Handin 4

Thomas Vinther & Jens Kristian Refsgaard Nielsen

25-09-18

Young tableaus

A Young Tableau $Y = \{a_{i,j}\}_{i=1.j=1}^{m,n}$ satifies the criteria

$$a_{i,j} \leq a_{l,k} \iff 1 \leq i \leq \ell \leq m \ \vee 1 \leq j \leq k \leq n \tag{0.1}$$

because each row and column is sorted.
a. Draw a 4×4 Young tableau containing the elements $9, 16, 3, 2, 4, 8, 5, 14, 12$

$$\begin{bmatrix} 2 & 3 & 4 & 5 \\ 8 & 9 & 12 & 14 \\ 16 & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty \end{bmatrix}$$

We Simply fill up the matrix with the lowest numbers in the first row and continue down the the rows.
b. Argue that an $m \times n$ Young tableau Y is empty if Y[1,1] $= \infty$. Argue that Y is full (contains mn elements) if Y[m,n] $< \infty$.

If Y[1,1] $= \infty$ we treat Y[1,1] as an nonexistent element, pr. definition. As all the rows and the columns are in sorted order, no other element can exist in the rest of Y. Looking at Y[m,n] $< \infty$ we see that the entirety of Y is full, as once again we remember that all the rows and the columns are in sorted order, and the last entry to be filled is then obviously Y[m,n]. I.e.

$$\begin{aligned} a_{1,1} &\overset{(0.1)}{\leq} a_{j,k} \forall 1 \leq j \leq m \forall 1 \leq k \leq n \\ a_{m,n} &\overset{(0.1)}{\geq} a_{j,k} \forall 1 \leq j \leq m \forall 1 \leq k \leq n \end{aligned}$$

c. Give an algorithm to implement EXTRACT-MIN on a nonempty $m \times n$ Young tableau that runs in $O(m + n)$ time.

The Young tableau is by defintion sorted in each row and column and all we therefore need to do to extract our minimum is to extract Y[1,1] as this will always be our smallest element in Y. After extracting Y[1,1] and replacing with ∞ we of course need to resort Y and place the now smallest element in Y[1,1]. This element is either next to Y[1,1] (on the position to the right) or below it. Otherwise Y could not be in sorted order in each column and row. That is exactly what we check for in SortTabelau and as we move down either a row or a column, the smallest element of the now $(m-1) \times n$ or $m \times (n-1)$ matrix of Y must then once again be either next to it (to the right) or below it. Any out of bounds calls is defined to give back ∞ This code is heavily influenced by the HEAP-Extract-MAX code in CLRS page 163.

```
EXTRACT-MIN-TABLEAU(Y)
$min = Y[1,1]$
$Y[1,1] = \infty$
SortTableau(Y,1,1)
return $min$
```

```
SortTableau(Y,i,j)
$right = Y[i, j+1]$
$under = Y[i+1, j]$
if $right < Y[i,j]$ & & $right < under$
    exchange $right$ with $Y[i,j]$
    SortTableau(i,j+1)
if $under < Y[i,j]$
exchange $under$ with $Y[i,j]$
    SortTableau(i+1,j)
```

The first if sentence checks that the *right* element is actually the smallest of *right* and *under*. If they are equal we simply take *under*. The algorithm will stop as we reach a non exsistent element or we reach $Y[m,n]$ as the out of bounds call will return ∞.
The running time for each if-sentence is bounded by m and n and the EXTEACT-MIN-TABLEAU is only in constant time, and we therefore get that this algorithm will run in $O(n+m)$ time.

d.Show how to insert a new element into a nonfull $m \times n$ Young Tableau in $O(m+n)$ time.

As we know that the Young tableau is nonfull and once again that each row and column is sorted, and a non esisting element has the value ∞, the position Y[m,n] must be empty. We insert our new element into position Y[m,n].

```
INSERT-KEY-TABLEAU(Y,key,m,n)
$Y[m,n] = key$
KEY-INSERT-SORT(Y,m,n)
```

As we call KEY-INSERT-SORT we check in the first if statement: if the element to the left is larger than our newly inserted element and which is largest (the one above or to the left). We take the largest element and either shift it one column to the right, or in the second if statement move it down one row, thereby still maintaining the sorted order as we have only moved the largest of the two. If they are equal we simply take the one to the left. We then call KEY-INSERT-SORT again, for the new position we moved our inserted key to. We define any out of bounds call as simply $-\infty$

```
KEY-INSERT-SORT(Y,i,j)
$over = Y[i-1, j]$
$left = Y[i, j-1]$
if $left > Y[i,j]$ & & $left > over$
    exchange $left$ with $Y[i,j]$
    KEY-INSERT-SORT(Y,i,j-1)
if $over > Y[i,j]$
    exchange $over$ with $Y[i,j]$
    KEY-INSERT-SORT(Y,i-1,j)
```

This algorithm also runs in $O(m+n)$ time, as the first if statement is bounded by n, and the second by m.

e. Using no other sorting method as a subroutine, show how to use an $n \times n$ Young tableau to sort n^2 number in $O(n^3)$ time.

First we simply use our INSERT-KEY-TABLEAU algorithm. On a $n \times n$ matrix this will take $O(2n) = O(n)$ time for each number, there are n^2 which gives us $O(n^3)$ time. We then use our algorithm EXTRACT-MIN-TABLEAU which will also take $O(2n) = O(n)$ time for each number, that also gives us $O(n^3)$. In total we will have a running time of $O(n^3) + O(n^3) = O(n^3)$

f. Give an $O(m+n)$-time algorithm to determine whether a given number is stored in a given $m \times n$ Young Tableau.
We start the call with LOOK(Y,m,1,a)

```
LOOK(Y,i,j,a)
if $i < 1$ or $j > n$
    return false.
if $a < Y[i,j]$
    LOOK(Y,i-1,j,a)
if $a > Y[i,j]$
    LOOK(Y,i,j+1,a)
return true.
```

Correctness: We once again recall that a Young Tableau is sorted in both rows and columns. Therefore if the last number in a column, is larger than the number we are looking for, we can disregard the entire column.

$$a \geq a_{m,j} \overset{(0.1)}{\geq} a_{\ell,j} \; \forall \ell = 1, \dots, m$$

If the first number in a row is smaller than the number we are looking for, we can disregard the entire row as:

$$a \leq a_{i,1} \overset{(0.1)}{\leq} a_{i,\ell} \; \forall \ell = 1, \dots, n$$

That is exactly what we do in the last two if statements. In the first we check whether or not the a we are looking for could be in the last row, we simply check if a is smaller than the first element in that row, if it is strictly smaller, it cannot be in that row, due to (0.1) we then recursively call LOOK where we start one row higher up. We make a similiar argument for the next if statement, but this time we consider the columns. If both if statements are false, we clearly have $a = Y[i,j]$

The first if statement ensures that the algorithm terminates in final time. In the worst case the algorithm makes $m + n - 1$ recursive calls to itself, which gives us the desired $O(m + n)$