Olympic Tin Học Sinh Viên OLP & ACM-ICPC

Nguyễn Quản Bá Hồng*

Ngày 3 tháng 4 năm 2025

Tóm tắt nội dung

This text is a part of the series *Some Topics in Advanced STEM & Beyond*:
URL: `https://nqbh.github.io/advanced_STEM/`.
Latest version:

- *Olympic Tin Học Sinh Viên OLP & ICPC.*
  PDF: URL: `https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/NQBH_OLP_ICPC.pdf`.
  TEX: URL: `https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/NQBH_OLP_ICPC.tex`.
- Codes:
  - C: `https://github.com/NQBH/advanced_STEM_beyond/tree/main/OLP_ICPC/C`.
  - C++: `https://github.com/NQBH/advanced_STEM_beyond/tree/main/OLP_ICPC/C++`.
  - Python: `https://github.com/NQBH/advanced_STEM_beyond/tree/main/OLP_ICPC/Python`.

Mục lục

*A Scientist & Creative Artist Wannabe. Website: `https://nqbh.github.io`. GitHub: `https://github.com/NQBH`.
E-mail: nguyenquanbahong@gmail.com, hong.nguyenquanba@umt.edu.vn. Bến Tre & Hồ Chí Minh Cities, Việt Nam.

1 Basic Competitive Programming – Lập Trình Thi Đấu Cơ Bản

Resources – Tài nguyên.

1. [Laa20]. Antti Laaksonen. *Guide to Competitive Programming: Learning & Improving Algorithms Through Contests.*
2. [Thư+21a]. Trần Đan Thư, Nguyễn Thanh Phương, Đinh Bá Tiến, Trần Minh Triết. *Nhập Môn Lập Trình.*
3. [Thư+21b]. Trần Đan Thư, Nguyễn Thanh Phương, Đinh Bá Tiến, Trần Minh Triết, Đặng Bình Phương. *Kỹ Thuật Lập Trình.*
4. [TTK21]. Trần Đan Thư, Đinh Bá Tiến, Nguyễn Tấn Trần Minh Khang. *Phương Pháp Lập Trình Hướng Đối Tượng.*

1.1 The art of handling inputs & formatting outputs – Nghệ thuật xử lý các dạng đầu vào & định dạng các dạng đầu ra

To handle various types of inputs & format various types of outputs, see, e.g.:

- Peking University Judge Online for ACM/ICPC (POJ)/FAQ. See, e.g., [Laa20; Laa24, Chap. 2, Subsect. 2.1.1, pp. 10–11].

To compile a C++ program in Linux, run in Terminal:

```
$ g++ -O2 -Wall program_name.cpp -o program_name
$ ./program_name
```

or if you want to transfer input file into it & print output into Terminal screen:

```
$ ./program_name < program_name.inp
```

or if you want to transfer input file into it & print output into a file:

```
$ ./program_name < program_name.inp > program_name.out
```

- Geeks4Geeks/`std::endl` vs. `\n` in C++: `https://www.geeksforgeeks.org/endl-vs-n-in-cpp/`.
- `i++` vs. `++i`: StackOverflow/Is there a performance difference between i++ & ++i in C?

1.2 Repeat/Loop – Lặp

1.3 String data – Kiểu dữ liệu chuỗi

1.4 Array data – Kiểu dữ liệu mảng

Về mặt toán học, kiểu dữ liệu mảng là dãy số hữu hạn $(a_i)_{i=1}^n = (a_1, a_2, \ldots, a_n)$. Về mặt Tin học, kiểu dữ liệu mảng được ký hiệu bởi `a[1..n]`.

1 ([Đức22], 141., pp. 140–141: Count digit – Đếm chữ số)**.** *Cho dãy số n số nguyên dương $A[1..n]$ & 1 chữ số k. Đếm số lần xuất hiện chữ số k trong dãy A đã cho. E.g., với dãy $A[] = (11, 12, 13, 14, 15)$, thì chữ số $k = 1$ xuất hiện 6 lần trong dãy A.*

Input. *Dòng đầu tiên của đầu vào chứa số nguyên $T \in \mathbb{N}^\star$ cho biết số bộ dữ liệu cần kiểm tra. Mỗi bộ dữ liệu gồm: (i) Dòng đầu chứa lần lượt $n, k \in \mathbb{N}$ là số phần tử trong dãy $A[]$ & chữ số k. (ii) Dòng thứ 2 chứa n số nguyên cách nhau 1 dấu cách, mô tả các phần tử của dãy A.*

Output. *Ứng với mỗi bộ dữ liệu, in ra 1 dòng chứa kết quả của bài toán tương ứng với bộ dữ liệu đầu vào đó.*

Constraint. $1 \le T \le 100, 1 \le n \le 100, 0 \le k \le 9, 1 \le A[i] \le 1000,\ \forall i = 1, \ldots, n$.

