Young tableaus(applications of the heap properties)

Kexin Ding

Please check the integrated code in young.py

1.Draw 4×4 tableau containing the elements {9,16,3,2,4,8,5,14,12}

2	3	5	14
4	8	9	16
12	∞	∞	∞
∞	∞	∞	∞

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]<∞.

If Y[1,1] = ∞, it means all elements in the first row and column are ∞. Y[1,1] is the smallest element in Y. The rest element of Y will be larger than Y[1,1]. If Y[1,1] = ∞, other elements in Y will also equal to ∞ to satisfy the rules of Young tableau. Hence, the Young tableau Y is empty.

If Y[m, n] < ∞, it means the last "children" element(the bottom right element) is not ∞. And this bottom right element is the largest element in the Y. Each element is smaller than Y[m, n]. If Y[m, n] is smaller than ∞, each element in Y will smaller than ∞. Hence Y is not empty.

3. Give an algorithm to implement EXTRACT-MIN.

Time complexity:

$$\begin{aligned} T(p) &= T(p-1) + O(1) \\ &= T(p-2) + O(1) + O(1) \\ &= T(p - (p - 1)) + (p - 1) * O(1) \\ &= T(1) + (p - 1) * O(1) \\ &= p * O(1) \\ &= O(p) \\ &= O(m + n) \end{aligned}$$

Result:

```
ExtractMin
Young tableaus:
        2          3          5          14
        4          8          9          16
       12        INF        INF         INF
      INF        INF        INF         INF
Minimum number be extracted:  2
```

Code:

def Young(Y, i, j, m, n):

```
        x = i
        y = j
        if i < m and Y[i][j] > Y[i + 1][j]:
                x = i + 1
                y = j

        if j < n and Y[x][y] > Y[i][j + 1]:
                x = i
                y = j + 1

        if x != i or y != j:
                Y[i][j], Y[x][y] = Y[x][y], Y[i][j]
                Young(Y, x, y, m, n)

def ExtractMin(Y, m, n):
        tmp = Y[0][0]
        Y[0][0] = sys.maxsize
        young(Y, 0, 0, m, n)
        return tmp
```

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

Time Complexity:

The similar algorithm as 3. But an inversing process.

$$\begin{aligned} T(p) &= T(p-1) + O(1) \\ &= T(p-2) + O(1) + O(1) \\ &= T(p - (p - 1)) + (p - 1) * O(1) \\ &= T(1) + (p - 1) * O(1) \\ &= p * O(1) \\ &= O(p) \\ &= O(m + n) \end{aligned}$$

Result:

```
insert value to build Young tableaus
Young tableaus after inserting values
        2           3           5          14
        4           8           9          16
       12         INF         INF         INF
      INF         INF         INF         INF
```

Code:

```
def Inverse(Y, m, n):
        i = m
        j = n
        if m>0 and Y[m][n] < Y[m-1][n]:
```

```
        i = m - 1
        j = n

    if n>0 and Y[i][j] < Y[m][n-1]:
        i = m
        j = n - 1

    if i!=m or j!=n:
        Y[m][n], Y[i][j] = Y[i][j], Y[m][n]
        Inverse(Y, i, j)

def Insert(Y, m, n, value):
    if m < 0 and n < 0:
        return
    Y[m][n] = value
    Inverse(Y, m, n)
```

5. Using no other sorting method as a subroutine, show how to use an n×n Young tableau to sort n2 numbers in O(n3) time.

Inserting is actually an in-order operation. It will maintain the order in its algorithm. For inserting one element, we need to spend O(m + n) = O(n + n) = O(n). For inserting n^2 elements, we need to spend $n^2 O(n)$ = 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.

Time complexity:

$$
\begin{aligned}
T(p) &= T(p - 1) + O(1) \\
&= T(p - 2) + O(1) + O(1) \\
&= T(p - (p - 1)) + (p - 1) * O(1) \\
&= T(1) + (p - 1) * O(1) \\
&= p * O(1) \\
&= O(p) \\
&= O(m + n)
\end{aligned}
$$

Result:

```
Sorting
previous list is:
[9, 16, 3, 2, 4, 8, 5, 14, 12]
Sorted list is:
[2, 3, 4, 5, 8, 9, 12, 14, 16]


Searching
array is:
[9, 16, 3, 2, 4, 8, 5, 14, 12]
Search number: 3  search result:  True
Search number: 50  search result:  False
```

Code:

```
def YoungSearch(Y, i, j, m, n, value):
    if Y[i][j] == value:
        return True
    x = i
    y = j
    if i > 0 and Y[i][j] > value:
        x = i - 1
        y = j

    if j < n and Y[i][j] < value:
        x = i
        y = j + 1

    if x==i and y==j:
        return False


    return YoungSearch(Y,x,y,m,n,value)
def search(Y, m, n, value):
    return YoungSearch(Y, m, 0, m, n, value)
```