Assignment : Young tableaus, one of the famous applications of the heap properties

Due Date: 03/17/2019

1)

2	4	5	16	
3	8	9	2147483647	
12	14	2147483647	2147483647	
2147483647	2147483647	2147483647	2147483647	

(Note: empty values in tableaus are **to be considered ∞ values**).

2) Argue that an m×n Young tableau Y is empty if Y[1,1]=∞. Argue that Y is full (contains mn elements) if Y[m,n]<∞.

Ans.

A young tableaus inherently maintains the property that any element is greater than or equal to its parent nodes (left node and node above).With that we can conclude, it has a smallest element at the top left corner of tableaus.

Hence, if Y[1,1]=∞ then we can say that the smallest element in the tableaus is ∞ and as there exists no element greater than ∞, the rest tableaus entries are empty. Hence its logical to argue that Young tableau Y is empty if Y[1,1]=∞.

On the other hand, if the last / bottom most element at right corner (Y[m][n]) is less than ∞ that implies, there is no more space to accommodate any new elements and all the elements currently present in tableaus are set of finite numbers (<∞.) Hence we imply that Y is full.

3) Give an algorithm to implement EXTRACT-MIN on a nonempty m×n Young tableau that runs in O(m+n) time. Your algorithm should use a recursive subroutine that solves an m×n problem by recursively solving either an (m−1)×n or an m×(n−1) subproblem. (Hint: Think about MAX-HEAPIFY.) Define T(p) where p=m+n, to be the maximum running time of EXTRACT-MIN on any m×n Young tableau. Give and solve a recurrence relation for T(p) that yields the O(m+n) time bound.

Ans.

Young tableaus has its smallest element stored at the top left corner. We remove it to get the smallest element in the tableaus, once we remove smallest element (Y[1][1]), either we can shift / re-shuffle all the elements in tableaus to maintain young tableaus

property (to have smallest number at Y[1][1] again from the remaining elements), in this approach since all the elements in rows as well as columns need to be traversed until the young tableaus property is retained, we'd get O(mn) time complexity in worst case.

However, to get better time complexity we can replace Y[1][1] with +∞ and compare with its neighboring element (either in row or column) and swap it till the young tableaus property is retained again. We achieve this by writing a recursive function by either traversing through row (**(m−1)×n**) or column (**m×(n-1)**) and swap with the smallest element till the tableaus property is retained.

Algorithm:

1> Copy the top left element Y[1][1] as the minimum element and replace it with int MAX value.
2> Int I,j=0
3> Till I < =m-1 or if tableaus property is retained, check if top element is smaller
4> If Y [i][j] > Y[i+1][j] then swap and i+=1
5> Till j <= n-1 or if tableaus property is retained, check if right element is smaller
6> If Y [i][j] > Y[i][j+1] then swap and j+=1

Below is the T.C analysis of this approach:

Consider p= (m+n).
Hence, **T (p) = T(p)+O(1)**

we either traverse through row (m) or column (n) such that it takes O(m+n) and we swap with smallest element in O(1).

By expanding the expression,

T(p)= T(p)+O(1)

=T(p-1) +O(1) +O(1)

=T(p-2)+O(1)+O(1)+O(1) …

(when we reach a condition where T(p-2) reaches O(1), expression would look like p*(O(1)) , hence T.C= O(p))

=O (p)

=O (m+n)

4)Show how to insert a new element into a nonfull m×n Young tableau in O(m+n) time

Ans.

To insert the new element in the tableaus, we insert the element at the bottom right corner of the tableaus and similar to Q(3), we compare it with its neighbors and swap when smaller element found till the young tableaus property is retained. Similar to Q(3), we would either compare it recursively with the element from same row or same column (i.e either left parent or top parent). Hence T.C. calculation is similar to Q(3), with T.C. **=O (m+n).**

Algorithm to insert new element:

1> Check if i==m and j==n , if yes, declare tableaus is full and no more insertions possible.
2> Else insert the element at the next available position.
3> Int I,j initialize to current max of row and current max of column resp.
4> Till I >0 or if tableaus property is retained, check if top element is larger
5> If Y [i][j] < Y[i-1][j] then swap and i-=1
6> Till j >0 or if tableaus property is retained, check if left element is greater
7> If Y [i][j] < Y[i][j-1] then swap and j-=1

5)Using no other sorting method as a subroutine, show how to use an n×n Young tableau to sort n^2numbers in O(n^3) time.

Ans.

In nxn empty tableaus, we first insert **n^2** elements and after every new element insert we put it in appropriate position so as to maintain young tableaus property .

To sort the newly inserted element we take **O (n+n)** time as explained in Q(3 and 4), we call function recursively to check either left parent or top parent of the element and swap till neither of them are smaller than the element.

Hence T.C to sort each new element = O (n+n)

= O(2n) …[ignore constant coefficient]

= O(n)

We've **n^2** elements in total and each of them after insertion takes O(n) time to sort , Hence for all **n^2** elements it'd take **n^2 X O(n) time**, that gives **T.C of O(n^3).**

6)Give an O(m+n)-time algorithm to determine whether a given number is stored in a given m×n Young tableau.

Ans.

This is a searching technique to find if the element is present In the tableaus. We start from the bottom left corner of the tableaus and check if current element is the element we're trying to search (say **Value**) if yes, return true and if current element < Value them move right or if current element > Value then we move up and recursively call search till either the element is found or all elements in tableaus are travelled.

Below is the algorithm for search:

1> Start with i=m-1 and j=0 (i.e. Y[m-1][0])
2> Continue till i<=0 and j <= n-1.
3> Check if Y[i][j] == value //value is the element to be searched
4> If yes, return true;
5> Else check if Y[i][j] < value
6> If yes, then j++
7> Else i--;
8> IF all tableaus elements traversed and element not found, return false.

As for each step in element search we either traverse along the row (m-1)*n or column m*(n-1) , Hence it takes **O(m+n) time** to compute search.

7) Implement all above-mentioned algorithms

Ans. (Please see attached source code).

References and Citation:

1> https://walkccc.github.io/CLRS/Chap06/Problems/6-3/
2> https://github.com/gzc/CLRS/blob/master/C06-Heapsort/young.cpp
3> https://en.wikipedia.org/wiki/Young_tableau
4> https://cs.stackexchange.com/questions/76728/extract-min-algorithm-for-young-tableau
5> https://github.com/gzc/CLRS/blob/master/C06-Heapsort/problem.md
6> http://bricesson.net/data-structures/85-young-tableaus-one-of-the-famous-applications-of-the-heap-properties.html