Towards optimal sorting of 16 elements

Marcin PECZARSKI

Institute of Informatics, University of Warsaw,
ul. Banacha 2, 02-097 Warszawa, Poland
email: marpe@mimuw.edu.pl

Abstract. One of the fundamental problem in the theory of sorting is to find the pessimistic number of comparisons sufficient to sort a given number of elements. Currently 16 is the lowest number of elements for which we do not know the exact value. We know that 46 comparisons suffices and that 44 do not. There is an open question if 45 comparisons are sufficient. We present an attempt to resolve that problem by performing an exhaustive computer search. We also present an algorithm for counting linear extensions which substantially speeds up computations.

1 Introduction

We consider sorting by comparisons. One of the fundamental problem in that area is to find the pessimistic number $S(n)$ of comparisons sufficient to sort n elements. Steinhaus posed this problem in [8]. Knuth considered it in [4]. From the *information-theoretic lower bound*, further denoted by ITLB, we know that $S(n) \geq \lceil \log_2 n! \rceil = C(n)$. Ford and Johnson discovered [2] an algorithm, further denoted by FJA, which nearly and sometimes even exactly matches $C(n)$. Let $F(n)$ be the worst case number of comparisons in the FJA. It holds $S(n) = F(n) = C(n)$ for $n \leq 11$ and $n = 20, 21$. The FJA does not achieve the ITLB for $12 \leq n \leq 19$ and infinitely many $n \geq 22$. Carrying an exhaustive computer search, Wells discovered in 1965 [9, 10] that the FJA is optimal for 12 elements and $S(12) = F(12) = C(12) + 1 = 30$. Kasai et al. [3] computed $S(13) = F(13) = C(13) + 1 = 34$ in 1994, but that result was not widely

---

Computing Classification System 1998: F.2.2

Mathematics Subject Classification 2010: 06A07, 68P10

Key words and phrases: optimal sorting, poset, computer aided proof

known. It was discovered again a few years later [5], independently, extending the Wells method. Further improvement of the method led to show in years 2003–2004 [6, 7] that it holds $S(n) = F(n) = C(n) + 1$ for $n = 14, 15, 22$, similarly.

In this paper we consider the case $n = 16$. This is now the lowest number of elements for which we do not know the exact value of $S(n)$. The previous results could suggest that $S(16) = F(16) = C(16) + 1 = 46$. However Knuth conjectures that $S(16) = C(16) = 45$. He does not believe that the FJA is optimal for 16 elements. He wrote [4]: "There must be a way to improve upon this!" We present recently obtained results[1] aiming to compute the value of $S(16)$. It is very unlikely that someone will find it by pure theoretical consideration. It seems that the only promising way leads by performing an exhaustive computer search supported by cleaver heuristics.

The paper is organized as follows. In Section 2 we introduce notation used throughout the paper. In Section 3 we briefly describe the algorithm we use to resolve if there exists a sorting algorithm for a given number of elements and comparisons. We analyse why the ITLB is not achieved for 13, 14 and 15 elements in Section 4. We present the newest results for 16 elements in Section 5. In Section 6 we compare the computation complexity of the previous cases and the case of 16 elements. Finally, in Section 7, we present the algorithm for counting linear extensions which substantially improves the algorithm from Section 3.

2 Notation

We denote by $U = \{u_0, u_1, \ldots, u_{n-1}\}$ an n-element set to be sorted. Sorting of the set U is represented as a sequence of posets $(P_c = (U, R_c))_{c=0,1,\ldots,C}$, where R_c is a partial order relation over a set U. Sorting starts from the total disorder $P_0 = (U, R_0)$, where $R_0 = \{(u, u) : u \in U\}$. After performing c comparisons we obtain a poset $P_c = (U, R_c)$. Sorting should end with a linear order P_C. Assume that elements u_j and u_k are being compared in step c. Without loss of generality we can assume that $(u_j, u_k) \notin R_{c-1}$ and $(u_k, u_j) \notin R_{c-1}$. Suppose the answer to the comparison is that *element u_j is less than element u_k*. Then we obtain the next poset $P_c = (U, R_c)$, where the relation R_c is the transitive closure of the relation $R_{c-1} \cup \{(u_j, u_k)\}$. We denote this by $P_c = P_{c-1} + u_j u_k$.

By $e(P)$ we denote the number of linear extensions of a poset $P = (U, R)$.

