NHỮNG YẾU TỐ TOÁN HỌC NÂNG CAO HIỆU QUẢ CỦA CÁC THUẬT GIẢI TRONG MỘT SỐ BÀI TOÁN THỰC TẾ

Nguyễn Thị Phi Doan[1]
Email: doannp.dttt@hou.edu.vn

Ngày tòa soạn nhận được bài báo: 02/06/2025
Ngày phản biện đánh giá: 08/09/2025
Ngày bài báo được duyệt đăng: 15/10/2025

DOI: 10.59266/houjs.2025.718

Tóm tắt: *Bài báo trình bày vai trò của các yếu tố toán học trong việc nâng cao hiệu quả thuật toán và phần mềm giải quyết một số bài toán thực tế. Thông qua các ví dụ như hàm sơ cấp, dãy số Fibonaci và bài toán đê chắn sóng, tác giả chỉ ra cách các mô hình toán học được áp dụng để phân tích và tối ưu hóa. Ngoài ra, bài báo còn đề xuất các kỹ thuật như biến đổi công thức, tổ chức dữ liệu, phân rã bài toán và điều chỉnh tham số để cải thiện hiệu quả tính toán trong thực tiễn.*

Từ khoá: *mô hình toán học, thuật toán, toán học ứng dụng*

I. Đặt vấn đề

Trong kỷ nguyên số, khả năng chuyển hóa các mô hình toán học thành chương trình máy tính hiệu quả là yếu tố then chốt trong toán học và tin học ứng dụng. Với những bài toán đòi hỏi dẫn xuất kết quả cụ thể (một số, dãy số, văn bản, kết luận lô gic,…), trên quan điểm toán học, được coi là giải quyết xong nếu dẫn xuất được công thức hay chỉ ra quy trình biến đổi để nhận được kết quả. Đã có rất nhiều thuật giải cho các mô hình toán học được đưa ra trong (Bently, J., 2016; Burden, R.L. and Faires, J.D., 2015; Cormen, T.H. và cộng sự, 2009; Knuth, D., 1968; Nguyễn Cảnh Toàn, 2005; Press, W.H. và cộng sự, 2007; Trần Đức Thọ, 2018). Tuy nhiên, trên thực tế, trong quá trình xử lý các bài toán ta phải làm việc với một máy tính hữu hạn cả về tốc độ, bộ nhớ và khả năng biểu diễn chính xác. Do đó việc xây dựng một thuật toán tối ưu là cần thiết, nó đòi hỏi những người làm toán ứng dụng cần phải đưa ra được những lời giải cho bài toán phải đảm bảo tính chính xác, độ phức tạp tối thiểu.

Ta bắt đầu với một ví dụ về khai triển Taylor của hàm số sin *x* như sau: Xét khai triển Taylor với hàm sin *x* tại $x_o = 0$ có dạng (Abramowitz, Mi. and Stegun, I. A., 1970).

$$\sum_{n=0}^{+\infty}(-1)^n \cdot \frac{x^{2n+1}}{(2n+1)!}. \qquad (1)$$

[1] Trường Đại học Mở Hà Nội

Đây là chuỗi luỹ thừa vô hạn hội tụ tới hàm sin x, hội tụ tuyệt đối với bán kính hội tụ trong khoảng $(-\infty, +\infty)$ và sai số tính toán so với giá trị của hàm không vượt quá giá trị tuyệt đối của số hạng đầu tiên bị bỏ qua (tức là phần dư trong khai triển $r \approx o(x^{2n+1})$. Trong các bài toán thực tế cần tính toán với giá trị của hàm sin x ta thấy, với x nhận giá trị khá bé (gần với 0) thì ta có thể xấp xỉ sin $x \approx x$, tuy nhiên với x nhận giá trị khá lớn thì công thức xấp xỉ trên không còn đúng nữa. Vậy ta cần phải xác định được giá trị thứ n của khải triển để sao cho khi ngắt bỏ phần dư thì độ chính xác vẫn đạt được yêu cầu của bài toán đặt ra và đảm bảo thời gian xử lý của máy tính đối với thuật toán phải tối thiểu.

Ngoài ra việc xây dựng các thuật toán khác nhau cho một bài toán cũng dẫn đến độ chính xác và độ phức tạp tính toán sẽ khác nhau. Ta xét ví dụ về thuật toán tìm dãy số Fibonacci (Brillhart, J., Montgomery, P. L. and Silverman, R. D., 1988; Cohn, J. H. E.,1964) sau: Xét bài toán (*số cánh hoa*): Tìm dãy số $f_0, f_1, f_2, \ldots$ trong đó

$$\begin{cases} f_0 = f_1 = 1, \\ f_n = f_{n-1} + f_{n-2}, \ \forall n \geq 2. \end{cases} \qquad (2)$$

Dãy số trên được gọi là dãy số Fibonacci, dãy số này xuất hiện trong những bông hoa, hầu hết các bông hoa có số cánh hoa là một trong các số: 3, 5, 8, 13, 21, 34, 55 hoặc 89. Hoa muồng Hoàng Yến có 5 cánh, hoa phi yến thường có 5 hoặc 8 cánh, hoa Dã Quỳ có 13 cánh, hoa cúc thường có 34, hoặc 55 hoặc 89 cánh. Hiện nay, bằng công nghệ biến đổi gen, người ta tạo được một loại hoa mới rất đẹp có số cánh hoa là số Fibonacci fn.

