Kick Start 2022 - Round C

Analysis: Range Partition

Let us simplify the problem statement. The problem is to partition '$\mathbf{N}$', the set of first $\mathbf{N}$ positive integers $(1, 2, \ldots, \mathbf{N})$ into two subsets, S_{Alan} and $S_{Barbara}$ (for Alan and Barbara), with A and B as their sums respectively, such that $\frac{A}{B} = \frac{\mathbf{X}}{\mathbf{Y}}$ and $A + B = \frac{\mathbf{N}(\mathbf{N}+1)}{2}$ (where $\frac{\mathbf{N}(\mathbf{N}+1)}{2}$ is the sum of first $\mathbf{N}$ positive integers , let us call it Sum_N).

Test Set 1

We can check all the possible partitions of '$\mathbf{N}$' into two subsets S_{Alan} and $S_{Barbara}$ (where S_{Alan} is non-empty), to see if we encounter a partition where $\frac{A}{B} = \frac{\mathbf{X}}{\mathbf{Y}}$. If we encounter such a partition, we can conclude that it is `POSSSIBLE` to partition '$\mathbf{N}$' as it is asked in the problem statement, and return this partition as the answer. If no such partition is encountered, we can conclude that the answer is `IMPOSSIBLE`.

Since there are $2^{\mathbf{N}} - 1$ ways we can partition '$\mathbf{N}$' this way, and each partition takes $O(N)$ time to check for the conditions mentioned in the problem statement. The time complexity of this solution is $O(2^{\mathbf{N}} \times N)$.

Test Set 2

Since $A = Sum_N \times (\frac{\mathbf{X}}{\mathbf{X}+\mathbf{Y}})$ and $B = Sum_N \times (\frac{\mathbf{Y}}{\mathbf{X}+\mathbf{Y}})$. For A and B to be integers, $Sum_N \pmod{(\mathbf{X}+\mathbf{Y})} \equiv \mathbf{0}$. If $Sum_N \pmod{(\mathbf{X}+\mathbf{Y})} \not\equiv \mathbf{0}$ then it is `IMPOSSIBLE` to partition '$\mathbf{N}$' into S_{Alan} and $S_{Barbara}$.

In what follows we will use a Greedy algorithm to form S_{Alan}. The proof that such a partition is always `POSSIBLE` if $Sum_N \pmod{(\mathbf{X}+\mathbf{Y})} \equiv \mathbf{0}$, can be given by mathematical induction.

Algorithm

Let us define a function `def partition(N, PartitionSum)` which returns a partition from the set of first $\mathbf{N}$ positive integers which sums up to the `PartitionSum`.

```
def partition(N, PartitionSum):
  assert(N >= 0 and PartitionSum >= 0)
  if (PartitionSum == 0 or N == 0):
    return []
  // Greedily pick that largest available number to form the PartitionSum.
  if(N > PartitionSum):
    return partition(N-1, PartitionSum)
  else
    return [N] + partition(N-1, PartitionSum-N)
```

Proof by Induction

Base case

Given '$\mathbf{N}$', the set of first $\mathbf{N}$ positive integers. For $\mathbf{N} = 1$, the possible values of $PartitionSum = [0, 1]$. We can see that the algorithm works correctly, as it returns an empty set for $PartitionSum = 0$, and greedily chooses $[1]$ from the set for $PartitionSum = 1$.

Inductive Step

We want to show that if partitions can be formed greedily for $\mathbf{N} = K - 1$ and $PartitionSum = [0, 1, 2, \ldots, Sum_{K-1}]$ using the algorithm mentioned above, then the partitions can also be formed in the same way for $\mathbf{N} = K$ and $PartitionSum = [0, 1, 2, \ldots, Sum_K]$.

Assume the induction hypothesis that we can form partitions greedily for $\mathbf{N} = K - 1$ and $PartitionSum = [0, 1, 2, \ldots, Sum_{K-1}]$ using the above algorithm, and a partition is denoted by `partition(K-1,PartitionSum)`.

For $\mathbf{N} = K$, possible values of $PartitionSum$ are $PartitionSum = [0, 1, 2, \ldots, Sum_K]$.

Now say, for the case when $K \le PartitionSum \le Sum_K$, to form the partition, we can greedily select $[K]$ and merge it with `partition(K-1,PartitionSum-K)`. This is possible because $PartitionSum - K \le Sum_{K-1}$, so we know `partition(K-1,PartitionSum-K)` exists (our assumption from the inductive step).

Now for the other case when, $0 \le PartitionSum < K$, we can choose `partition(K-1,PartitionSum)` to form this partition as $K - 1 \le Sum_{K-1}$, hence `partition(K-1,PartitionSum)` exists (our assumption from the inductive step).

Since, both the Base case and Inductive Step have been proven as true, by mathematical induction the greedy algorithm mentioned above works for every $\mathbf{N}$ and $PartitionSum = [0, 1, 2, \ldots, Sum_N]$.

We can use the above algorithm to form S_{Alan} by calling `partition(N, A)`, what remains after picking elements for S_{Alan}, would be $S_{Barbara}$, as $A + B = Sum_N$. The time complexity of this algorithm is $O(N)$.