T213 - Game Developer

KK Chan {trashcan}

2021-04-18

香港電腦奧林匹克競賽
Hong Kong Olympiad in Informatics

Background

Problem Idea by KK Chan

Preparation by KK Chan, Gabriel Liu, Percy Wong and Tony Wong

Problem statement

Given N integers V_i

Also given a tree rooted at node 0

Assign m_i to node i where $m_{1..N}$ is a permutation of $1..N$ such that

Denote R_i by the value assigned to the i-th node, $R_i = V_{mi}$

Denote P_i as the parent of node i, $P_i <= i - 1$

$R_i >= R_{Pi}$

Find the lexicographically largest sequence $R_{1..N}$

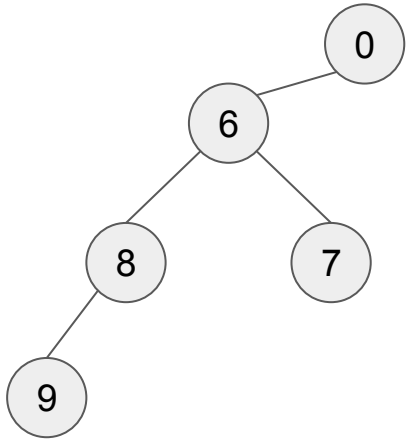
Example

Input:

```
4
6 7 8 9
0 1 1 2
```

Output:

```
6 8 7 9
```



Statistics

Attempts	Max	Mean	Std Dev
42	100	16.666	25.6

Subtasks						
16: 21	4: 22	21: 4	6: 4	14: 4	34: 3	5: 2

First solve by **dbsgame** at 2:56

SUBTASKS

For all cases:
$1 \le N \le 10^6$
$1 \le V_i \le 10^9$
$0 \le P_i \le i - 1$, for all $1 \le i \le N$

	Points	Constraints
1	16	$1 \le N \le 2 \cdot 10^5$ V_i are distinct
2	4	$1 \le N \le 8$
3	21	$1 \le N \le 200$
4	6	$1 \le N \le 1000$
5	14	$1 \le N \le 5000$
6	34	$1 \le N \le 2 \cdot 10^5$
7	5	No additional constraints

Subtask 2

Subtask 2 (4%): 1 <= N <= 8

- Exhaust all possible sequence of R_i with next_permutation
- Check if the sequence is valid and record the largest one
- Time complexity: O(N * N!)

Subtask 1

Subtask 1 (16%): 1 <= N <= 2e5, V_i are distinct

- Notice that when V_i are distinct, we can greedily assign each node with the largest value of available V_i in postorder transversal.
- Time complexity: O(N) for DFS, O(N log N) for sorting V

```
int cnt = 0;
void dfs(int u) {
  for (int v : g[u]) {
    dfs(v);
  }
  if (u != 0)
    ans[u] = V[cnt++];
}

// sort V by descending order in main
```

Subtask 3

Subtask 3 (21%): 1 <= N <= 200

- Notice that when V_i is not distinct, the greedy from Subtask 1 is no longer valid.
- You may have noticed that during contest when testing Sample 2.

