An Optimal Algorithm for the Range Maximum-Sum Segment Query Problem

Kuan-Yu Chen and Kun-Mao Chao

Department of Computer Science and Information Engineering
National Taiwan University, Taipei, Taiwan 106
{r92047, kmchao}@csie.ntu.edu.tw

Abstract

We are given a sequence A of n real numbers which is to be preprocessed. In the Range Maximum-Sum Segment Query (RMSQ) problem, a query is comprised of two intervals $[i, j]$ and $[k, l]$ and our goal is to return the maximum-sum segment of A where the staring index of the segment lies in $[i, j]$ and the ending index lies in $[k, l]$. We provide the rst known optimal algorithm with $O(n)$ preprocessing time and $O(1)$ query time.

Keywords: RMQ, maximum sum interval, sequence analysis

1 Introduction

Sequence analysis in bioinformatics has been studied for decades. One important line of investigation in sequence analysis is to locate the biologically meaningful segments, like conserved regions or GC-rich regions in DNA sequences. A common approach is to assign a real number(also called scores) to each residue, and then look for the maximum-sum or maximum-average segment [3, 5, 10].

Ruzzo and Tompa [12] proposed a linear time algorithm for nding all maximal scoring subsequences. Huang [9] extended the well-known recurrence relation used by Bentley [2] for solving the maximum sum consecutive subsequence problem, and derived a linear-time algorithm for computing the optimal segments with lengths at least L. Lin, Jiang, and Chao [10] and Fan *et al.* [5] studied the maximum-sum segment problem with length at least L and at most U.

In this paper, we consider a more general problem in which we wish to nd the maximum-sum segment whose staring and ending indices of the segment lie in given intervals. By preprocessing the sequence in $O(n)$ time, each query can be answered in $O(1)$ time. This also yields an alternative linear-time algorithm for computing the maximum-sum segment with length constraints.

The rest of the paper is organized as follows. Section 2 gives the formal de nition of the RMSQ problem and introduces the RMQ techniques [6]. Section 3 coping with the special case of the RMSQ problem(called the SRMSQ problem). Section 4 gives the optimal algorithm for the RMSQ problem.

2 Problem Definition and Preliminaries

The input is a sequence $A = \langle a_1, a_2, \ldots, a_n\rangle$ of (not necessarily positive) real numbers. The maximum-sum segment of A is the contiguous subsequence having the greatest total sum, where the sum of a subsequence $S(i, j) = \langle a_i, \ldots, a_j\rangle$ is simply $w(S(i, j)) = \sum_{k=i}^{j} a_k$. For simplicity, throughout the paper, the term "subsequence" will be taken to mean "contiguous subsequence". To avoid ambiguity, we disallow nonempty, zero-sum pre x or su x (also called ties) in the maximum-sum segments. We de ne $c[i] = \sum_{k=1}^{i} a_k$ as the cumulative sum of A for all $1 \leqslant i \leqslant n$ and $c[0] = 0$. Notice that $w(S(i, j)) = c(j) - c(i - 1)$.

As an example, consider the input sequence $A = \langle 4, -5, 2, -2, 4, 3, -2, 6\rangle$. The maximum-sum segment of A is $M = \langle 4, 3, -2, 6\rangle$, with a total sum of 11. There is another subsequence tied for this sum by appending $\langle 2, -2\rangle$ to the left end of M, but this subsequence is not the maximum-sum segment, since it has a nonempty zero-sum pre x.

2.1 Problem De nition

We start with a special case of the RMSQ problem, called the SRMSQ problem.

Problem 1. *A Special Case of the RMSQ problem(SRMSQ)*
Structure to Preprocess: $A = \langle a_1, a_2, \ldots, a_n \rangle$ *is a sequence of* n *real numbers.*
Query: *For an interval* $I = (i, j)$, $1 \leqslant i \leqslant j \leqslant n$, *SRMSQ*$_A(i, j)$ *returns a pair of indices* (x, y), $i \leqslant x \leqslant y \leqslant j$, *such that segment* $S(x, y)$ *is the maximum-sum segment of the subsequence* $\langle a_i, \ldots, a_j \rangle$.

A naïve algorithm is to build a $n \times n$ table storing answers to all of the $O(n^2)$ possible queries. Each entry (i, j) in the table represents an interval (i, j) so we only have to ll in the upper-triangular part of the table. By applying the well known linear time algorithm for nding the maximum-sum segments of a sequence, we have an $O(n^3)$ preprocessing algorithm. Notice that answering a SRMSQ query now requires just one lookup to the table. To achieve $O(n^2)$ preprocessing rather than the $O(n^3)$ naïve preprocessing, we use the online property of the algorithm, lling the table row-by-row. In the paper, our algorithm can achieve $O(n)$ time and space preprocessing, and $O(1)$ time for each query.