- Input: `https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/input/count_digit.inp`.
- Output: `https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/output/count_digit.out`.
- Python: `https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/Python/count_digit.py`.
- C++: ?

2 ([Đức22], 141., pp. 140–141: Count digit – Đếm chữ số)**.** *Cho dãy số nguyên $a[1], a[2], \ldots, a[n]$. Thực hiện nhiệm vụ: Chia dãy thành 2 phần trái & phải, trong đó phần trái gồm $\frac{n}{2}$ phần tử đầu tiên & phần phải gồm các phần tử còn lại. Tính tổng các phần tử của mỗi phần, cuối cùng tính & in ra tích 2 tổng tìm được.*

Input. *Dòng đầu tiên của đầu vào chứa $t \in \mathbb{N}^\star$ cho biết số bộ dữ liệu cần kiểm tra. Mỗi bộ dữ liệu gồm: (i) Dòng đầu chứa $n \in \mathbb{N}^\star$ cho biết số phần tử của dãy. (ii) Dòng 2 chứa n số nguyên cách nhau bởi dấu cách, là các phần tử của dãy.*

Output. *Ứng với mỗi bộ dữ liệu, in ra 1 dong chứa kết quả của bài toán tương ứng với bộ dữ liệu đầu vào đó.*

Constraint. $1 \le t \le 100, 1 \le n \le 100, 1 \le A[i] \le 100, \forall i = 1, \ldots, n$.

- Input: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/input/prod_left_right_sums.inp.
- Output: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/output/prod_left_right_sums.out.
- Python: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/Python/prod_left_right_sums.py.
- C++: ?

2 Olympic Tin THCS & THPT

3 ([Tru23], HSG12 Tp. Hà Nội 2020–2021, Prob. 1, p. 80: Find mid – Tìm giữa). *(a) Cho $l, r \in \mathbb{N}^\star$. Tìm $m \in [l, r) \cap \mathbb{N}^\star$ để chênh lệch giữa tổng các số nguyên liên tiếp từ l đến m & tổng các số nguyên liên tiếp từ $m+1$ đến r là nhỏ nhất. (b) Mở rộng cho $l, r \in \mathbb{Z}$. (c⋆) Thay tổng bởi tổng bình phương, tổng lập phương, tổng lũy thừa bậc $a \in \mathbb{R}$.*

Input. *2 số $l, r \in \mathbb{N}^\star$, $l < r \le 10^9$.*

Output. *Gồm 1 số nguyên duy nhất là m thỏa mãn.*

Limits. *Subtask 1: 60% các test có $l < r \le 10^3$. Subtask 2: 40% các test còn lại có $l < r \le 10^9$.*

- Input: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/input/find_mid.inp.
- Output:
- C++: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/C++/find_mid.cpp.
- Python: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/Python/find_mid.py.

3 VNOI

4 (gcd in Pascal triangle – ƯCLN trong tam giác Pascal, https://oj.vnoi.info/problem/gpt). *Tam giác Pascal là 1 cách sắp xếp hình học của các hệ số nhị thức vào 1 tam giác. Hàng thứ $n \in \mathbb{N}$ của tam giác bao gồm các hệ số trong khai triển của đa thức $f(x, y) = (x+y)^n$. I.e., phần tử tại cột thứ k, hàng thứ n của tam giác Pascal là $C_n^k = \binom{n}{k}$, i.e., tổ hợp chập k của n phần tử $0 \le k \le n$. Cho $n \in \mathbb{N}$. Tính $\mathrm{GPT}(n)$ là ƯCLN của các số nằm giữa 2 số 1 trên hàng thứ n của tam giác Pascal.*

Input. *Dòng đầu ghi T là số lượng test. T dòng tiếp theo, mỗi dòng ghi 1 số nguyên n.*

Output. *Gồm T dòng, mỗi dòng ghi $\mathrm{GPT}(n)$ tương ứng.*

Constraint. $1 \le T \le 20$, $2 \le n \le 10^9$.

