Proceedings of the 8th International Conference on Applied Informatics
Eger, Hungary, January 27–30, 2010. Vol. 1. pp. 379–386.

Quick Testing of Random Sequences*

Antal Iványi, Imre Kátai

Dept. of Computer Algebra of Eötvös Loránd University

Abstract

Let ξ be a random integer sequence, having uniform distribution

$$\mathbf{P}\{\xi = (i_1, i_2, \ldots, i_n) = 1/n^n\} \quad \text{for} \quad 1 \leq i_1, i_2, \ldots, i_n \leq n.$$

A realization $(i_1, i_2, \ldots, i_n)$ of ξ is called *good*, if its elements are different. We present seven algorithms which decide whether a given realization is good. The investigated problem is connected with design of experiments [3, 13] and with testing of the solutions of Latin [1, 4, 10, 11, 12, 18, 19, 21] and Sudoku puzzles [2, 5, 7, 8, 9, 20, 22, 23, 24, 25, 27].

In this short paper (which contains the talk [15]) we have space only to define seven algorithms (using the pseudocode of [6]) and to summarize the basic properties of their running times (see Table 1 at the end of the paper). The proofs of the assertions can be found in [16].

Keywords: random sequences, efficient algorithms

MSC: 68Q25

1. Algorithms

The inputs of the following seven algorithms are n (the length of the sequence $\mathbf{s}$) and $\mathbf{s}$ $(= (s_1, s_2, \ldots, s_n)$: a sequence of nonnegative integers with $1 \leq s_j \leq n$ for $1 \leq j \leq n)$ in all cases. The output is always a logical variable g (its value is True, if the input sequence is good, and False otherwise).

The working variables are usually the cycle variables i and j.

1.1. Definition of the algorithm Forward

Forward compares the first (i_1), second (i_2), ..., last but one (i_{n-1}) element of the realization with the following elements until the first collision or until the last pair of elements.

*The research was supported by the project TÁMOP-4.2.1.B-09/1/KMR–2010-003 of Eötvös Loránd University.

FORWARD$(n, \mathbf{s})$

```
01 g ← TRUE
02 for i ← 1 to n − 1
03     do for j ← i + 1 to n
04         do if s[i] = s[j]
05             then g ← FALSE
06                  return g
07 return g
```

The number of the necessary comparisons in line 04 is $C_{\text{best}}(n, \text{FORWARD}) = 1 = \Theta(1)$ in the best case, and $C_{\text{worst}}(n, \text{BACKWARD}) = n(n-1)/2 = \Theta(n^2)$ in the worst case.

Using Lemma 1.1 [14, 17] one can show that the expected number of the necessary comparisons is $C_{\text{exp}}(n, \text{FORWARD}) = n = \Theta(n)$.

Lemma 1.1. *Let η_n be a random variable defined for $n = 1, 2, \ldots$ and $k = 1, 2, \ldots, n$ as*

$$P(\eta_n = k) = P(i_1, i_2, \ldots, i_k \text{ are different and } i_{k+1} = i_j \text{ for some } j, \text{ where } 1 \le j \le k).$$

Then

$$E(\eta_n) = \left(\frac{\pi n}{2}\right)^{1/2} - 1/3 + \epsilon_n,$$

where ϵ_n tends to zero when n tends to infinity.

1.2. Definition of algorithm BACKWARD

BACKWARD compares the second (i_2), third (i_3), ..., last (i_n) element of the realization with the previous elements until the first collision or until the last pair of elements.

BACKWARD$(n, \mathbf{s})$

```
01 g ← TRUE
02 for i ← 2 to n
03     do for j ← i − 1 downto 1
04         do if s[i] = s[j]
05             then g ← FALSE
06                  return g
07 return g
```

The number of the necessary comparisons in line 04 is $C_{\text{best}}(n, \text{BACKWARD}) = 1 = \Theta(1)$ in the best case, and $C_{\text{worst}}(n, \text{BACKWARD}) = n(n-1)/2 = \Theta(n^2)$ in the worst case.

Using Lemma 1.1 one can show that the expected number of the necessary comparisons is $C_{\text{exp}}(n, \text{BACKWARD}) = n = \Theta(n)$.

Although the order of growth of the expected number of the necessary comparisons is the same for FORWARD and BACKWARD, the concrete values are different, if $n \ge 4$.

