CS577 HW2

09/24/2015

Keith Funkhouser

Haruki Yamaguchi

Input: $A[0 \dots n-1]$ of positive integers, where $n > 0$.

Output: Maximum value of $m(i,j) * (j-i+1)$, where $0 \leq i \leq j \leq n-1$ and $m(i,j) \doteq \min_{i \leq k \leq j} A[k]$.

Procedure $FindMaxRect(A)$

$max \leftarrow FindMaxRectRec(A, 0, n-1)$

return max

Input: A is the same as above, and $0 \leq start \leq end \leq n-1$.

Output: Maximum value of $m(i,j) * (j-i+1)$, where $start \leq i \leq j \leq end$ and $m(i,j) \doteq \min_{i \leq k \leq j} A[k]$.

Procedure $FindMaxRectRec(A, start, end)$

If $start = end$ **then**

return $A[start]$

$mid \leftarrow \frac{start + end}{2}$

$max_L \leftarrow FindMaxRectRec(A, start, mid)$

$max_R \leftarrow FindMaxRectRec(A, mid + 1, end)$

$max \leftarrow Max(max_L, max_R)$ //Assume that $Max(a,b)$ returns a larger one of a, b.

$l \leftarrow mid$

$r \leftarrow mid + 1$

While $l \geq start$ **and** $r \leq end$

If $A[l] < A[r]$ **then**

$value \leftarrow A[r]$

While $r \leq end$ **and** $value \leq A[r]$

$r \leftarrow r + 1$

Else If $A[l] > A[r]$ **then**

$value \leftarrow A[l]$

```
        While $l \geq start$ and $A[l] \geq value$
            $l \leftarrow l - 1$
    Else // $A[l] = A[r]$
        $value \leftarrow A[r]$
        While $value \leq A[r]$ and $r \leq end$
            $r \leftarrow r + 1$
        While $A[l] \geq value$ and $l \geq start$
            $l \leftarrow l - 1$
    $m \leftarrow value * (r - l - 1)$
    $max \leftarrow Max(max, m)$

While $l \geq start$
    $value \leftarrow A[l]$
    While $l > start$ and $value \leq A[l - 1]$
        $l \leftarrow l - 1$
    $l \leftarrow l - 1$
    $m \leftarrow value * (r - l - 1)$
    $max \leftarrow Max(max, m)$

While $r \leq end$
    $value \leftarrow A[r]$
    While r < end and $value \leq A[r + 1]$
        $r \leftarrow r + 1$
    $r \leftarrow r + 1$
    $m \leftarrow value * (r - l - 1)$
    $max \leftarrow Max(max, m)$
Return $max$
```

Proof of correctness

Claim: The procedure $FindMaxRectRec(A, start, end)$ returns a maximum value of $m(i,j) * (j - i + 1)$, where $start \le i \le j \le end$ and $m(i,j) \doteq \min_{i \le k \le j} A[k]$

The base case: It is invoked when A has only one element and returns a value of the element. This is correct since $m(i,j)$ is just a value of the element here, and the maximum value is $m(i,j) * (j - i + 1) = m(\mathrm{i,j}) * 1$.

Find a maximum value between $start$ and end: "The maximum value of a rectangle under pictogram" is the largest area of:

- The largest rectangle in left hand side
- The largest rectangle in right hand side
- The largest rectangle lying between left hand side and right hand side

The first two values are given by the recursion calls.

Then, the procedure must determine a value of all rectangles that lies between both left and right side.

The examination starts with two indexes l, r of elements that are next to a border; i.e., $l = mid, r = mid + 1$.

If $A[l] < A[r]$, move r to the right side until it finds r' such that $A[l] > A[r']$ or reaches to the end of right side. If such $A[r']$ is found, then the area of rectangle with height $A[r]$ is $A[r] * (r' - l - 1)$ since $\forall A[i] \ge A[r]$ for $r < i < r'$. In addition, rectangles with such $A[i]$ don't need to be determined since those rectangles can't extend to the left side (if $A[i] > A[r]$) or are identical to the one with r (if $A[i] = A[r]$). If $A[l] > A[r]$, the same method is done in the left side.

If $A[l] = A[r]$, move both l and r toward the end of each side until they find a smaller element or hit the end and determine the area.

If the area of determined rectangle is larger than the current maximum value, assign it as a new current max.

If one side is exhausted ($l = start - 1$ or $r = end + 1$), the first loop is terminated. The area calculation still gives a correct value even if either or both l, r are out of boundary by 1. At this point, **at least one half side of each rectangle whose value exists in that side is determined.**

Then either second or third loop determines the rectangles in the remaining side if they still exist. Assuming that $l > start$ after the first loop, the procedure gets into the second loop (and later skips the third loop). In the loop, move l to the left side until it finds l'such that $A[l] > A[l']$ or reaches to the end. If such $A[l']$ is found, then the area of rectangle with height $A[l]$ is $A[l] * (r - l' - 1)$ since $\forall A[i] \ge A[l]$ for $l' < i < l$, and again, rectangles with $A[i]$ don't need to be determined. If the area is larger than current max, update it. At the time loop is terminated, **the other side of each rectangle whose value exists in that side is determined.**

At the end, the area of each possible rectangle is determined, and the maximum value is returned.

Termination

Recursive call: The termination of a recursive part of $FindMaxRectRec(A, start, end)$ can be proven in the same way as a merge sort. The boundary of the procedure is defined by $start$ and end, and the bound is halved in each recursive call since it can be either $FindMaxRectRec(A, start, mid)$ or $FindMaxRectRec(A, mid + 1, end)$, where $mid = \frac{start+end}{2}$. At last, as a base case, the recursive calls terminate when $start = end$.

Since $start \leq end$ and mid is assigned by integer division, $start$ will never become larger than end. For example, let $start = n$ and $end = n + 1$, so that $mid = \frac{n+n+1}{2} = \frac{2n+1}{2} = n$ for integer division. Then the next recursive calls are $FindMaxRectRec(A, n, n)$ and $FindMaxRectRec(A, n + 1, n + 1)$.

Iteration: There are three types of while loops in the procedure.

The first type has two loop conditions: $l \geq start$ and $r \leq end$. Since either or both l is decremented by 1 and/or r is Incremented by 1 in each iteration and since $0 \leq start \leq end \leq n - 1$, the loop eventually terminates.

The second type has $l \geq start$ and the third type has $r \leq end$ as its loop condition, but they follow the same rule as the first type, as its variable is decremented/incremented for each iteration.

Proof of runtime

Since each recursive call halves its boundary, the depth of recursive calls is $\boldsymbol{logn}$. In each call, an array is iterated through, but each element is visited at most one time. Since the area of rectangle is calculated in a constant time, its runtime is $\boldsymbol{cn}$, where c is some constant. Therefore, the complexity of this algorithm is $\boldsymbol{O}(\boldsymbol{nlogn})$.

Optimization

The first comparison to find a rectangle between left and right sides (when $l = mid$ and $r = mid + 1$) is redundant since it actually calculates a rectangle that lies only in one side, unless $\mathrm{A}[l] = A[r]$. Therefore, following lines can be added right before the first while loop.

If $A[l] < A[r]$ **then**

While $A[l] < A[r]$ **and** $r \leq end$

$r \leftarrow r + 1$

Else If $A[l] > A[r]$ **then**

While $A[l] > A[r]$ **and** $l \geq start$

$l \leftarrow l - 1$