EECS 2101 Assignment1 Solution

Huanrui Cao **219256809** **saikoro@my.yorku.ca**

Problem 1 . Squeeze an array:

```
int curIndex = 0;

for (int i = 0; i < ints.length - 1; i++) {
    if (ints[i] != ints[i + 1])  ints[curIndex++] = ints[i];
}

if (ints.length > 0) ints[curIndex++] = ints[ints.length - 1];

while (curIndex < ints.length) { ints[curIndex++] = -1; }
```

Initialize `curIndex` to 0.

When traversing the array from the beginning in the first for loop, if the `i`th element is not equal to the `i+1`th element, then element value of `curIndex` is made equal to the `i`th element, and then `curIndex` is incremented forward by one; if it is not, then `curIndex` remains unchanged, i.e., the elements that are equal are squeezed into the `curIndex` element until the next one is unequal.

As the traversal reaches the last part of the array, since the unequal elements do not exist after that (beyond the index), the traversal ends and then manually determines that the length of the array is greater than `1`, so that the value of the `curIndex` (which has been incremented in the traversal) is equal to the last value.

Finally, if `curInex` is still smaller than the original array length, the remaining elements are padded with `-1`.

Time complexity: $\mathcal{O}(n)$

Problem 2 . Longest plateau:

Traverse the array starting from the second element, if it is equal to the previous element, then it is a **flat** array. If the left boundary still satisfies the plateau condition (the left value does not exist: `i - curLen == 0` or is strictly smaller than the subarray element: `ints[i - curLen] > ints[i - curLen - 1]`), then `curLen` is incremented by 1.

```
for (int i = 1; i < ints.length; i++) {
    if ( ints[i] == ints[i - 1]
        && (i - curLen == 0 || ints[i - curLen] > ints[i - curLen - 1]))
    {
        curLen++;
    }
```

If it is not equal to the previous element, then it is the right boundary of the flat subarray. In this case, if curLen is larger than len and satisfies the plateau condition (the i_{th} element is strictly smaller than the $i-1^{th}$ element), then value is equal to the $i-1_{th}$ element, the `startIndex` is equal to `curIndex`, and the initial subarray length is equal to `curLen`.

Then reset `curIndex` to i, `curLen` to 1.

```
else {
    if (curLen > len && ints[i] < ints[i - 1]) {
                value = ints[i - 1];
                startIndex = curIndex;
                len = curLen;
    }
    curIndex = i;
    curLen = 1;
}
```

Traversing to the end requires manually determining whether the longest plateau condition is satisfied due to the non-existence of the right boundary of the flat subarray at the end of the array:

```
if (curLen > len &&
    (ints.length - curLen == 0
     || ints[ints.length - curLen] > ints[ints.length - curLen - 1]  )
   ) {
        value = ints[ints.length - 1];
        startIndex = curIndex;
        len = curLen;
}
```

In the if condition statement:

`curLen > len` Determines if the last child is the longest.

`ints.length - curLen == 0` handles the special case where the left boundary does not exist `ints[ints.length - curLen] > ints[ints.length - curLen - 1]` Determine if the left boundary satisfies the plateau condition.

Time complexity: $\mathcal{O}(n)$

Problem 3. Overlap/enclosure counts of windows:

Enclosure

The `Window a` encloses `Window b` condition is satisfied by the fact that all boundaries of a are outside (or coincide with) the boundaries of b. Then it is clear that

```
return (boolean) this.left <= w.left && this.right >= w.right && this.bottom
<= w.bottom && this.top >= w.top;
```

Overlap

If the overlap condition is not satisfied, then

> *(a.right $\leq$ b.left) $\cup$ (a.left $\geq$ b.right) $\cup$ (a.bottom $\geq$ b.top) $\cup$ (a.top $\geq$ b.bottom)*

take its converse proposition (math.), we have

```
return (boolean) this.left < w.right && this.right > w.left && this.bottom <
w.top && this.top > w.bottom;
```