1.3. Definition of algorithm LINEAR

LINEAR writes zero into the elements of an n length vector $\mathbf{v} = (v_1,\ v_2,\ \ldots,\ v_n)$, then investigates the elements of the realization and if $v[s_i] > 0$ (signalising a repetition), then stops, otherwise adds 1 to v_k.

```
LINEAR(n, s)
01 g ← TRUE
02 for i ← 1 to n
03     do v[i] ← 0
04 for i ← 1 to n
05     do if v[s[i]] > 0
06           then g ← FALSE
07                return g
08           else v[s[i]] ← v[s[i]] + 1
09 return g
```

LINEAR needs assignments in lines 03 and 08, and it needs comparisons in line 05. The number of assignments in line 03 is equals to n for arbitrary input and varies between 1 and n in line 08. The number of c omparisons also varies between 1 and n in line 08. Therefore the running time of RANDOM is $\Theta(n)$ in the best, worst and expected case too.

1.4. Definition of algorithm RANDOM

RANDOM generates random pairs of elements and tests them until it finds two identical elements or it tested all the possible pairs of $\mathbf{s}$. It uses the procedure RAN(k) [6] generating a random integer value distributed uniformly in the interval $[1, k]$.

```
RANDOM(n, s)
01 g ← TRUE
02 while g = TRUE
03       do i ← RAN(n^2)
04          j ← (RAN(n^2 − 1) + j)  (mod n^2)
05          if s_i = s_j
06            then g ← FALSE
07 return g
```

RANDOM needs only $1 = \Theta(1)$ time in the best case. In the worst case its running time can be arbitrary large but the probability of a large running time is small.

The problem of the expected case of RANDOM is known as the *coupon collector's problem* (see [6, page 109–110] or [26]).

Lemma 1.2. *The expected running time of* RANDOM *is*

$$C_{exp}(n, \text{RANDOM}) = \Theta(n).$$

.

Proof. Algorithm RANDOM can get two types of input: it gets a good input with probability $n!/n^n$ and a bad input with probability $(n^n - n!)/n^n$.

In the case of a good input the algorithm needs $n(n-1)/2$ different comparisons to observe that the investigated input is good. According to the known solution of the coupon collector's problem the expected number of the necessary comparisons is

$$C_{\text{good}} = n \sum_{i=1}^{n(n-1)/2} \frac{1}{i} = \Theta(n \log n). \tag{1.1}$$

If RANDOM gets a bad input, then

$$\sum_{i=0}^{\infty} \left(\frac{n-1}{n} \right)^i = 1 + n$$

is an upper bound of its expected running time, and so

$$C_{\text{exp}}(n, \text{RANDOM}) \leq n = \frac{n!}{n^n} \Theta(n \log n) + \frac{n^n - n!}{n^n}(1 + n) = \Theta(n).$$

□

1.5. Definition of algorithm TREE

TREE builds a random search tree from the elements of the realization and finishes the construction of the tree if it finds the following element of the realization in the tree (then the realization is not good) or it tested the last element too without a collision (then the realization is good).

```
TREE(n, s)
01 g ← TRUE
02 let s[1] be the root of a tree
03 for i ← 2 to n
04     if [s[i] is in the tree
05         then g ← FALSE
06              return
07         else  insert s[i] in the tree
08 return g
```

The worst case running time of TREE appears when the input contains different elements in increasing or decreasing order. Then the result is $\Theta(n^2)$. The best case is when the first two elements of **s** are equal: tin this case $C_{\text{best}}(n, \text{TREE}) = 1 = \Theta(1)$.

Using the known fact that the expected height of a random search tree is $\Theta(\log n)$ and Lemma 1.1 we can get that the order of the expected running time is $\sqrt{n} \log n$.

Lemma 1.3. *The expected running time of* TREE *is*

$$C_{exp}(n, \text{TREE}) = \Theta(\sqrt{n} \log n).$$

1.6. Definition of algorithm GARBAGE

This algorithm is similar to LINEAR, but it works without the setting zeros into the elements of a vector requiring linear amount of time.

