한국지능시스템학회 논문지 2010, Vol. 20, No. 4, pp. 455-462

단순하고 스테이블한 머징알고리즘

On a Simple and Stable Merging Algorithm

김복선[1] · 쿠츠너 아네[2]
Pok-Son Kim and Arne Kutzner

[1]국민대학교 수학과
E-mail: pskim@kookmin.ac.kr
[2]한양대학교 정보시스템학과
E-mail: kutzner@hanyang.ac.kr

요 약

단순하고 스테이블한 머징알고리즘의 비교횟수와 관련된 worst case 복잡도를 분석한다. 복잡도 분석을 통해 소개되는 알고리즘이 m 과 n, $m \le n$ 사이즈의 두 수열에 대해 $O(m\log(n/m))$의 비교횟수를 요구하는 사실을 증명한다. 그래서 병합에 있어서의 하계가 $\Omega(m\log(n/m))$이라는 사실로부터 우리의 알고리즘이 비교횟수와 관련해 점근적 최적 알고리즘에 해당함을 추론가능하다.
worst case 복잡도 증명을 위해 모든 입력수열로 구성된 정의구역을 두개의 서로소인 집합으로 나눈다. 그런 후 서로소인 각각의 집합으로 부터 특수한 subcase를 구별한 후 이들 subcase 각각에 대해 점근적 최적성을 증명한다. 이 증명을 바탕으로 나머지 모든 경우에 대한 최적성 또한 추론 또는 증명 가능함을 소개한다. 이로써 우리는 복잡도 분석이 까다로운 알고리즘에 대해 투명한 하나의 해를 제시한다.

키워드 : 스테이블 머징, 미니멈 스토리지, 알고리즘복잡도.

Abstract

We investigate the worst case complexity regarding the number of comparisons for a simple and stable merging algorithm. The complexity analysis shows that the algorithm performs $O(m\log(n/m))$ comparisons for two sequences of sizes m and n $m \le n$. So, according to the lower bound for merging $\Omega(m\log(n/m))$, the algorithm is asymptotically optimal regarding the number of comparisons.
For proving the worst case complexity we divide the domain of all inputs into two disjoint cases. For either of these cases we will extract a special subcase and prove the asymptotic optimality for these two subcases. Using this knowledge for special cases we will prove the optimality for all remaining cases. By using this approach we give a transparent solution for the hardly tractable problem of delivering a clean complexity analysis for the algorithm.

Key Words : stable merging, minimum storage, complexity of algorithms

1. Introduction

Merging denotes the operation of rearranging the elements of two adjacent sorted sequences of sizes m and n, so that the result forms one sorted sequence of $m+n$ elements. An algorithm merges two adjacent sequences with *minimum storage* [1] when it needs $O(\log^2(m+n))$ bits additional space at most. It is regarded as *stable*, if it preserves the initial ordering of elements with equal value.

접수일자 : 2010년 3월 6일
완료일자 : 2010년 6월 20일
이 논문은 2008년 정부(교육과학기술부)의 재원으로 한국학술진흥재단 (KRF-2008-531-D00020)과 2010년도 국민대학교 교내연구비 지원을 받아 수행된 연구임.

There are two significant lower bounds for merging. The lower bound for the number of assignments is $m+n$ because every element of the input sequences can change its position in the sorted output. As shown by Knuth in [1] the lower bound for the number of comparisons is $\Omega(m\log\frac{n}{m})$, where $m \le n$.

The simple standard merge algorithm is rather inefficient, because it uses linear extra space and always needs a linear number of comparisons. The Recmerge algorithm of Dudzinski and Dydek [2] is minimum storage merging algorithm that is asymptotically optimal regarding the number of comparisons. It performs the merging by a binary partitioning of both input sequences which operates as the foundation of a rotation that is followed by two recursive calls.

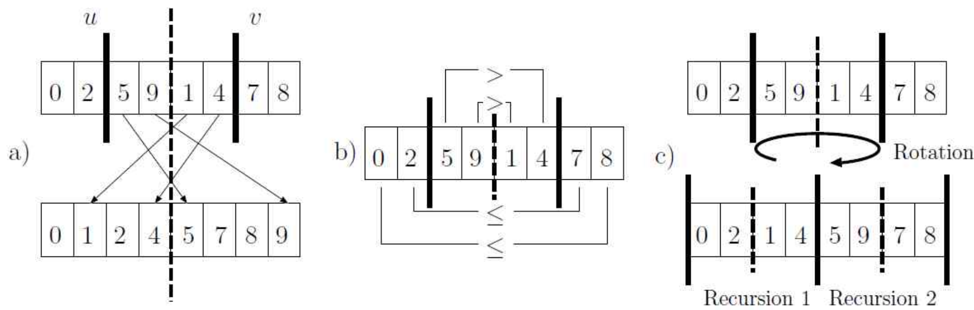

Fig. 1. SymMerge example

Another class of merging algorithms is the class of *in place* merging algorithms, where the external space is restricted to a constant amount merely. Recent work in this area are the publications [3, 4, 5, 6, 7], that describe algorithms which are all asymptotically optimal regarding the number of comparisons as well as assignments. However, these algorithms are structurally quite complex and rely heavily on other concepts, as e.g. Kronrod's idea of an internal buffer [8], Mannila and Ukkonen's technique for block rearrangements [9] and Hwang and Lin's merging algorithm [10].