Ta có thể xây dựng thuật giải để tìm dãy số Fibonacci như sau:

Dữ liệu: Vào từ thiết bị vào chuẩn của hệ thống gồm một dòng chứa số nguyên dương n ($1 \leq n \leq 10^7$).

Kết quả: Đưa ra thiết bị ra chuẩn của hệ thống số lượng cánh hoa tính được theo modun 10^9+7.

Ví dụ:

INPUT
8

OUTPUT
34

Giải thuật:

Theo công thức lặp đã nêu với dãy số Fibonacci việc lập trình rất đơn giản, ngắn gọn. Chương trình sau đưa ra thời gian tính (loại bỏ thời gian nhập dữ liệu).

```
# Fibonacci
import time
n = int(input(" n = "))
start_time = time.time()
p = 1_000_000_007
fa, fb = 1, 1
for i in range(n-1):
    fn = (fa + fb)%p
    fa, fb = fb, fn
print("n =",n)
print(fb)
print("--- %s seconds ---" % (time.
time() - start_time))
```

Kết quả:

```
n = 10^7
Kết quả: 640540120
Thời gian: --- 15.817691326141357
seconds ---
```

Trong thuật giải trên với n = 10^7 thì ta chỉ cần khoảng 16 giây là tìm được

tất cả các số Fibonacci trong khoảng này, nhưng nếu tăng lên n = 10^{18} thì chương trình cần nhiều tháng để chạy ra kết quả. Tuy nhiên, nếu tính số Fibonacci theo công thức:

$$\begin{bmatrix}1 & 1\\ 1 & 0\end{bmatrix}^n = \begin{bmatrix}f_{n+1} & f_n\\ f_n & f_{n-1}\end{bmatrix} \qquad (3)$$

Và áp dụng giải thuật tính nhanh luỹ thừa độ phức tạp của giải thuật sẽ là O(log$n$) và thời gian thực hiện với sẽ không quá 0.003 giây.

Chương trình trên C++ theo công thức dạng ma trận với ứng dụng sơ đồ tính nhanh lũy thừa:

```
#include <bits/stdc++.h>
#define NAME “fib.”
using namespace std;
ifstream fi (NAME”inp”);
ofstream fo (NAME”out”);
uint64_tn,a,ans,fib,t[4]={0,1,1,1},
r[4]={0,1,1,1},tg[4],p;
void mp2(uint64_t x[],uint64_t y[],
uint64_t z[])
{tg[0]=(x[0]*y[0]+x[1]*y[2])%p;
 tg[1]=(x[0]*y[1]+x[1]*y[3])%p;
 tg[2]=(x[2]*y[0]+x[3]*y[2])%p;
 tg[3]=(x[2]*y[1]+x[3]*y[3])%p;
z[0]=tg[0];z[1]=tg[1];z[2]=tg[2];
z[3]=tg[3];
}
int main()
{
 fi>>n>>p;
 while(n)
 {if(n&1) mp2(r,t,r);
  mp2(t,t,t);
  n>>=1;
 }
 ans=r[2]%p;
 fo<<ans;
fo<<”\nTime:  “<<clock()/(double)
1000<<” sec”;
}
```

Kết quả thực hiện chương trình với giá

n=10^{18}
Kết quả: 153691709
Time: 0.003 sec

II. Phương pháp nghiên cứu

Trong bài báo này, xuất phát từ các bài toán thực tế như: bài toán xây thả nhím bê tông cho để chắn sóng biển, bài toán phát chương trình quảng cáo xen lẫn chương trình truyền hình, bài toán năng lượng, bài toán truy đuổi tội phạm. Tác giả sử dụng phương pháp phân tích và tổng hợp lý thuyết để nghiên cứu các mô hình toán học, nghiên cứu các yếu tố toán học quyết định và tối ưu hoá về mặt toán học. Sau đó xây dựng thuật toán và lập trình mô phỏng cho kết quả của bài toán. Và từ đó có thể đưa ra các kết luận cho việc ứng dụng trong thực tế.

III. Các yếu tố nâng cao hiệu quả của thuật toán và phần mềm

3.1. Biến đổi, rút gọn công thức trước khi tính toán

Việc biến đổi, rút gọn công thức cho phép dễ dàng áp dụng các giải thuật tìm kiếm có hiêu quả. Xét bài toán “đê chắn sóng” (Phạm Văn Giáp, Nguyễn Hữu Đầu, 2000; Silvester, R., 1974) có dạng

sau: Để bảo vệ vùng bờ biển có giá trị kinh tế cao ở một vịnh người ta xây dựng một con đê chắn sóng chắn cửa vịnh. Đoạn thẳng nối hai bên của cửa vịnh được chia thành ô cùng độ dài. Đê gồm nhiều đoạn liên tục được thả "Nhím biển" (xem trong hình 1), mỗi đoạn bao gồm một số ô liên tục. Giữa 2 đoạn phải có ít nhất một ô trống.