We assume that $e(P + u_j u_k) = e(P)$ and $e(P + u_k u_j) = 0$ if elements u_j, u_k are in relation, i.e., if $(u_j, u_k) \in R$.

3 The algorithm

In this section we remember briefly the algorithm which answers if sorting of a given poset P_0 can be finished in C comparisons. The algorithm was invented in [9, 10] and improved in [5] and later in [6]. We present the next improvement to the algorithm in Section 7. The algorithm has two phases: forward steps and backward steps.

In the forward steps we consider a sequence of sets $(S_c)_{c=0,1,...,C}$. The set S_0 contains only the poset P_0. In step c we construct the set S_c from the set S_{c-1}. Every poset $P \in S_{c-1}$ is examined for every unrelated pair (u_j, u_k) in order to verify whether it can be sorted in the remaining $C - c + 1$ comparisons. As the result of the comparison of u_j and u_k one can get one of two posets $P_1 = P + u_j u_k$ or $P_2 = P + u_k u_j$. If the number of linear extensions of P_1 or P_2 exceeds 2^{C-c} then by the ITLB it cannot be sorted in the remaining $C - c$ comparisons. It follows that in this case, in order to finish sorting in $C - c + 1$ comparisons, elements u_j and u_k should not be compared in step c. If the number of linear extensions of both P_1 and P_2 do not exceed 2^{C-c} then we store one of them in the set S_c, namely that with greater number of linear extensions. If both have the same number of linear extensions we choose P_1 arbitrarily. We do not store isomorphic posets or a poset which dual poset is isomorphic to some already stored poset.

If some set S_c in the sequence appears to be empty then we conclude that the poset P_0 cannot be sorted in C comparisons. Such results are received for 12 and 22 elements and $C = C(n)$ [6], where the set S_{23} and S_{40} is empty, respectively. Wells reported [10] that for $n = 12$ only the set S_{24} is empty. Those results mean that $S(n) > C(n)$ for $n = 12, 22$. If the set S_C is not empty after performing forwards steps, we cannot conclude about sorting of the poset P_0. In that case we continue with backward steps.

In the backward steps we consider the sequence of sets $(S_c^*)_{c=0,1,...,C}$. We start with the set $S_C^* = S_C$ which contains only a linear order of the set U. In step c, where $c = C - 1, C - 2, \ldots, 0$, we construct the set S_c^* from the set S_{c+1}^*. The set S_c^* is a subset of the set S_c and contains only posets which can be sorted in the remaining $C - c$ comparisons. Poset $P \in S_c$ is stored in S_c^* iff there exists in P a pair of unrelated elements (u_j, u_k) such that poset $P_1 = P + u_j u_k$ or poset $P_2 = P + u_k u_j$ belongs to the set S_{c+1}^* (as previously we identify isomorphic and dual posets) and both posets are sortable in $C - c - 1$

comparisons. Therefore we store the poset P in the set $\mathcal{S}_c^*$ iff both $P_1, P_2 \in \mathcal{S}_{c+1}^*$ or $P_1 \in \mathcal{S}_{c+1}^*$ and P_2 is sortable in $C - c - 1$ comparisons or $P_2 \in \mathcal{S}_{c+1}^*$ and P_1 is sortable in $C - c - 1$ comparisons. Sortability of P_1 or P_2 can be checked recursively using the same algorithm.

If some set $\mathcal{S}_c^*$ in the sequence appears to be empty then we conclude that the poset P_0 cannot be sorted in C comparisons. On the other hand, if the set $\mathcal{S}_0^*$ is not empty, it contains the poset P_0 and we conclude that the poset P_0 can be sorted in C comparisons. For $n = 13, 14, 15$ and $C = C(n)$ we received that the set $\mathcal{S}_{15}^*$ is empty [5, 6, 7], which means that $S(n) > C(n)$ for $n = 13, 14, 15$. We analyze those results in detail in the next section.
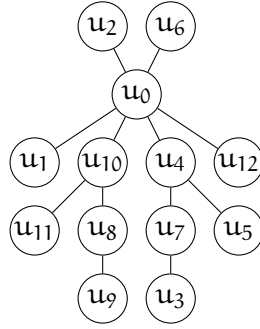