In [11] we presented a stable minimum storage merging algorithm called SymMerge and investigated its worst case complexity regarding the number of comparisons as well as assignments. However, the complexity analysis was restricted only to a special case called "Maximum spanning case". Furthermore the method taken for the complexity analysis was quite complex.

In this paper we complete our proof based on a new *simplified* method for proving the worst case complexity. Consequently we get the result that the SymMerge algorithm performs $O(m\log\frac{n}{m})$ comparisons for two sequences of sizes m and n $(m \le n)$. According to the lower bound $\Omega(m\log\frac{n}{m})$ mentioned above, we can imply SymMerge is asymptotically optimal regarding the number of comparisons.

For proving the worst case complexity we divide the domain of all inputs into two disjoint classes (cases), for later reference denoted by case I and case II. For either of these cases we will extract a special subcase and prove the asymptotic optimality for these subcases. Then the optimality of the special subcase of case I logically implies the optimality of case I in general. Further, based on the optimality for the special subcase of case II, we will prove the optimality for all remaining cases of case II.

2. The SymMerge Algorithm

We start with a brief introduction of the merging method of the SymMerge algorithm presented in [11]. Let us assume that we have to merge the two sequences $u=(0,2,5,9)$ and $v=(1,4,7,8)$. When we compare the input with the sorted result, we can see that in the result the last two elements of u occur on positions belonging to v, and the first two elements of v appear on positions belonging to u (see Fig. 1 a)). So, 2 elements were exchanged between u and v. The kernel of SymMerge is to compute this number of side-changing elements efficiently and then to exchange such a number of elements. More accurately, if we have to exchange p $(p \ge 0)$ elements between sequences u and v, we move the p greatest elements from u to v and the p smallest elements from v to u, where the exchange of elements is realized by a rotation. Then by recursive application of this technique to the arising subsequences we get a sorted result. Fig. 1 illustrates this approach to merging for our above example.

We will now focus on the process of determining the number of elements to be exchanged. This number can be determined by a process of symmetrical comparisons of elements that happens according to the following principle:

We start at the leftmost element in u and at the rightmost element in v and compare the elements at these positions. We continue doing so by symmetrically comparing element-pairs from the outsides to the middle. Fig. 1 b) shows the resulting pattern of mutual comparisons for our example. There can occur at most one position, where the relation between the compared elements alters from 'not greater' to 'greater'. In Figure 1 b) two thick lines mark this position. These thick lines determine the number of side-changing elements as well as the bounds for the rotation mentioned above.

So far we introduced the computation of the number of side-changing elements as linear process of symmetric comparisons. But this computation may also happen in the style of a binary search. Then only $\lfloor \log(\min(|u|,|v|)) \rfloor + 1$ comparisons are necessary to compute the number of side-changing elements.

2.1 Formal Definition

Let u and v be two adjacent ascending sorted sequences. We define $u \le v$ $(u < v)$ iff $x \le y$ $(x < y)$ for all elements $x \in u$ and for all elements $y \in v$.

We merge u and v as follows:

If $|u| \le |v|$, then

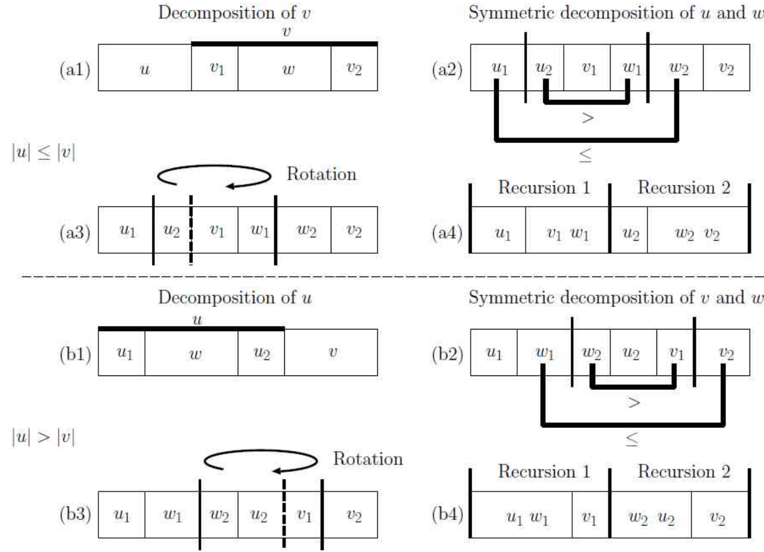

Fig. 2. Illustration of SymMerge

(a1) we decompose v into v_1wv_2 such that $|w|=|u|$ and either $|v_2|=|v_1|$ or $|v_2|=|v_1|+1$.