Problem 2. *the Range Maximum-Sum Segment Query problem(RMSQ)*
Structure to Preprocess: $A = \langle a_1, a_2, \ldots, a_n \rangle$ *is a sequence of* n *real numbers.*
Query: *For two intervals* $S = (i, j)$ *and* $E = (k, l)$, $1 \leqslant i \leqslant j \leqslant n$ *and* $1 \leqslant k \leqslant l \leqslant n$, *RMSQ*$_A$*(i,j,k,l) returns a pair of indices* (x, y), *such that* $w(S(x, y))$ *is maximized for* $i \leqslant x \leqslant j$ *and* $k \leqslant y \leqslant l$.

This is a generalized version of the SRMSQ problem because when $i = k$ and $j = l$ we are actually querying SRMSQ$_A(i, j)$. The naïve algorithm will have to build a 4-dimensional table and the time complexity for preprocess will achieve $\Omega(n^4)$. This is de nitely ine cient for practical use. We will also provide an algorithm with $O(n)$ preprocessing time and $O(1)$ query time.

2.2 The RMQ Techniques

We next introduce an important technique, called RMQ, used in our algorithm. We are given a sequence $A = \langle a_1, a_2, \ldots, a_n \rangle$ to be preprocessed. A Range Minima Query(RMQ) speci es an interval I and the goal is to nd the index k with minimum value a_k for $k \in I$.

Lemma 1. *The RMQ problem can be solved in* $O(n)$ *time and space prerpocessing and* $O(1)$ *time per query. [1, 6]*

The well known algorithm for the RMQ problem is to rst construct the Cartesian tree(de ned by Vuillemin 1980) of the sequence, then be preprocessed for LCA(Least Common Ancestor) queries [8, 4]. This algorithm can be easily modi ed to output the index k for which a_k achieves the minimum or the maximum. We denote RMQ$_{min}$ as the minimum query and RMQ$_{max}$ as the maximum query. That is, RMQ$_{min}(A, i, j)$ will return index k such that a_k achieves the minimum for $i \leqslant k \leqslant j$, and RMQ$_{max}(A, i, j)$ will return index k such that a_k achieves the maximum. For the correctness of our algorithm if there are more than two minimums(maximums) in the query interval, it always outputs the rightmost(leftmost) index k for which a_k achieves the minimum(maximum). This can be done by constructing the Cartesian tree in a particular order.

3 Coping with the SRMSQ Problem

A key idea to solve the SRMSQ problem is to view the problem in the sense of the cumulative sum. For convenience of later proof, we give the following de nition.

Definition 1. *Let* A *be any nonempty real number sequence. We de ne* $l[j]$ *for each index* j *of* A *to be the largest index* k *such that* $c[k] \geq c[j]$ *and* $1 \leqslant k \leqslant j - 1$. *But if no such* k *exists, that is,* $c[j] > c[k]$ *for all* $1 \leqslant k \leqslant j - 1$, *we assign* $l[j] = 0$.

Such largest index $l[j]$ and the cumulative sum $c[j]$ for each index j of A can be computed by the PREPROCESS1 algorithm as illustrated in Figure 1.

```
Algorithm PREPROCESS1
Input: An array of $n$ real numbers $A[1 \dots n]$.
Output: A length $n$ array $l[\cdot]$ and a length $n+1$ array $c[\cdot]$
1   $c[0] \leftarrow 0$;
2   for $j \leftarrow 1$ to $n$ do
3       $c[j] \leftarrow c[j-1] + A[j]$;
4       $l[j] \leftarrow j-1$;
5       while $c[l[j]] < c[j]$ and $l[j] > 0$ do
6           $l[j] \leftarrow l[l[j]]$;
7       end while
8   end for
```

Figure 1: Algorithm for computing $c[\cdot]$ and $l[\cdot]$

Lemma 2. *Let A be any nonempty real number sequence. If segment $S(i, j)$ is the maximum-sum segment of A, then index i is the largest index such that the cumulative sum $c[i-1]$ is minimized for $l[j] \leqslant i-1 < j$(that is, $c[i-1]$ is the rightmost minimum).*

Proof. Suppose not, then either (1) index $i-1$ lies in the interval $[0, l[j]-1]$ or (2) index $i-1$ lies in the interval $[l[j], j-1]$ but is not the largest index such that the cumulative sum $c[i-1]$ is minimized for $l[j] \leqslant i-1 < j$. We discuss it as follows.

