Finding pattern matchings for permutations

Louis Ibarra
Dept. of Computer Science
Hill Center, Busch Campus
Rutgers University
Piscataway, NJ 08855
ibarra@paul.rutgers.edu

January 19, 1995

Abstract

Given a permutation P of $\{1, \ldots, k\}$ and T of $\{1, \ldots, n\}$, the pattern matching problem for permutations is to determine whether there is a length k subsequence of T whose elements are ordered in the same way as the elements of P. We present an $O(kn^4)$ time and $O(kn^3)$ space algorithm for finding a match of P into T or determining that no match exists, given that P is *separable*, i.e. contains neither (2, 4, 1, 3) nor (3, 1, 4, 2) as a subpattern.

1 Introduction

Let $T = (t_1, \ldots, t_n)$ be a permutation of $\{1, \ldots, n\}$, called the *text*, and let $P = (p_1, \ldots, p_k)$ be a permutation of $\{1, \ldots, k\}$, called the *pattern*. A *match* of P into T is a subsequence T' of T that is ordered in the same way as P, i.e. $T' = (t_{i_1}, ..., t_{i_k})$ such that $i_1 < \cdots < i_k$ and $t_{i_r} < t_{i_s}$ iff $p_r < p_s$ for all $1 \leq r, s \leq k$. If there is a match of P into T, then we say that T *contains* P. The *pattern matching problem* for permutations is to decide if there is a match of P into T. The general decision problem is NP-complete, but there is a polynomial time algorithm when P is *separable*, i.e. P contains neither the pattern (2, 4, 1, 3) nor (3, 1, 4, 2) [1]. The significance of separable permutations is the following.

Given a permutation $P = (p_1, \ldots, p_k)$, a *separating tree* for P is a binary tree with the following properties: (1) the leaves are $(p_1, \ldots, p_k)$ in this order, (2) for any node V, the leaves of the subtree rooted at V form a subrange of $\{1, \ldots, k\}$, i.e. if the leaves of the subtree are $(p_{i_1}, \ldots, p_{i_j})$, then $\{p_{i_1}, \ldots, p_{i_j}\} = \{l, l+1, \ldots, l+m\}$ for some $1 \leq l \leq k$ and $0 \leq m \leq k-l$. For example, a separating tree of $P = (6, 3, 2, 1, 4, 5)$ is (6, ((3, (2, 1)), (4, 5))). The set of leaves of the subtree rooted at V is the *range* of V. It follows from the definition that if node V has left child V_l and right child V_r, then the range of V_l immediately precedes or succeeds the range of V_r; in the first case, V is a *positive* node and in the second case, V is a *negative* node. The key result is that a permutation P is separable iff P has a separating tree [1, 2]. Furthermore, there is a linear time algorithm to decide if a permutation P is separable and if so, to construct a separating tree for P [1].

The algorithm for the pattern matching problem for permutations in [1] counts the number of matches of P into T and runs in $O(kn^6)$ time with $O(kn^4)$ space, given that P is separable. The algorithm uses dynamic programming by defining a set of subproblems for each node of P's separating tree, so that the subproblems at an internal node can be solved from the subproblem

solutions for its children. In this paper, we present an algorithm that finds a match of P into T or determines that no match exists, in $O(kn^4)$ time with $O(kn^3)$ space, given that P is separable. Our approach is also dynamic programming, but there are fewer subproblems per node of the tree and computing the solutions for an internal node from the solutions for its children is done more quickly than in [1].

We will use some additional definitions. We slightly generalize the definition of pattern matching for permutations as follows. Let $Q = (q_1, \ldots, q_l)$ be a sequence of distinct positive integers. We define a match of Q into $T = (t_1, \ldots, t_n)$ in the same way as a match of P into T, namely, a *match* of Q into T is a subsequence $T' = (t_{i_1}, \ldots, t_{i_l})$ of T such that $i_1 < \cdots < i_l$ and $t_{i_r} < t_{i_s}$ iff $q_r < q_s$ for all $1 \leq r, s \leq l$. If $T' = (t_{i_1}, \ldots, t_{i_l})$ is a match of Q into T, then we say that $t_{i_1}, \ldots, t_{i_l}$ are its *matched* elements. For brevity, we use V to mean both a node V and the sequence of leaves of the subtree rooted at V. Thus, we can refer to a match of V into T and to its matched elements.

