Wroclaw University of Technology, Faculty of Management

SS 2008 Academic Work Neisse University

Algorithm Complexity Project

The Cycle Detection Algorithms

Author: Piotr Puczyński

Statistics:

Pages	Paragraphs	Lines	Words	Pictures
8	128	258	1227	1

Appendixs:

1) Source code of implementation of cycle detection algorithms (source: author's study)

Jelenia Góra 16.04.2007

Contents

Abstract

The cycle detection is an algorithmic problem of finding cycle of iterated values of specific type in results of function *f* that maps a specific set *S* to itself and any initial value of x_0 in set *S*.

This topic is very interesting because of way that the algorithms can be applied in IT, security and for testing purposes. This work describes the idea of cycle detection as essential mathematical formulas.

The main algorithms covered by this work are Floyd's cycle-finding algorithm and Brent's algorithms. Both are represented in lower level programming language implementation in the appendix. Very important part of the work is also analysis of the algorithms' complexity and cost.

This work gives the view over the idea of the tortoise and hare problem representation as well as tries to support theory by practical examples and to show applications in science and information technology.

Introduction

What is a cycle?

A cycle is repeating part in the sequence. It's possible to prove that it exists using the idea:

$$Let\ S\ be\ a\ finite\ set\ of\ cardinality\ n$$
$$A\ pseudo-random\ function\ f(x) \rightarrow S$$

$$Define\ sequence\ a_i: a_{i+1} = f(a_i)$$

It's easy to see that such a sequence must cycle after at most *n* iterations of the function *f* (called pseudo – random function).

The cycle is to be defined by two parameters:

$$\mu, \lambda$$

- The cycle length – μ
- The length of the tail (the part of the sequence that does not cycle) – λ

The Cycle detection theory

Naïve and costly for the algorithm way to find the cycle could be to store subsequent elements of the sequence and every iteration check for duplicates in the loop. That means the storage requirement is:

$$O(\mu + \lambda)$$

It is not a good way to construct such an algorithm. When we deeply thing it's visible that if we find two elements of the sequence that:

$$a_i = a_j$$

Then:

$$|i - j|$$

Must be a multiple of the cycle length by the definition of the cycling sequence:

$$1.\ a_{\lambda+m} = a_{\lambda+m+km\mu}$$

The difference of indices is equal to $k\mu$ that is the multiple of the cycle length. This fact is used in cycle detection algorithms.

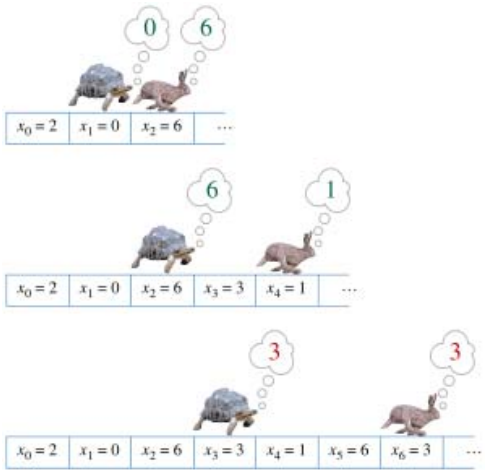
Cycle detection algorithms

Floyd's cycle-finding algorithm

Robert W Floyd (1936 – 2001) was a eminent computer scientist. He published a lot of papers as a full professor at Stanford University. Probably he is known from introducing algorithm to find the shortest patch in a graph.
The Floyd's cycle-finding algorithm was invented in 1967. The algorithm can detect cycles from sequences stored in data structures or generated on the fly.
Perhaps the most widely known variant of Floyd's cycle-finding algorithm is Pollard's rho algorithm, which is used in integer factorization process that uses pseudo-random sequences to factor integers. There is also an algorithm for calculating discrete logarithms based on Floyd's cycle-finding algorithm.
Algorithm uses storage:

$$O(1)$$

And idea of so called *tortoise and hare*:
.



