

Cycle Detection: Floyd's Algorithm

Nitin Verma
mathsanew.com

August 19, 2021

Let f be a function from a finite set S to the same set. So, for any element e_0 of S , each of the elements obtained as $f(e_0), f(f(e_0)), f(f(f(e_0))), \dots$ will also belong to S . For any $i \geq 0$, let e_i denote the element obtained after applying function f over e_0 i times. That is,

$$e_i = f(f(\dots \{i \text{ times} \} f(e_0) \dots)) = f^i(e_0)$$

Clearly, all e_i are in S . Since S is a finite set, not all e_i can be distinct. Say, $e_0, e_1, e_2, \dots, e_u$ are all distinct for some u , and $e_{u+1} = e_l$ for some $l \leq u$. Due to this, $e_{u+2} = f(e_{u+1}) = f(e_l) = e_{l+1}$, similarly $e_{u+3} = e_{l+2}$, and likewise all subsequent e_i will belong to this “cycle”: $e_l, e_{l+1}, \dots, e_u$.

We will denote by n ($n \geq 1$) the number of elements in this cycle: $n = u - l + 1$, and so $u = l + n - 1$. The following diagram depicts this situation; each arrow indicates application of function f :

$$e_0 \rightarrow e_1 \rightarrow e_2 \rightarrow \dots \rightarrow e_l \rightarrow e_{l+1} \rightarrow e_{l+2} \rightarrow \dots \rightarrow e_{l+n-1}$$


Since any e_i (including $i \geq l + n$) equals one of the above $l + n$ distinct elements, it can be associated with one particular index among these $l + n$ elements' indices. We will refer to it as the “index of e_i ”. For $i < l$, the index of e_i is simply i , and for $i \geq l$, the index is:

$$l + (i - l) \bmod n \tag{1}$$

Given the initial element e_0 and function f , we need to find l and n . In this article, we discuss the *Floyd's Cycle Detection Algorithm* for this

Copyright © 2021 Nitin Verma. All rights reserved.
Article revised from its first version (April 18, 2020).

problem, named after R. W. Floyd ([1]: section 3.1, exercise 6). It is also called *Tortoise-Hare Algorithm* [2].

Many other situations can transform to this generic problem, including a well-known problem of how we can detect cycle in a linked-list (described later in section “Detecting Cycle in a Linked-List”).

The Algorithm

To be able to find n and l , this algorithm first finds out some element which belongs to the cycle. To find such element, it observes that there must exist some integer $t \geq 1$ such that e_t and e_{2t} are equal, i.e. have the same index. For e_t and e_{2t} to be equal, they both must belong to the cycle. That is:

$$t \geq l \quad (2)$$

Let us try to find out more about such t . The index of e_t and e_{2t} are same iff (due to (1)):

$$\begin{aligned} l + (t - l) \bmod n &= l + (2t - l) \bmod n \\ \Leftrightarrow t \bmod n &= 0 \\ \Leftrightarrow t &= \text{a multiple of } n \end{aligned} \quad (3)$$

Due to (2), (3) and $t \geq 1$, t is a positive multiple of n which is at least l . There are infinitely many such t , but let's try to find out the smallest of them. Now onward, we will simply use ' t ' to denote this smallest t .

So we are looking for the smallest positive multiple of n , which is at least l . For the trivial case of $l = 0$, t is n . For $l > 0$ and $n \mid l$, t is l .

Now consider the case of $n \nmid l$ (which also implies $l > 0$). So, $0 < l \bmod n < n$. Also, $l - l \bmod n$ is a multiple of n . Hence the required t is: $(l - l \bmod n) + n = l + n - l \bmod n$.

For the case of $l = 0$ also, $t = n$ can be written as $l + n - l \bmod n$. This reduces the number of cases we have to deal with while working with t . We can summarize our findings as:

$$t = \begin{cases} l, & l > 0 \text{ and } n \mid l \quad (\text{case (a)}) \\ l + n - l \bmod n, & l = 0 \text{ or } n \nmid l \quad (\text{case (b)}) \end{cases} \quad (4)$$

Notice that $t \leq l + n$ always. We now understand how t relates to l and n , but these quantities are still unknown. To find t and locate e_t , the algorithm works as follows.

It iterates over the possible values of $t = 1, 2, 3, \dots$ while checking if the required t is reached, i.e. if $e_t = e_{2t}$. For that, it maintains two references (pointers) R_1 and R_2 at e_t and e_{2t} respectively, for the current t . When both R_1 and R_2 point to the same element $e_t = e_{2t}$, it knows that the required t is reached.

Below is an implementation of this algorithm in C. The above described part of this method will be referred as “Cycle-Searching” (the other two parts “Find n ” and “Find l ” will be discussed below). Whenever a reference R is updated to $f(R)$, we will call it one “step” taken by R .

```

/* Parameter f is a function pointer.
   Output values of n and l are returned via pointers pn and pl.

   The elements e[i] are referred using type "void *". So,
   function f has input and output type as "void *". */

void floyd(void *e0, void* f(void*), int *pn, int *pl)
{
    void *R1, *R2, *R;
    int i, count, t, n, l;

    /***** Cycle-Searching *****/

    R1 = f(e0);
    R2 = f(f(e0));
    t = 1;

    /* loop-invariant: (R1 = e{t}) AND (R2 = e{2t}) */

    while(R1 != R2)
    {
        R1 = f(R1);
        R2 = f(f(R2));
        t++;
    }

    /* (R1 = R2), so (e{t} = e{2t}) holds */

```

```

/***** Find n *****/

R1 = f(R1);
count = 1;

while(R1 != R2)
{
    R1 = f(R1);
    count++;
}

/* (R1 = e{t}) holds again */

n = count;

/***** Find l *****/

R = e0;
i = 0;

while(i < t-n)
{
    R = f(R);
    i = i + 1;
}

/* (R = e{t-n}) holds */

count = 0;

/* (R1 = e{t}) AND (R = e{t-n}) holds */

while(R != R1)
{
    R = f(R);
    R1 = f(R1);
    count++;
}

/* (R = R1 = e{l}) holds */

l = (t-n) + count;

*pn = n;
*pl = l;
}

```

Finding n

When the cycle-searching loop terminates, both R_1 and R_2 are at element e_t ($= e_{2t}$). We know that e_t belongs to the cycle. Now, if R_1 is stepped n times, it must again meet R_2 . So, we can iteratively step R_1 till it meets R_2 , while counting the steps. This step count will be n .

