GetAhead - Interview Practice 4

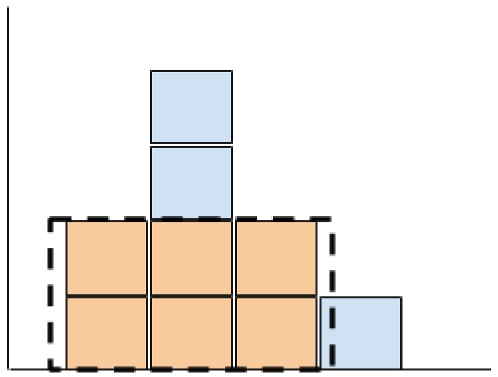
Histograms and Areas

Given an array of non-negative integers that represent the bars (y value) in a histogram (with the array index being the x value), find the rectangle with the largest area under the curve and above the x-axis (i.e. the largest rectangle that fits *inside* the histogram). Return the pair of array indices that represent the rectangle.

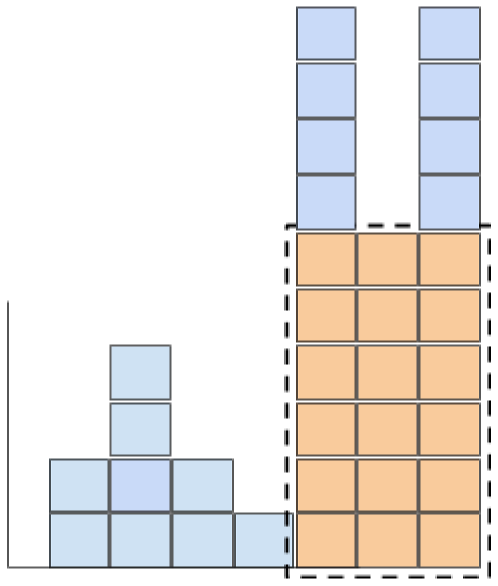
Test Cases

Note that there may be other valid answers.

For array `[2,4,2,1]`, the largest area is 6, with height 2, and width from indices 0 to 2:



For array `[2,4,2,1,10,6,10]` the largest area is 18, with height 6 and width from indices 4 to 6.



Solution

JAVA

```
// JAVA SOLUTION
import java.util.Stack;
// This is a solution for the Histograms and Areas problem.
// It is partially taken from
https://www.geeksforgeeks.org/largest-rectangle-under-histogram/

public class FindMaxRectangle {
   private int histogram[];
   private int histogramLen;
   private Stack<Integer> stack;
   private int max_area = 0; // Initialize max area
   private int left_index = 0; // The left index of the max area rectangle
   private int right_index = 0; // The right index of the max area rectangle


   public FindMaxRectangle(int histogram[]) {
       this.histogram = histogram;
       this.histogramLen = histogram.length;
       this.stack = new Stack<>();
   }

   public int getRightIndex() {
       return this.right_index;
   }
```

```
public int getLeftIndex() {
    return this.left_index;
}

private void calculateAndUpdateArea(int top, int current_position) {
    int right_position = current_position - 1;
    int left_position = stack.empty() ? 0 : stack.peek() + 1;
    int rect_width = right_position - left_position + 1;
    int rect_height = histogram[top];
    int area = rect_height * rect_width;
    if (max_area < area) {
        max_area = area;
        left_index = left_position;
        right_index = right_position;
    }
}

public int findMaxRectangle() {
    int top; // To store top of stack

    // Run through all bars of given histogram
    for (int current_position = 0;
         current_position < histogramLen;
         current_position++) {

        // If this bar is lower than top of stack, then calculate area
        // of rectangle with stack top as the smallest (or minimum height) bar.
        while (!this.stack.empty() &&
               (this.histogram[this.stack.peek()] >=
                this.histogram[current_position])) {
            top = this.stack.peek(); // store the top index
            this.stack.pop(); // pop the top
            // Calculate the area with hist[top] stack as smallest bar
            // update max area, if needed
            calculateAndUpdateArea(top, current_position);
        }
        this.stack.push(current_position);
    }

    // Now pop the remaining bars from stack and calculate area with
    // every popped bar as the smallest bar
    while (!this.stack.empty()) {
        top = this.stack.peek();
        this.stack.pop();
        calculateAndUpdateArea(top, histogramLen);
    }
```

```
        return max_area;
    }
```

