FINDING THE PERMUTATIONS CORRESPONDING TO A GIVEN YOUNG TABLEAU

Erkki Mäkinen and Jyrki Nummenmaa

Abstract. We consider the problem of finding all the permutations corresponding to a given Young tableau. We introduce a recursive algorithm, deleting the elements from the tableau in reverse order which were inserted by the Schensted algorithm.

1. Introduction. Studying Young tableaux and their relationship with permutations provides an excellent topic in undergraduate data structures and combinatorics courses. This paper concentrates on the problem of finding the permutations related to a given Young tableau. While the algorithms have been presented before, at least implicitly, in the literature, we believe that our presentation gives an easy-to-understand view to an interesting problem that can be utilized in teaching. Further information about Young tableaux can be found in [2, 4, 6]. Our interest in Young tableaux originates from The International Olympiad in Informatics 2001 [5].

2. Young Tableaux. A Young tableau of shape $(n_1, n_2, \ldots, n_m)$, where $n_1 \geq n_2 \geq \cdots \geq n_m > 0$, is an arrangement of $n_1 + n_2 + \cdots + n_m$ distinct integers in an array with m rows such that in row i there are n_i elements, each row is in increasing order from left to right, and each column is in increasing order from top to bottom. Basic results about Young tableaux can be found in [4].

There is a close connection between Young tableaux and permutations. Given a permutation $P = (p_1, \ldots, p_n)$, the corresponding Young tableau is constructed by inserting the elements $p_1, \ldots, p_n$ one by one in an originally empty tableau. Inserting p_i is performed as follows. First, find in row 1 the least element greater than p_i, if any, and place p_i in the spot of that element. If p_i is greater than all elements in row 1, insert it to be the last element of row 1, and halt. Otherwise, continue by inserting the element, say r, replaced by p_i, in row 2. Again, halt if r is the greatest in the row. Otherwise, continue similarly with the next replaced element and the next row, until a row is found where all elements are smaller than the new element to be inserted or a new row is started. This insertion algorithm is called the *Schensted algorithm*.

Consider, for example, constructing the Young tableau for the permutation

$$\begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 3 & 5 & 4 & 9 & 8 & 2 & 7 & 6 & 1 \end{pmatrix}.$$

After inserting 3, 5, 4, 9, 8, 2, and 7, the tableau has the form shown in Figure 1.

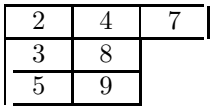

Figure 1. The tableau after inserting 3, 5, 4, 9, 8, 2, and 7 (in this order).

Inserting 6 causes first the replacement of 7 by 6 in row 1. Then, 8 is replaced by 7 in row 2, and 9 is replaced by 8 in row 3. Finally, a new row is started with 9 as the only element. The resulting tableau is shown in Figure 2.

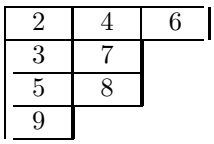

Figure 2. The tableau of Figure 1 after inserting 6.

Inserting 1 in the tableau obtained causes changes only in the first column: 2 replaced by 1, 3 replaced by 2, 5 replaced by 3, 9 replaced by 5, and a new row with 9 as the only element is started.

The resulting tableau is shown in Figure 3.

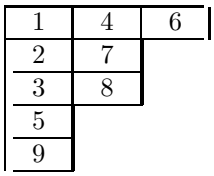

Figure 3. The complete tableau.

The following well-known theorem establishes a 1-to-1 correspondence between permutations and Young tableaux.

Theorem 2.1. [6] Given a pair (P, Q) of Young tableaux having the same shape with elements $1, 2, \ldots, n$, there is a unique permutation p corresponding to (P, Q) such that inserting the elements of p in an originally empty tableau by using the Schensted algorithm gives P and, moreover, Q contains the order in which the positions of P are filled.