Hình 1. Hình ảnh nhím biển của đê chắn sóng

Theo thiết kế, đê phải có một đoạn độ dài *k* ô, 2 đoạn độ dài *k - 1* ô, 3 đoạn độ dài *k - 2* ô,…, đoạn độ dài 1 ô. Hãy xác định *k* lớn nhất có thể chọn.

Dữ liệu: Vào từ thiết bị vào chuẩn gồm một dòng chứa số nguyên .

Kết quả: Đưa ra thiết bị ra chuẩn số nguyên tìm được.

Ví dụ:

INPUT
7

OUTPUT
2

Giải thuật: *Phân tích và biến đổi mô hình toán học, Tìm kiếm nhị phân.*

Xét số lượng ô cần thiết để có đoạn đê chắn sóng dài nhất độ dài k, khi đó số lượng các đoạn chắn sóng là

$$1 + 2 + 3 + \cdots + k = \sum_{i=1}^{k} i = \frac{k(k+1)}{2}. \quad (4)$$

Số lượng ô cần thiết để trống sẽ là không ít hơn $\frac{k(k+1)}{2} - 1$. Do đó số lượng ô thuộc các đoạn đê sẽ là

$$k + 2.(k-1) + 3.(k-2) + \cdots + (k-1).2 + k.1 = \frac{k(k+1)(k+2)}{6}.$$

Như vậy tổng số lượng ô cần có sẽ là

$$f(k) = \frac{k(k+1)(k+2)}{6} + \frac{k(k+1)}{2} - 1 = \frac{k(k+1)(k+5)}{6} - 1.$$

Ta thấy *f(k)* là hàm đơn điệu tăng trong miền $k \geq 0$. Giá trị *k* có thể tìm bằng giải thuật tìm kiếm nhị phân. Lưu ý khi xác định miền tìm kiếm để tránh tràn ô khi tính *f(k)*: với $n \leq 10^{18}$, không thể lớn hơn 2.10^6. *Độ phức tạp của giải thuật: O*(ln *n*).

Chương trình

```
#include <bits/stdc++.h>
#define NAME "dike."
#define Times fo<<"\nTime: "<<clock()/(double)1000<<" sec"
using namespace std;
ifstream fi (NAME"inp");
ofstream fo (NAME"out");
int64_t n,l,r,m,sum,mx=2000000;

int main()
```

```
{
   fi>>n;
    l=0; r=min(n+1,mx);
     while(l<r-1)
      {
       m=(r+l)/2;
       sum=m*(m+1)*(m+5)/6-1;
         if(sum>n) r=m; else l=m;
       }
   fo<<l;
   Times;
}
```

3.2. Khai thác khả năng tổ chức dữ liệu của công cụ

Hiệu quả nhiều giải thuật phụ thuộc và được quyết định một cách chủ yếu vào việc khai thác các cấu trúc dữ liệu do các hệ thống lập trình cung cấp. Ví dụ, xét bài toán "quảng cáo" như sau: Chương trình TV được phát đan xen với quảng cáo. Mỗi quảng cáo có thể xuất hiện nhiều lần trong tháng. Steve không có mấy cảm tình với thông tin quảng cáo đan xen. Theo Steve, giới thiệu sản phẩm mới là cần thiết và đã có những kênh TV riêng, còn việc phát quảng cáo đan xen với nội dung chính của một chương trình chỉ là một nỗ lực với mục đích "*Bán cát cho người Ả rập và bán tủ lạnh cho người Eskimo*". Tuy vậy, ta đang sống trong một thế giới thực, bộ máy truyền thông cũng cần phải có thêm thu nhập, ta phải chấp nhận sự tồn tại của quảng cáo đan xen, phải thích nghi với thực tế và Steve đã tìm ra một cách thích nghi tuyệt vời - sử dụng quảng cáo như một công cụ rèn luyện trí nhớ. Khi xuất hiện một quảng cáo, Steve dựa vào một số đặc trưng trên màn hình, tính ra một số nguyên dương tương ứng với quảng cáo đó (tương tự như ánh xạ hàm băm) và xác định đây là quảng cáo mới, lần đầu tiên mình thấy hay là quảng cáo cũ, đã thấy trước đó.

Ở một cuốn sổ, Steve ghi số tính được tương ứng với quảng cáo thứ . Ở một cuốn sổ khác Steve ghi số 0 nếu đây là lần đầu tiên quảng cáo xuất hiện và ghi số 1 trong trường hợp ngược lại.

Sau một thời gian rèn luyện Steve có một khả năng quan sát tuyệt vời và trí nhớ tốt. Tất cả quảng cáo nêu trong tháng đều được xác định đúng lần xuất hiện đầu tiên!

Hãy xác định dãy số 0, 1 ghi trong cuốn sổ thứ 2 của Steve.