You don't necessarily have to write test code in an interview, but you are still expected to provide meaningful test cases and try some manually.

```
    public static void main(String[] args) {
        int hist1[] = {2, 4, 2, 1};
        FindMaxRectangle finder1 = new FindMaxRectangle(hist1);
        int maxRectangle = finder1.findMaxRectangle();
        System.out.println("The maximum area is " + maxRectangle +
                           " with indices " + finder1.getLeftIndex() +
                           " " + finder1.getRightIndex());

        int hist2[] = {2, 4, 2, 1, 10, 6, 10};
        FindMaxRectangle finder2 = new FindMaxRectangle(hist2);
        int maxRectangle2 = finder2.findMaxRectangle();
        System.out.println("The maximum area is " + maxRectangle2 +
                           " with indices " + finder2.getLeftIndex() +
                           " " + finder2.getRightIndex());
    }

}
```

C++

```
// C++ Solution

#include <cassert>
#include <stack>
#include <utility>
#include <vector>

class LargestRectangleCalculator {
 public:
  LargestRectangleCalculator(std::vector<int> histogram)
      : histogram_(histogram) {}
  ~LargestRectangleCalculator() = default;

  std::pair<int, int> GetLargestRectangularAreaUnderCurve() {
    CalculateLargestRectangle();
    return std::pair<int, int>(low_index_, high_index_);
  }
  void PushCurrentPosOnStack(int current_pos) {
    positions_.push(current_pos);
  }

  void PopFromStacksAndUpdateMaxArea(int current_pos) {
    assert((!positions_.empty()));

    // The largest rectangle so far with the height corresponding to the top
    // of stack will have its left index as the index right after the second to
    // top element in the stack. Its right index will be right before the
    // the current position.
    int top_position = positions_.top();
    positions_.pop();

    int right_index = current_pos -1;
    int left_index = positions_.empty() ? 0: positions_.top() + 1;
    int rectangle_width = right_index - left_index + 1;
    int rectangle_height = histogram_[top_position];

    if (rectangle_width * rectangle_height > max_area_) {
      max_area_ = rectangle_width * rectangle_height;
      low_index_ = left_index;
      high_index_ = right_index;
    }
  }

  void CalculateLargestRectangle() {
    if (histogram_.empty()) return;

    for (int current_pos = 0; current_pos < histogram_.size(); current_pos++) {
      while (!positions_.empty() && histogram_[positions_.top()] >=
             histogram_[current_pos]) {
        PopFromStacksAndUpdateMaxArea(current_pos);
      }
      PushCurrentPosOnStack(current_pos);
    }
    // Pop all remaining values from the stacks.
    while (!positions_.empty()) PopFromStacksAndUpdateMaxArea(histogram_.size());
  }

 private:
  std::vector<int> histogram_;
  std::stack<int> positions_;
  int max_area_ = -1;
  int low_index_ = -1;
  int high_index_ = -1;
};
```

You don't necessarily have to write test code in an interview, but you are still expected to provide meaningful test cases and try some manually.

```
int main(int argc, char** argv) {
  LargestRectangleCalculator calculator_1(std::vector<int>({}));
  assert((calculator_1.GetLargestRectangularAreaUnderCurve() ==
          std::pair<int, int>(-1, -1)));

  LargestRectangleCalculator calculator_2(std::vector<int>({2, 4, 2, 1}));
  assert((calculator_2.GetLargestRectangularAreaUnderCurve() ==
         std::pair<int, int>(0, 2)));

  LargestRectangleCalculator calculator_3(
      std::vector<int>({2, 4, 2, 1, 10, 6, 10}));
  assert((calculator_3.GetLargestRectangularAreaUnderCurve() ==
         std::pair<int, int>(4, 6)));

  LargestRectangleCalculator calculator_4(std::vector<int>({2, 2, 2, 2}));
  assert((calculator_4.GetLargestRectangularAreaUnderCurve() ==
         std::pair<int, int>(0, 3)));

  LargestRectangleCalculator calculator_5(std::vector<int>({8,0,1,1}));
  assert((calculator_5.GetLargestRectangularAreaUnderCurve() ==
         std::pair<int, int>(0, 0)));
  return 0;
}
```