(a2) we decompose u into u_1u_2 ($|u_1| \geq 0$, $|u_2| \geq 0$) and w into w_1w_2 ($|w_1| \geq 0$, $|w_2| \geq 0$) such that $|u_1|=|w_2|$, $|u_2|=|w_1|$ and $|u_1| \leq |w_2|$, $u_2 > w_1$.

(a3) we rotate $u_2v_1w_1$ to $v_1w_1u_2$.

(a4) we recursively merge u_1 with v_1w_1 as well as u_2 with w_2v_2. Let u' and v' be the resulting sequences, respectively.

else

(b1) we decompose u into u_1wu_2 such that $|w|=|v|$ and either $|u_2|=|u_1|$ or $|u_2|=|u_1|+1$.

(b2) we decompose v into v_1v_2 ($|v_1| \geq 0$, $|v_2| \geq 0$) and w into w_1w_2 ($|w_1| \geq 0$, $|w_2| \geq 0$) such that $|v_1|=|w_2|$, $|v_2|=|w_1|$ and $|w_1| \leq |v_2|$, $w_2 > v_1$.

(b3) we rotate $w_2u_2v_1$ to $v_1w_2u_2$.

(b4) we recursively merge u_1w_1 with v_1 as well as w_2u_2 with v_2. Let u' and v' be the resulting sequences, respectively.

$u'v'$ then contains all elements of u and v in sorted order.

Fig. 2 contains an accompanying graphical description of the process described above. The steps (a1) and (b1) manage the situation of input sequences of different length by cutting a subsection w in the middle of the longer sequence as ``active area''. This active area has the same size as the shorter of either input sequences. The decomposition formulated by the steps (a2) and (b2) can be achieved efficiently by applying the principle of the symmetric comparisons between the shorter sequence u (or v) and the active area w. After the decomposition step (a2) (or (b2)), the subsequence $u_2v_1w_1$ (or $w_2u_2v_1$) is rotated so that we get the subsequences $u_1v_1w_1$ and $u_2w_2v_2$ ($u_1w_1v_1$ and $w_2u_2v_2$).

In [11] we presented an implementation of the SymMerge algorithm in Pseudocode which shows the algorithm is easy to implement.

Stability

During the symmetric decomposition of u and w (w and v) $u_1 \leq w_2$ and $u_2 > w_1$ ($w_1 \leq v_2$ and $w_2 > v_1$) always hold. The treatment of pairs of equal elements as part of the "outer blocks" (u_1, w_2 in (a2) and w_1, v_2 in (b2)) avoids the exchange of equal elements and so any reordering of these. Hence the following corollary holds:

Corollary 2.1. SymMerge is stable.

Minimum Storage Property

The decomposition steps (a1) and (a2) ((b1) and (b2)) satisfy the properties $|v_1|=|v_2|$ or $|v_1|+1=|v_2|$, $|u_1|=|w_2|$ and $|w_1|=|u_2|$ ($|u_1|=|u_2|$ or $|u_1|+1=|u_2|$, $|v_1|=|w_2|$ and $|w_1|=|v_2|$). Therefore the following corollary holds:

Corollary 2.2. After applying the decomposition steps (a1) and (a2) (or (b1) and (b2)) it holds $|u_1|+|v_1|+|w_1|= \lfloor (|u|+|v|)/2 \rfloor$ and $|u_2|+|w_2|+|v_2|= \lceil (|u|+|v|)/2 \rceil$.

The following theorem holds trivially.

Theorem 2.3. The recursion-depth of SymMerge is bounded by $\lceil \log(m+n) \rceil$.

Corollary 2.4. SymMerge is a minimum storage algorithm.

3. Worst Case Complexity regarding the number of comparisons

We start this section with a short overview of the proof's structure. First we will divide the domain of all inputs into two disjoint classes (cases). For either of these classes we will extract a special subclass and prove the asymptotic optimality for these two subclasses. Using this knowledge for special cases we will prove the optimality for all remaining cases.