Dữ liệu: Vào từ thiết bị vào chuẩn:

+ Dòng đầu tiên chứa một số nguyên n,($1 \le n \le 10^5$),

+ Dòng thứ 2 chứa n số nguyên $a_1,a_2,\ldots,a_n$ ($1 \le a_i \le 10^9$,i=1,…,n).

Kết quả: Đưa ra thiết vị ra chuẩn trên một dòng n số ghi trong cuốn sổ thứ 2, các số cách nhau một dấu cách.

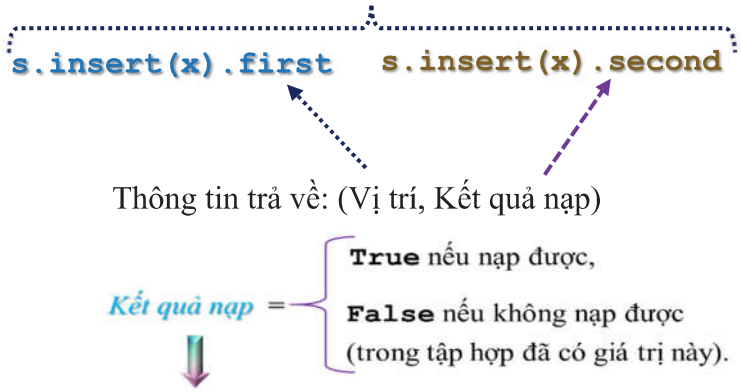
Ví dụ:

INPUT	OUTPUT
8 11313831	01011011

Giải thuật: *Cấu trúc dữ liệu tập hợp*

Sử dụng tập hợp **set<int> s**,

Nạp phần tử vào tập hợp: **s.insert(x)**,



Độ phức tạp của giải thuật:

```
// Blurb Set
#include <bits/stdc++.h>
using namespace std;
int n,x;
set<int> s;
int main()
{
   freopen("input.txt", "r", stdin);
  freopen("output.txt", "w", stdout);
      ios_base::sync_with_stdio(0);
cin.tie(0);
   cin>>n;
   for(int i=0; i<n; ++i)
   {
     cin>>x;
      if(s.insert(x).second)cout<<"0
"; else cout<<"1 ";
   }
}
```

3.3. Phân rã thành các bài toán con đơn giản và có cùng mô hình toán học

Nhiều bài toán ở dạng ban đầu đòi hỏi nhiều công sức để thiết kế giải thuật cũng như lập trình. Tuy vậy không ít bài toán cho phép phân rã mỗi bài đã cho thành một số bài toán con có cùng mô hình toán học và dễ dàng lập trình. Vấn đề còn lại chỉ còn là tổng hợp các kết quả nhận được từ những bài toán con để có kết quả cuối cùng. Kỹ thuật này đặc biệt hiệu quả khi đòi hỏi phải xử lý nhiều truy vấn với cùng một bộ dữ liệu cơ sở và các truy vấn có thể cho ở chế độ ***offline*** hoặc ***online***!

Xét bài toán "năng lượng" có dạng sau: Quá trình trang bị kiến thức phải đi kèm với việc cũng cố thể lực. Chính vì vậy chương trình của trại hè tin học có bạn tham gia, bên cạnh các xê mi na giải thuật còn có các buổi thi kiểm tra thể lực. Nội dung của các cuộc kiểm tra là xuất phát từ vị trí x_0 người thi phải dùng bóng ném đổ bình đặt ở các vị trí $x,x_1,x_2,\ldots,x_n$ nằm trên cùng một đường thẳng và đi qua điểm x_0. Bóng được đặt ở các vị trí $x_0+kd,k=0,\pm1,\pm2,\ldots$. Số lượng bóng ở mỗi vị trí là đủ nhiều nhưng người chơi chỉ có thể được phép ném bóng từ nơi đặt bóng và ném đổ bình ở vị trí tùy chọn. Do đã được tập luyện nhiều từ trước nên cú ném của các bạn trẻ là rất chính xác, hạ mục tiêu đã chọn ngay từ lần ném đầu tiên hướng tới nó. Tuy vậy, từ vị trí tọa độ x, để ném đổ bình ở vị trí x_i tùy chọn người chơi phải tốn ít nhất $(x-x_i)^2$ đơn vị năng

lượng. Ngoài ra, do thể chất khác nhau nên người thứ j cần tốn t_j đơn vị để vượt qua quảng đường độ dài d. Nhà ăn phục vụ theo nguyên tắc tự chọn với số lượng món ăn khác nhau rất phong phú và ở mỗi món ăn đều ghi rõ số đơn vị năng lượng mà nó cung cấp cho cơ thể.

Hãy xác định số năng lượng tối thiểu mà mỗi người phải có để vượt qua bài kiểm tra thể lực.

Dữ liệu: Vào từ thiết bị vào chuẩn:

+ Dòng đầu tiên chứa số nguyên – số lượng bình cần ném đổ ($1 \le n \le 3.10^5$).