Given a single Young tableau P, there are usually several permutations that could produce P. The number of Young tableaux of a given shape containing the integers $1, 2, \ldots, n$ is $n!$ divided by the product of the hook lengths of its elements [1]. (The *hook length* of an element x in a Young tableau is the number of elements counting from the bottom of a column to a given element and then to the right end of the row.)

Given a Young tableau P, we can always create a permutation that produces P by reading the rows from right to left starting from the bottom row and proceeding towards the top row. If P is the tableau in Figure 3, then this permutation corresponds to the pair (P, Q_1) where Q_1 is the tableau in Figure 4.

1	2	3
4	5	
6	7	
8		
9		

Figure 4. Tableau Q_1.

Another permutation is obtained by reading the columns from bottom to top starting from the rightmost column and proceeding to the left (excluding the two special cases where these methods result in the same permutation). The corresponding pair of Young tableaux is (P, Q_2), where Q_2 is the tableau in Figure 5.

In the original Schensted algorithm, it is easy to work backward from a given pair (P, Q) of tableaux to get the original permutation, because the Q tableau tells us which cell was added last and therefore, which cell to remove first. In this paper we show how all possible permutations can be found starting from the P tableau alone.

1	6	9
2	7	
3	8	
4		
5		

Figure 5. Tableau Q_2.

3. Producing the Permutations. A straightforward approach to solve the problem is to write a recursive algorithm that, in reverse order, deletes elements from a given tableau until the tableau becomes empty

again and the elements are back in the permutation. We start by making some observations concerning Young tableaux.

First, we notice that the element last inserted is always in the first row. When finding the element y which was bumped down from the spot of the last inserted element x, we know that $x < y$. More precisely, we have the following little result.

Proposition 3.1. If x is the last inserted item, y is the item replaced by x, and z is the item following x in the same row (provided that such an element exists), then we have $z > y > x$.

When all the elements left in the tableau are in the topmost row, we know that they were inserted in the reserve order in which they appear in the row.

The process of bumping down an element can halt when an element is inserted in the end of a row. Further, since each column of a Young tableau is always in ascending order from top to bottom, we notice that when bumped down, an element never moves to the right. It follows that row i is always at least as long as row $i + 1$.

Proposition 3.2. Given a Young tableau produced by the Schensted algorithm, the element last bumped down is in the end of a row that is longer than the row below it or in the end of the last row.

The cells containing the elements referred to in Proposition 3.2 are called *removable cells*. As an example, consider the tableau in Figure 3. The removable cells in this tableau contain the elements 6, 8, and 9.

By the definition of Young tableaux we have the following proposition.

Proposition 3.3. A cell can be deleted from a Young tableau if and only if it is removable.

The remarks above suggest that we should delete the cells in bottom-up style, that is, by starting from the removable cells. This gives us the following recursive algorithms.

```
BumpUp(P: tableau; x: element; r: row);
begin
  if r = 1
  then
    delete x from P and insert it in the beginning of the resulting
    permutation;
    MakeEmpty(P)
  else
    let y be the unique element (by Proposition 3.1) in row r − 1
    that could have been replaced by x by the Schensted algorithm;
    replace y by x;
    if r = 2
      then
        insert y in the beginning of the resulting permutation;
        MakeEmpty(P)
    else
        BumpUp(P,y,r − 1)
end
```

```
MakeEmpty (P: tableau);
begin
  if there are elements left in P
  then for each removable cell c in P do
    let x be the element in c and let c be in row r;
    delete c;
    BumpUp(P,x,r);
    remove the first element from the permutation and insert it back
    to P using the Schensted algorithm;
  else output the resulting permutation found
end
```

The desired permutations can now be computed by calling $MakeEmpty(P)$ where P is the input tableau.

Consider again the tableau in Figure 3 as an example. The removable cells contain the elements 6, 8, and 9. Starting with 6 would cause 6 to be removed from the tableau since it is in the first row. Starting with 8 or 9 would, in turn, cause recursive calls for the $BumpUp$ procedure. If the removable cell is originally in the ith row of the tableau, deleting an element from the tableau requires $i - 1$ $BumpUp$ calls.