Figure 1: The poset P_{16}, $e(P_{16}) = 113400$

4   The previous cases

The computer experiment for $n = 13$ and $C = C(n)$ returns that the set $\mathcal{S}_{15}^*$ is empty, which means that $S(13) = F(13) = C(13) + 1 = 34$ [5]. In that experiment the set $\mathcal{S}_{16}^*$ contains only one poset P_{16}, whose Hasse diagram is shown in Figure 1. The poset P_{16} can be obtained from a poset contained in the file $\mathcal{S}_{15}$ in two ways:

- we compare elements u_0 and u_{10} in the poset $P_{15}' \in \mathcal{S}_{15}$ shown in Figure 2(a); if $u_0 > u_{10}$ we obtain the poset P_{16}; if $u_0 < u_{10}$ we obtain the poset Q_{16}' shown in Figure 3(a);

- we compare elements u_0 and u_6 in the poset $P_{15}'' \in \mathcal{S}_{15}$ shown in Figure 2(b); if $u_0 < u_6$ we obtain the poset P_{16}; if $u_0 > u_6$ we obtain the poset Q_{16}'' shown in Figure 3(b).

(a) $e(P'_{15}) = 222750$ (b) $e(P''_{15}) = 238140$
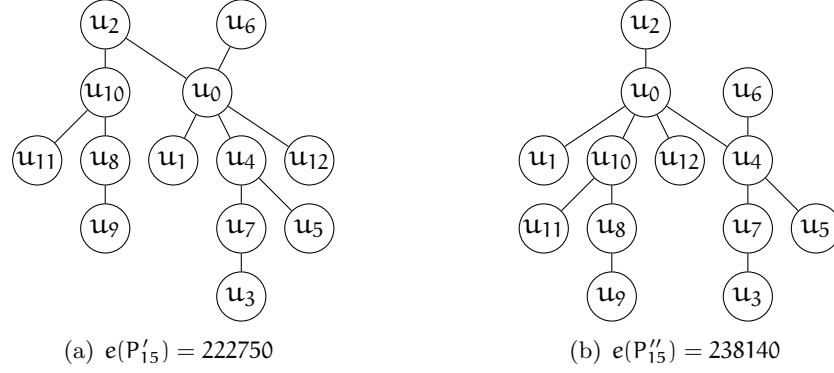
Figure 2: The posets P'_{15} and P''_{15}

Neither the poset P'_{15} nor the poset P''_{15} can be stored in the file $\mathcal{S}^*_{15}$, because neither the poset Q'_{16} nor the poset Q''_{16} can be sorted in the remaining $C-16 = 17$ comparisons. It is quite surprising that the posets Q'_{16}, Q''_{16} cannot be sorted. The poset Q'_{16} has less linear extensions than the poset P_{16}, which intuitively should make it easier to sort. Indeed, the poset Q''_{16} has more linear extensions than the poset P_{16}, which intuitively makes it harder to sort. On the other hand, there are known the two largest elements of the poset Q''_{16}, which intuitively makes it easier to sort. The poset P_{16} is sortable in 17 comparisons because of its symmetry.

Similar results were received in the computer experiments for $n = 14, 15$ and $C = C(n)$, i.e., $S(14) = F(14) = C(14) + 1 = 38$ and $S(15) = F(15) = C(15) + 1 = 42$. In both cases the file $\mathcal{S}^*_{16}$ contains only one poset, namely the poset P_{16} extended by one isolated element u_{13} (for $n = 14$) or two isolated elements u_{13}, u_{14} (for $n = 15$), respectively. In both cases the file $\mathcal{S}^*_{15}$ is empty and the reason is the same. The posets P'_{15}, Q''_{15}, Q'_{16}, Q''_{16} extended by u_{13} or u_{13}, u_{14} are observed, respectively, and neither Q'_{16} nor Q''_{16} is sortable in the remaining $C - 16$ comparisons. Note that for $n = 14$ we have $C - 16 = C(n) - 16 = 21$ and for $n = 15$ we have $C - 16 = C(n) - 16 = 25$.