+ Dòng thứ 2 chứa n số nguyên $x_1, x_2, \ldots, x_n$ ($0 \le x_i \le 10^9, i = 1,2\ldots,n$).

+ Dòng thứ 3 chứa 2 số nguyên x_0 và d ($0 \le x_0 \le 10^9, 1 \le d \le 2.10^6$).

+ Dòng thứ 4 chứa số nguyên m ($1 \le m \le 6.10^5$).

+ Dòng thứ j trong m dòng sau chứa số nguyên t_j ($0 \le t_j \le 10^8$).

Kết quả: Đưa ra thiết bị ra chuẩn tổng năng lượng tối thiểu cần thiết của mỗi người để hoàn thành bài kiểm tra thể lực, mỗi giá trị đưa ra trên một dòng dưới dạng số nguyên và theo trình tự tương ứng ở dữ liệu vào.

Ví dụ:

Input
4 100 2 101 666 9 10 5 777 1 2 15 10

Output
49597 91 159 1043 703

Giải thuật: *Tổng tiền tố-hậu tố*

Xét trường hợp $x_0 = 0$:

Giả thiết chiến lược người chơi là:

+ Thực hiện k lần chuyển động tới nơi có bóng với độ dài đường đi là $x = k.d$,

+ Trên đường đi ném đổ các bình một cách tối ưu,

+ Từ vị trí cuối của đường đi: Ném đổ các bình còn lại.

Chi phí năng lượng tối ưu để ném đổ bình ở vị trí x_i trên đường chuyển động là

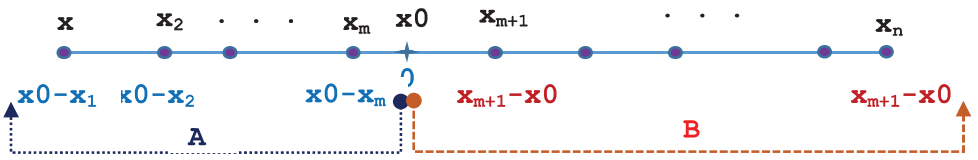
$$(\min\{x_i \% d, x_i - x_i \% d\})^2 \qquad (5)$$

Gọi tổng các chi phí cần thực hiện trên đường đi là ***left***. Ký hiệu tập các chỉ số của các phần tử $x_i > x$ là ***right*** $= \sum_{i \in R} x_i$, ***right2***$= \sum_{i \in R} x_i^2$, $f(k)$ là hàm tổng chi phí năng lượng để hoàn thành cuộc thi.

$$f(k) = k.t + left + \sum_{i \in R} (x - x_i)^2 = k.t + left + x^2.|R| - 2x.right + right2. \qquad (6)$$

Vấn đề còn lại chỉ là tìm $\min_{k=0 \div \max(\frac{x}{d})} f(k)$.

Trường hợp x_0 bất kỳ: Bằng phương pháp biến đổi tọa độ, từ bài toán đã cho đưa về việc giải 2 bài toán A và B với điểm xuất phát tọa độ 0 và tổng hợp để có kết quả cần tìm.



Việc tổng hợp kết quả có thể dựa trên phương pháp tìm kiếm tam phân hoặc sẽ hiệu quả hơn nhiều nếu dùng phương pháp tìm kiếm nhị phân dựa trên đạo hàm $f'(k)$, khi đó độ phức tạp của giải thuật sẽ là $O(mlogn)$.

