Problem Tutorial: "Deleting"

Let's consider(hopefully) slow solution first — $dp[l][r]$ is the smallest value for deleting numbers from l to r. If number l was deleted in pair with number i then we have two cases: 1. $r = i$, in this case minimum value is $max(cost[l][r], dp[l+1][r-1])$.

2. $i < r$, then segments $[l, i]$, $[i+1, r]$ are independent and value is $max(dp[l][i], dp[i+1][r])$.

So slow solution which works in $O(n^3)$ is to compute this dp table, doing transitions in $O(n)$ time.

To speed it up, we will compute dp values in increasing order. Also, we will make forward transitions(so we will try to do transitions from already computed values to uncomputed ones). Suppose that now we consider segment $[l, r]$ and we are trying to make transitions from it. It's easy to do transition of the first type — we can set $dp[l-1][r+1]$ to $dp[l][r]$ if $cost[l-1][r+1] < dp[l][r]$ and $dp[l-1][r+1]$ is still not computed.

Transitions of the second type are harder. Let's assume that segment l, r is left one in transition. Then, we need to find numbers $k > r$ such that:

1. Value of $dp[l][k]$ is still not computed.

2. Value of $dp[r+1][k]$ is computed.

Then we are able to set $dp[l][k]$ to $dp[l][r]$.

To do this, we will use the best structure in the world — bitset. So, for each l we will maintain bitset of already computed values and uncomputed values. Then, finding k is just operation of AND. After this we will iterate over all k(for C++ users you can use Find_first() and Find_next(p) in std::bitset), and update the value of dp for them. Note that we will update the value of dp for each segment at most once, so this operations take $O(\frac{n^3}{64})$ in total.

There are some implementation notes:

1. Input is rather large, so using fast reading methods(for example, fread) can help a lot.

2. You can write solution without priority queue, because costs are small. Moreover, since they are disticnt, you can iterate over dp values in increasing order and, if current value is equal to z, then the only place where new value using first transition type can appear is l, r such that $dp[l][r] = z$. After that, you can do transitions of the second type, starting from this segment, you can do them in query(simple one, not priority) like structure, since all updated values will also have value of dp equal to z.

This optimizations are not needed to fit you solution in time(we have solution with priority queue and cin reading method), but are certainly useful in case you solution is a little bit unoptimal.