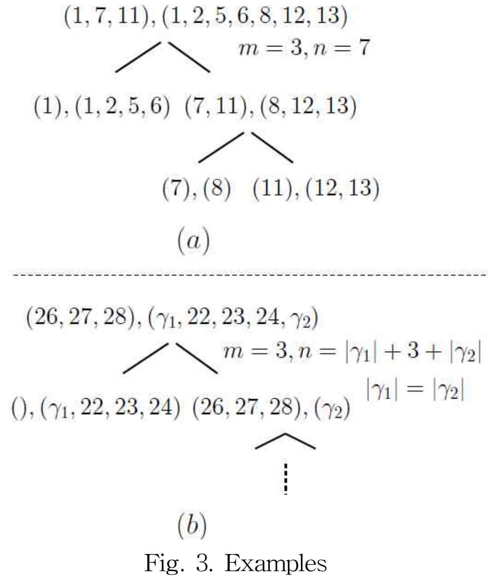

Fig. 3. Examples

Unless stated otherwise, let us denote $m=|u|$, $n=|v|$, $m \le n$, $m=2^k$, i. e. $k=\log m$. Further let m_j^i and n_j^i denote the sizes of sequences merged on the ith recursion level where the index j denotes the order of the merged sequences. Initially (on the recursion level 0), it holds $m_1^0=m$ and $n_1^0=n$. On the next recursion level 1, (m_1^0,n_1^0) is divided into two pairs (children nodes). We denote these by (m_1^1,n_1^1), (m_2^1,n_2^1) where the sequences of lengths m_1^1 and m_2^1 are merged with the sequences of lengths n_1^1 and n_2^1 respectively. On the recursion level i, the sequences of lengths $m_1^i,m_2^i,m_3^i,\cdots$ and m_j^i are merged with the sequences of lengths $n_1^i,n_2^i,n_3^i,\cdots$ and n_j^i respectively, where it holds $2 \le j \le 2^i$.

We divide the domain of all inputs into two disjoint classes as follows:

Case I - Every internal node (m_j^i,n_j^i) with $m_j^i>1$ is divided into two pairs $(m_{j'}^{i+1},n_{j'}^{i+1})$ and $(m_{j''}^{i+1},n_{j''}^{i+1})$, where it holds $m_{j'}^{i+1} \ge 1$ and $m_{j''}^{i+1} \ge 1$. Further all inputs of size $(1,n)$ for all $n \ge 1$ belong to this class. Case (a) of Fig. 3 shows such an example.

Case II - Complement of Case I; During the computation, some node (m_j^i,n_j^i) with $m_j^i>1$ is divided into two pairs $(m_{j'}^{i+1},n_{j'}^{i+1})$ and $(m_{j''}^{i+1},n_{j''}^{i+1})$, where $m_{j'}^{i+1}=0$ and $m_{j''}^{i+1}=0$. Case (b) of Fig. 3 shows such an example.

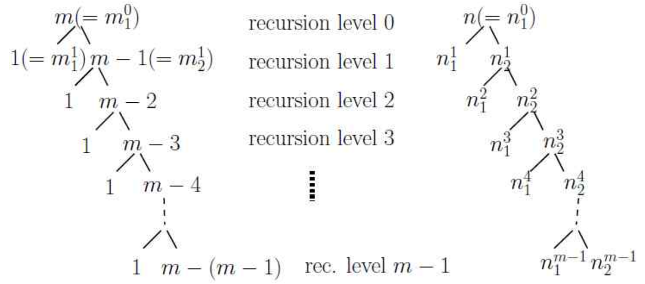

Fig. 4. Maximum spanning case

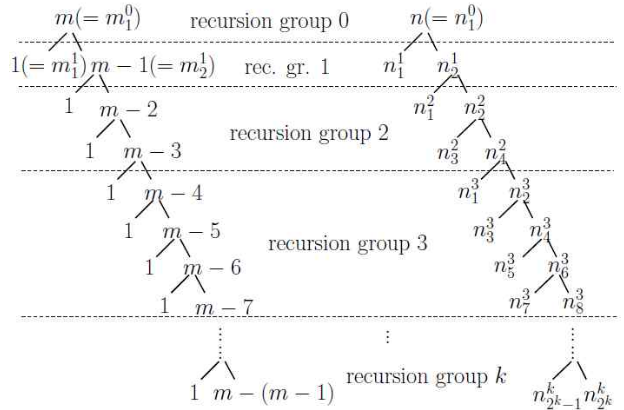

Fig. 5. Construction of recursion groups

First we show SymMerge performs $O(m\log(n/m+1))$ comparisons for case I.

Case I:

We begin by considering the complexity for a special case called maximum spanning case. In this case (m,n) is partitioned into $(1(=m_1^1),n_1^1)$ and $(m-1(=m_2^1),n_2^1)$ and each (m_2^i,n_2^i), $i=1,2,\cdots,m-2$ is partitioned into $(1(=m_1^{i+1}),n_1^{i+1})$ and $(m_2^i-1(=m_2^{i+1}),n_2^{i+1})$.