Chú ý: Để lập trình có hiệu quả cần thiết kỹ bản đồ lưu trữ dữ liệu.

```
// WI46 Energy V2 Opt
#pragma GCC optimize("Ofast,unroll-loops")
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
using ld = long double;
using pii = pair<int, int>;
#define bigint __int128
#define all(a) (a).begin(), (a).end()

const ll inf = 2e18;
long long soft(bigint x) {
    return x >= inf ? inf : x;
}
struct Solver {
    int n;
    vector<int> a;
    vector<int> u_ind;
    int d;
    vector<ll> pref_sum_opt;
    vector<bigint> suf_2_sum;
    vector<ll> suf_sum;
    Solver() {}
    Solver(vector<int> a_, int d) : a(a_), d(d) {
        sort(all(a));
        if (a.empty() || a[0] != 0) a.insert(a.begin(), 0);
        n = a.size();
        pref_sum_opt.resize(n + 1);
        for (int i = 0; i < n; ++i)
            pref_sum_opt[i + 1]=pref_sum_opt[i]+(ll)min(a[i]% d,
                d-a[i] % d) * min(a[i] % d, d - a[i] % d);
        suf_2_sum.resize(n + 1);
        suf_sum.resize(n + 1);
        for (int i = n - 1; i >= 0; --i) {
            suf_2_sum[i] = suf_2_sum[i + 1] + (ll)a[i]*a[i];
            suf_sum[i] = suf_sum[i + 1] + a[i];
        }
        for (int i = 1; i < n; ++i)
            if ((a[i] + d - 1) / d != (a[i - 1] + d - 1) / d)
                u_ind.push_back(i - 1);
        u_ind.push_back(n - 1);
    }
    bool func_der(ll last_ind, ll steps, int t) {
        return t + (bigint)(steps * (n - last_ind - 1)
            * d - suf_sum[last_ind + 1]) * (2 * d) < 0;
    }
    long long func(ll last_ind, ll steps, int t) {
        return soft(steps * t + pref_sum_opt[last_ind + 1]
            + suf_2_sum[last_ind + 1]
```

```
                + (bigint)(n - last_ind - 1)
                  * steps * d * steps * d -
(bigint)2
                * suf_sum[last_ind + 1] *
steps * d);
        }
        long long solve_with_formula(int
pl, int t) {
            if (pl >= n) return inf;
            long long spl = (a[pl] + d - 1) / d;
             long long spr = (pl + 1 >= n ?
spl : (a[pl+1]+d-1)/d);
            if (n - pl - 1 == 0)
                return min({func(pl, spl, t),
func(pl, spr, t)});
            long long res = inf;
             long long x0 = -(t - (bigint)2 *
suf_sum[pl + 1] * d)
                / ((bigint)2 * (bigint)(n - pl
- 1) * d * d);
            x0 = min(max(spl, x0), spr);
            for (long long steps = max(spl,
x0 - 1); steps
                    <= min(x0 + 1, spr);
++steps)
                res = min(res, func(pl, steps,
t));
            return res;
        }
        long long solve_with_l(int pl, int
t) {
            if (pl >= n) {
                return inf;
            }
            long long spl = (a[pl] + d - 1) / d;
            long long spr = (pl + 1 >= n ?
spl : (a[pl+1]+d-1)/d);
            while (spl + 1 < spr) {
              long long cm = (spl + spr) / 2;
                if (func_der(pl, cm, t)) spl
= cm;
                else  spr = cm;
            }
            return min({func(pl, spl, t),
func(pl, spr, t)});
        }
        long long solve(int t) {
            int pl = 0;
            int pr = u_ind.size();
            while (pl + 1 < pr) {
                int pm = (pl + pr) / 2;
                    if (func_der(u_ind[pm],
(a[u_ind[pm]]+d-1)/d, t))
                    pl = pm;
                else pr = pm;
            }
            long long res = solve_with_
formula(u_ind[pl], t);
            if (pl + 1 != u_ind.size())
                res = min(res, func(u_ind[pl
+ 1],
                    (a[u_ind[pl + 1]] + d - 1)
/ d, t));
            return res;
        }
    };
    int main() {
        freopen("input", "r", stdin);
        freopen("output.a", "w", stdout);
            ios_base::sync_with_stdio(0);
```

```
cin.tie(0);
        int n;
        cin >> n;
        vector<int> a(n);
        for (int& i : a) cin >> i;
        int x0, d;
        cin >> x0 >> d;
        vector<int> a1, a2;
        for (int i = 0; i < n; ++i)
          if (a[i] <= x0) a1.push_back(x0
- a[i]);
            else a2.push_back(a[i] - x0);
        Solver s1(a1, d);
        Solver s2(a2, d);
        int q;
        cin >> q;
        for (int i = 0; i < q; ++i) {
          int t;
          cin >> t;
            long long r1 = s1.solve(t) +
s2.solve(2 * t);
          long long r2 = s1.solve(2 * t) +
s2.solve(t);
          cout << min(r1, r2) << '\n';
        }
    }
```

Kết quả thực hiện: Thời gian thực hiện với bộ dữ liệu kích thước tối đa đã cho: 0.45 sec.

3.4. Thay đổi tham số của mô hình toán học

Phần lớn các bài toán tin học đòi hỏi xử lý trong một trường giá trị nhất định, thường gặp hơn cả là trường số nguyên. Trong quá trình xử lý, nếu một vài tham số thay đổi giá trị, một số phép xử lý sẽ cho kết quả trung gian ngoài trường đang xét và dẫn đến kết quả sai hoặc quá trình xử lý trở nên cực kỳ phức tạp. Ngay khi làm việc trong trường số thực, việc quản lý tích lũy sai số làm tròn cũng là vấn đề cực ký khó khăn.

Một trong những giải pháp hữu hiệu là, dựa trên quan hệ hàm giữa các đại lượng - thay đổi thích hợp giá trị một số đại lượng khác để duy trì hiệu ứng đã cho. Điều này sẽ làm giảm đáng kể khối lượng lập trình cần thực hiện và không làm thay đổi độ phức tạp của giải thuật.

Ta có bài toán minh họa sau, bài toán "truy đuổi": Nhận được thông báo có xe chở hàng buôn lậu quốc cấm hiện đang ở ki lô mét thứ của quốc lộ, đồn biên phòng đóng ở ki lô mét 0 lập tức cho xe truy đuổi (Hình 2). Bọn buôn lậu cũng đã phát hiện ra là bị truy đuổi và không từ một thủ đoạn nào để tìm cách trốn thoát. Trên xe của bọn buôn lậu có k thùng phuy dầu máy. Chúng quyết định khi cần thiết, tại các đoạn đường dốc hiểm trở sẽ đổ dầu ra đường làm xe truy đuổi buộc phải giảm tốc độ, mỗi lần sẽ phải đổ hết cả một thùng phuy. Có n điểm có thể đổ dầu cản trở xe của lực lượng truy đuổi, điểm thứ i ở ki lô mét x_i và sẽ làm cho xe truy đuổi phải mất thêm a_i thời gian để vượt qua đoạn đường bị đổ dầu ($i = 1,...,n$). Tốc độ xe của bọn buôn lậu là v_1, tốc độ xe của đồn biên phòng là v_2.