(1) Suppose index $i-1$ lies in the interval $[0, l[j]-1]$:

When $l[j] = 0$, it's obvious that $i-1$ cannot lie in the interval $[0, l[j]-1]$. When $l[j] > 0$, we have $w(S(i, l[j])) = c[l[j]] - c[i-1] \geq c[j] - c[i-1] = w(S(i, j))$. If equality holds, then $w(S(l[j]+1, j)) = c[j] - c[l[j]] = 0$. So $S(l[j]+1, j)$ would be a zero-sum pre x of $S(i, j)$. Thus, $w(S(i, l[j]))$ must be strictly greater than $w(S(i, j))$. But, this contradicts to the fact that $S(i, j)$ is the maximum-sum segment of A. So index $i-1$ cannot lie in the interval $[0, l[j]-1]$.

(2) Suppose index $i-1$ lies in the interval $[l[j], j-1]$:

If $c[i-1]$ is not minimized for $l[j] \leqslant i-1 < j$. That is, there exists an index k, $k \neq i$ and $l[j] \leqslant k-1 < j$, such that $c[k-1] < c[i-1]$. Then we have $w(S(k, j)) = c[j] - c[k-1] > c[j] - c[i-1] = w(S(i, j))$. This contradicts to the fact that $S(i, j)$ is the maximum-sum segment of A. So the cumulative sum $c[i-1]$ must be minimized for $l[j] \leqslant i-1 < j$. We further suppose that $c[i-1]$ is not the rightmost minimum, that is, there exists an index $k' > i$ such that $c[k'-1]$ is also a minimum. Then $w(S(i, k'-1)) = c[k'-1] - c[i-1] = 0$. So $S(i, j)$ has a zero-sum pre x $S(i, k'-1)$ which also contradicts to the de nition of the maximum-sum segment. Hence, index i must be the largest index such that $c[i-1]$ is minimized for $l[j] \leqslant i-1 < j$. $\square$

Definition 2. *Let A be any nonempty real number sequence. We de ne the "good partner", $p[j]$, of each index j in A as the largest index k such that $c[k-1]$ is minimized for $l[j] \leqslant k-1 < j$. And segment $S(p[j], j)$ is called a candidate segment of A at j. We also de ne $m[j]$ to be the sum of the candidate segment of A at j, that is $m[j] = w(S(p[j], j))$.*

By Lemma 2, we know that each pair $(p[j], j)$ constitutes a candidate solution of the maximum-sum segment of A, that is, segment $S(p[j], j)$. The relationship between $l[j]$ and $p[j]$ as de ned above is illustrated in Figure 2. The left side of the gure shows the case that there exists a largest index $l[j]$ such that $c[l[j]] \geq c[j]$ and $1 \leqslant l[j] \leqslant j$. And the right side of the gure shows the case that $c[j]$ is the unique maximum of $c[k]$ for all $1 \leqslant k \leqslant j$.

The good partner $p[j]$ and the sum of the candidate segment $m[j]$ for each index j of A can be computed by the PREPROCESS2 algorithm as illustrated in Figure 3.

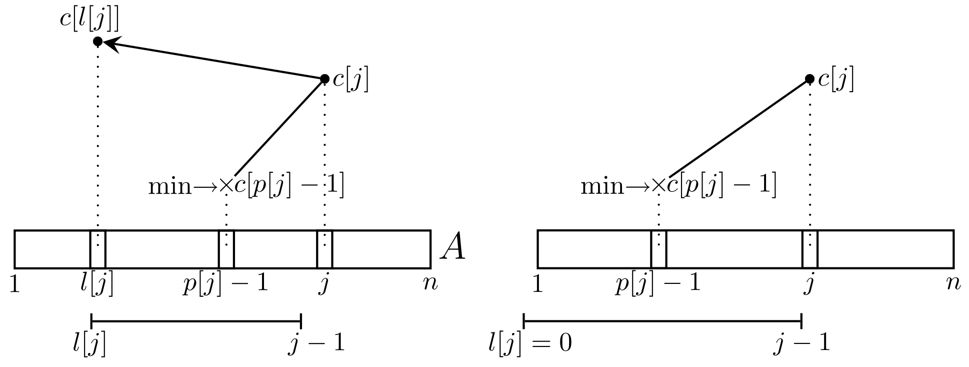

Figure 2: An illustration for $l[\cdot]$ and $p[\cdot]$.