Subcase I.1: Maximum spanning case

Figure 4 shows the partitioning for the maximum spanning case. On the recursion level i, a sequence of length $m_1^i=1$ $(m_2^i=m-i)$ is merged with a sequence of length $n_1^i(n_2^i)$. Note that we need $\lfloor \log n_1^i \rfloor +1$ comparisons for merging the sequences (pair) of lengths $(1,n_1^i)$ completely. Further, by the merging method of SymMerge, such a merging is done over several recursion levels. However, for the convenience of complexity analysis we will consider the overall number of required comparisons $\lfloor \log n_1^i \rfloor +1$ at once on each corresponding recursion level i.

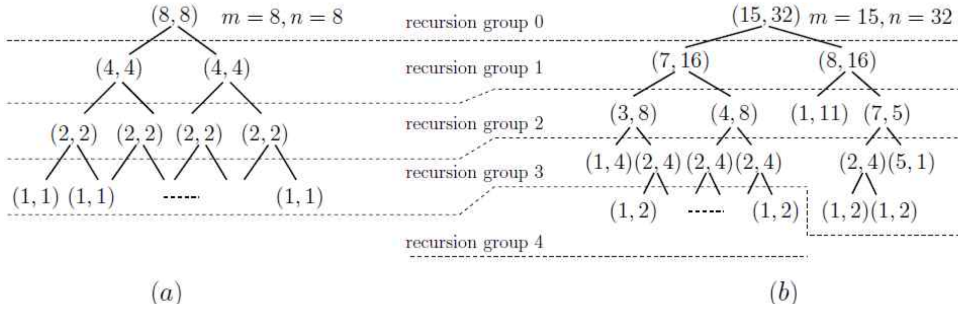

Fig. 6. Examples for partitioning the set of all merging pairs into $k+1$ groups

Theorem 3.1. The SymMerge algorithm needs $O(m\log(n/m+1))$ comparisons for the maximum spanning case.

Proof. The binary search of the recursion level 0 requires $\lfloor \log m \rfloor + 1 \le \log(m+n)$ comparisons. For the recursion level 1 we need $\lfloor \log n_1^1 \rfloor + 1 + \lfloor \log(\min\{m_2^1, n_2^1\}) \rfloor + 1 \le 2 \cdot \log((m+n)/2)$ comparisons since $m_1^1 + n_1^1 = \lfloor (m+n)/2 \rfloor$ and $m_2^1 + n_2^1 = \lceil (m+n)/2 \rceil$ by Corollary 2.2. Further it holds for all $i \ge 1$ each (m_2^i, n_2^i) is divided into two pairs (m_1^{i+1}, n_1^{i+1}) and (m_2^{i+1}, n_2^{i+1}) where it holds $m_1^{i+1} + n_1^{i+1} = \lfloor (m_2^i + n_2^i)/2 \rfloor$ and $m_2^{i+1} + n_2^{i+1} = \lceil (m_2^i + n_2^i)/2 \rceil$. Therefore the overall number of comparisons for all m levels does not exceed $\log(m+n) + 2\sum_{i=1}^{m-1}\log((m+n)/2^i) = \log(m+n) + 2((m-1)\log(m+n) - \sum_{i=1}^{m-1} i) = O(m\log(n/m+1))$.

Now we consider the complexity for *case I* in general. Fig. 5 shows all m levels of the maximum spanning case can be partitioned into $k+1$ recursion groups, say recursion group 0, recursion group 1, $\cdots$, recursion group k, so that each recursion group i, $(i = 1,2,\cdots,k)$ comprises 2^{i-1} recursion levels (Here, we assume $m = 2^k$ i.e. $k = \log m$). Note that if $m = 2^k$ then $m-1 = 2^0 + 2^1 + 2^2 + \cdots + 2^{k-1} = 2^0 + 2^1 + \cdots 2^{\log m - 1}$. Accordingly we get the following corollary:

Corollary 3.2. For the maximum spanning case each recursion group i, $(i = 0,1,2,\cdots,k)$ covers 2^i merging pairs (nodes).

There is a further special case where each merging pair (subsequence merging) always triggers two non-empty merging pairs. In this case each recursion group i is exactly equal to the recursion level i. Therefore recursion groups and recursion levels are identical. Case (a) of Fig. 6 shows such an example. Eventually, because of the condition of case I that every internal node (m_j^i, n_j^i) with $m_j^i > 1$ is divided into two pairs $(m_{j'}^{i+1}, n_{j'}^{i+1})$ and $(m_{j''}^{i+1}, n_{j''}^{i+1})$, with $m_{j'}^{i+1} \ge 1$ and $m_{j''}^{i+1} \ge 1$, we can always construct $k+1$ recursion groups so that each recursion group i, $i = 0,1,2,\cdots k$ covers at most 2^i pairs. Thus the following theorem holds:

Theorem 3.3. For case I the SymMerge algorithm needs $O(m\log(n/m+1))$ comparisons.