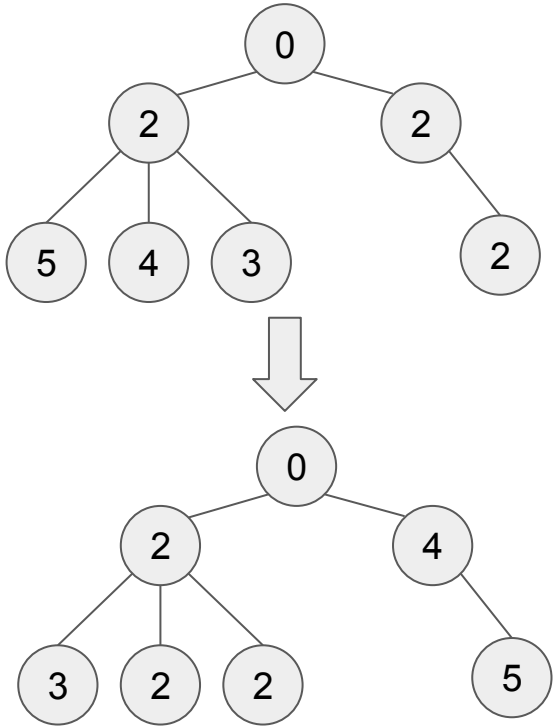
Subtask 3 Key Observations

- Observe that to assign a value `x` to node i with subtree size `sz[i]` to be valid, there must exist at least `sz[i]` number of values y remaining such that `y >= x`.
- Subtask 1's greedy solution doesn't work when V_i is not distinct because instead of using `sz[i]-th` largest elements, we can use value `x` more than once, which may reserve larger elements for later nodes to achieve better answer!

Example 1

- The greedy in Subtask 1 outputs:
  - 2 2 5 4 3 2
- But instead, 4 and 5 can be exchanged for the two 2s on the right.
- Therefore, it is not necessarily optimal to always use the largest value

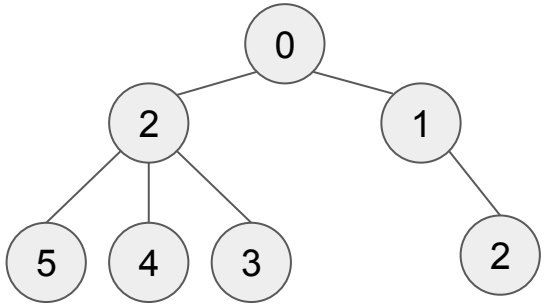
```
Inputs:
6
5 4 3 2 2 2
0 0 1 1 1 2
```



Example 2

- The greedy in Subtask 1 outputs:
  - 2 1 5 4 3 2
- We cannot exchange them this time to obtain an even more optimal answer.
- There is no simple way to determine exactly which values should be chosen when assigning the parent

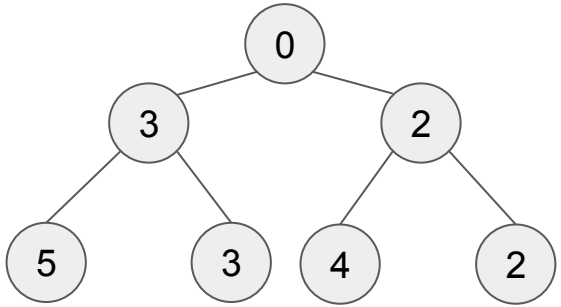
```
Inputs:
6
5 4 3 2 2 1
0 0 1 1 1 2
```



Example 3

- The greedy in Subtask 1 outputs:
  - 3 2 5 3 4 2
- However, the value 3, 4 could be exchanged here. Since we are using any DFS order of assigning, node 4 will take a larger value first we may cause the answer to be not optimal.

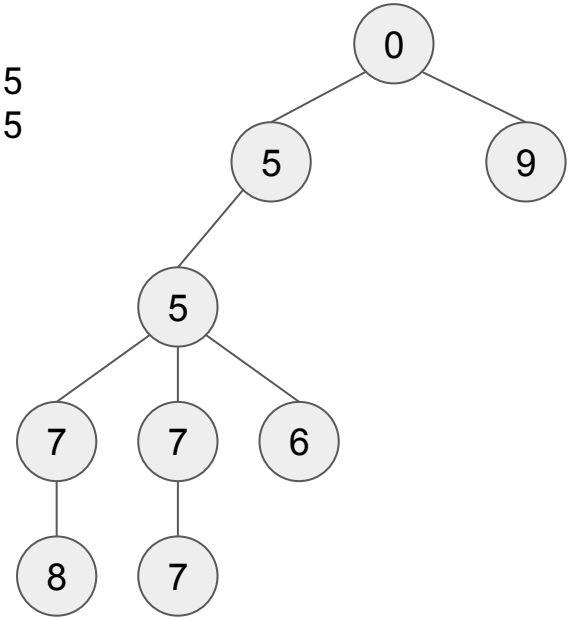
```
Inputs:
6
5 4 3 3 2 2
0 0 1 2 1 2
```



