Question 1

a) What are the possible solutions for the problem?
   1. Construct a tree step by step. Each tree node has two elements and two pointers: node number, attack power, first child and next sibling. Following the input, we construct tree from node 1. For input u, v, w, we search node u in the existing tree, and adding node v with attack power w to the node u. Then traverse the tree and calculate the minmax hp.
   2. Using an array to store node. Each tree node has only one element and two pointers: attack power, first child and next sibling. At first, we construct all node into the array. We can visit each node by array index. Then following the input to construct tree. For input u, v, w, we find the node in the array and let u be v's parent and assign w to v's attack power. Then calculate hp grom root to each leaf and return the minmax hp.

b) How do you solve this problem?
   Tree Node: 1 element and 2 pointers, attack power, first child and next sibling. The default number of attack power is zero, default pointer is *nullptr*.

   Initialize: we first assign an array with n elements. Then we fill the array with n node.

   Construct tree: for the input u, v, w, by accessing `array[u-1]` we can reach the node u, then we add v to the u. We first check whether node u's first child is empty or not. If it is empty, we directly add v to u. If it is not empty, we move to u's first child, and check whether its next sibling is empty or not until we find a empty next sibling and add v to it.

   Find minmax hp: we have a global variable `hp_max` to store the minimax hp. we can actually find that the MP before passing one edge is exactly the depth of that node. So, we just need to traverse the tree from the root to each leaf, which is node 1 with zero hp, and calculate each node's hp and update the `hp_max`. After we traverse the whole tree, we output the `hp_max`.

c) Why is your solution better than others?
   If we use the solution 1 as I mentioned above, it will lead to runtime error when constructing the tree, since there might exist two nodes that are not in the tree yet but with edges. Also, each time it inserts the node into the tree it needs to find the node in the tree from root. My solution assigns all nodes at first to the array and can be accessed by index, which is efficient. My solutions tree node has 1 element and 2 pointers, no need to store the node number since each node can be accessed by index.

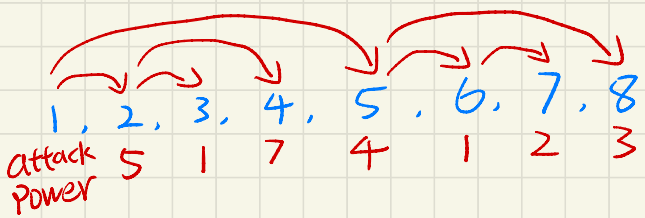
node : attack Power
first child
next sibling

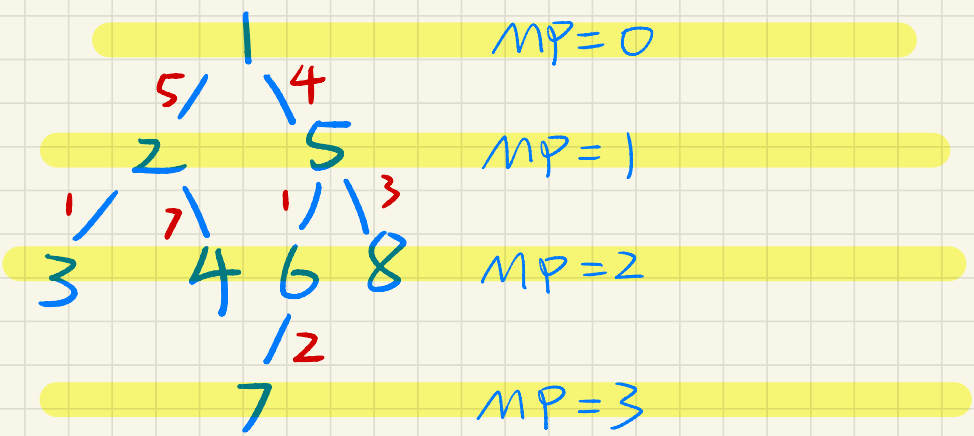
node 1 ~ node n

array [node 1, node 2, ---- node n]
index 0 1 .... n-1

Sample Input 2

```
8 7 1
1 2 5
2 3 1
2 4 7
1 5 4
5 6 1
6 7 2
5 8 3
```





HP = last_HP + max(0, attack Power - MP)