Algorithm PREPROCESS2
Input: An array of n real numbers $A[1 \ldots n]$, array $c[\cdot]$ and array $l[\cdot]$.
Output: Two length n arrays, $p[\cdot]$ and $m[\cdot]$.

```
1  Apply $RMQ_{min}$ preprocess to array $c[\cdot]$.
2  for $j \leftarrow 1$ to $n$ do
3      $p[j] \leftarrow RMQ_{min}(c, l[j], j-1) + 1$;
4      $m[j] \leftarrow c[j] - c[p[j]-1]$;
5  end for
```

Figure 3: Algorithm for computing $p[\cdot]$ and $m[\cdot]$

Lemma 3. *Let A be any nonempty real number sequence of length n. If index i satis es $m[i] \geq m[k]$ for all $1 \leqslant k \leqslant n$, then $S(p[i], i)$ is the maximum-sum segment.*

Proof. Suppose on the contrary, segment $S(s,t)$, $(s,t) \neq (p[i], i)$, is the maximum-sum segment. By Lemma 2, we have $s = p[t]$. So $m[t] = w(S(p[t],t)) = w(S(s,t)) > w(S(p[i],i) = m[i]$ which contradicts to $m[i]$ is the maximum value. □

Lemma 3 tells us, once we have computed $m[j]$ and $p[j]$ for each index j of A. To nd the maximum-sum segment of A, we only have to retrieve the index i such that $m[i] \geq m[k]$ for all $1 \leqslant k \leqslant n$. Then, $S(p[i], i)$ is the maximum-sum segment of A. Lemma 4-6 will show some important properties of the candidate segments.

Lemma 4. *Let A be any nonempty real number sequence. If $p[j]$ is the good partner of index j and $p[j] < j$, then $c[p[j]-1] < c[k] < c[j]$ for all $p[j]-1 < k < j$.*

Proof. Suppose not. That is, there exists an index k', $p[j]-1 < k' < j$, such that $c[k'] \leq c[p[j]-1]$ or $c[k'] \geq c[j]$. (1) If $c[k'] \leq c[p[j]-1]$. By de nition of $p[j]$, we know that $l[j] \leqslant p[j] < j$. Since $p[j]-1 < k'$, we have $l[j] < k' < j$ and $c[k'] \leq c[p[j]-1]$. This contradicts to the de nition of $p[j]$ to be the largest index and $c[p[j]-1]$ is minimized for $l[j] \leqslant p[j]-1 < j$. (2) If $c[k'] \geq c[j]$, then by de nition of $p[j]$, we have $k' \leq p[j]-1 < j$. So $k' \leq p[j]-1 < k'$. A contradiction occurs. □

That is, $c[p[j]-1]$ is the unique minimum and $c[j]$ is the unique maximum of $c[k]$ for all $p[j]-1 \leqslant k \leqslant j$. The following lemma shows the nesting property of the candidate segments. See Figure 5. Notice that, the pointer of each index j points to the position of $p[j]$.

```
Algorithm PREPROCESS OF SRMSQ_A(i, j)
1   Run algorithm PREPROCESS1 to compute array $c[\cdot], l[\cdot]$ of A.
2   Run algorithm PREPROCESS2 to compute array $p[\cdot], m[\cdot]$ of A.
3   Apply $\text{RMQ}_{max}$ preprocess to array $m[\cdot]$.
4   Apply $\text{RMQ}_{min}$ preprocess to array $c[\cdot]$.

Algorithm QUERY OF SRMSQ_A(i, j)
1   r ← $\text{RMQ}_{max}(m, i, j)$
2   if $p[r] < i$ then
3       $r_1 \leftarrow \text{RMQ}_{min}(c, i-1, r-1) + 1$;
4       $r_2 \leftarrow \text{RMQ}_{max}(m, r+1, j)$;
5       if $c[r] - c[r_1 - 1] < m[r_2]$ then
6           OUTPUT ($p[r_2]$,$r_2$);
7       else
8           OUTPUT ($r_1, r$);
9       end if
10  else
11      OUTPUT ($p[r], r$);
12  end if
```

Figure 4: Algorithm for the SRMSQ problem.

Lemma 5. *Let A be any nonempty real number sequence. For two indices i and j, $i < j$, if $p[i]$ is the good partner of i and $p[j]$ is the good partner of j, then it cannot be the case that $p[i] < p[j] \leq i < j$.*

Proof. Suppose $p[i] < p[j] \leq i < j$ holds. By Lemma 4, we have
(1) $c[p[i]-1] < c[k'] < c[i] \quad \forall \quad p[i]-1 < k' < i$
(2) $c[p[j]-1] < c[k''] < c[j] \quad \forall \quad p[j]-1 < k'' < j$
Since $p[j]-1 < i$, we can substitute $p[j]-1$ for k' in (1) $\Rightarrow c[p[i]-1] < c[p[j]-1] < c[i]$.
Similarly, we can substitute i for k'' in (2) $\Rightarrow c[p[j]-1] < c[i] < c[j]$.
Then we have
(3) $c[p[i]-1] < c[p[j]-1] < c[k''] < c[j] \quad \forall \quad p[j]-1 < k'' < j$
(4) $c[p[i]-1] < c[k'] < c[i] < c[j] \quad \forall \quad p[i]-1 < k' < i$
By (3) and (4), we have
(5) $c[p[i]-1] < c[k'''] < c[j] \quad \forall \quad p[i]-1 < k''' < j$
So, if there exists an index k such that $c[k] \geq c[j]$, then $k < p[i]-1$. We also have by (5) that $c[p[i]-1] < c[p[j]-1]$ which is a contradiction to that $c[p[j]-1]$ minimizes $c[l]$ for $k < l < j$. If there is no such index k, then $c[p[i]-1] < c[p[j]-1]$ is a contradiction to that $c[p[j]-1]$ minimizes $c[l]$ for $l < j$. □