The “Find n ” part in method *floyd()* implements this approach.

Finding l

We have R_1 at e_t . After l steps, it will be at index (due to (1)):

$$l + (t + l - l) \bmod n = l + t \bmod n = l \quad \{\text{since } n \mid t\}$$

Also, say another reference R is initialized to point to e_0 . It will be at index l after l steps. So, to find l , we can iteratively step R_1 and R till they meet, while counting the steps. This step count will be l .

The above approach can be made more efficient. Due to (4), $(t - n)$ can be expressed as:

$$t - n = \begin{cases} l - n, & l > 0 \text{ and } n \mid l \quad (\text{case (a)}) \\ l - l \bmod n, & l = 0 \text{ or } n \nmid l \quad (\text{case (b)}) \end{cases} \quad (5)$$

Clearly, $0 \leq (t - n) \leq l$. Now consider a reference R at element e_{t-n} and R_1 which is already at e_t . For case (a) and (b) above, R will require n and $l \bmod n$ steps respectively to reach e_l . Now consider equation (4): adding n and $l \bmod n$ to t for case (a) and (b) respectively will make it $l + n$, and e_{l+n} is simply e_l (due to (1)).

Thus, for case (a), both R and R_1 after taking n steps, must point to e_l . For case (b), they must point to e_l after taking $l \bmod n$ steps. So we can find l as follows.

Initialize R at e_{t-n} . This can be done by stepping R from e_0 $(t - n)$ times. R_1 is already at e_t . Now, iteratively step R and R_1 till they meet (they will meet at e_l), while counting the steps. Say, the step count comes out to be x . Adding x to $(t - n)$ will give us the total steps taken by R to reach e_l from e_0 , which must be l .

The “Find l ” part in method *floyd()* implements this approach.

The other approach mentioned in the beginning of this section steps both R and R_1 l times. This approach steps R l times, but steps R_1 only x times where x is n (case (a)) or $l \bmod n$ (case (b)).

Detecting Cycle in a Linked-List

A well-known related problem is to find whether a given linked-list contains a cycle or not. The generic cycle detection problem easily transforms to this problem, where elements e_i are nodes of the linked-list, with e_0 as the head node, and function f maps a node to its next node or to NULL to indicate end of the linked-list (case of no cycle).

Since there may not be a cycle in this problem, we will need to additionally check for f returning NULL in the cycle-searching loop of method *floyd()*. Upon seeing NULL, we will terminate the loop and declare no cycle found.

Other Step-Counts for References

In the cycle-searching part of method *floyd()*, every time we progress R_1 and R_2 by 1 and 2 steps respectively. Suppose we instead progress them by p and q steps with $p > q$, and call the corresponding references R_p and R_q . Will these two references ever meet inside the cycle?

They will meet if there exists $s \geq 1$ such that e_{ps} and e_{qs} are equal and so belong to the cycle. That is, $ps \geq l$ and $qs \geq l$, which is equivalent to (since $p > q$) $qs \geq l$. So:

$$s \geq \frac{l}{q} \tag{6}$$

Due to (1), indices of e_{ps} and e_{qs} are same iff:

$$\begin{aligned} l + (ps - l) \bmod n &= l + (qs - l) \bmod n \\ \Leftrightarrow ((p - q)s) \bmod n &= 0 \\ \Leftrightarrow (p - q)s &= \text{a multiple of } n \end{aligned} \tag{7}$$

It is in fact possible to find $s \geq 1$ which satisfies both (6) and (7). An example is $s = mn$ where m is a positive integer large enough such that $mn \geq l/q$. So we can conclude that R_p and R_q will always meet after some number of iterations, for any p and q with $p > q$.

Note that for $p = 2$ and $q = 1$, equations (6) and (7) are equivalent to (2) and (3) respectively. ■

References

- [1] D. E. Knuth. *The Art of Computer Programming*, Vol 2, Third Edition. Addison-Wesley (1997).
- [2] Wikipedia. *Cycle Detection*. https://en.wikipedia.org/wiki/Cycle_detection.

ЭТОТ КОНТЕНТ КУПЛЕН НА САЙТЕ **SKLADCHIK.ORG**

ЧТО ТАКОЕ КЛУБ «СКЛАДЧИК»?



Платформа

Это платформа, где каждый день тысячи людей собираются вместе, чтобы общими усилиями находить, приобретать и изучать курсы интересные именно им.



Сообщество

Это сообщество из 500 000 людей, открывших для себя выгодный способ быть в тренде самых актуальных и получать ценные знания по минимальной цене.



Библиотека

Это крупнейшая библиотека инфопродуктов в которой можно найти практически любой курс или тренинг продававшийся за последние 10 лет, а также уникальные авторские инфопродукты, которые не получится найти больше нигде.



**ТЫ ЕЩЕ
НЕ В КЛУБЕ?**
**ПРИСОЕДИНЯЙСЯ
И НАЧИНАЙ ЭКОНОМИТЬ!**