Hãy xác định thời gian tối đa bọn buôn lậu có thể trì hoãn trước khi bị bắt. Thời điểm bọn buôn lậu bị bắt là khi 2 xe ở cùng một địa điểm, thậm chí nếu đó là thời điểm xe bỏ chạy đang đổ dầu ra

đường! Nếu không thể đuổi kịp bọn buôn lậu thì đưa ra thông báo "**inf**".

Dữ liệu: Vào từ file văn bản CHASE.INP:

+ Dòng đầu tiên chứa 2 số nguyên n và k ($1 \le n,k \le 10^5$).

+ Dòng thứ 2 chứa 2 số nguyên v_1 và v_2 ($1 \le v_1,v_2 \le 1000$).

+ Dòng thứ 3 chứa số nguyên s ($0 \le s \le 10^8$).

+ Dòng thứ i trong n dòng sau chứa 2 số nguyên x_i và a_i ($0 \le x_i \le 10^8, 0 \le a_i \le 1000, x_i \le x_{i+1}, i=1,\ldots,n-1$).

Kết quả: Đưa ra file văn bản CHASE.OUT một số thực với độ chính xác 10^{-6}- thời gian tính được hoặc thông báo "**inf**" nếu không thể đuổi kịp xe buôn lậu.

Ví dụ:

INPUT
6 2 1 2 3 0 1 5 2 7 3 10 4 11 5 12 6

OUTPUT
13.000000

Giải thuật:

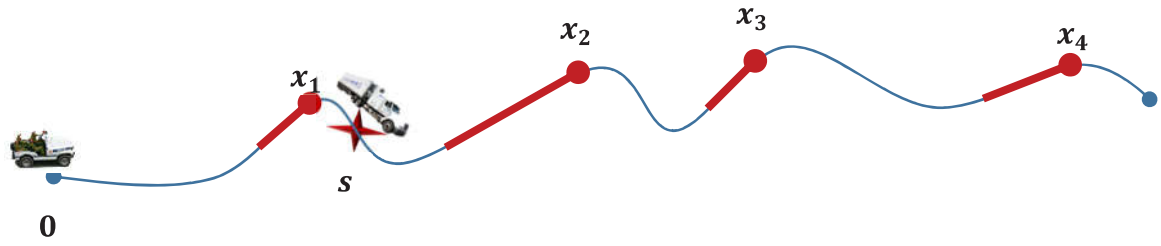
Nhận xét: Cần phân lập, xử lý các trường hợp riêng:

+ Trường hợp s=0, thời gian truy đuổi là 0,

+ Trường hợp s>0 và $v_1 \ge v_2$ - không thể đuổi kịp,

+ Trường hợp còn lại: tính thời gian truy đuổi.

Xử lý trường hợp 3 (đuổi kịp): Nếu khoảng cách giữa 2 xe là **dist** thì thời gian 2 xe gặp nhau sẽ là $\frac{dist}{v2-v1}$.



Hình 2. Mô tả lộ trình quá trình truy đuổi tội phạm

Lọc và loại bỏ các x_i<s: m=min{i | $x_i \ge$ s}. Ban đầu, khoảng cách 2 xe ***dist*** = s, nếu xe bỏ chạy đổ dầu ở điểm x_i thì thời gian truy đuổi sẽ tăng thêm một lượng là a_i, điều này tương đương như không có hiện tượng đổ dầu, nhưng khoảng cách ban đầu của 2 xe tăng thêm một khoảng là $a_i . v_2$. Vấn đề phải giải quyết là khi nào xe trốn chạy đổ dầu và đổ ở đâu để kéo dài nhất thời gian trốn chạy.

Gọi d là khoảng cách gia tăng do hiệu ứng đổ dầu, d ban đầu bằng 0. Lần lượt xét các điểm $x_i, i=m,\ldots,k$, nếu xe trốn chạy tới điểm này trước xe truy đuổi thì nạp a_i vào hàng đợi q, trong trường hợp ngược lại, lấy a_i từ đầu hàng đợi ra khỏi hàng đợi và chỉnh lý lại d.

Hàng đợi q được tổ chức theo kiểu **priority_queue<int>**, vì vậy giá trị lấy ra là lớn nhất trong các khả năng lựa chọn, thời gian truy đuổi sẽ kéo dài nhiều nhất.