5   The case of 16 elements

In this section we describe an attempt to find for $n = 16$ a sorting algorithm better than the FJA or to exclude the existence of such algorithm. Before starting a long time computation it was checked if the scenario from

(a) $e(Q'_{16}) = 109350$ (b) $e(Q''_{16}) = 124740$
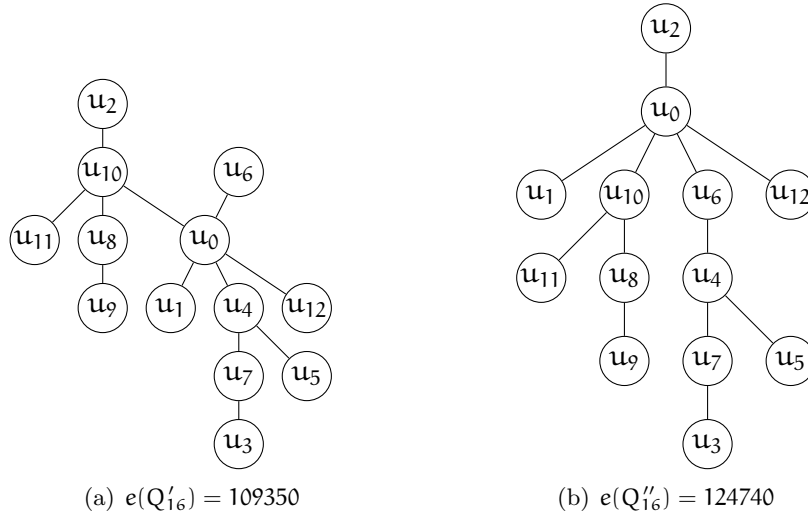
Figure 3: The posets Q'_{16} and Q''_{16}

the previous section repeats for $n = 16$. The posets Q'_{16}, Q''_{16} were extended by three isolated elements u_{13}, u_{14}, u_{15}. As previously, the experiment returned that neither the poset Q'_{16} nor the poset Q''_{16} can be sorted in the remaining $C - 16 = C(n) - 16 = 29$ comparisons. Of course, this result does not exclude the existence of the desired algorithm.

To find the exact value of $S(16)$ the algorithm from Section 3 with improvement from Section 7 is applied. Because the search space is very reach, the problem is divided into smaller subproblems. Let $T(k)$ be the number of elements which were compared (touched) by a sorting algorithm in the first k comparisons. Observe that $T(k_1) \leq T(k_2)$ for $k_1 < k_2$. A sorting algorithm for 16 elements, using at most $C(16) = F(16) - 1 = 45$ comparisons, is examined for possible values of $T(k)$.

The first experiment returned that if $S(16) = 45$ then it holds $T(15) < 16$. Note that for the FJA we have $T(k) = 16$ for $k \geq 8$. Hence a hypothetical algorithm, using for 16 elements pessimistically less comparisons than the FJA, must be complete different from the FJA. It must differ from the FJA already before the 9th comparison. This is quite surprising, when we look at regular structure of the first 15 comparisons in the FJA. The next experiment showed that if $S(16) = 45$ then $T(15) > 11$, which is already not surprising.

n	Pentium II 233 MHz 2002	Pentium III 650 MHz 2003	Opteron 246 2 GHz 2004	Core 2 Duo 2.13 GHz 2007
13	10 hr. 30 min.	41 min.	10 min. 44 sec.	46 sec.
14		391 hr. 37 min.	44 hr. 10 min.	4 hr. 31 min.
15			17554 hr.	

Table 1: Computation times

6 Computation complexity

Computation complexity of the method groves exponentially. The case $S(13)$ needed in year 2002 [5] more than 10 hours of CPU time. The value of $S(14)$ was computed one year later (published in 2004 [5]) and took about 392 hours on faster computer and using improved algorithm, which could solve $S(13)$ in about 40 minutes. Further progress in hardware allowed to compute the value of $S(15)$ in year 2004 (published only in 2007 [7]) using about 17500 hours of CPU time. Each next case required significant improvements in the algorithm or hardware. The progress is presented in Table 1. One can argue that the comparison is not fair, because the machines used in the experiments are different. The purpose of this table is to show an overall improvement in software and hardware, and to give a filling, how difficult the case of 16 elements could be. The about 10 times improvement observed between the second last and the last column is due mainly to the algorithm described in the next section. Note that for Core 2 Due processor both cores were used in parallel. A few years of CPU time was used up to now to search for an algorithm achieving the ITLB for $n = 16$. The computation that $T(15) < 16$ and $T(15) > 11$ took about 20000 and 7000 hours, respectively. Computation for the next case $T(15) = 12$ is currently in progress. It used up to now more than 25000 hours.