2 The algorithm

The input is a permutation $T = (t_1, \ldots, t_n)$, a separable permutation $P = (p_1, \ldots, p_k)$ and a separating tree for P.

For every node V, we define the values $L(V, i, j, x)$ and $H(V, i, j, x)$ for $1 \leq i \leq j \leq n$ and $1 \leq x \leq n$ as follows.

$L(V, i, j, x) = \max \{0\} \cup \{y : y$ is the smallest matched element in a match of V into $(t_i, \ldots, t_j)$ with every matched element $\leq x\}$

Thus, $L(V, i, j, x) > 0$ iff there is a match of V into $(t_i, \ldots, t_j)$ with every matched element $\leq x$ and furthermore, $L(V, i, j, x)$ is the maximin matched element over all such matches. Notice that for all $x < y$, $L(V, i, j, x) \leq L(V, i, j, y)$.

$H(V, i, j, x) = \min \{n+1\} \cup \{y : y$ is the largest matched element in a match of V into $(t_i, \ldots, t_j)$ with every matched element $\geq x\}$

Thus, $H(V, i, j, x) < n + 1$ iff there is a match of V into $(t_i, \ldots, t_j)$ with every matched element $\geq x$ and furthermore, $H(V, i, j, x)$ is the minimax matched element over all such matches. Notice that for all $x < y$, $H(V, i, j, x) \leq H(V, i, j, y)$.

We now show how to compute the L, H values. We assume that all L (resp. H) values are initialized to 0 (resp. $n + 1$). The pseudo-code uses the subroutines update$_L$ and update$_H$ (where update$_H$ is symmetric to update$_L$ with $<$ replacing $>$).

```
update_L(val, y):
    if y > val
    then val ← y
```

Let V be a leaf. The following pseudo-code computes $L(V, i, j, x)$ for fixed i, j and for all x.

```
for k = i to j do                                   (1)
    L(V, i, j, t_k) ← t_k

best = L(V, i, j, 1)
for x = 2 to n do                                   (2)
    if best < L(V, i, j, x)
        best ← L(V, i, j, x)
    update_L(L(V, i, j, x), best)
```

The loop in (1) considers every match of V into $(t_i, \ldots, t_j)$. When (1) halts, the non-zero elements of $(L(V,i,j,1), \ldots, L(V,i,j,n))$ satisfy the rule $x < y \Rightarrow L(V,i,j,x) \leq L(V,i,j,y)$, but it is possible that a non-zero is followed by a zero. When (2) halts, no non-zero is followed by a zero. (The code for $H(V,i,j,x)$ is symmetric: $>$ replaces $<$ and the second loop runs from $n-1$ down to 1.)

Let V be a node with left child V_l and right child V_r. If $i = j$, then $L(V,i,j,x)$ is 0 and $H(V,i,j,x)$ is $n+1$. If $i < j$, then the following pseudo-code computes $L(V,i,j,x)$ and $H(V,i,j,x)$ for all x from the values of $L(V_l,i,j,x)$ and $H(V_r,i,j,x)$. (We have omitted the final loop for H, which is symmetric to (2).)

```
for k = i + 1 to j do                                        (1)
      for x = 2 to n do
            if V is a positive node
            then  low = L(V_l, i, k − 1, x − 1)
                  high = H(V_r, k, j, x)
            else  high = H(V_l, i, k − 1, x)
                  low = L(V_r, k, j, x − 1)
            if low > 0 and high < n + 1
            then  update_L(L(V, i, j, high), low)
                  update_H(H(V, i, j, low), high)

best = L(V, i, j, 1)
for x = 2 to n do                                            (2)
      if best < L(V, i, j, x)
            best ← L(V, i, j, x)
      update_L(L(V, i, j, x), best)
```

We show by induction that the algorithm correctly computes the L values for V; the correctness proof for the H values is symmetric.

