CS 381: Assignment #3

Due on Thursday, October 16th, 2014

Prof. Grigorescu 12:00pm

Yao Xiao(xiao67)

Problem 1

The original solution will only find an interval which overlap the given one. The procedure will stop as long as an overlapped interval was found. The new one requires finding the left subtree after the interval is found since the interval search tree is using low-endpoint as key, the real solution might appears at left subtree. So the question becomes find a node where the left subtree doesn't contain the given interval.

Min-interval-search-tree(x,i)
x = root[T]
while x= nil[T] and i does not overlap int[x]
if left[x] = nil[T] and max[left[x]] $\geq$ low[i]
x=left[x]
else
x=right[x]

After found the current x, which is the first interval we found, we iterated the left subtree.

If x=nil return nil
while max[x]$\geq$low[i] and x$\neq$nil
if max[left[x]]$\geq$low[i] x=left[x]
else if overlapped return x
else x=right[x]

Problem 2

Basically we need to traverse the tree using left,right and max.

Print-Intervals(x,i) while x$\neq$nil if i overlaps x print i
if max[left[x]]>=low[i] x=left[x]
if low[left[x]]<=high[i] x=right[x]

Problem 3

Apperantly It's the same question as the exam.

Problem 4

Since it's a dynamic set for a set of integers, I will use a red black tree to implement it and using the integer as the key.

Insert: Same as red-black tree $O(lgn)$
Delete Same as red-black tree $O(lgn)$
More Features:
Since the min gap can only exist on the node with the biggest element of left subtree and the smallest element of its right sub tree. So We update keep updating three value. Min-value records the node.min=min(node,node− >left.min) where can be updated for every insert and every delete. Max-value records the node.max=max(node,node− >right.max).

Problem 4 continued on next page...

Then set up a min_gap for each node. Min_gap represent the minimum gap for this subtree
$mingap = min(node->left.mingap, node->right.mingap, abs(value-node->left.max), abs(value-node->right.min)$
The complexity of looking up is $O(1)$

Problem 5

The way I will implement it is using two stack. Using one stack as enqueue, whenever a new element is added, push it into enqueue stack. The other stack is used to dequeue, every time a dequeue is called, the instack will push everything to dequeue stack and push everything out. The cost for each is $O(4n) = O(n)$