544. Output Contest Matches

Dec. 10, 2017 | 11.2K views

During the NBA playoffs, we always arrange the rather strong team to play with the rather weak team, like make the rank 1 team play with the rank n_{th} team, which is a good strategy to make the contest more interesting. Now, you're given **n** teams, you need to output their **final** contest matches in the form of a string.

The **n** teams are given in the form of positive integers from 1 to n, which represents their initial rank. (Rank 1 is the strongest team and Rank n is the weakest team.) We'll use parentheses('(', ')') and commas(',') to represent the contest team pairing - parentheses('(' , ')') for pairing and commas(',') for partition. During the pairing process in each round, you always need to follow the strategy of making the rather strong one pair with the rather weak one.

Example 1:

```
Input: 2
Output: (1,2)
Explanation:
Initially, we have the team 1 and the team 2, placed like: 1,2.
Then we pair the team (1,2) together with '(', ')' and ',', which is the final answer.
```

Example 2:

```
Input: 4
Output: ((1,4),(2,3))
Explanation:
In the first round, we pair the team 1 and 4, the team 2 and 3 together, as we need to
And we got (1,4),(2,3).
In the second round, the winners of (1,4) and (2,3) need to play again to generate the
And we got the final answer ((1,4),(2,3)).
```

Example 3:

```
Input: 8
Output: (((1,8),(4,5)),((2,7),(3,6)))
Explanation:
First round: (1,8),(2,7),(3,6),(4,5)
Second round: ((1,8),(4,5)),((2,7),(3,6))
Third round: (((1,8),(4,5)),((2,7),(3,6)))
Since the third round will generate the final winner, you need to output the answer ((
```

Note:

1. The **n** is in range [2, 2^{12}].
2. We ensure that the input **n** can be converted into the form 2^k, where k is a positive integer.

Approach #1: Simulation [Accepted]

Intuition

Let `team[i]` be the correct team string of the `i`-th strongest team for that round. We will maintain these correctly as the rounds progress.

Algorithm

In each round, the `i`-th team becomes `"(" + team[i] + "," + team[n-1-i] + ")"`, and then there are half as many teams.

Java | Python

```python
class Solution(object):
    def findContestMatch(self, n):
        team = map(str, range(1, n+1))

        while n > 1:
            for i in xrange(n / 2):
                team[i] = "{1},{2}".format(team[i], team.pop())
            n /= 2

        return team[0]
```

Complexity Analysis

- Time Complexity: $O(N \log N)$. Each of $O(\log N)$ rounds performs $O(N)$ work.
- Space Complexity: $O(N \log N)$.

Approach #2: Linear Write [Accepted]

Intuition

Let's try to solve the problem in linear time. We can treat this problem as two separate problems: outputting the correct sequence of parentheses and commas, and outputting the correct team number. With a little effort, one can be convinced that a linear time solution probably exists.

Algorithm

Let's focus on the parentheses first. We can use recursion to find the answer. For example, when `N = 8`, let `R = log_2(N) = 3` be the number of rounds. The parentheses and commas look like this:

```
(((x,x),(x,x)),((x,x),(x,x)))
```

But this is just recursively

```
"(" + (sequence for R = 2) + "," + (sequence for R = 2) + ")"
= "(" + "((x,x),(x,x))" + "," + "((x,x),(x,x))" + ")"
```

Now let's look at the team numbers. For `N = 16`, the team numbers are:

`team = [1, 16, 8, 9, 4, 13, 5, 12, 2, 15, 7, 10, 3, 14, 6, 11]`

One thing we might notice is that adjacent numbers sum to `17`. More specifically, indices that are 0 and 1 (mod 2) sum to `17`. Also, indices 0 and 2 (mod 4) sum to `9`, indices 0 and 4 (mod 8) sum to `5`, and so on.

The pattern in general is: indices `0` and `2**r` (mod `2**(r+1)`) sum to `N * 2**-r + 1`.

If we want to find the next `team[i]`, then the lowest bit of `i` will help determine it's lower neighbor. For example, `team[12] = team[0b1100]` has lower bit `w = 4 = 0b100`, so `12` has lower neighbor `12 - w = 8`, and also those team numbers sum to `N / w + 1`.

Java | Python

```python
class Solution(object):
    def findContestMatch(self, n):
        team = []
        ans = []
        def write(r):
            if r == 0:
                i = len(team)
                w = i & -i
                team.append(n/w+1 - team[i-w] if w else 1)
                ans.append(str(team[-1]))
            else:
                ans.append("(")
                write(r-1)
                ans.append(",")
                write(r-1)
                ans.append(")")

        write(int(math.log(n, 2)))
        return "".join(ans)
```

Complexity Analysis

- Time Complexity: $O(N)$. We print each of $O(N)$ characters in order.
- Space Complexity: $O(N)$.

Analysis written by: @awice.

Rate this article:

Previous Next

Comments: 9

Sort By

Type comment here... (Markdown is supported)

Preview Post

UnderDog 15 May 7, 2018 8:09 AM

@zwang186 I agree that the second for loop (outer one) will run log(n) times. But the third for loop (inner one) will not run n/2 times everytime we enter the outer for loop. Let's look at it in the following way:

1. Value = n: Inner loop will run n/2 times.

Read More

12 Share Reply

SHOW 1 REPLY

lancewang 99 June 25, 2018 12:47 AM

Both are linear time:
1 + 2 + ... + n / 2 = n - 1;

17 Share Reply

SHOW 6 REPLIES

RazorQ 5 January 9, 2019 10:26 PM

The first solution should have both O(n) space and time complexity. For the time, it is similar as the time complexity for quick select. n + n/2 + n/4 + ... < 2n.

4 Share Reply

SHOW 1 REPLY

peaceCoder 452 February 24, 2018 4:38 AM

Hi @awice. I am unable to understand your lowest bit logic and in general, what is your algorithm intuition. Can you please provide a more detailed example and show the intuition.

2 Share Reply

SHOW 1 REPLY

esoh 1 August 24, 2018 2:16 AM

To see the pattern identified in the second solution, we can work our way up from the most basic example. (Working backwards from the algorithm provided in the second solution)
For N = 2.

1. team 1 fills up index 0.

Read More

1 Share Reply

SHOW 3 REPLIES

njucscc 51 February 25, 2018 3:22 AM

First solution should be O(n) space complexity.

1 Share Reply

SHOW 1 REPLY

zerustech 200 January 6, 2019 11:04 AM

Explanation of the `lower bits` logic

The pattern in general is `team[0 mod 2**r] + team[2^ mod 2**r] = N/2^ + 1`, so we have the

Read More

0 Share Reply

zwang186 66 May 7, 2018 7:10 AM

@Saksham_Sharma no, only space is O(n). time is nlogn no doubt. look at the second loop, each time n/=2, that's logN, and the inside look, each time from 0 to n/2, that's n/2, which means O(n). the first loop outside is O(n). so the total is O(n) + O((N/2)(LogN)) which is O(NlogN)

0 Share Reply

madno 245 August 29, 2019 11:09 AM

Recursive solution as you can derive during an interview:

```
//i          2)        consecutive pair sum=2+i=3, n=8,  single known value=1, n
extValCalculated=sum-val=3-1=2, pos(val)=n/2+pos(nextVal)
```

Read More

0 Share Reply