Lemma 1 *If $L(V,i,j,z) > 0$ when the algorithm halts, then there is a match of V into $(t_i, \ldots, t_j)$ with every matched element $\leq z$ and with smallest matched element $\geq L(V,i,j,z)$.*

Proof Suppose $L(V,i,j,z) > 0$ when the algorithm halts. There are two cases.

Case a: (2) does not increase $L(V,i,j,z)$. Consider the last iteration of (1) which increases $L(V,i,j,z)$. Since $low > 0$ and $high < n+1$ in this iteration, then by induction there is a match of V into $(t_i, \ldots, t_j)$ with largest matched element $= high$ and smallest matched element $= low$. By choice of iteration, $z = high$. Thus, $L(V,i,j,z) = L(V,i,j,high) = low$ when the iteration ends. Since (2) does not increase $L(V,i,j,z)$, then there is a match of V into $(t_i, \ldots, t_j)$ with largest matched element $= z$ and smallest matched element $= L(V,i,j,z)$, which proves the claim.

Case b: (2) does increase $L(V,i,j,z)$. Then there is a smallest y such that $y < z$ and $L(V,i,j,y) > 0$ when (1) halts and $L(V,i,j,y) = L(V,i,j,z)$ when (2) halts. Since y is the smallest, then $L(V,i,j,y)$ is not increased by (2). By case a, there is a match of V into $(t_i, \ldots, t_j)$ with largest matched element $= y < z$ and smallest matched element $= L(V,i,j,y) = L(V,i,j,z)$, which proves the claim. □

Lemma 2 *If there is a match of V into $(t_i, \ldots, t_j)$ with every matched element $\leq z$, then $L(V, i, j, z) > 0$ when the algorithm halts and furthermore, $L(V, i, j, z) \geq$ the smallest matched element.*

Proof We consider the case when V is a positive node; the argument when V is a negative node is symmetric. Suppose there is a match of V into $(t_i, \ldots, t_j)$ with every matched element $\leq z$ and smallest matched element $= y$. Then for some k, there is a match of V_l into $(t_i, \ldots, t_{k-1})$ with smallest matched element $= y$ and a match of V_r into $(t_k, \ldots, t_j)$ with largest matched element $\leq z$, and furthermore there is an x such that every matched element of V_l is $\leq x - 1$ and every matched element of V_r is $\geq x$. By induction, $L(V_l, i, k-1, x-1) \geq y$ and $H(V_r, k, j, x) \leq z$. Consider the iteration of (1) corresponding to these values of k, x, and let $low', high'$ be the values of $low, high$ in this iteration, respectively. When this iteration ends, $L(V, i, j, high') \geq low' = L(V_l, i, k-1, x-1) \geq y$. Since an L value is never decreased, then $L(V, i, j, high') \geq y$ when (1) halts. Since $high' = H(V_r, k, j, x) \leq z$, then $L(V, i, j, z) \geq L(V, i, j, high') \geq y$ when (2) halts, which proves the claim. □

Once all the L, H values are computed, there is a match of P into T iff $L(root, 1, n, n) > 0$ iff $H(root, 1, n, 1) < n + 1$. (In fact, $L(root, 1, n, n)$ and $H(root, 1, n, 1)$ are the best lower and upper bounds on the matched elements of a match of P into T, respectively.) We can recover a match by recording the values of k and x whenever $L(V, i, j, x)$ and $H(V, i, j, x)$ are changed by update.

Since there are k leaves and computing the L, H values for each leaf requires $O(n)$ time for each i, j pair, then computing the L, H values for all leaves requires $O(kn^3)$ time. Since there are $O(k)$ nodes in the tree and computing the L, H values for an internal node requires $O(n^2)$ time for each i, j pair, then computing all the L, H values is done in $O(kn^4)$ time. Thus, the algorithm solves the pattern matching problem for permutations in $O(kn^4)$ time using $O(kn^3)$ space.

References

[1] P. Bose, J.F. Buss, and A. Lubiw. Pattern matching for permutations. In *Proc. Workshop on Algorithms and Data Structures, Lecture Notes in Computer Science 709*, pages 200–209, New York, 1993. Springer-Verlag.

[2] D.G. Corneil, H. Lerchs, and L.S. Burlingham. Complement-reducible graphs. *Discrete Applied Math*, 3:163–174, 1981.