Notice further that our algorithm does not fix the order in which the removable cells are handled. In what follows, we trace the execution of $MakeEmpty(P)$, where P is the tableau in Figure 3 starting from the removable cell containing the element 9. The element 9 is first deleted from

the tableau. Then, it is put in the spot of 5, which in turn, is put in the spot of 3. The element 3 is put in the spot of 2, and now we are in the second row of P. The unique element in the first row is 1, which means that 2 is put in the spot of 1 and 1 is inserted in the beginning of the resulting permutation. The tableau is now as shown in Figure 2. Again, there are three removable cells containing the elements 6, 8, and 9, which means that the execution is further divided into three branches. We trace the branch starting by deleting the removable cell containing 9. The element 9 is put in the spot of 8, which is put in the spot of 7. We are in the second row and the unique element in the first row is 6. The element 7 is put in the spot of 6 and 6 is inserted in the beginning of the resulting permutation. The tableau is in the form shown in Figure 1. Now there are only two removable cells, namely those containing 7 and 9. In both cases the next element to be inserted in the resulting permutation is 7, but the tableau obtained has a different shape depending on the chosen removable cell. If we choose to delete the removable cell containing 7, we obtain the tableau in Figure 6.

2	4
3	8
5	9

Figure 6. Tableau P after deleting the removable cell containing 7.

On the other hand, if we choose to delete the removable cell containing 9, we obtain the tableau in Figure 7.

2	4	8
3	9	
5		

Figure 7. Tableau P after deleting the removable cell containing 9.

In the former case, there is only one removable cell (the one containing 9), while in the latter case, there are three removable cells containing the elements 5, 8, and 9. Hence, the execution is again divided into three branches in the latter case. This process continues as long as there are elements left in the tableau. When the tableau is empty, the elements are in the resulting permutation in the correct order.

We now observe how our algorithm will proceed, given the tableau of Figure 3 as input and assuming that it treats the removable cells in order starting from the bottommost row. This means that the algorithm will first start bumping up 9, which will replace 5 in row 4, which in turn will

replace 3 in row 3, which in turn replaces 2 in row 2, and 2 will replace 1 and cause 1 to be the first element to be added to the permutation. After this, *MakeEmpty* will be called with the input tableau shown in Figure 8.

2	4	6
3	7	
5	8	
9		

Figure 8. The tableau of Figure 3 after deleting 1.

The algorithm will, again, start up with the cell containing 9. After bumping up 9, 8, and 7, and adding 6 to the permutation, we get the tableau in Figure 9.

2	4	7
3	8	
5	9	

Figure 9. The tableau of Figure 8 after deleting 6.

In the following round, 9 and 8 are bumped up and 7 is added to the permutation. Then, 5 and 3 are bumped up, and 2 is added to the permutation, and then 9 is bumped up and 8 is added to the permutation. The tableau obtained is shown in Figure 10.

3	4	9
5		

Figure 10. The tableau of Figure 9 after deleting 7, 2, and 8.

Then, 5 is bumped up and 4 is added to the permutation. Now there is only one row remaining and it contains the numbers 3, 5 and 9, and we know that they must have been inserted in this order. Therefore, the first permutation is (3 5 9 4 8 2 7 6 1).

To find the next permutation, we must reverse to a point where there were at least two removable cells, which is the tableau of Figure 10. There, the previously selected removable cell contains 5, but another removable cell contains 9. By selecting 9 first and continuing as before, we get the next permutation: (3 5 4 9 8 2 7 6 1), and this branch of computation with 9 selected, also gives us the permutation (5 3 4 9 8 2 7 6 1), before we have

to backtrack further. The interested reader is encouraged to download and try out the Pascal source programs for this implementation [7].

Having demonstrated the way our algorithm works by examples, we shall now prove the algorithm correct.

Theorem 3.1. Let P be a Young tableau. Calling $MakeEmpty(P)$ produces correctly all such permutations p that the Schensted algorithm outputs P, when given p as an input.