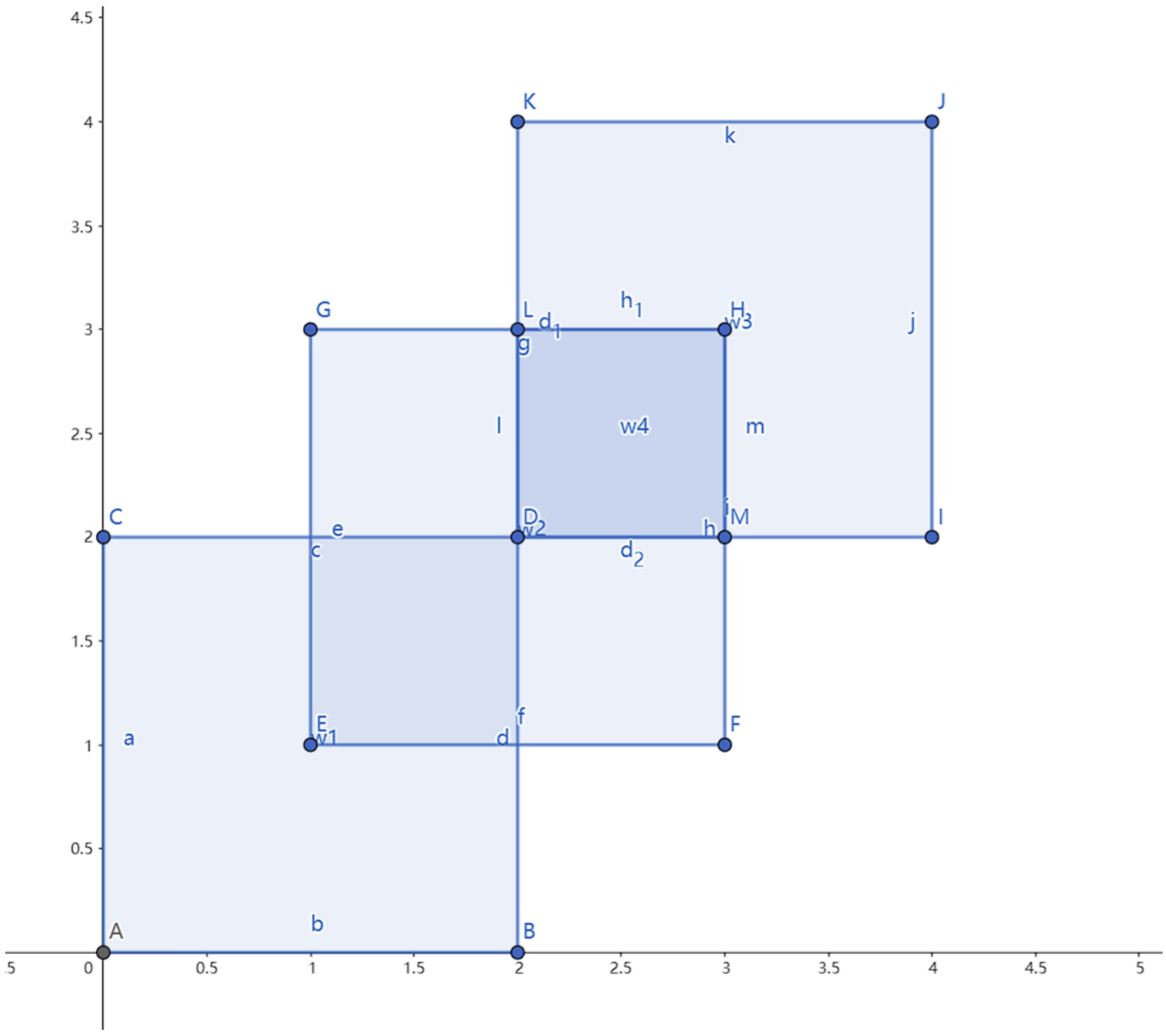
Counts

Since relationships occur in pairs when counting, traversal requires only combining rather than arranging:

```
int result = 0;
        for (int i = 0; i < windows.length; i++) {
            for (int j = i + 1; j < windows.length; j++) {
                if (windows[i].overlaps(windows[j])) {
                    result++;
                }
            }
        }
return result;
```

Time complexity: $(n-1)+...+2+1 = \sum_{k=2}^{n}(k-1) = \frac{1}{2}(n^2-n) < \mathcal{O}(n^2)$

Test Windows:



K
J
k
G
L
d1
h1
H
w3
j
g
l
w4
m
C
e
D
w2
h
M
I
c
d2
E
w1
f
d
F
a
A
b
B