7 Counting linear extensions

The most time consuming part of the algorithm presented in Section 3 is counting linear extensions of a given poset. In this section we describe the algorithm for counting linear extensions which is inspired by [1] and which substantially improves computations. For a given poset $P = (U, \prec)$ the algorithm computes $e(P)$ and the table $t[j, k] = e(P + u_j u_k)$ for $j \neq k$.
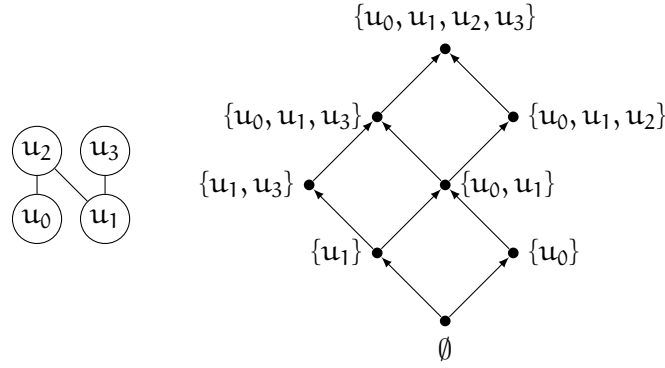
Figure 4: A poset and the graph of its downsets

Let $P = (U, \prec)$ be a poset. A subset $D \subseteq U$ is called a down set of the poset P if for each $x \in D$ all elements $y \in U$ preceding x (i.e., $y \prec x$) also belong to D. We consider a directed acyclic graph $\mathcal{G}$ whose nodes are all downsets of P. For two nodes D_1 and D_2 there is an edge (D_1, D_2) if there exists $x \in U \setminus D_1$ such that $D_2 = D_1 \cup \{x\}$. An example of a poset and its graph of downsets is shown in Figure 4, where $U = \{u_0, u_1, u_2, u_3\}$.

Let $d(D)$ denote the number of linear extensions of the poset $(D, \prec)$ which is the poset P reduced to the down set D. Let $u(D)$ denote the number of linear extensions of the poset $(U \setminus D, \prec)$ which is the poset P reduced to the complementary set of the down set D. We have [1]

$$d(D) = \sum_{(X,D)} d(X), \tag{1}$$

where the sum is taken over all edges (X, D) in the graph $\mathcal{G}$ incoming to the node D. We assume $d(\emptyset) = 1$. Observe that $d(U) = e(P)$. All values of $d(D)$ are computed using the DFS in the graph $\mathcal{G}$, starting at the node U and going down, i.e., in the opposite direction to the edges. Similarly, it holds [1]

$$u(D) = \sum_{(D,X)} u(X), \tag{2}$$

where the sum is taken over all edges (D, X) in the graph $\mathcal{G}$ outgoing from the node D. We assume $u(U) = 1$. Observe that $u(\emptyset) = e(P)$. All values of $u(D)$ are computed using the second DFS in the graph $\mathcal{G}$, starting at the node $\emptyset$
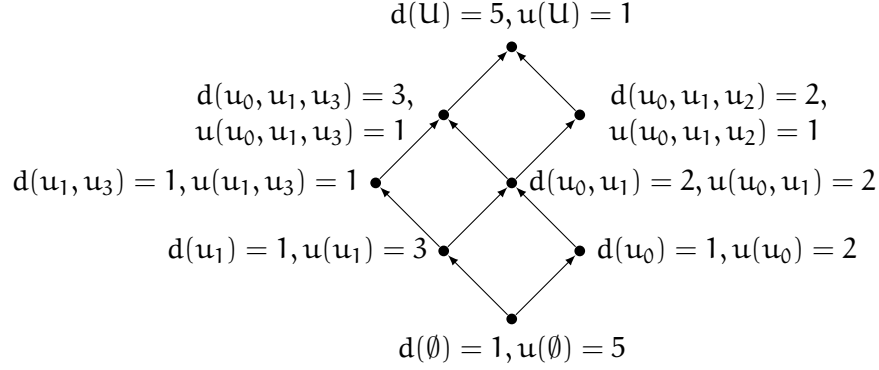
$$d(U) = 5, u(U) = 1$$

