Kick Start 2022 - Round G

Analysis: Happy Subarrays

For simplicity, let us denote subarray of array $\mathbf{A}$ starting from index i and ending at index j, $(j \geq i)$ as $A(i, j)$.

Test Set 1

$A(i, j)$ is a happy subarray iff all of its prefix sums are non-negative, *i.e.*

$$\begin{aligned} \mathbf{A_i} &\geq 0 \\ \mathbf{A_i} + \mathbf{A_{i+1}} &\geq 0 \\ \mathbf{A_i} + \mathbf{A_{i+1}} + \mathbf{A_{i+2}} &\geq 0 \\ &\vdots \\ \mathbf{A_i} + \mathbf{A_{i+1}} + \mathbf{A_{i+2}} + \cdots + \mathbf{A_j} &\geq 0 \end{aligned}$$

We can observe that:

- Observation 1: If $A(i, j)$ is a happy subarray then all its prefix arrays $A(i, k)$, such that $i \leq k \leq j$ are also happy subarray.
- Observation 2: If $A(i, j)$ is *not* a happy subarray then all subarrays $A(i, k)$, such that $k \geq j$ are also *not* happy subarray.

We can iterate over all subarrays with a left index i. For a fixed left index i, we can iterate over the right index j such that the subarray sum is non-negative. As soon as we find an index j such that subarray sum of $A(i, j)$ is less than 0, we can stop and increase the left index.

Here is a sample code in C++.

```
int ans = 0;
for(int i = 1; i <= N; i++) {
  int cur_sum = 0;
  for(int j = i; j <= N; j++) {
    cur_sum += A[j];
    if(cur_sum < 0)
      break;
    ans += cur_sum;
  }
}
```

The overall time complexity of the above solution would be $O(\mathbf{N}^2)$, which is fast enough for test set 1.

Test Set 2

Let us denote subarray sum of $A(i, j)$ as $S(i, j)$ and prefix sum of array $\mathbf{A}$ till i^{th} index as $P(i)$,

$$\begin{aligned} S(i, j) &= \mathbf{A_i} + \mathbf{A_{i+1}} + \mathbf{A_{i+2}} + \cdots + \mathbf{A_j} \\ P(i) &= \mathbf{A_1} + \mathbf{A_2} + \mathbf{A_3} + \cdots + \mathbf{A_i} \end{aligned}$$

The prefix sum array P of array $\mathbf{A}$ can be computed in $O(\mathbf{N})$ by iterating over the array from left to right:

$$P(i) = \begin{cases} 0 & i = 0 \\ P(i-1) + \mathbf{A_i} & i > 0 \end{cases}$$

By the definition of a prefix array, we can easily note that $S(i, j) = P(j) - P(i-1)$

For every index i, let us compute $nsv(i)$ (nearest smaller value), the smallest index j such that $j \geq i$ and subarray sum of $A(i, j)$ is less than 0. If there is no such index we can simply set $nsv(i) = \mathbf{N} + 1$. Finding smallest index j on right of i, such that the subarray sum $A(i, j)$ is less than 0

$$\begin{aligned} \mathbf{A_i} + \mathbf{A_{i+1}} + \mathbf{A_{i+2}} + \cdots + \mathbf{A_j} &< 0 \\ S(i, j) &< 0 \\ P(j) - P(i-1) &< 0 \\ P(j) &< P(i-1) \end{aligned}$$

is same as finding the smallest index j, $j \geq i$ and $P(j) < P(i-1)$. This can be done using small modification in [All nearest smaller values algorithm]() in $O(\mathbf{N})$.

All subarrays which starts at index l and end at index j, such that $l \leq j < nsv(l)$ would have non-negative sum. Sum of all such subarrays starting at index l and extending at max to index r, $r = nsv(l) - 1$ is same as the sum of below expressions:

$$\begin{aligned} \mathbf{A_l} &= P(l) - P(l-1) \\ \mathbf{A_l} + \mathbf{A_{l+1}} &= P(l+1) - P(l-1) \\ \mathbf{A_l} + \mathbf{A_{l+1}} + \mathbf{A_{l+2}} &= P(l+2) - P(l-1) \\ &\vdots \\ \mathbf{A_l} + \mathbf{A_{l+1}} + \mathbf{A_{l+2}} + \cdots + \mathbf{A_r} &= P(r) - P(l-1) \end{aligned}$$

On simplification, sum of all subarray $A(i, j)$ such that $i = l$ and $i \leq j \leq r$

$$sum(l) = (P(l) + P(l+1) + P(l+2) + \cdots + P(r)) - (r - l + 1) \times P(l-1)$$

The first term $P(l) + P(l+1) + P(l+2) + \cdots + P(r)$ can be computed by pre-calculating the prefix sum array of P.

Let us denote prefix sum of P as PP, *i.e.* $PP(i) = P(1) + P(2) + \cdots + P(i)$. The above sum can be simplified as:

$$\begin{aligned} sum(l) &= PP(r) - PP(l-1) - (r - l + 1) \times P(l-1) \\ ans &= \sum_{l=1}^{N} sum(l) \\ ans &= \sum_{l=1}^{N} PP(r) - PP(l-1) - (r - l + 1) \times P(l-1) \end{aligned}$$

Nearest smaller value on right for each index in prefix array $nsv(i)$ can be computed in $O(\mathbf{N})$. Sum of all subarrays with fixed left index and moving right index can be computed in $O(1)$, if we have pre computed prefix sum array of $\mathbf{A}$ *i.e.* P and prefix sum array of P *i.e.* PP.

Precomputation of prefix sum arrays can be done in $O(\mathbf{N})$. The overall time complexity of the above solution would be $O(\mathbf{N})$