Python

Let's solve this problem in a recursive way: in each iteration, we answer the question: "what's the largest rectangle that we could fit here if we used *all* the bars?". This rectangle would be determined by the lowest bar, so we need to find the smallest value and compute the area of the rectangle (height of the lowest bar x number of bars, i.e. length of the histogram). Then compute the maximum area _without this bar_. How? Recursively, using the lowest bar as a pivot. We split the problem into two smaller ones: (a) bars to the left and (b) bars to the right of the pivot, and compute their areas. The largest rectangle is the maximum of these three subproblems: (a) the area given this lowest bar, (b) the maximum area we can find to the left of this lowest bar, and (c) the maximum area we can find to the right of this lowest bar. Base case for recursion: we are left with a single bar, case in which the area of the rectangle is the height of the bar.

An online solution can be found here.

Questions to ask ourselves:

1. DOes the recursive stack fits in memory? (yes)

Suggestions for live-coding this exercise:

1. There are faster solutions, including an O(n) approach. However, in an interview we don't necessarily want to find the optimal solution; we want something that's good enough given that we have to come up with a solution while being time constrained. In particular, the O(n) solution that can be found online is arguably difficult to come up with if we don't know the problem already. Our solution here is conceptually simpler, and still fast enough (logarithmic time complexity).
2. Start with the solution that only computes the area, to discuss the "core" of the solution. We can then worry about keeping track of the indexes (we can see this as a follow-up), and modify the solution accordingly.

Suggestions for advanced Pythonistas.

1. Using numpy.argmax() instead of writing our own find_minimum().
2. memoryview to avoiding the copies of the slice operator. First we need to transform our list with: `histogram  = memoryview(array.array('h', histogram))`, after which recursive calls such as `histogram[:index]` will not create copies.

```python
import math
from typing import List
import unittest


def find_minimum(values: List[int]) -> int:
  """Return the index of the minimum element."""
  index = -1
  minimum = math.inf
  for i, n in enumerate(values):
    if n < minimum:
      index = i
      minimum = n
  return index


def maximum_rectangle(histogram: List[int]) -> int:
  """Find the area of the maximum rectangle under the curve."""

  if not histogram:
    return 0
  if len(histogram) == 1:
    return histogram[0]
  index = find_minimum(histogram)
  return max(
      maximum_rectangle(histogram[:index]), histogram[index] * len(histogram),
      maximum_rectangle(histogram[index + 1:]))
```

This version is a modification of the previous solution, also keeping track of the indices of the columns that give us the maximum rectangle that fits inside the histogram.

```
import math
import operator
from typing import List, NamedTuple
import unittest


class Solution(NamedTuple):
  area: float
  start: int
  end: int


def find_minimum(values: List[int], start: int, end: int) -> int:
  """Return the index of the minimum element between [start, end]."""
  index = -1
  minimum = math.inf
  for i in range(start, end + 1):
    n = values[i]
    if n < minimum:
      index = i
      minimum = n
  return index


def maximum_rectangle(histogram, start=None, end=None) -> Solution:
  """Return (i, j, area): the [i, j] columns that give us the max rectangle."""

  if start is None:
    start = 0
  if end is None:
    end = len(histogram) - 1

  if start > end:
    return Solution(-math.inf, start, end)

  if start == end:
    return Solution(histogram[start], start, end)
  index = find_minimum(histogram, start, end)
  solutions = set()

  # Maximum area of the left subproblem.
  solutions.add(maximum_rectangle(histogram, start=start, end=index - 1))

  # Maximum area taking the current minimum as the highest bar.
  solutions.add(Solution(histogram[index] * (end - start + 1), start, end))

  # Maximum area of the right subproblem.
  solutions.add(maximum_rectangle(histogram, start=index + 1, end=end))

  return max(solutions, key=operator.attrgetter("area"))
```

You don't necessarily have to write test code in an interview, but you are still expected to provide meaningful test cases and try some manually.

```
class HistogramTests(unittest.TestCase):

  def test_maximum_rectangle(self):
    self.assertEqual(maximum_rectangle([6, 2, 5, 4, 5, 1, 6]),
                     Solution(12, 2, 4))
    self.assertEqual(maximum_rectangle([2, 4, 2, 1]),
                     Solution(6, 0, 2))
    self.assertEqual(maximum_rectangle([2, 4, 2, 1, 10, 6, 10]),
                     Solution(18, 4, 6))


if __name__ == "__main__":
  unittest.main()
```