Proof. For any input (u,v) in case I, let S be the set consisting of all merging pairs (nodes) which arise during the computation. Then S can be partitioned into $k+1$ recursion groups such that each recursion group i, $i = 0,1,2,\cdots k$ has at most 2^i pairs and needs at most $2^i\log((m+n)/2^i)$ comparisons. Hence for all levels, the required number of comparisons is less than $\sum_{i=0}^{k} 2^i\log((m+n)/2^i) = \sum_{i=0}^{k} 2^i((\log(m+n)) - i)$ $= O(m\log(n/m+1))$ since $\sum_{i=0}^{k} i2^i = (k-1)2^{k+1} + 2$.

Case II: Complement of Case I

Now we investigate the worst case complexity for case II. We can state and prove the following basic results:

Lemma 3.4. Any node $(m_j^{\lceil \log\frac{n+m}{m} \rceil}, n_j^{\lceil \log\frac{n+m}{m} \rceil})$ on the recursion level $\lceil \log\frac{n+m}{m} \rceil$ satisfies $m_j^{\lceil \log\frac{n+m}{m} \rceil} + n_j^{\lceil \log\frac{n+m}{m} \rceil} \le m$.

Proof. As shown in Corollary 2.2, each call of the SymMerge decomposes the input sequence into two output subsequences with equal size. Therefore $m_j^{\lceil \log\frac{n+m}{m} \rceil} + n_j^{\lceil \log\frac{n+m}{m} \rceil} \le (n+m)/2^{\lceil \log\frac{n+m}{m} \rceil} \le m$.

Lemma 3.5. If $m = 2^k$, then $\lfloor \log m \rfloor + 1 + \sum_{j=1}^{2}(\lfloor \log m_j^1 \rfloor + 1) + \sum_{j=1}^{2^2}(\lfloor \log m_j^2 \rfloor + 1) + \cdots + \sum_{j=1}^{2^k}(\lfloor \log m_j^k \rfloor + 1) = O(m)$.

Proof.

$\lfloor \log m \rfloor + 1 + \sum_{j=1}^{2}(\lfloor \log m_j^1 \rfloor + 1) + \sum_{j=1}^{2^2}(\lfloor \log m_j^2 \rfloor + 1)$
$+ \cdots \sum_{j=1}^{2^k}(\lfloor \log m_j^k \rfloor + 1) \le \sum_{j=0}^{k}(2^j \cdot \log \frac{m}{2^j} + 2^j) = 4m$
$- \log m - 3 = O(m)$.

Similar to case I, we consider a special case, here the case $u < v$ or $v < u$, i.e. the merging represents the identity or a simple block rotation from uv to vu.

Subcase II.1: Case of $u < v$ or $v < u$ (identity or block rotation)
Since both cases have the same complexity, without loss of generality we assume $u < v$. To get the result vu, on the recursion level 1 it has to hold $m_1^1 = 0$, $m_2^1 = m$, $n_1^1 = \left\lfloor \frac{n}{2} + \frac{m}{2} \right\rfloor$, $n_2^1 = \left\lceil \frac{n}{2} - \frac{m}{2} \right\rceil$. If $m_2^1 \le n_2^1$, on the next recursion level we get $m_1^2 = 0$, $m_2^2 = m$, $n_1^2 = \left\lfloor \frac{1}{2} n_2^1 + \frac{m}{2} \right\rfloor$, $n_2^2 = \left\lceil \frac{1}{2} n_2^1 - \frac{m}{2} \right\rceil$. For the recursion level i, m_2^i has to be equal to m so long as we have $m = m_2^{i-1} \le n_2^{i-1}$. Accordingly, we get the followings:

Theorem 3.6. Let vu be the merged result of the input uv. Then the SymMerge algorithm needs $O(m \log(n/m+1))$ comparisons until reaching the recursion level $\left\lceil \log \frac{n+m}{m} \right\rceil$.

Proof. Every recursive call up to the recursion level $\left\lceil \log \frac{n+m}{m} \right\rceil$ requires $\lfloor \log m \rfloor + 1$ comparisons. Thus the overall number of comparisons until reaching the maximal indivisible depth of m is $(\lfloor \log m \rfloor + 1)(\left\lceil \log \frac{n+m}{m} \right\rceil) = O(\log m \log(n/m+1))$

Theorem 3.7. Let vu be the merged result of the input uv. Then the SymMerge algorithm needs $O((\log m)^2)$ comparisons for merging the sequences on the recursion level $\left\lceil \log \frac{n+m}{m} \right\rceil$.

