Q1：
The first question's logic is pretty clear.

Firstly, we know that the question asks us to get the HP at the beginning, which is at the end of a path. And also the question told us the final HP and MP is 0. It is simple to think that the question can be solved in a reverse way: firstly we start from the root, then walk along the path till the end of a path to get HP and MP. There will be different results because different paths, and choose the maximum one.

Now the only thing that is important to think is how do we walk from a node to the next node. Here is what we get from the question and is the main logic of the question: suppose at node A, MP = a, HP = b. we are going to node B, during this edge, attack point is w. what does not change is MP will be a+1 after going. As for HP, if a+1>w-------the HP does not change, it is still b. if a+1 < w, the HP will be b+(w-a-1).   Just remember to think things in a reverse way and do the calculation.

Now we code, the first thing to be clear is use DFS search(it is a simple recursive), to get HP&MP after every little edge, eventually to get the final HP,MP at the end of every path walk. And to get the final solution, we don't need to remember every final MP&HP, here we set a max_hp, every time we walk to one path's end, compare the max_hp and this hp, then let max_hp be the larger one in them. Finally return max_hp

Another thing that needs us to think is that this tree need to record attack point for every edge. This is easy to solve, we already know that we want to DFS along the tree, so from A TO B this edge the attack point should be attack point of A, in order to compute more easily. Here we don't set the tree as taught in the textbook. Instead we set this attack point in A's children(this is a dictionary). May be A can go to a lot of nodes(called A's children), so set A.children[child] = attackpoint. When we are building the tree, add attackpoint along the way is an easy thing to do.

Q2:
The question is too long! Let's firstly clear the question to this:
We want a maximum sum subarray

1. What kind of subarray? It is chosen from a **circular** array, each number has an attribute---its shelf number.

2. Does the subarray has any restrictions? 1. Every consecutive two shelf numbers can not be separated by more than 1(here we need to reconsider for index0); 2. every shelf number can not have same number of numbers in the subarray.

3. How do we get the circular array? We get input numbers, then assign them into shelves using there ID%k, then place in descending order of their IDs into shelves. Remember that 0 and k-1 are connected.

As for me I have tried a lot of ways to deal with this circular thing. And I think it is hard to think of a simple way very quickly. In summary, I have tried to get subarrays directly by iterate for many rounds, it is not time efficient. I have tried to connect two subarrays together when they are at the beginning and the end, there are always mistakes and hard to consider. FINALLY!   I figured out a solution:
Since we just want to know the maximum sum, we expand the array into a bigger one. For example, the original is(in it i just write shelf numbers for existing numbers) [0,2,2,3,5]when k=6(6 shelves in total), now I expand it into [0,2,2,3,5,6,8,8,9,11]. it means to keep the value unchanged, only add there shelf number by k and copy them to the end.

WHY? It is simpler to find the all subarrays!! for arrays that need to cross 0 index, like [3,5,0,2]it is transferred and can be found as [3,5,6,8], and there sums equal. Just find subarrays that there length is <= bagsize and get the smallest number is OK.
BUT we have to notice one thing: bagsize can > n, which means if bagsize is really large, and when situation is perfect we can only get n at most. But if we expand the array, and do not put any restrictions, then we may add same elements. So we have to add one thing: when the subarray length is n but still no more than bagsize, we stop.

Now the most difficult part is solved, we start coding:
Firstly deal with input, and let them be (shelf number, value)in ascending ID order. Put them all in a list. Then expand it, just put each one to be(shelf number +k, value), and add it to the end.
Then find maximum sum. We set boolean value valid(this is to see if the next number's shelf -this shelf <=1) and overflow(if the numbers taken is over n) to test whether to stop in the iteration.

For to keep shelves separation <=1, we set a previous shelf and move it along we get additional number, everytime compare the shelf number with previous number, it True, then let previous number be this shelf number.

Also we set a dictionary to record the shelf number counts, every time we take a number , the specific shelf number's value +1. and we check if the length of shelf_counts equals length of set of shelf_counts' all values, that means no same number in the shelf numbers' values.

We start at every number in the expanded list, and end is start+bagsize, but stop when end = = start +n(do not add the same number again in the bag). and compare after adding every number to keep the max_sum always the biggest. Even when we start again, we do not erase the max_sum , just keep it.
After we over the start at every number in the list, we can get the max_sum that is correct.