Now, we are about to establish the relationship between sequence A and its subsequence Q. The following key lemma will show that a candidate segment S of A is still a candidate segment of Q if Q contains the whole candidate segment S.

Lemma 6. *Let $A = \langle a_1, \ldots a_n\rangle$ be any nonempty real number sequence and $Q = \langle a_s, \ldots a_t\rangle$ be any subsequence of A. If $p[j]$ is the good partner of index j for sequence A and $s \leq p[j] \leq j \leq t$, then $p[j]$ is still the good partner of index j for sequence Q.*

Proof. Let $c\ [j]$ be the cumulative sum of each index j in Q. Then, $c\ [j] = \sum_{k=s}^{j} a_k = \sum_{k=1}^{j} a_k - \sum_{k=1}^{s-1} a_k = c[j] - c[s-1]$ for all $s-1 \leqslant j \leqslant r$. Let $l\ [j]$ be the largest index k such that $c\ [k] \geq c\ [j]$ for all $s \leqslant k \leqslant j-1$. If no such k exists, we assign $l\ [j] = s-1$. Our goal is to prove that $p[j]$ is the largest index k that minimizes $c\ [k-1]$ for all $l\ [j] \leqslant k-1 < j$.
(1) If $l[j] \geq s$. Since $l[j]$, by de nition, is the largest index k such that $c[k] \geq c[j]$ for all $1 \leqslant k \leqslant j-1$. So $l[j]$ is the largest index k such that $c\ [k] = c[k] - c[s-1] \geq c[j] - c[s-1] = c\ [j]$ for all $1 \leqslant s \leqslant k \leqslant j-1$. Hence, we have $l\ [j] = l[j]$. And $p[j]$, by de nition, is the largest index k that minimizes $c[k-1]$ for all $l[j] \leqslant k-1 < j$. So $p[j]$ is the largest index k that minimizes $c\ [k-1] = c[k-1] - c[s-1]$ for all $l\ [j] = l[j] \leqslant k-1 < j$.
(2) If $l[j] < s$. That is, $c[k] < c[j]$ for all

$s \leqslant k \leqslant j-1$. So, $c\ [k] = c[k] - c[s-1] < c[j] - c[s-1] = c\ [j]$ for all $s \leqslant k \leqslant j-1$. Hence, we have $l\ [j] = s-1$. Since $p[j]$ is the largest index k that minimizes $c[k-1]$ for all $l[j] \leqslant k-1 < j$. So, $p[j]$ is the largest index k that minimizes $c\ [k-1] = c[k-1] - c[s-1]$ for all $l[j] \leqslant s-1 \leqslant k-1 < j$. That is, $p[j]$ is the largest index k that minimizes $c\ [k-1]$ for all $l\ [j] \leqslant k-1 < j$. □

Corollary 1. *Let $A = \langle a_1, \dots a_n \rangle$ be any nonempty real number sequence and $Q = \langle a_s, \dots a_t \rangle$ be any subsequence of A. If $p[j]$ is the good partner of index j for sequence A, $s \leq p[j] \leq j \leq t$, and $m[j]$ is the sum of the candidate segment of A at j. Then $m[j]$ is also the sum of the candidate segment of Q at j.*

Proof. A direct result of Lemma 6. □

Now, we are ready to present our algorithm for the SRMSQ problem(See Figure 4). Let A be a sequence of n real numbers and $[i,j]$ is the query interval, where $1 \leqslant i \leqslant j \leqslant n$.

For instance, see Figure 5, the input sequence A has 15 elements. suppose we are querying SRMSQ$_A(3,7)$. The QUERY OF SRMSQ algorithm in Figure 4 will rst query the index r such that $m[r]$ is maximized for $3 \leqslant r \leqslant 7$(line 1). In this case, $r = 5$, which means candidate segment $S(p[5],5)$ has the largest sum compared to other segments. The left end of $S(p[5],5)$, $p[5] = 3$, lies in the interval $[3,7]$. The algorithm executes line 11, output $(p[5],5)$, which means segment $S(3,5)$ is the maximum-sum segment of the subsequence $A[3\dots7]$.