Example 4

- The greedy in Subtask 1 outputs:
  - 5 5 6 8 7 7 9 7
- We can actually not use 9 in the left subtree and instead use it on the right subtree, this requires “shifting” our choice of 5 to the rightmost and use a continuous range of value

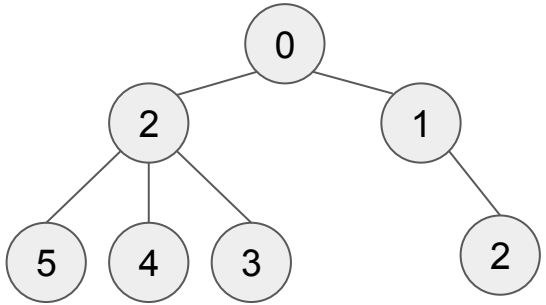
```
Inputs:
8
9 8 7 7 7 6 5 5
0 0 1 3 3 3 4 5
```



Example 2 revisited

- The greedy in Subtask 1 outputs:
  - 2 1 5 4 3 2
- Can we shift our choice of 2 to the rightmost and let left subtree use “4 3 2 2”? But it will not be optimal.

```
Inputs:
6
5 4 3 2 2 1
0 0 1 1 1 2
```



Small conclusion

- Reserving a continuous range is not correct
- We cannot predetermine which exact values have to be used for subtree

Subtask 3

- “Observe that to assign a value `x` to node i with subtree size `sz[i]` to be valid, there must exist at least `sz[i]` number of value y remaining such that `y >= x`”. We denote this value as **usable**.
- Sort the value of V in descending order
- For node i = 1..N (O(N))
- For each value in V (O(N))
- Check if this value is usable ($O(N^2)$) ← How?

Subtask 3

- When assigning a value in node i, we try:
  - for j = i + 1 to N (nodes haven't assigned value)
    - Assign smallest possible value not used & fulfill the constraints($R_i >= R_{Pi}$)
    - This can be done by linear search(O(N)) or binary search(O(log N))
- Time complexity for this part: $O(N^2)$ or O(N log N)
- Total time complexity: $O(N^4)$ or (N^3 log N)

Subtask 4 Key Observations

- We don’t have to decide which value should be used for the children yet, we just need to reserve at least `sz[i]` number of value for this subtree.

Subtask 4

- Observe that to assign a value x to node i with subtree size `sz[i]` to be valid, there must exist at least `sz[i]` number of value y remaining such that y >= x, we denote this value is **usable**.
- Sort the value of V in descending order
- For node i = 1..N (O(N))
- For each value in V (O(N))
- Check if this value is usable (O(N)) ← Faster checking!
- If this value is usable and not distinct, try to use the rightmost one.
- Reserve sz[i] value larger than the first usable value
- Time complexity $O(N^3)$

Subtask 4 Check usable

- We denote `hold[i]` as # of values before position `i` is reserved by `i`
- Suppose we want to use value at `idx`
- Obviously, if this value is already used by some other node, this is not usable
- We accumulate the total # of reserved value from `hold[i]` by using prefix sum
- If `idx == i`, we also add `sz[i]` reserved value to the prefix sum
- If the prefix sum is larger than `i + 1` (# of values), using this value at `idx` is impossible
- Otherwise, after checking for all `i`, this is a usable value

Subtask 5

Subtask 5 (14%): 1 <= N <= 5000

- Notice that if a value x is usable, then all value smaller than must also be usable, this is monotonic so we can binary search on the position of x instead of linear search.
- Time complexity: $O(N^2 \log N)$