Beside the cycle variable i GARBAGE uses as working variable also a vector $\mathbf{v} = (v_1, v_2, \ldots, v_n)$. Interesting is that $\mathbf{v}$ is used without initialization, that is its initial values can be arbitrary integer numbers.

```
GARBAGE(n, s)
01 g ← TRUE
02 for i ← 1 to n
03     do if v[s[i]] < i and s[v[s[i]]] = s[i]
04           then g ← FALSE
05                return g
06           else v[s[i]] ← i
07 return g
```

Lemma 1.4. *The expected running time of* GARBAGE *is*

$$C_{exp}(n, \text{GARBAGE}) = \Theta(\sqrt{n}).$$

1.7. Definition of algorithm MODULAR

MODULAR handles the queues $Q_0,\ Q_1,\ \ldots,\ Q_{m-1}$ (where $m = \lceil \sqrt{n} \rceil$ and puts the element s_j into the Q_i if s_j gives a residue i mod m. MODULAR tests weather s_j appeared earlier in Q_i using linear search.

For the simplicity let us suppose that n is a square.

```
MODULAR(n, s)
01 g ← TRUE
02 let s[1] be the root of a tree
03 for i ← 2 to n
04     if [s[i] is in the tree
05       then g ← FALSE
06            return
07       else insert s[i] in the tree
08 return g
```

Lemma 1.5. *The expected running time of* MODULAR *is*

$$C_{exp}(n, \text{MODULAR}) = \Theta(\sqrt{n}).$$

In the proof of this lemma central role plays Lemma 1.1.

1.8. Summary

Table 1 summarises the basic properties of the running times of the investigated algorithms.

Index and Algorithm	$T_{\text{best}}(n)$	$T_{\text{worst}}(n)$	$T_{\text{exp}}(n)$
1. FORWARD	$\Theta(1)$	$\Theta(n^2)$	$\Theta(n)$
2. BACKWARD	$\Theta(1)$	$\Theta(n^2)$	$\Theta(n)$
3. LINEAR	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
4. RANDOM	$\Theta(1)$	$\Theta(n^2 \log n)$	$\Theta(n)$
5. TREE	$\Theta(1)$	$\Theta(n^2)$	$\Theta(\sqrt{n} \log n)$
6. GARBAGE	$\Theta(1)$	$\Theta(n)$	$\Theta(\sqrt{n})$
7. MODULAR	$\Theta(1)$	$\Theta(n\sqrt{n})$	$\Theta(\sqrt{n})$

Table 1: The running times of the investigated algorithms in best, worst and expected cases

We used in our calculations the RAM computation model [6]. If the investigated algorithms run on real computers then we have to take into account also the limited capacity of the memory locations and the increasing execution time of the elementary arithmetical and logical operations.

Acknowledgements. The authors thank Tamás F. Móri, Péter Burcsi, and Attila Kiss, teachers of Eötvös Loránd University for the useful consultation.

References

[1] ADAMS, P., BRYANT, D., BUCHANAN, M. Completing partial Latin squares with two filled rows and two filled columns. *Electron. J. Combin.* Vol. 15, No. 1 (2008), Research paper 56, 26 pp. MR2398848 (2009h:05035). http://www.combinatorics.org/.

[2] BAILEY, R. A., CAMERON, P. J., CONNELLY, R., Sudoku, gerechte designs, resolutions, affine space, spreads, reguli, and Hamming codes. *Amer. Math. Monthly,* Vol. 115, No. 5 (2008), 383–404. MR2408485. http://www.math.cornell.edu/∼connelly/bcc_sudokupaper.pdf.

[3] BEHRENS, W. U. Feldversuchsanordnungen mit verbessertem Ausgleich der Bodenunterschiede, *Zeitschrift für Landwirtschaftliches Versuchs- und Untersuchungswesen,* Vol. 2, (1956), 176–193.

[4] BUCHANAN, H. L. II, FERENCAK, M. N. On completing Latin squares. *J. Combin. Math. Combin. Comput.* Vol. 34 (2000), 129–132. MR1772791 (2001b:05042). http://www.cs.elte.hu/ annalesm/

[5] CHEN, Z., Heuristic reasoning on graph and game complexity of sudoku. 6 pages. ARXIV, 2010. http://arxiv.org/PS_cache/arxiv/pdf/0903/0903.1659v1.pdf.

[6] CORMEN, T. H., LEISERSON, C. E., RIVEST, R. L., STEIN, C., *Introduction to Algorithms.* The MIT Press, 2009.