Suppose we are querying SRMSQ$_A(6,12)$, RMQ$_{max}(m,6,12)$ will return index 9(line 1). Since $p[9] = 3 < 6$, line 3-9 will be executed. In line 3, RMQ$_{min}(c,5,8)$ will return index 8. In line 4, RMQ$_{max}(m,10,12)$ will return index 11. In line 5, since $c[9] - c[8] = 6 < m[11] = 8$, the QUERY OF RMSQ algorithm will output $(p[11],11)$, which means $S(11,11)$ is the maximum-sum segment of the subsequence A[6. . . 12].

Theorem 1. *Algorithm QUERY OF SRMSQ$_A(i,j)$ will output the maximum-sum segment of the subsequence $S(i,j)$.*

Proof. Let $m\ [k]$ be the sum of the candidate segment of $S(i,j)$ at k and $c\ [k]$ be the cumulative sum of $S(i,j)$ for $i-1 \leqslant k \leqslant j$. We have $c\ [k] = c[k] - c[i-1]$, $i-1 \leqslant k \leqslant j$. Let $p\ [k]$ be the good partner of index k of Q for all $i \leqslant k \leqslant j$. Let index r satisfy $m[r] \geq m[k]$ for all $i \leqslant k \leqslant j$(line 1).

(A) If $p[r] \geq i$: Our goal is to show that $m^{[}r] \geq m\ [k]$ for all $i \leqslant k \leqslant j$, and then apply Lemma 3 to complete the proof. We rst consider each index k' such that $i \leqslant k' \leqslant j$ and $i \leq p[k'] \leq k' \leq j$. By corollary 1, we can deduce that $m\ [k'] = m[k'] \leq m[r] = m\ [r]$. We next consider each index k'' such that $i \leqslant k'' \leqslant j$ and $p[k''] < i$. Since $p[k''] < i \leq k''$, we can apply Lemma 4 and get
$c[p[k'']-1] < c[k] < c[k''] \quad \forall \quad p[k'']-1 < k < k''$ (1)
Since $i \leq p\ [k''] < k''$ by de nition of $p\ [k'']$, we have $p[k'']-1 < i-1 \leq p\ [k'']-1 < k''$. So, we can substitute $p\ [k'']-1$ for k in (6) and get $c[p[k'']-1] < c[p\ [k'']-1]$. Hence, we can deduce that $m\ [k''] = c\ [k''] - c\ [p\ [k'']-1] = c[k''] - c[p\ [k'']-1] < c[k''] - c[p[k'']-1] = m[k'']$ Moreover, $m[k''] \leq m[r] = m\ [r]$. So, we have $m\ [k''] < m\ [r]$. Till now, we have shown that $m[r] \geq m\ [k]$ for all $i \leqslant k \leqslant j$. By Lemma 3, $S(p[r],r)$ is the maximum-sum segment of $S(i,j)$(line 11).

(B) If $p[r] < i$: We rst consider each index k' such that $i \leqslant k' < r$. Since $p[r] < i \leq r$, we can apply Lemma 4 and obtain
$c[p[r]-1] < c[k] < c[r] \quad \forall \quad p[r]-1 < k < r$ (2)
Since $c[k] < c[r]$ for all $p[r]-1 < i \leqslant k < r$, we have $c\ [k] = c[k] - c[i-1] < c[r] - c[i-1] = c\ [r] \quad \forall \quad i \leqslant k < r$. For any segment $S(k,k')$, $i \leqslant k < r$, since $w(S(k,k')) = c\ [k'] - c\ [k] < c\ [r] - c\ [k] = w(S(k,r))$, it's not hard to see that $S(k,k')$ cannot be the maximum-sum segment.

We next consider each index k'' such that $r < k'' \leq j$. By Lemma 5, we know that it cannot be the case $p[r] < p[k''] \leq r < k''$. If $p[k''] \leq r$, then it must be the case that $p[k''] \leq p[r] < r < k''$. Since $p[k'']-1 \leq r < k''$ and $p[k'']-1 \leq p[r]-1 < k''$, we can apply Lemma 4 and get $c[p[k'']-1] < c[r] < c[k'']$ and $c[p[k'']-1] < c[p[r]-1] < c[k'']$. So, $m[k''] = c[k''] - c[p[k'']-1] > c[r] - c[p[r]-1] = m[r]$ which contradicts to that $m[r] \geq m[k]$ for all $i \leqslant k \leqslant j$. So, $p[k''] \nleq r$, that is, $p[k''] > r$. Hence, by corollary 1, we have $m\ [k''] = m[k'']$ for $r < k'' \leq j$. Let index r_2 satis es $m[r_2] \geq m[k]$ for all $r+1 \leqslant k \leqslant j$(line 4). It's not hard to see that either $S(r_1,r)$ or $S(p[r_2],r_2)$ is the maximum-sum segment of $S(i,j)$. By Lemma 3, the one with greater sum is the maximum-sum segment of $S(i,j)$(line 5-9). □

