Quiz 2: 27 October, 2015

ESO207: Data Structures and Algorithms

Time 50 minutes Max Marks 27

Instructions: *Please write clearly. Clearly write the question number.*

Question 1. [Marks 10].

A 1-player board game uses a chess like board with n rows and m columns. A number (positive or negative) is written on each square. Initially the game piece is placed on square $(1,1)$ (top left corner). The player can move the piece either to left or right or down one box each time, without revisiting any box. The player can continue to play as long as the piece is in side the board. The initial score is zero and the value in each visited square is added to the score. The goal of the game is to maximize the score.

Design an efficient (dynamic programming based) algorithm to determine the highest possible score, for a given board and numbers written on the squares. Justify the correctness of the recurrence relation. Derive the time complexity of your algorithm.

Note: It is sufficient to describe the recurrence relation by clearly defining all the terms. **No** need to write the pseudocode.

Solution

Define three score functions: *scoreTop*, $scoreLeft$, and *scoreRight* with following interpretations. $scoreTop(i,j)$ will denote the maximum possible score over all paths that start at $(1,1)$ and end at (i,j) approaching from $(i-1,j)$. $scoreLeft(i,j)$ is similarly defined over paths that reach (i,j) from $(i,j-1)$ and *scoreRight* is that for paths that reach (i,j) from $(i,j+1)$.

Now a path that ends at $(i,0)$ will have maximum score $\max\{scoreTop(i,1), scoreRight(i,1)\}$. Similarly the maximum score for a path that ends at $(i,m+1)$ will be $\max\{scoreTop(i,m)$, $scoreLeft(i,m)\}$. In case the path terminates at $(n+1,j)$, then the best score will be $\max\{scoreTop(n,j), scoreLeft(n,j), scoreRight(n,j)\}$.

Finally we describe the recurrence relation for these functions. Let $w(i,j)$ denote the number written on block (i,j).

$scoreTop(1,1) = scoreLeft(1,1) = w(1,1)$.

$scoreTop(1,j) = scoreRight(1,j) = -\infty$ and $scoreLeft(1,j) = scoreLeft(1,j-1) + w(1,j)$.

$scoreLeft(i,1) = -\infty$, $scoreTop(i,1) = \max\{scoreTop(i-1,1), scoreRight(i-1,1)\} + w(i,1)$, and $scoreRight(i,1) = \max\{scoreTop(i,2), scoreRight(i,2)\} + w(i,1)$.

$scoreRight(i,n) = -\infty$, $scoreTop(i,n) = \max\{scoreTop(i-1,n), scoreLeft(i-1,n)\} + w(i,n)$, and $scoreLeft(i,n) = \max\{scoreTop(i,n-1), scoreLeft(i,n-1)\} + w(i,n)$.

For the remaining (i,j),
$scoreTop(i,j) = \max\{scoreTop(i-1,j), scoreLeft(i-1,j), scoreRight(i-1,j)\} + w(i,j)$,
$scoreRight(i,j) = \max\{scoreTop(i,j+1), scoreRight(i,j+1)\} + w(i,j)$, and $scoreLeft(i,j) = \max\{scoreTop(i,j-1), scoreLeft(i,j-1)\} + w(i,j)$.

Since each entry can be computed in $O(1)$ time the total time complexity will be $O(nm)$.

Question 2. [Marks 5+2+5+5].

Following algorithm outputs the values stored in each node of a BST rooted at vertex u, in sorted order. We assume that u is not a null vertex.

(a) Write this algorithm (pseudocode) without using recursion.

(b) If the tree has n vertices, then what is the number of transitions (passage from parent to a child or from a child to its parent) in the execution of the algorithm you have written in part (a)?

(c) In a Red-Black tree additional information has been stored in each node, namely, the count of the vertices in its subtree (including itself) (for example, the root contains the total number of

```
if u·Left ≠ null then
  | SortedAll(u·Left);
end
Output(u·value);
if u·Right ≠ null then
  | SortedAll(u·Right);
end
```

Algorithm 1: SortedAll(u)

vertices in the tree). Write the pseudocode for an algorithm which outputs all the values from i-th to the j-th element in sorted order, using $SortedAll()$ as a subroutine. You may use recursion.

(d) Give a tight upperbound for the number of transitions in the *non-recursive* version of the algorithm of part (c). Do not use big-O notation. Justify your answer.

Hint: To help analyse, you can think of a color coding of the nodes: a node is white its subtree contains no value in the range; black if all values in its subtree are in the range; grey otherwise.

Solution

(a) The iterative version of the algorithm is given in Algorithm 2.

(b) The algorithm shows that the passage from parent to a child and passage from each child to parent happens only once. Hence total number of passages is two times the number of edges of the tree. Since the tree has $n-1$ edges, the number is $2(n-1)$.

(c)

Assume that *root* points to the root node. Algorithm 3 gives the recursive algorithm to output the elements in the range.

(d) The search algorithm will reach the first element in the range and subsequently it will output the range element in linear time as seen in part (b). So the total time will be at most $tree-depth+2(j_0-i_0) = 2logn+2(j_0-i_0)$.

```
u · Parent := @;
/* @ is a special symbol used for convenience                    */
while x ≠ @ do
    if prev = parent(x) AND x · Left ≠ null then
        prev := x;
        x := x · Left;
    else
        if prev = parent(x) AND x · Right ≠ null then
            prev := x;
            x := x · Right;
        else
            if prev = x · Parent then
                Output(x · value);
                prev := x;
                x := x · Parent;
            else
                if prev = x · Left AND x · Right ≠ null then
                    Output(x · value);
                    Parent := x;
                    x := x · Right;
                else
                    if prev = x · Left AND x · Right = null then
                        Output(x · value);
                        prev := x;
                        x := x · Parent;
                    else
                        prev := x;
                        x := x · Parent;
                    end
                end
            end
        end
    end
end
```

Algorithm 2: IterativeSortedAll(u)

```
if j_0 < i_0 then
  | Return;
end
k := (root · Left) · number;
if k ≥ i_0 then
  | l := min{j_0, k};
  | RangeSearch(root · Left, i_0, l);
  | if j_0 > k then
  |   | Output(root · value);
  |   | RangeSearch(root · Right, 1, j_0 − k − 1);
  | end
else
  | if i_0 = k + 1 then
  |   | Output(root · value)
  | end
  | ;
  | l := max{i_0, k + 2};
  | RangeSearch(root · Right, l, j_0 − k − 1);
end
```

Algorithm 3: RangeSearch$(root, i_0, j_0)$