$$d(u_0, u_1, u_3) = 3,$$
$$u(u_0, u_1, u_3) = 1$$

$$d(u_0, u_1, u_2) = 2,$$
$$u(u_0, u_1, u_2) = 1$$

$$d(u_1, u_3) = 1, u(u_1, u_3) = 1$$

$$d(u_0, u_1) = 2, u(u_0, u_1) = 2$$

$$d(u_1) = 1, u(u_1) = 3$$

$$d(u_0) = 1, u(u_0) = 2$$

$$d(\emptyset) = 1, u(\emptyset) = 5$$

Figure 5: The numbers of linear extensions of the downsets and they complementary sets

		k			
		0	1	2	3
	0	–	2	5	4
j	1	3	–	5	5
	2	0	0	–	2
	3	1	0	3	–

Table 2: The values of $t[j, k]$

and going up. Values of $d(D)$ and $u(D)$ for the graph in Figure 4 are shown in Figure 5. The curly braces are omitted for clarity, e.g., instead of $d(\{u_0\})$ we write $d(u_0)$. The table t can be computed from the equation

$$t[j, k] = \sum_{(V,W)} d(V)u(W), \tag{3}$$

where the sum is taken over all edges (V, W) in the graph $\mathcal{G}$ such that $W = V \cup \{u_j\}$ and $u_k \in U \setminus W$. For a proof see [1]. This computation is done altogether with the second DFS. For the graph in Figure 4 the values $t[j, k]$ are included in Table 2.

For a given poset on an n-element set its graph of downsets can have up to 2^n nodes. We implemented the graph as a table of the size 2^n. The table is indexed by downsets. The index is the characteristic function of the set D,

i.e., the index is the n-bit number, where bit j is set iff $u_j \in D$. Graph $\mathcal{G}$ is not constructed explicitly. When we proceed a node D all incoming and outgoing edges are easily computable from a poset representation. We hold at position D in the table only two numbers $d(D)$, $u(D)$ and visited time stamp $v(D)$ needed to implement the DFS. We initialize the table only once at the beginning of the program by setting all $v(D) = 0$. We also hold the global visited time stamp v_t initialized to 0. Starting a new DFS we increment the time stamp v_t. If we proceed a node D and $v_t > v(D)$ then it means that the node D was not yet visited in the current DFS run. If $v_t = v(D)$ then the node was already visited. We do not need to reinitialize the table before the next DFS. This is very important and decreases running time. The algorithm is very efficient for small n, because with a high probability the whole graph resides in a processor cache memory.

References

[1] K. De Loof, H. De Meyer, B. De Baets, Exploiting the lattice of ideals representation of a poset, *Fund. Inform.* **71** (2006) 309–321. ⇒221, 222, 223

[2] L. Ford, S. Johnson, A tournament problem, *Amer. Math. Monthly* **66** (1959) 387–389. ⇒215

[3] T. Kasai, S. Sawato, S. Iwata, Thirty four comparisons are required to sort 13 items, in: *Logic, Language, and Computation: Festschrift in Honor of Satoru Takasu* (eds. N. D. Jones, M. Hagiya, M. Sato), *Lecture Notes* in Comput. Sci. **792** (1994) 260–269. ⇒215

[4] D. E. Knuth, *The Art of Computer Programming. Vol. 3. Sorting and Searching* (second edition), Addison–Wesley, Reading, MA, 1998. First edition: Addison–Wesley, Reading, MA, 1973. ⇒215, 216

[5] M. Peczarski, Sorting 13 elements requires 34 comparisons, in: *Proceedings of the 10th Annual European Symposium on Algorithms* (eds. R. Möhring, R. Raman), *Lecture Notes* in Comput. Sci. **2461** (2002) 785–794. ⇒216, 217, 218, 221

[6] M. Peczarski, New results in minimum-comparison sorting, *Algorithmica* **40** (2004) 133–145. ⇒216, 217, 218

[7] M. Peczarski, The Ford–Johnson algorithm still unbeaten for less than 47 elements, *Inform. Process. Lett.* **101** (2007) 126–128. ⇒216, 218, 221

[8] H. Steinhaus, *Mathematical Snapshots*, Dover Publications, Mineola, NY, 1981. First edition: Oxford University Press, New York, NY, 1950. ⇒215

[9] M. Wells, Applications of a language for computing in combinatorics, in: *Proceedings of the 1965 IFIP Congress, Information Processing* **65**, North-Holland, Amsterdam, 1966, 497–498. ⇒215, 217

[10] M. Wells, *Elements of Combinatorial Computing*, Pergamon Press, Oxford, 1971. ⇒215, 217