Proof. By Corollary 2.2 and 3.4 the recursion depth of the sequences on the recursion level $\left\lceil \log \frac{n+m}{m} \right\rceil$ is bounded by $\lceil \log m \rceil$. On each recursion level the number of required comparisons is less than $(\lfloor \log m \rfloor + 1)$. Thus the overall number of required comparisons is less than $\lceil \log m \rceil \cdot (\lfloor \log m \rfloor + 1) = O((\log m)^2)$.

Accordingly, by Theorem 3.6 and Theorem 3.7 the following corollary holds:

Corollary 3.8. Let vu (or uv) be the merged result of the input uv. Then the SymMerge algorithm needs $O(\log m \log(n/m+1))$ comparisons for $n \ge m(m-1)$ and $O((\log m)^2)$ comparisons otherwise.

Now we consider the complexity for *case II* in general. In this case we can always divide all merged pairs (internal nodes) (m_j^i, n_j^i) generated during the computation into two disjoint parts; part (a) consisting of all nodes (m_j^i, n_j^i) with $m_j^i = 1$ and all nodes (m_j^i, n_j^i) which are partitioned into two nodes $(m_{j'}^{i+1}, n_{j'}^{i+1})$ and $(m_{j''}^{i+1}, n_{j''}^{i+1})$ with $m_{j'}^{i+1} > 0$ and $m_{j''}^{i+1} > 0$, part (b) consisting of all remaining nodes (m_j^i, n_j^i), $m_j^i \ge 2$ which is partitioned into two nodes $(m_{j'}^{i+1}, n_{j'}^{i+1})$ and $(m_{j''}^{i+1}, n_{j''}^{i+1})$ with $m_{j'}^{i+1} = 0$ or $m_{j''}^{i+1} = 0$.
In subcase II.1 (case of $u < v$ or $v < u$), all merged pairs (nodes) occurring during the computation belong to only part (b). As already shown in Theorem 3.6 and Theorem 3.7 it needs less than $(\lfloor \log m \rfloor + 1)(\left\lceil \log \frac{n+m}{m} \right\rceil) + \lceil \log m \rceil \cdot (\lfloor \log m \rfloor + 1) = O(m \log(n/m+1))$ comparisons.
Now, for better understanding the complexity analysis, we consider an additional case that is similar to case II.1. The case is described as follows: (m,n) is partitioned into (m_1^1, n_1^1) and (m_2^1, n_2^1) with $m_1^1 > 1$, $m_2^1 > 1$ and the remaining computations of the both nodes (m_1^1, n_1^1) and (m_2^1, n_2^1) correspond to the identity or block rotation. In this case, except of the node belonging to the recursion level 0, i.e. (m,n), all remaining nodes belong to part (b). Similar to subcase II.1, we can analyze the complexity for part (b) of this case. So, it needs less than $\left\lceil \log \frac{n+m}{m} \right\rceil \cdot (\lfloor \log m \rfloor + 1 + \sum_{j=1}^{2}(\lfloor \log m_j^1 \rfloor + 1) + \sum_{j=1}^{2^2}(\lfloor \log m_j^2 \rfloor + 1))$. Further by Lemma 3.5 it holds $\left\lceil \log \frac{n+m}{m} \right\rceil \cdot (\lfloor \log m \rfloor + 1 + \sum_{j=1}^{2}(\lfloor \log m_j^1 \rfloor + 1) + \sum_{j=1}^{2^2}(\lfloor \log m_j^2 \rfloor + 1))$
$< \left\lceil \log \frac{n+m}{m} \right\rceil \cdot (\lfloor \log m \rfloor + 1 + \sum_{j=1}^{2}(\lfloor \log m_j^1 \rfloor + 1)$
$+ \sum_{j=1}^{2^2}(\lfloor \log m_j^2 \rfloor + 1)) + \cdots + \sum_{j=1}^{2^k}(\lfloor \log m_j^k \rfloor + 1))$
$= O(m \log(n/m+1))$ comparisons.

Theorem 3.9. For case II the SymMerge algorithm needs $O(m \log(n/m+1))$ comparisons.

Proof. First we divide all internal nodes of any computation of case II into two disjoint parts; part (a) and part (b). By Theorem 3.3, part (a) can be partitioned into at most $k+1$ recursion groups such that each recursion group i, $i = 0,1,2,\cdots,k$ covers at most 2^i pairs

and needs at most $2^i \log((m+n)/2^i)$ comparisons. Hence the complexity for part (a) is bounded by $O(m\log(n/m+1))$. Now we consider the required number of comparisons for part (b). The set of all nodes belonging to part (b) is a proper subset of some set $\{(m,n),(m_1^1,n_1^1),(m_2^1,n_2^1),(m_1^2,n_1^2),\cdots,(m_1^k,n_1^k),\ \cdots\}$ where it holds $m_j^i > 0$ for all i and j. Thus the number of all required comparisons for part (b) does not exceed

$$\left\lceil \log\frac{n+m}{m} \right\rceil \cdot (\lfloor \log m \rfloor + 1 + \sum_{j=1}^{2}(\lfloor \log m_j^1 \rfloor + 1)$$
$$+ \sum_{j=1}^{2^2}(\lfloor \log m_j^2 \rfloor + 1) + \cdots + \sum_{j=1}^{2^k}(\lfloor \log m_j^k \rfloor + 1))$$

$= O(m\log(n/m+1))$ by Lemma 3.4 and Lemma 3.5.

