

Complexity Analysis

- Time and Space Complexity: $O(1)$.

Analysis written by: @awice.

Rate this article:

Previous Next

Comments: 21

Sort By

Type comment here... (Markdown is supported)

Preview Post

SleezyBunny ★ 138 July 31, 2018 8:59 AM
What the hell is this. Cannot understand.
42 Share Reply
SHOW 1 REPLY

hello_world_cn ★ 240 September 30, 2018 12:39 PM
Modify approach 1 a little so that it's easier to understand:

```
class Solution {
    public int maxA(int N) {
```

Read More
16 Share Reply
SHOW 2 REPLIES

brucezu ★ 37 December 29, 2017 7:09 AM
worst solution article I ever seen
15 Share Reply

OneDayMore ★ 13 January 29, 2018 4:30 AM
It's better to rewrite the article.
8 Share Reply
SHOW 1 REPLY

LArch ★ 34 January 22, 2018 7:57 AM
Solutions are correct. The writing is unclear.
7 Share Reply

Evercode ★ 115 July 26, 2019 8:13 PM
worse solution I ever seen on LeetCode
2 Share Reply

jainsakshi22 ★ 2 July 7, 2019 3:57 PM
def max_a(n)
dp = Array.new(n+1, 0)
dp[0] = 0
dp[1] = 1
dp[2] = 2
Read More
2 Share Reply

habanero ★ 11 April 24, 2018 4:43 PM
Can I do "Ctrl-A" and "Ctrl-C" multiple times? The problem seems to assume that I select all and copy only once and paste for the rest.
2 Share Reply
SHOW 3 REPLIES

ofLucas ★ 1283 February 22, 2018 5:05 PM
<Multiply (k+1 keypresses): Multiply M by k>
is wrong. In order to multiply current `M` letters on the screen k times, you need k + 2 key presses: one key2, one key3, and k key4.
2 Share Reply
SHOW 2 REPLIES

sriramoman ★ 119 May 28, 2019 6:35 AM
Very poor quality solutions. It doesn't even explain how greedy is employed (That's a tag of the problem).
And it doesn't explain the correlation of Add, Multiply and the keystrokes in terms of the 3 keystrokes that are required to accomplish a complete cycle of select/copy/paste.
And the math solution is the worst. In 40 minutes of interview time, which candidate will have time to
Read More
1 Share Reply