699. Falling Squares

DescriptionHintsSubmissionsDiscussSolution

On an infinite number line (x-axis), we drop given squares in the order they are given.

The `i`-th square dropped (`positions[i] = (left, side_length)`) is a square with the left-most point being `positions[i][0]` and sidelength `positions[i][1]`.

The square is dropped with the bottom edge parallel to the number line, and from a higher height than all currently landed squares. We wait for each square to stick before dropping the next.

The squares are infinitely sticky on their bottom edge, and will remain fixed to any positive length surface they touch (either the number line or another square). Squares dropped adjacent to each other will not stick together prematurely.

Return a list `ans` of heights. Each height `ans[i]` represents the current highest height of any square we have dropped, after dropping squares represented by `positions[0], positions[1], ..., positions[i]`.

Example 1:

```
Input: [[1, 2], [2, 3], [6, 1]]
Output: [2, 5, 5]
Explanation:

After the first drop of positions[0] = [1, 2]:
_aa
_aa
-------
The maximum height of any square is 2.

After the second drop of positions[1] = [2, 3]:
__aaa
__aaa
__aaa
_aa__
_aa__
--------------
The maximum height of any square is 5.
The larger square stays on top of the smaller square despite where its center
of gravity is, because squares are infinitely sticky on their bottom edge.

After the third drop of positions[1] = [6, 1]:
__aaa
__aaa
__aaa
_aa
_aa___a
--------------
```

The maximum height of any square is still 5.

Thus, we return an answer of [2, 5, 5].

Example 2:

Input: [[100, 100], [200, 100]]
Output: [100, 100]
Explanation: Adjacent squares don't get stuck prematurely - only their bottom edge can stick to surfaces.

Note:

- 1 <= positions.length <= 1000.
- 1 <= positions[i][0] <= 10^8.
- 1 <= positions[i][1] <= 10^6.

Seen this question in a real interview before?

```
class Solution {
public:
    typedef struct SegNode
{
     int left,mid,right;
     int max;
     bool modified;
     struct SegNode *leftchild, *rightchild;
     SegNode(int left, int right, int max)
     {
          this->left = left;
          this->right = right;
          this->mid = (left+right)/2;
          this->max = max;
          this->leftchild = NULL;
          this->rightchild = NULL;
          this->modified= false;
     }
}SegNode;

void pushdown(SegNode *node)
{
     if(node->leftchild!=NULL)
     {
          if(node->modified)
          {
               node->leftchild->modified = true;
               node->leftchild->max = node->max;
```

```
                node->rightchild->modified = true;
                node->rightchild->max = node->max;
           }
     }else
     {
           node->leftchild = new SegNode(node->left,node->mid,node-
>max);
           node->rightchild = new SegNode(node->mid+1,node-
>right,node->max);
     }
}
void insert(SegNode *node, int left, int right, int value)
{
     if(left<=node->left && right>=node->right)
     {
           if(value>node->max)
           {
                node->max = max(value,node->max);
                node->modified = true;
           }
     }else{
           pushdown(node);
           if(left<=node->mid) insert(node->leftchild,left,min(node-
>mid,right),value);
          if (right>node->mid) insert(node->rightchild,max(node-
>mid+1,left),right,value);
          node->max = max(node->leftchild->max,node->rightchild->max);
          node->modified = false;
     }
}

int querry(SegNode *node,int left, int right)
{
     int ans;
     if((left<=node->left && right>=node->right) || node-
>leftchild==NULL)
     {
           ans = node->max;
     }else{
           pushdown(node);
           ans = INT_MIN;
           if(left<=node->mid && node->leftchild!=NULL)
                ans = max(ans,querry(node->leftchild,left,min(node-
>mid,right)));
           if(right>node->mid && node->rightchild!=NULL)
                ans = max(ans,querry(node->rightchild,max(left,node-
>mid+1),right));
     }
```

```
     return ans;
}

vector<int> fallingSquares(vector<pair<int, int>>& positions) {
    if(positions.size()==0) return {};
    vector<int> ans;
    int left = 0; int right=100000000;
    SegNode root(left,right,0);
    int oldPeek=0;
    for(auto pos:positions)
    {
          int curMax=querry(&root,pos.first, pos.first+pos.second-1);
          int newMax=curMax+pos.second;
          oldPeek = max(oldPeek,newMax);
          ans.push_back(oldPeek);
       insert(&root,pos.first, pos.first+pos.second-1, newMax);
     }
     return ans;
}
};
```