Độ phức tạp của giải thuật: $O(n\log(n))$ (sử dụng hàng đợi ưu tiên).

```
// Chase
#include <bits/stdc++.h>
using namespace std;
#define fastInp ios_base::sync_with_stdio(0); cin.tie(0); cout.tie(0);
int n,k,s,v1,v2,x[100001], a[100001],m;
priority_queue<int> q;
int64_t t,d=0;
double ans;

int main()
{
   freopen("input","r",stdin);
       freopen("output","w",stdout);
fastInp;
   cin>>n>>k>>v1>>v2>>s;
                for(int i=0;i<n;++i) cin>>x[i]>>a[i];
   if(s==0){cout<<0; return 0;}
    if(v1>=v2){cout<<"inf"; return 0;}
          m=n; for(int i=0;i<n;++i) if(x[i]>=s){m=i; break;}
   while(k>0)
   {
      for(int i=m;i<n;++i)
        {
                  if((double)(x[i]-s)/v1>=(double)(x[i]+d)/v2)break;
          ++m; q.push(a[i]);
        }
      --k;
          if(!q.empty())t=q.top(); else break;
          d+=t*v2; q.pop();
      }
      ans=(double)(s+d)/(v2-v1);

cout<<fixed<<setprecision(6)<<ans;
}
```

IV. Kết luận

Từ một bài toán thực tế, để giải được trên máy tính cần phát biểu lại dưới dạng mô hình toán học và từ đó xác định chọn giải thuật và cách triển khai giải thuật. Hiệu quả của lời giải phụ thuộc không những vào giải thuật đã chọn mà phụ thuộc nhiều vào cách triển khai. Những giải thuật ứng dụng trong lập trình thuộc lĩnh vực toán rời rạc và lý thuyết về cấu trúc dữ liệu. Cách triển khai giải thuật là nghệ thuật lập trình và phụ thuộc rất nhiều vào tố chất của người lập trình. Để có thể triển khai các thuật giải của các mô hình toán trong thực tế một cách hiệu quả thì ta cần: các kiến thức sâu về toán nói chung và toán rời rạc nói riêng, nắm vững công cụ lập trình, tay nghề và kinh nghiệm lập trình.

Trong bài báo này tác giả chỉ xin liệt kê ra một số yếu tố toán học góp phần nâng cao hiệu quả của thuật giải trong các bài toán thực tế, đưa ra độ phức tạp về tính toán trong các thuật giải của các bài toán thực tế, về độ chính xác và tính tối ưu về mặt toán học của các mô hình toán này đã được trình bày trong các tài liệu tham khảo tương ứng. Bài báo còn hạn chế là chưa đưa ra được mô hình biểu đồ so sánh về thời gian chạy thực tế và bộ nhớ chiếm dụng khi thực hành trên máy tính.

Tài liệu tham khảo:

[1]. Abramowitz, Mi. and Stegun, I. A. (1970). *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. New York: Dover Publications. Ninth printing.

[2]. Bently, J. (2016). Programming pearls (Second edition). ACM Press, New York.

[3]. Brillhart, J., Montgomery, P. L. and Silverman, R. D. (1988). Tables of Fibonacci and Lucas Factorizations. *Math. Comput.* 50, 251-260.

[4]. Burden, R.L. and Faires, J.D (2015): *Numerical Analysis*, Brooks Cole, 10th edition.

[5]. Cohn, J. H. E. (1964). On square Fibonacci numbers. *The Journal of the London Mathematical Society*, 39: 537–540, doi:10.1112/jlms/s1-39.1.537.

[6]. Cormen, T.H. et al (2009). *Introduction to Algorithms*, MIT Press, 3rd edition.

[7]. Knuth, D. (1968). Fundamental Algorithms (Volume 1). First edition, xxi+634pp, ISBN 0-201-03801-3.

[8]. Nguyễn Cảnh Toàn (2005). Phương pháp Toán học ứng dụng, NXB Giáo dục.

[9]. Phạm Văn Giáp, Nguyễn Hữu Đầu (2000). *Bể cảng và đê chắn sóng*. NXB Xây dựng. Hà nội.

[10]. Press, W.H. et al (2007). *Numerical Recipes: The Art of Scientific Computing*, Cambridge University Press.

[11]. Silvester, R. (1974). Coastal engineering. Sedimentation, estuaries, tides, effluents, and modelling. New York.

[12]. Trần Đức Thọ (2018). Toán học trong kỹ thuật và công nghệ, ĐH Quốc gia TP.HCM.

MATHEMATICAL ELEMENTS ENHANCING THE EFFICIENCY OF ALGORITHMS IN SOLVING REAL-WORLD PROBLEMS

***Nguyen Thi Phi Doan*[2]**

Abstract: *This paper highlights the crucial role of mathematical elements in enhancing the efficiency of algorithms and software used to solve real-world problems. Through representative examples such as elementary functions, the Fibonacci sequence, and the breakwater problem, the author analyzes how mathematical models can be applied to support analysis, design, and solution optimization. In addition, the paper proposes several general techniques including formula transformation, data organization, problem decomposition, and parameter adjustment to improve computational performance and increase practical applicability.*

Keywords: *algorithm, applied mathematics, mathematical model*

---

[2] Hanoi Open University