Subtask 5, but better

Subtask 5 (14%): 1 <= N <= 5000

- Consider number of value usable before position `i`

Subtask 5

- For node 1, the subtree size is 4, so value 2 at position 4 is usable.
- We can shift to position 6 instead for more possibility when considering the siblings.
- We need to take away 4 usable value before position 6, so we can confirm that for every value after position 6, there are 4 less value usable.

```
6
5 4 3 2 2 2
0 0 1 1 1 2
```

i	1	2	3	4	5	6
value	5	4	3	2	2	2
# of usable value before position i	1	2	3	4	5	6

Subtask 5

- We need to take away 4 usable value before position 6, so we can confirm that for every value after position 6, there are 4 less values usable.
- But then what is the actual # of usable value at position before 6?
- It is actually the suffix minimum of the array.

```
6
5 4 3 2 2 2
0 0 1 1 1 2
```

i	1	2	3	4	5	6
value	5	4	3	2	2	2
# of usable value before position i	1	2	3	4	5	2
Suffix minimum	1	2	2	2	2	2

Subtask 5

- Why suffix minimum?
- When we are doing the range update operation, we can only confirm the change made to the position after position `i`. For the value before position `i`, they became invalid, as we do not know for sure which value would be used by a subtree.
- However, "# of usable value before position `i`" must be monotonic, if we denote this array by `F[i]`, `F[i] - 1 = F[i-1] <= F[i]`.
- By maintaining the suffix minimum, we can obtain the "actual" value of F.

Subtask 5

- For node 2, it has subtree size 2, the first position usable is at position 2.
- Minus 2 from 2 to the end of array
- We removed number without deciding which value the children will be used.

```
6
5 4 3 2 2 2
0 0 1 1 1 2
```

i	1	2	3	4	5	6
value	5	4	3	2	2	2
# of usable value before position i	1	0	1	2	3	0
Suffix minimum	0	0	0	0	0	0

Subtask 5

- For node 3, its parent is node 1, node 1 reserved 3 values for its children, we have to undo this change by adding back 3 from the position where the parent of a node reserves value to the end of array.

```
6
5 4 3 2 2 2
0 0 1 1 1 2
```

i	1	2	3	4	5	6
value	5	4	3	2	2	2
# of usable value before position i	1	0	1	2	3	3
Suffix minimum	0	0	1	2	3	3

Subtask 5

- Now, node 3 have subtree size of 1 and it will be taking the value 3 at position 3.
- This guarantees us to always take the largest possible value, making the answer lexicographically largest.
- Minus 1 from position 3 to end of array.

```
6
5 4 3 2 2 2
0 0 1 1 1 2
```

i	1	2	3	4	5	6
value	5	4	3	2	2	2
# of usable value before position i	1	0	0	1	2	2
Suffix minimum	0	0	0	1	2	2

Subtask 5

- We can formalize the algorithm like this:
- For each node i = 1..N
- If its parent is not 0, undo the change made by its parent by adding back the usable values, except its parent's took away value, i.e. sz[P[i]] - 1
- Find the leftmost value such that # of usable value before >= subtree size
- Shift to the rightmost one if there multiple same value
- Denote pos as the position of the value we use for this node
- Minus the subtree size of i from pos..N
- If its parent is not 0, minus the subtree size of i from its parent subtree size

Subtask 5

- Therefore we need two operations, query for suffix minimum and range updates.
- We can do this operation in O(N) naively.
- Time complexity: $O(N^2)$

Subtask 6

Subtask 6 (34%): 1 <= N <= 2e5

- Maintain the previous array by a segment tree, speeding up the range query and range update to O(log N)
- Binary search for the first position `pos` such that `query(pos, N) >= sz[i]`
- Time complexity: $O(N \log^2 N)$

Subtask 7

Subtask 7 (5%): No additional constraints

- We can just binary search while traversing the segment tree.
- Time complexity: O(N log N)

- P.S. O(N $\log^2$ N) solution using bottom up segment tree (zkw 線段樹) should not be able to pass this subtask, we intentionally make it TLE.