Journal of Modern Mathematics and Statistics 2 (2): 70-72, 2008
ISSN: 1994-5388

On a Randomized Comparison of Randomized String Matching and Naive String Matching

Soubhik Chakraborty
Department of Applied Mathematics, BIT Mesra, Ranchi-835215, India

Abstract: The present study makes a randomized comparison of randomized string matching and naive string matching and shows why it may not be wise to opt for randomization. This study supplies a theoretical justification for the experimental results.

Key words: Randomized string matching, naive string matching, discrete uniform distribution, geometric distribution, DNA strand, algorithm

INTRODUCTION

In biological studies, it is a common practice to compare the DNA of two (or more) different organisms. A strand of DNA consists of a string of DNA molecules called bases. These possible bases can be Adenine, Guanine, Cytosine and Thymine. One can conveniently denote each of these bases by their initial letters, namely, A, G, C and T, hence every strand of DNA will be a string like GACCTTCAGA which is fixed for a particular organism but will be different for another organism for which it may be TACCGGAAATG. If two DNA strands are found similar in some sense, for example one string turning out to be a substring of the other, the corresponding organisms would be said to be closely related in a genetic sense.

There are several algorithms for determining whether one string is a substring of the other. Cormen *et al.* (2001) has provided a helpful review. The present note makes a randomized comparison of randomized string matching and naive string matching and shows why it may not be wise to opt for randomization. The code used in this study is just for the biological application indicated above but the reader can easily generalize it for other situations. Further literature on randomized algorithms can be found in Bach (1991), Blum and Micali (1984), Borodin and Hopcroft (1985) and Motwani and Raghavan (1995) and the references cited therein.

MATERIALS AND METHODS

Below is given a user friendly QBASIC code to do the study indicated in the study.

```
REM randomized comparison of randomized string matching and naive string matching
  CLS
  RANDOMIZE TIMER
  m = INT(RND * 10000) + 1
  n  = INT(RND * 10000) + 1
5 IF m < n THEN SWAP m, n
  FOR i = 1 TO m
  x = RND
  IF x < .25 THEN f$ = f$ + "A"
  IF x >= .25 AND x < .5 THEN f$ = f$ + "C"
  IF x >= .5 AND x < .75 THEN f$ = f$ + "G"
  IF x >= .75 AND x <= 1 THEN f$ = f$ + "T"
  NEXT i
  k = INT(RND * (m - n + 1)) + 1
  s$ = MID$(f$, k, n)
  REM randomized string matching begins
  FOR i = 1 TO 10000
   z = INT(RND * (m - n + 1)) + 1
   t$ = MID$(f$, z, n)
   IF s$ = t$ THEN GOTO 10
   NEXT i
   PRINT "no match in 10000 random trials": GOTO 20
10 PRINT "match found at"; i; "-th trial for the randomized case"
20 REM naive string mathing begins
   FOR i = 1 TO m - n + 1
   t$ = MID$(f$, i, n)
   IF s$ = t$ THEN GOTO 30
   NEXT i
   PRINT "no match found": GOTO 40
30 PRINT "match found at"; i; "-th trial for the naive case"
   PRINT "length of the larger string"; m
   PRINT "length of the smaller string"; n
40 END
```

The RND function gives an approximate continuous uniform variate in the range [0, 1]. We could also write x<1 in place of x<=1 as 1 will not be generated by the random number generator (which generates an approximate U [0, 1] variate by dividing random remainders {0, 1, 2...m-1} by m where m is some large integer). The library function MID$(g$, p, r) extracts a substring of length r starting from position p in the string g$.

RESULTS AND DISCUSSION

The experimental results upon running the QBASIC code (given in the previous study) in our system are given in Table 1.

Interpretation: It is clear from Table 1 that randomized string matching does not necessarily perform better than naïve string matching even when the smaller string is present in the larger string. Only on 6 occasions out of 30 trials it performed better. These are marked with double asterisk (**). On one occasion it performed nearly as good as naive matching marked with a single (*). On three occasions no match could be found after 10000 trials (marked by (!)). Readers are encouraged to write independent codes or take more trials with the one we have given. We will instead concentrate on a theoretical justification for all this in the study.

Theoretical justification: Let X be a discrete random variable giving the number of trials to detect the match in the naïve case. Obviously X can take values 1, 2, 3... m-n+1, where m>n, with identical probability 1/(m-n+1) for each value. X has a discrete uniform distribution.

It is easy to see that E(X) = (m-n+2)/2 and that. Var(X) = $(m-n)^2/12$. Here if the match occurs at the k-th position from left, we must have X=k, k=1, 2, 3...m-n+1.

