CODE FESTIVAL 2016 Tournament Round 3 A 問題 解説

yutaka1999

まず、$M = N$ のときは高橋君が覚えている情報は $1 \leq p_1 \leq ... \leq p_K$ に等しくなるので、動的計画法（以下、DP と略す）を行うことで解くことができる。具体的には、$\{p_n\}$ の要素として i 個選んだ時の i 回目までの得点の総和の最大値を $1, 2, ..., N$ の順に更新していけばよい。このとき、$O(NK)$ で解くことができる。

次に、部分点 2 以降を考える。このときも DP を行うことができるが、高橋君の覚えている情報に従うために、以下のように少し変える。

$$dp_{i,j} := (p_j = i) \text{ となるときの、} j \text{ 回目までの得点の総和の最大値}$$

この dp の更新は $dp_{i,1} = A_i, dp_{i,j} = j \cdot A_i + \max dp_{k,j-1} (1 \leq i - k \leq M, j \geq 2)$ であるから、愚直に更新すると $O(NMK)$ となり、部分点 2 を解くことができる。

ここで、j が小さい順に dp の値を埋めていくことを考えると、更新のボトルネックになっている max の計算は連続する区間なので、Segment Tree などを用いて、$O(\log N)$ で求めることができる。よって、$O(NK \log N)$ で解くことができる。さらに、この連続する区間の左右の端点が単調増加になっているため、スライド最小値を用いることができ、これにより各 max を線形で求めることができる。よって、$O(NK)$ で解くことができ、満点を取ることができる。

CODE FESTIVAL 2016 Tournament Round 3 B 問題 解説

yutaka1999

答えを二分探索することを考える。つまり、圧縮して得られる数が X 以上かどうかの判定方法を考える。圧縮される数列 $(a_1, a_2, ..., a_K)$ に対して、$b_i = \max(a_i, a_{i+1})$ とするとき、

$$b_i \geq X \Leftrightarrow (a_i \geq X) \text{ または } (a_{i+1} \geq X)$$

であり、$b_i = \min(a_i, a_{i+1})$ とするとき、

$$b_i \geq X \Leftrightarrow (a_i \geq X) \text{ かつ } (a_{i+1} \geq X)$$

であるから、最終的に残る数が X 以上であるかどうかは、$c_i = (a_i \geq X ? 1 : 0)$ とおいて、c_i に対して圧縮したときに、残る数が 1 であるかどうかと一致する。

よって、0 と 1 からなる数列 c に対しての圧縮が行うことができればよい。そこで、c の連続する 0 の区間の集合を X_0、c の連続する 1 の区間の集合を X_1 として、圧縮するたびにこれらがどのように変化をするかを考える。ただし、0 を含む c_0 が連続する区間を $[0, s]$ としたとき、$[0, s]$ の代わりに $[-\inf, s]$ の区間を考え、同様に、$n-1$ を含む c_{n-1} が連続する区間を $[t, n-1]$ としたとき、$[t, n-1]$ の代わりに $[t, \inf]$ の区間と考えることにする。

i) ステップが M のとき

X_0 の要素 $[s, t]$ は $[s, t-1]$ になり、X_1 の要素 $[u, v]$ は $[u-1, v]$ になる。ただし、X_0 の要素が空集合になる場合は消え、隣り合う X_1 の区間が併合される。

ii) ステップが m のとき

X_0 の要素 $[s, t]$ は $[s-1, t]$ になり、X_1 の要素 $[u, v]$ は $[u, v-1]$ になる。ただし、X_1 の要素が空集合になる場合は消え、隣り合う X_0 の区間が併合される。

よって、X_0, X_1 に含まれる各区間が消えるタイミングと併合されるタイミングを適切に管理することができれば良い。各区間は長さが短い順に消えていくので、 priority queue などを用いて各区間を長さが短い順に管理し、長さが 0 になっている区間ができる