[7] CROOK, J. F. A pencil-and-paper algorithm for solving Sudoku puzzles. *Notices Amer. Math. Soc.,* Vol. 56 (2009), 460–468. MR2482305 (2010c:05022). http://www.ams.org/notices/200904/tx090400460p.pdf

[8] Dahl, G. Permutation matrices related to Sudoku. *Linear Algebra Appl.* Vol. 430 (2009), 2457–2463. MR2508304 (2010d:05022). http://www.sciencedirect.com/science/journal/00243795

[9] Dénes, J., Keedwell, A. D., *Latin Squares. New Developments in the Theory and Applications*. North-Holland, Amsterdam, 1991.

[10] Easton, T., Parker, R. G. On completing Latin squares. *Discrete Appl. Math.* Vol. 113, No. 2-3 (2001), 167–181. MR1857774 (2002j:05029)

[11] Euler, R. On the completability of incomplete Latin squares. *European J. Combin.* Vol. 31 (2010), 535–552. MR2565345. http://www.sciencedirect.com/science/journal/01956698.

[12] Hajirasouliha, I., Jowhari, H., Kumar, R., Sundaram, R. On completing Latin squares. *Lecture Notes in Comput. Sci.*, Vol. 4393 (STACS2007), 524–535. Springer, Berlin, 2007. MR2362490 (2008k:68122). http://www.springerlink.com/content/105633/.

[13] Heppes, A., Révész, P. A new generalization of the concept of latin squares and orthogonal latin squares and its application to the design of experiments (in Hungarian). *Magyar Tud. Akad. Mat. Int. Közl.* 1 (1956), 379–390.

[14] Iványi, A., Kátai, I., Estimates for speed of computers with interleaved memory systems, *Annales Univ. Sci. Budapest., Sectio Mathematica*, Vol. 19 (1976), 159–164.

[15] Iványi, A., Kátai, I., Quick testing of random variables. In: *Abstracts of ICAI'2010.* http://icai.ektf.hu/index.php?p=11.

[16] Iványi, A., Kátai, I., Testing of random matrices, *Annales Univ. Sci. Budapest., Sectio Computatorica,* submitted.

[17] Knuth, D. E., *The Art of Computer Programming. Vol. 1. Fundamental Algorithms* (third edition). Addison–Wesley, 1997. MR0378456 (51 #14624).

[18] Kuhl, J. S., Denley, T. On a generalization of the Evans conjecture. *Discrete Math.* Vol. 308, No. 20 (2008), 4763–4767. MR2438179 (2009i:05047). http://www.sciencedirect.com/science/journal/0012365X.

[19] Kumar, S. R., Russell, A., Sundaram, R. Approximating Latin square extensions. *Algorithmica* Vol. 24, No. 2 (1999), 128–138. MR1678015 (99m:68088). http://www.springerlink.com/content/1432-0541/.

[20] Lorch, J. Mutually orthogonal families of linear Sudoku solutions. *J. Aust. Math. Soc.* Vol. 87, No. 3 (2009), 409–420. MR2576574. http://journals.cambridge.org/action/displayJournal?jid=JAZ.

[21] Öhman, L.-D. A note on completing Latin squares. *Australas. J. Combin.* Vol. 45 (2009), 117–123. MR2554529. http://ajc.maths.uq.edu.au/.

[22] Provan, J. S., Sudoku: strategy versus structure. *Amer. Math. Monthly* Vol. 116, No. 8 (2009), 702–707. http://www.jstor.org/journals/00029890.html

[23] Sander, T. Sudoku graphs are integral. *Electron. J. Combin.* Vol. 16, No. 1 (2009), Note 25, 7 pp. MR2529816. http://www.combinatorics.org/.

[24] Thom, D., SUDOKU ist NP-vollständig. PhD Dissertation. Stuttgart, 2007.

[25] Vaughan, E. R., The complexity of constructing gerechte designs. *Electron. J. Combin.*, Vol. 16 (2009), no. 1, paper R15, pp. 8. MR2475538 (2009k:05041). http://www.combinatorics.org/.

[26] Wikipedia, Coupon collector's problem.
http://en.wikipedia.org/wiki/Coupon_collector%27s_problem.

[27] Xu, C., Xu, W., The model and algorithm to estimate the difficulty levels of Sudoku puzzles. *J. Math. Res.,* Vol. 11, No. 2 (2009), 43–46.
http://www.ccsenet.org/journal/index.php/jmr/article/viewFile/3732/3336

Antal Iványi, Imre Kátai
1117 Budapest, Pázmány Péter sétány 1/C
e-mail:
tony@compalg.inf.elte.hu
katai@compalg.inf.elte.hu