Picture 1: ***Tortoise and hare idea***

The algorithm finds equation 1. By running two instances for the sequence in parallel one as twice "fast" as other. For instance one pointer moves two iterations when the other moves just one. Then, when:

$$a_m = a_{2m}$$

Then any divisor of

$$2m - m = m$$

Must be length of the cycle. If m is composite, let the algorithm continue running until it finds more values of m so the greatest common divisor of all collected values is the length of the cycle (or the candidates for μ are reducing).

The best performance of the algorithm possible is

$$\lambda \Leftrightarrow \lambda > 1 \text{ comparisons}$$

Because *tortoise* must to get to the beginning of the cycling part.
On the other hand the worst – case performance is

$$\lambda + \frac{\mu}{2} \text{ comparisons}$$

This is because *tortoise* cannot get more then to the half of the *way* around the loop and doesn't meet *hare*.
The implementation of this type of algorithm can be found in function *floyd()* (see: appendix 1.).

Brent's algorithm

Richard Peirce Brent is an Australian mathematician and computer scientist, born in 1946. He works at Australian National University.
He described similar to Floyd's algorithm solution which also uses two pointers in the sequence. However it's different because is searches for such *i-power*:

$$2^i > \lambda + \mu$$

It has two advantages compared to the Floyd's original algorithm: it finds the correct length of λ in the cycle directly, rather than needing to search for it in a subsequent stage, and its steps involve only one evaluation of *f* rather than three.
It uses:

$$O(\lambda + \mu)$$

Tests (the same like Floyd's one) and:

$O(1)$

Storage space.
It's possible to show that the number of tests for Brent's algorithm cannot be higher from Floyd's algorithm. In fact Brent's algorithm is less cost demanding.
The implementation of this type of algorithm can be found in function *brent()* (see: appendix 1.).

Comparison of algorithms

There was an experiment conducted using both above-mentioned algorithms (see: source code in appendix 1.) and very precise time measurement. The results of the experiment are showed in Table 1.

μ	Floyd (sec)	Brent (sec)
10	0.00352021	0.00366569
1000	0.00404709	0.00352482
1000000	0.175316	0.124318
75000000	10.2279	9.77249

Table 1: *Floyd's and Brent's algorithms comparison experiment results for cycle where* $\lambda = 0$ *(source: author's research)*

It's easy to see that Floyd's algorithm cost is lower on the same machine just for very short cycle length. Brent's algorithm takes advantage for longer lengths. In this experiment Brent's algorithm is faster not more that 15% of cost.
This is very good result considering similar algorithms complexity.

Applications of cycle detection algorithms

The cycle determination could be used to measure the strength of a pseudo-random numbers generator. However some generators' cycle could be found just because of its inertial state.
What is very fruitful that exist a lot of other algorithms in numbers theory science which use the cycle detection, e.g. Pollard's rho algorithm for integer factorization and his related lambda algorithm for the discrete logarithm problem.

The cycle detection algorithm can also be used to attack for security algorithm and hash-secure functions.
In software development this kind of algorithm can be useful to find infinite loops for the thread in application as well as in robotics it's possible easier to determine the machine cycles using such helpful algorithm.
It's possible also to find application in applications in computational group theory: determining the structure of an Abelian group from a set of its generators.

Conclusion

The cycle detection is pretty much important issue in mathematical field of algorithms. The reason is that it's applications give the possibilities to analyze other algorithms and create the useful feedback for the scientist such as function period.
It is also very important topic in information technology, particularly in security audit of cryptology methods.

References

Floyd's cycle-finding algorithm
http://www.spiritus-temporis.com/floyd's-cycle-finding-algorithm/

Picture 1 source:
http://en.wikipedia.org/wiki/Cycle_detection

Teske, Edlyn (1998), "A space-efficient algorithm for group structure computation", Mathematics of Computation

R. Sedgewick, T.G. Szymanski, A.C. Yao, The complexity of finding cycles in periodic functions, SIAM