Recursive solution to 1625C: Road Optimization

I saw a bunch of solutions and explanations to this problem in the discuss section, but none of them related to the recursive approach, which is more intuitive to me, so I decided to write it down myself!

Let us first understand what the problem wants. We are given n speed boards or rather time boards, which tells, when we are at position[i] we need to cover each unit of distance in X[i] minutes till we reach position[i+1]. When we are at position N (the end point), position[i+1] is L, which is the distance of the final destination from the start point. In short words we need to find Summation of i=2 to N ((position[i] - position[i-1])*X[i-1]) + (L-position[N])*X[N].

Position array denotes the coordinates of the speed boards and X array denotes the minuted written in the speedboards.

The problem is we need to remove atmost k speedboards such that, the total time of the journey from start till L becomes minimum.

What should be our recursive approach?

We can define our function as f(a, b, c), where a is the current index of the coordinate array where we stand, b is the number of speedboards that we can still remove and c is the last index where there is a speedboard standing. 'c' is required since we have already removed (k-b) speedboards and we don't know if the last removed speedboard is in index a-1 or not.

This method works and we indeed get the right answer but the constraints don't allow us to have a 3-dimensional DP. We cannot declare an array of 500*500*500, when the space limit is given as 128mb. Hence we need to optimise.

Space Optimising our recursive function:

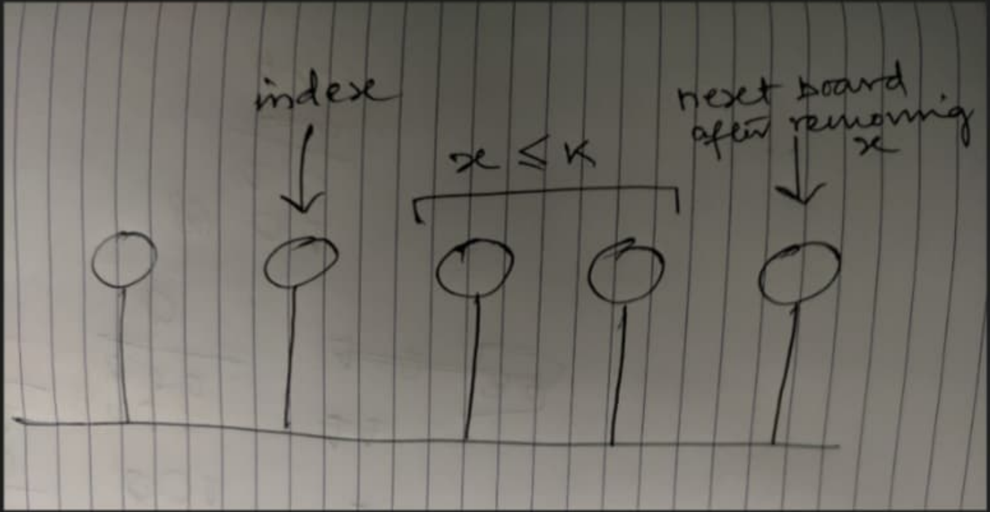
So, now we know that we need to delete a parameter from the function such that it becomes a 2-dimensional DP and will easily fit into our space requirements. The task is to decide what parameter to delete.

We had the function $f(a, b, c)$.
We cannot delete a, since we do need the current index where we are standing.
We cannot delete b as well, as there is no other way we can derive the number of speedboards we can delete from a or c.
So, the only remaining parameter is c, which is the last index, where we still have a speedboard.

Here, there is no chance, if we remove c, we can know what speedboard we have removed. We just reverse what we were doing. We will remove speedboards from (the index where we are standing + 1) till N.

Hence, we introduce a for loop inside our function, where the loop runs from $i = 0$ to b, and remove i speedboards after a, if and only $(a+i+1) <= N + 1$, since we can only remove uptill the last speedboard. We can calculate the time easily after removing i speedboards, the time would be the distance between the coordinate after the (a+i)-th position and coordinate of the a-th position, since i-speedboards have been removed between a and (a+i+1). In mathematical terms, it would be (position[a+i+1]-position[a])*X[a], since, it would now take X[a] minutes to travel from position[a] to position[a+i+1].

An image representation for better understanding:



So, our final function stands at:

```
int f(int index, int k)   // index = current position, k = remaining removals
{
   if(index > n)     // index exceeds the last position , we return 0
     return 0;
   int ans=INT_MAX;
   for(int i = 0; i <= k; i++)
   {
     // since the last speedboard is at N, we check till N+1, we assign position[N+1]
     as L initially, since it is the final destination
     if(index + i + 1 <= N + 1)
         ans=min(ans, dp(index + i + 1, k - i));   // the next index to be checked is after removing i-speedboards, k
                                                    becomes k - i.
   }
   return ans;
}
```

After this, one can easily memoize the above function using a 2-dimensional array and storing ans in it, before returning.
Hope you guys liked my explanation!