Hence we conclude the following corollary by Theorem 3.3 and Theorem 3.9.

Corollary 3.10. The SymMerge algorithm is asymptotically optimal regardingthe number of comparisons.

4. Experimental Work

As already shown in [11], we did some benchmarking with the unfolded version of the SymMerge algorithm and compared it with the implementations of three other merging algorithms. As first competitor we chose the merge_without_buffer-function contained in the C++ Standard Template Libraries (STL) [12]. The second competitor was taken from [13]. As third competitor we took the classical standard algorithm. The result of our evaluation has shown that the SymMerge algorithm is very efficient and so might be of high practical interest.

5. Conclusion

We proved that the SymMerge algorithm is asymptotically optimal regarding the number of comparisons. The proof gained its simplicity from the special characteristic of the SymMerge algorithm to map a merging of size $l(l=m+n)$ always to two mergings of size $l/2$. This "even splitting" is an interesting property of its own, it allows e.g. even load balancing in the context of parallel architectures. By repeatedly evenly splitting the input, the task of merging can be distributed over several processing units without disturbing the overall asymptotic optimality regarding comparisons.

References

[1] D. E. Knuth, *The Art of Computer Programming*, Addison-Wesley, Vol. 3: Sorting and Searching, 1973.

[2] K. Dudzinski and A. Dydek. "On a stable storage merging algorithm." *Information Processing Letters*, Vol. 12, pp. 5-8, February 1981.

[3] A. Symvonis, "Optimal stable merging," *Computer Journal*, Vol. 38, pp. 681-690, 1995.

[4] V. Geffert, J. Katajainen and T. Pasanen, "Asymptotically efficient in-place merging," *Theoretical Computer Science*, Vol. 237, No. 1/2, pp. 159-181, 2000.

[5] J. Chen. "Optimizing stable in-place merging," *Theoretical Computer Science*, Vol. 302, No. 1/3, pp. 191-210, 2003.

[6] P. S. Kim and A. Kutzner, "On Optimal and Efficient in-Place Merging," *SOFSEM 2006, Lecture Notes in Computer Science, Springer*, Vol. 3831, pp. 350-359, 2006.

[7] P. S. Kim and A. Kutzner, "Ratio Based Stable in-Place Merging," *TAMC 2008, Lecture Notes in Computer Science, Springer*, Vol. 4978, pp. 246-257, 2008.

[8] M. A. Kronrod, "An optimal ordering algorithm without a field operation," *Dokladi Akad. Nauk SSSR*, Vol. 186, pp. 1256-1258, 1969.

[9] H. Mannila and E. Ukkonen, "A simple linear-time algorithm for in situ merging," *Information Processing Letters*, Vol. 18, pp. 203-208, 1984.

[10] F. Hwang and S. Lin, "A simple algorithm for merging two disjoint linearly ordered sets," *SIAM J. Comput.*, Vol. 1, no. 1, pp. 31-39, 1972.

[11] P. S. Kim and A. Kutzner, "Stable Minimum Storage Merging by Symmetric Comparisons," *ESA 2004, Lecture Notes in Computer Science, Springer*, Vol. 3221, pp. 714-723, 2004.

[12] C++ Standard Template Library, "http://www.sgi.com/tech/stl."

[13] K. Møllerhøj AND C.U. Søttrup, "Undersøgelse og implementation af effektiv inplace merge," tech. rep., *CPH STL Reports 2002-06, Department of Computer Science*, University of Copenhagen, Denmark, June 2002.

저 자 소 개

김복선 (Pok-Son Kim)
한국지능시스템학회 협력이사
현재 국민대학교 수학과 부교수

제 18권 2호 참조

쿠츠너 아네 (Arne Kutzner)
1999년 : 독일 University of Frankfurt 컴퓨터과학부 공학박사
2001년 : 독일 Dresdner 은행
2002년 : DLogistics Korea 대표이사
2003년 ~ 2008년 : 서경대학교 전자상거래과 교수
2009년 ~ 현재 : 한양대학교 제II 공대 정보시스템학과 부교수

관심분야 : Algorithm, Algorithm Engineering, Programming Languages
E-mail : kutzner@hanyang.ac.kr