Question 2

a) What are the possible solutions for the problem?
   1. Using a hash map and linked list to store the goods in each shelf in descending order. Then we connect all linked list to form a circle linked list. Traverse the linked list and find out the max value.
   2. Using an array to store goods in each shelf. Traverse the array to find the max value for n times. After one iteration, move the first element to the last element and repeat the progress.

b) How do you solve this problem?
   Hash table: at the beginning we initialize a k size node pointer array to store the goods in each shelf.

   Add items: we first store all items data in a node. Then we add then to each shelf's vector and sort then into descending order. Finally, we add each node to the hash table one by one, in this way we get k linked lists that denote each shelf's items.

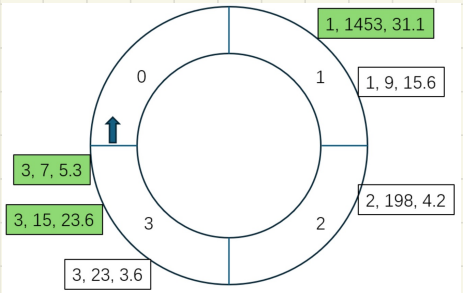
   Connect shelf: in this step, we connect the linked lists from head to tail to form a circle linked list. Then we traverse the circle and delete the node or say shelf that can be ignore. After that we traverse the circle to get a circle length denote the circle linked list's length.

   Max value: we check the bag size first, if it is larger than n then set it to n. Then we initialize value for the first round, and using an array to store the number of items taken from each shelf. Then is the iteration, move the head pointer and tail pointer. Each time we only need to minus the tail value and add the head value, then check the number of items taken from each shelf. If it is valid, we update the max value. When the max value is 0, it denotes that we can not take the bag size's items, so we use recursions, minus 1 to the bag size and repeat the progress. Finally return the max value.

   Check same: using an unordered set to check the array has the same number (except 0) or not. It returns true or false.

   Output: using `cout << fixed << setprecision(1) << result;`

c) Why is your solution better than others?
   My solution can solve the question efficiently since it doesn't need to add the value from start for each iteration, it only minus a value and add a value for each iteration. At the same time my solution can use recursion to prevent the issue that when all values at current bag size is invalid.



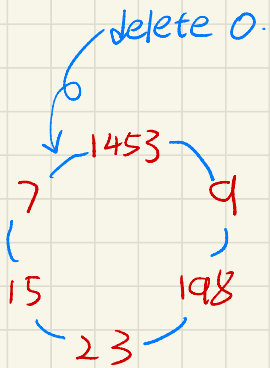
Hash table (node)

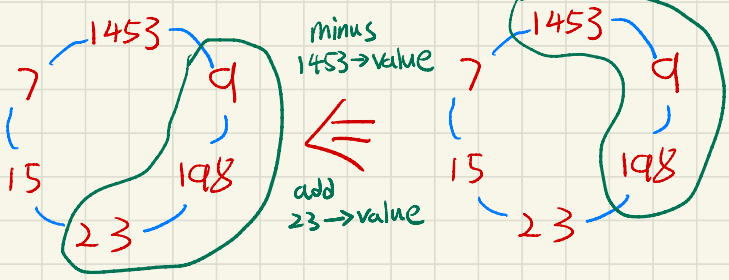
index/shelf: 0 1 2 3

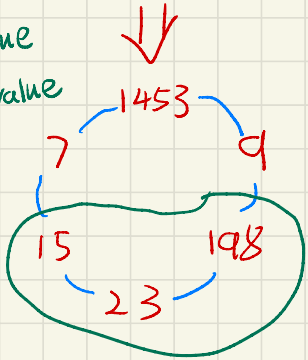
[0, 1453, 198, 23] ⇒

9 15 7

linked list







minus
9→value

⇒

add
15→value

Find max value

check valid.

if max=0 ⇒ bag size -1

repeat.