j	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
a_j	-	9	-10	4	-2	4	-5	4	-3	6	-11	8	-3	4	-5	3
$c[j]$	0	9	-1	3	1	5	0	4	1	7	-4	4	1	5	0	3
$l[j]$	-	0	1	1	3	1	5	5	7	1	9	9	11	9	13	13
$p[j]$	-	1	2	3	4	3	6	7	8	3	10	11	12	11	14	15
$m[j]$	-	9	-10	4	-2	6	-5	4	-3	8	-11	8	-3	9	-5	3

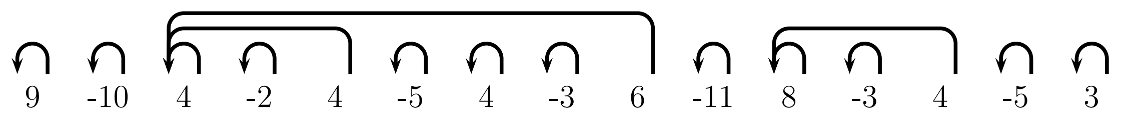

Figure 5: The candidate segment S(p[j],j) of each index j.

```
Algorithm Preprocess of RMSQ
1  Apply SRMSQ preprocess.
2  Apply RMQ$_{max}$ preprocess to $c[\cdot]$.

Algorithm Query of RMSQ$_A(i, j, k, l)$
1  if $j \leq k$ then
2      OUTPUT (RMQ$_{min}(c, i-1, j-1)+1$,RMQ$_{max}(c, k, l)$)
3  else
4      $(r_1,r'_1) \leftarrow$ (RMQ$_{min}(c, i-1, k-1)+1$,RMQ$_{max}(c, k, l)$);
5      $(r_2,r'_2) \leftarrow$ (RMQ$_{min}(c, k, j-1)+1$,RMQ$_{max}(c, j, l)$);
6      $(r_3,r'_3) \leftarrow$ SRMSQ$_A(k, j)$;
7      OUTPUT $(r_m,r'_m)$ such that $c[r_m]-c[r'_m]$ is maximized for $1 \leqslant m \leqslant 3$;
8  end if
```

Figure 6: Algorithm for the RMSQ problem.

Lemma 7. *Algorithm PREPROCESS1 runs in $O(n)$ time.*

Proof. It can be shown by a simple amortized analysis. The total number of operations of the algorithm is clearly bounded by $O(n)$ except for the while-loop body of Steps 5-7. In the following, we show that the amortized cost of the while-loop is a constant. Therefore, the overall time required by the loop is $O(n)$. We de ne the *potential function* of A after the ith iteration of the for-loop to be $\Phi(i)$, i.e. the number of times pointer $l[i]$ may be advanced most. So we have $\Phi(i) \geq 0$ always holds in every iteration. Suppose that the pointer $l[\cdot]$ is advanced c_i times in this period. Then the actual cost of the operations is $c_i + 1$. Observing that $\Phi(i) = \Phi(i-1) - c_i + 1$, the change of the potential of A during the ith iteration is $\Phi(i) - \Phi(i-1) = \Phi(i-1) - c_i + 1 - \Phi(i-1) = 1 - c_i$. The amortized cost is therefore calculated as $\hat{c_i} = c_i + 1 + \Phi(i) - \Phi(i-1) = 2$. Since exactly n iterations would be executed in the entire process, the while-loop spends at most overall $O(n)$ time. $\square$

Lemma 8. *Algorithm PREPROCESS2 runs in $O(n)$ time.*

Proof. By Lemma 1, the preprocessing time for RMQ is $O(n)$ and the query time is $O(1)$. So, the cost of each step in the for-loop is a constant and the overall time required is $O(n)$. $\square$

Theorem 2. *The SRMSQ problem can be solved in $O(n)$ preprocessing time and $O(1)$ query time.*

Proof. By Lemma 1, Lemma 7, and Lemma 8, the algorithms for preprocess all run in $O(n)$ time. So, the time complexity for the PREPROCESS OF SRMSQ algorithm is $O(n)$. Since each RMQ query in the QUERY OF SRMSQ algorithm takes $O(1)$ time, one can easily see that the query time for the QUERY OF SRMSQ algorithm is $O(1)$.