In contrast, for the randomized case, the probability of a detection in Y trials does not depend on where the match occurs. That is,

$$P(Y = r) = P(\text{r-1 failures followed by a success}) = q^{r-1}p;\ r = 1, 2, 3.....\infty$$

which is the probability mass function of geometric distribution with p = 1/(m-n+1) and q = 1-p as the probabilities of a success and a failure respectively in a single trial. Thus this probability does not depend on the specific instance of position of match; rather it depends on the maximum value this position can take, that is m-n+1, which is fixed no matter what the specific instance is. Using the methods of summing an arithmetico-geometric series, it is not difficult to show that E(Y) = 1/p = m-n+1 and that Var(Y) = q/p^2. We omit the details.

- Randomized case is worse on the average than the naïve case because E(Y)>E(X).
- To prove this we only need to show m-n+1>(m-n+2)/2
- or 2m-2n+2>m-n+2
- or m-n>0
- or m>n

Table 1: Randomized comparison of randomized string matching with naïve string matching

Larger string length	Smaller string length (present in the larger string)	Trial no. for match found (randomized case)	Trial no. for match found (naive case)
8703	1956	4181	3264
9175	3666	3542	2168
7915	2133	5223	1810
9559	8877	561	541 (*)
4584	4156	231	397 (**)
9904	3970	8058	1759
4103	2050	451	129
4272	1056	703	1795 (**)
9818	3270	5068	664
6914	136	5287	6098
8184	1639	No match found in 10000 trials!	598 (!)
8494	6263	1726	1134
7007	4858	1446	1075
7254	1416	3661	3803 (**)
6243	4826	1367	377
9450	9403	1	33 (**)
5905	4727	333	200
5229	190	No match found in 10000 trials!	3930 (!)
3311	3043	21	61 (**)
7915	5102	1930	1148
9360	246	6152	2330
7467	3481	No match found in 10000 trials!	1834 (!)
9996	5107	6311	1853
3975	1416	2908	830
2697	227	2320	1946
8903	1044	4105	2208
4637	3060	1201	817
8914	7573	400	934 (**)
4980	1249	2324	2018
6966	1073	7202	4705

which is already a given initial condition that holds without any loss in generality (see statement 5 in the code).

But in our study the comparison was also randomized which means m and n are themselves random variables whereas in the proof above m and n were fixed quantities. So now let m and n be random variables with E (m) = M and E (n) = N say. Obviously M and N are fixed. It is still logical to say that we can estimate E (X) and E (Y) by replacing m by M and n by N in the proof. Writing E (X)-estimate = M-N+1 and E (Y)-estimate = (M-N+2)/2 one may like to prove E (Y)-estimate>E (X)-estimate which similarly boils down to proving M>N. And this is true in that if m>n, we must have E (m)>E (n) as well. Q. E. D.!

REMARKS

- If n is small compared to m, the match can occur at more than one position. We are considering the first match from left for the naïve case and the first match detected through random trial (wherever it be) in the randomization case.
- For the worst case, that is when there is no match (not considered here), the naïve case will take m-n+1 trials for detecting with conviction that there is no match while for the randomized case one can only say that probably there is no substring and that too after a large number of trials (10000 in the BASIC code) Thus there is no way randomized string matching can challenge the naïve string matching in the worst case. However, algorithms better than the naive string matching for the worst case are available (Cormen *et al*., 2001). Since both naïve string matching and randomized string matching algorithms perform well in the average situations, hence this comparative study.

CONCLUSION

As a final comment, while it does make sense to opt for randomization when we are unable to make a good choice (UPSHOT: Nature is not an antagonist!), yet one must advise against taking a chance with chance (pun intended) whenever a sufficiently good choice is available, where sufficiently is to be gauged by the investigator.

REFERENCES

Bach, E., 1991. Realistic analysis of some randomized algorithms. J. Comput. Sys. Sci. [doi>10.1016/0022-0000(91)90038-7], 42: 30-53.

Blum, M. and S. Micali, 1984. How to generate cryptographically strong sequences of pseudo-random bits. SIAM J. Comput. [doi>10.1137/0213053], 13: 850-864.

Borodin, A. and J.E. Hopcroft, 1985. Routing, merging and sorting on parallel models of computation. J. Comput. Sys. Sci. [doi>10.1016/0022-0000(85)90008-X], 30: 130-145.

Cormen, T.H., C.E. Leiserson, R.L. Rivest and C. Stein, 2001. Introduction to Algorithms. IT Press, Cambridge, MA, U.S.A.

Motwani, R. and P. Raghavan, 1995. Randomized Algorithms. Cambridge University Press, New York.