Proof. For induction, it is clear that the algorithm works correctly when the input tableau contains only one element.

Assume, now, that the algorithm works correctly whenever the tableau contains $n-1$ elements. Suppose that the input tableau contains n elements. It suffices to show that the algorithm correctly finds all elements that could have been inserted last to the tableau and calls $MakeEmpty$ for all tableaux with one of those elements removed.

By Proposition 3.2, finding the removable cells is straightforward. Let a be the element in the removable cell chosen.

First of all, from the discussion above it is clear that by starting from a, calling $BumpUp$ will locate an item b in the top row such that the insertion of b would have pushed a to the removable cell. By Proposition 3.1, when working its way upward, $BumpUp$ will always find a unique item in the row above. We now want to show that for all items in the removable cells, we find a different item in the top row. If this would not be the case, then apparently we would be producing redundant permutations.

It is enough to see that whenever we move one row upwards, the unique elements of Proposition 3.1 are different for items in the current row. Let us assume, by contradiction, that this would not be the case. Then in some row we have items y_1 and y_2 such that x is the unique element of Proposition 3.1 for both of them. Without loss of generality, we assume $y_1 < y_2$. By Proposition 3.1, $x < y_1 < y_2 < z$. Now, y_1 and y_2 are in the same row and because $y_1 < y_2$, y_1 is to the left of y_2. Since items do not move to the right when they are bumped down, we know that x is no further right than y_1. As rows above are always at least as long as rows below, and columns are in ascending order, we know that there must be an item x' to the right of x such that $x < x' < y_2$. Therefore, x cannot be the unique item of Proposition 3.1 for y_2, and a contradiction follows.

Therefore, we have established a one-to-one relationship between the removable cells (respectively, their items) and items in the top row, whose insertion as the last item would have caused the creation of the removable cell with its item.

Finally, as the last inserted item is just inserted first in the permutation, removing it from the permutation recovers the permutation, and since $BumpUp$ does the reverse from the Schensted algorithm, calling the Schensted algorithm with the first item of the permutation recovers the tableaux to the form where it was before calling $BumpUp$. As $MakeEmpty$ calls

BumpUp for all elements in the removable cells, it finds all permutations correctly.

It is also possible to produce the permutations corresponding to a given Young tableau by using the transformations defined by Knuth [3].

Acknowledgements. The authors wish to thank Isto Aho, Tero Karras, Janne Kujala, and Samuli Laine for their help.

References

1. J. S. Frame, G. de B. Robinson, and R. M. Thrall, "The Hook Graphs of the Symmetric Group," *Canad. J. Math.,* 6 (1954), 316–324.
2. W. Fulton, *Young Tableaux: With Applications to Representation Theory and Geometry,* Cambridge University Press, Cambridge, England, 1997.
3. D. E. Knuth, "Permutations, Matrices, and Generalized Young Tableaux," *Pacific J. Math.,* 34 (1970), 709–727.
4. D. E. Knuth, *The Art of Computer Programming, Sorting and Searching,* Vol. 3, Second Edition, Addison-Wesley, Reading, MA, 1998.
5. J. Nummenmaa, E. Mäkinen, and I. Aho (eds.), "IOI'01. Report A-2001-7," Dept. of Computer and Information Sciences, University of Tampere, Tampere, Finland. (Available also at http://www.cs.uta.fi/reports/pdf/A-2001-7.pdf)
6. C. Schensted, "Longest Increasing and Decreasing Subsequences," *Canad. J. Math.,* 13 (1961), 179–191.
7. http://www.cs.uta.fi/~jyrki/young/

Mathematics Subject Classification (2000): 68R05, 05E10

Erkki Mäkinen
Department of Computer Sciences
P. O. Box 607
FIN-33014
University of Tampere
Tampere, Finland
email: em@cs.uta.fi

Jyrki Nummenmaa
Department of Computer Sciences
P. O. Box 607
FIN-33014
University of Tampere
Tampere, Finland
email: jyrki@cs.uta.fi