かどうかをステップを経るごとに確認すればよい。また、実際に長さが 0 になって消える際には、set などを用いて初めの状態の X_0, X_1 の集合を管理しておくことにより、消える区間をはじめからなかったものとして考えて、隣り合う適切な区間を併合すればよい。

CODE FESTIVAL 2016 Tournament Round 3 Problem A Editorial

yutaka1999

First, when $M = N$, the memory of Takahashi equivalents to $1 \leq p_1 \leq ... \leq p_K$, so the problem can be solved applying Dynamic Programming (DP). Specifically, update the maximum total score up to the i-th throw when i of the elements in $\{p_n\}$ are chosen, in the order $1, 2, ..., N$, in $\mathrm{O}(NK)$ time.

We will deal with the second partial score and onward. We can still apply DP, but we will introduce a minor change as follows, according to Takahashi's memory:

$$dp_{i,j} := \text{the maximum total score up to the } i\text{-th throw when } (p_j = i)$$

The recurrence relation is $dp_{i,1} = A_i, dp_{i,j} = j \cdot A_i + \max dp_{k,j-1} (1 \leq i - k \leq M, j \geq 2)$. By naively update the table in $\mathrm{O}(NMK)$ time, we can obtain the second partial score.

Now, consider filling table in the ascending order of j. Since the bottleneck of time complexity, the calculation of max, is performed on a consecutive segment, each value in the table can be found in $\mathrm{O}(\log N)$ time, and the problem can be solved in $\mathrm{O}(NK \log N)$ time. Furthermore, since the two endpoints of the consecutive segment is monotonically increasing, we can apply the "sliding minimum" technique to find each series of max in linear time. Therefore, the problem can be solved in $\mathrm{O}(NK)$ time, which is enough to earn full credit.

CODE FESTIVAL 2016 Tournament Round 3 Problem B Editorial

yutaka1999

We will perform binary search on the answer. That is, we will determine whether the resulting integer is at least X. Let the sequence on which some step of compression is performed be $(a_1, a_2, ..., a_K)$. When we let $b_i = \max(a_i, a_{i+1})$,

$$b_i \geq X \Leftrightarrow (a_i \geq X) \text{ or } (a_{i+1} \geq X)$$

Also, when we let $b_i = \min(a_i, a_{i+1})$,

$$b_i \geq X \Leftrightarrow (a_i \geq X) \text{ and } (a_{i+1} \geq X)$$

Therefore, whether the resulting integer is at least X, is equivalent to the following: consider the sequence c_i such that $c_i = (a_i \geq X ? 1 : 0)$. If we carry out the compression on this sequence, is the single integer obtained from the compression equal to 1?

Thus, we can solve the problem if we can carry out the compression on a binary sequence c. Let X_0 be the set of the contiguous segments consisting of 0 in c, and X_1 be the set of the contiguous segments consisting of 1 in c. We will consider how they are transformed during the compression. Here, we will treat the segment consisting of c_0 that covers 0, $[0, s]$, as $[-\inf, s]$. Similarly, we will treat the segment consisting of c_{n-1} that covers $n-1$, $[t, n-1]$, as $[t, \inf]$. *i*) When an 'M' step is performed

An element $[s, t]$ in X_0 becomes $[s, t-1]$, and an element $[u, v]$ in X_1 becomes $[u-1, v]$. If an element in X_0 becomes empty, it disappears and the neighboring segments in X_1 are merged. *ii*) When an 'm' step is performed

An element $[s, t]$ in X_0 becomes $[s-1, t]$, and an element $[u, v]$ in X_1 becomes $[u, v-1]$. If an element in X_1 becomes empty, it disappears and the neighboring segments in X_0 are merged.

Thus, we can simulate the process by maintaining when each segment in X_0 and X_1 disappears or merges. Since the segments disappears in ascending order of length,

we can use a data structure such as Priority Queue to maintain the segments in ascending order of length, and check in each step if there are segments whose length become 0. Also, when a segment disappears, we can use a data structure such as Set to maintain the sets of segments in X_0 and X_1 in the initial state, to merge the appropriate segments assuming the disappearing segment does not exist in the first place.