Implementation of skip Lists

1. chooseLevel:

   generate a random number

   Return # of trailing zeros.

Pseudocode:

```
mask ← (1 << maxLevel) - 1

        0000 1111111
             \_maxLevel_/

lev =
    Integer.numberOfTrailingZeros(
        random.nextInt() & mask)

if lev > maxLevel then
        return maxLevel + 1

else return lev
```

maxLevel of skip List is increased when chooseLevel function returns maxLevel + 1.

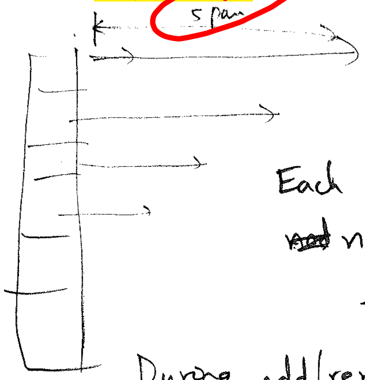
head/tail nodes can be created with max number of levels allowed, ever.

For n up to $2 * 10^9$: 32 is enough!

Indexing in Skip Lists

Linked lists: get(i) — bad. O(i)

Skip lists: get(i) — RT = O(log n) expected.



Each pointer ~~ned~~ next[i] also stores span[i]

During add/remove:

span needs to updated when something changes.

Reorganize: - not usually part of skip lists.

there for recursion practice.

Make a perfect skip list out of current list.

Tree.S8. Verify the validity of a given AVL tree (method in AVLTree class):

```
boolean verify( ):
    ( b, h, min, max ) ← verify( root )
     return b

// Bottom-up procedure to check validity of an AVL tree
// @return: boolean: is it an AVL tree?  int: height of tree, T: min value in tree, T: max value in tree
( boolean, int, T, T ) verify( t ):
   if t.element = null then return ( false, -1, null, null )
   if t.left = null   &&  t.right = null then  // leaf node
        if t.height = 0 then
            return ( true, 0, t.element, t.element )
   else if t.right = null then  // only left child
        ( lb, lh, lmin, lmax ) ←  verify( t.left )
        if lb  and  lh = 0  and  lmax < t.element then
            return ( true, 1, lmin, t.element )
   else if t.left = null then  // only right child
        ( rb, rh, rmin, rmax ) ←  verify( t.right )
        if rb  and  rh = 0  and  rmin > t.element then
            return ( true, 1, t.element, rmax )
   else  // Two child case
        ( lb, lh, lmin, lmax ) ←  verify( t.left )
        ( rb, rh, rmin, rmax ) ←  verify( t.right )
        h ← 1 + max( lh, rh )
        if lb and rb and  $|lh-rh| \le 1$  and  lmax < t.element  and  rmin > t.element and t.height = h then
              return ( true, h, lmin, rmax )
   return ( false, -1, null, null )
```

Tree.S9. Verify the validity of a given Red-Black tree:

```
boolean verify( ):    // Solution is written using null to represent $\pm\infty$
    ( isRB, color, blackHeight ) ←  verify( root, null, null )
    return isRB  and  color = Black

boolean within( lower, element, upper ):  // Check if lower < element < upper
    return (lower = null or lower < element)  and  (upper = null or upper > element)

( boolean, color, int ) verify( t, lower, upper ):
    if t = null then return ( true, Black, 0 )
    if t.element != null then
        ( leftIsRB, leftColor, lbh ) ← verify( t.left, lower, t.element )
        ( rightIsRB, rightColor, rbh ) ← verify( t.right, t.element, upper )
        if leftIsRB  and  rightIsRB  and  lbh = rbh and  within( lower, t.element, upper ) then
            if t.color = Black then  return ( true, Black, lbh + 1 )
            else  return ( leftColor = Black and rightColor = Black, Red, lbh )
return ( false, Red, -1 )
```