It's not hard to verify that the space complexity for these algorithms is also $O(n)$. ☐

4 Coping with the RMSQ Problem

The RMSQ problem is to answer the queries comprising of two intervals $[i, j]$ and $[k, l]$, where $[i, j]$ speci es the range of the starting index of the maximum-sum segment, and $[k, l]$ speci es the range of the ending index. Since it is meaningless if the range of the starting index is in front of the range of the ending index. So, without loss of generality, we assume that $i \leq k$ and $j \leq l$. We presents our algorithm as follows.

Theorem 3. *Algorithm QUERY OF RMSQ$_A(i, j, k, l)$ will output the correct answer.*

Proof. We discuss it under two possible conditions.
(1) Nonoverlapping($j \leq k$): Suppose the intervals $[i, j]$ and $[k, l]$ do not overlap. Since $w(S(x, y)) = c(y) - c(x-1)$, to maximize $S(x, y)$ is equivalent to maximize $c(y)$ and minimize $c(x-1)$ for $i \leq x \leq j$ and $k \leq y \leq l$. By applying the RMQ technique to preprocess $c[\cdot]$, the maximum-sum segment can be easily located.(line 2)
(2) Overlapping($j > k$): When it comes to the overlapping case, just to nd the maximum cumulative sum and the minimum cumulative sum might go wrong if the minimum is on the right of the maximum. We discuss it under 3 possible conditions for the maximum-sum segment $S(x, y)$. (a) Suppose $i \leq x \leq k$ and $k \leq y \leq l$, this is an nonoverlapping case. So we nd the minimum cumulative sum and the maximum cumulative sum.(line 4) (b) Suppose $k + 1 \leq x \leq j$ and $j \leq y \leq l$, this is also an nonoverlapping case. We nd the minimum cumulative sum and the maximum cumulative sum.(line 5) (c) Otherwise, $k + 1 \leq x \leq j$ and $k + 1 \leq y \leq j$, this is exactly the same as a SRMSQ$_A(k + 1, j)$ query.(line 6) The maximum sum segment $S(x, y)$ must be one of these three possible cases and have the largest sum(line 7). ☐

Theorem 4. *The RMSQ problem can be solved in $O(n)$ preprocessing time and $O(1)$ query time.*

Proof. The time for the RMQ$_{max}$ and the SRMSQ preprocesses are $O(n)$. So the PREPROCESS OF RMSQ algorithm costs $O(n)$ time. And the query time is $O(1)$ since each step in the QUERY OF RMSQ algorithm costs $O(1)$ time. ☐

Acknowledgements. We thank Yu-Ru Huang and An-Chiang Chu for helpful conversations. Quan-Yu Chen and Kun-Mao Chao were supported in part by an NSC grant 92-2213-E-002-059.

References

[1] M. A. Bender, and M. Farach-Colton. The LCA Problem Revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical INformatics*, 17: 88–94, 2000.

[2] J. Bentley, Programming Pearls - Algorithm Design Techniques, *CACM*, 865-871, 1984.

[3] K. Chung and H.-I. Lu. An Optimal Algorithm for the Maximum-Density Segment Problem. In *Proceedings of the 11th Annual European Symposium on Algorithms (ESA 2003)*, LNCS 2832, pp. 136-147, 2003.

[4] D. Harel and R. E. Tarjan. Fast Algorithms for Finding Nearest Common Ancestors. *SIAM J Comput.* Vol. 13, No 2, 1984.

[5] T.-H. Fan ,S. Lee, H.-I Lu, T.-S. Tsou, T.-C. Wang, and A. Yao. An Optimal Algorithm for Maximum-Sum Segment and Its Application in Bioinformatics. *CIAA*, LNCS 2759, pp. 251-257, 2003.

[6] H. Gabow, J. Bentley, and R. Tarjan. Scaling and Related Techniques for Geometry Problems. *Proc. Symp Theory of Computing*(STOC), 1984, 135-143.

[7] D. Gus eld, Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology. Cambridge University Press, 1999.

[8] B. Schieber and U. Vishkin. On Finding Lowest Common Ancestors: Simpli cation and Parallelization. *SIAM J Comput.* Vol. 17, No 6, 1988.

[9] Huang, X. (1994) An algorithm for identifying regions of a DNA sequence that satisfy a content requirement. *CABIOS*, **10**, 219-225.

[10] Y.-L. Lin, T. Jiang, and K.-M. Chao. Ef- cient Algorithms for Locating the Length-constrained Heaviest Segments with Applications to Biomolecular Sequence Analysis. *Journal of Computer and System Sciences*, 65, 570-586, 2002.

[11] S. Muthukrishnan, E cient Algorithms for Document Retrieval Problems. In *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 657–666, 2002.

[12] W. L. Ruzzo and M. Tompa. A Linear Time Algorithm for Finding All Maximal Scoring Subsequences. In *7th Intl. Conf. Intelligent Systems for Molecular Biology, Heidelberg, Germany*, Aug. 1999.