Phân tích. Công thức khai triển nhị thức Newton: $(a+b)^n = \sum_{i=0}^n C_n^i a^{n-i} b^i$, $\forall n \in \mathbb{N}$, see, e.g., Wikipedia/binomial theorem. Cần tính $\gcd(\{C_n^i; 1 \le i \le n-1\}) = \gcd(C_n^1, C_n^2, \ldots, C_n^{n-1})$. Chú ý mỗi hàng của tam giác Pascal có tính chất đối xứng nên chỉ cần xét "1 nửa" là đủ. Cụ thể hơn: $C_n^k = C_n^{n-k}$, $\forall k \in \mathbb{N}$, $k \le n$, nên

$$\{C_n^1, \ldots, C_n^{n-1}\} = \{C_n^1, \ldots, C_n^{\lfloor \frac{n}{2} \rfloor}\} = \begin{cases} \{C_n^1, \ldots, C_n^{\frac{n-1}{2}}\} & \text{if } n \not\vdots\, 2, \\ \{C_n^1, \ldots, C_n^{\frac{n}{2}}\} & \text{if } n \,\vdots\, 2, \end{cases}$$

nên thay vì xét $i = 1, \ldots, n-1$, chỉ cần xét $i = 1, \ldots, \lfloor \frac{n}{2} \rfloor$ là đủ.

Theorem 1.
$$\gcd\{C_n^i\}_{i=1}^{n-1} = \begin{cases} p & \text{if } n = p^k \text{ for some prime } p \text{ \& some } n \in \mathbb{N}^\star, \\ 1 & \text{if } n \ne p^k \text{ for all prime } p \text{ \& any } n \in \mathbb{N}^\star. \end{cases}$$

See also, e.g.:

- Mathematics StackExchange/gcd of binomial coefficients.

4 Recurrence Relation – Quan Hệ Hồi Quy

Resources – Tài nguyên.

1. [AB20]. Dorin Andrica, Ovidiu Bagdasar. *Recurrent Sequences: Key Results, Applications, & Problems.*

Let X be an arbitrary set. A function $f : \mathbb{N} \to X$ defines a *sequence* $(x_n)_{n=0}^\infty$ of elements of X, where $x_n = f(n)$, $\forall n \in \mathbb{N}$. The set of all sequences with elements in X is denoted by $X^\mathbb{N}$, while X^n denotes Cartesian product of n copies of X, where X will be chosen as $\mathbb{C}$, the Euclidean space $\mathbb{R}^m$, the algebra $M_r(A)$ of the $r \times r$ matrices with entries in a ring A, etc. The set $X^\mathbb{N}$ has numerous important subsets. E.g., when $X = \mathbb{R}$, the set of real numbers $\mathbb{R}^\mathbb{N}$ includes sequences which are bounded, monotonous, convergent, positive, nonzero, periodic, etc.

– Cho X là 1 tập hợp tùy ý. Một hàm $f : \mathbb{N} \to X$ định nghĩa một *dãy* $(x_n)_{n=0}^\infty$ các phần tử của X, trong đó $x_n = f(n)$, $\forall n \in \mathbb{N}$. Tập hợp tất cả các dãy có các phần tử trong X được ký hiệu là $X^\mathbb{N}$, trong khi X^n biểu thị tích Descartes của n bản sao của X, trong đó X sẽ được chọn là $\mathbb{C}$, không gian Euclidean $\mathbb{R}^m$, đại số $M_r(A)$ của các ma trận $r \times r$ có các phần tử trong vành A, v.v. Tập hợp $X^\mathbb{N}$ có nhiều tập con quan trọng. Ví dụ, khi $X = \mathbb{R}$, tập hợp các số thực $\mathbb{R}^\mathbb{N}$ bao gồm các dãy số bị chặn, đơn điệu, hội tụ, dương, khác không, tuần hoàn, v.v.

When $a \in X$ is fixed, in *implicit form*, a recurrence relation is defined by

$$F_n(x_n, x_{n-1}, \ldots, x_0) = a,\ \forall n \in \mathbb{N}^\star, \tag{1}$$

where $F_n : X^{n+1} \to X$ is a function of $n+1$ variables, $n \in \mathbb{N}^\star$. In general, the implicit form of a recurrence relation does not define uniquely the sequence $(x_n)_{n=0}^\infty$.

The *explicit form* of a recurrence relation is

$$x_n = f_n(x_{n-1}, \ldots, x_0),\ \forall n \in \mathbb{N}^\star, \tag{2}$$

where $f_n : X^n \to X$ is a function, $\forall n \in \mathbb{N}^\star$. The relations (2) give the rule to construct the term x_n of the sequence $(x_n)_{n\ge0}$ from the 1st term x_0: $x_1 = f_1(x_0), x_2 = f_2(x_1, x_0), \ldots$, i.e., (2) is a functional type relation.

In mathematics, a *recurrence relation* is an equation according to which the nth term of a sequence of numbers is equal to some combination of the previous terms, i.e.:

$$\begin{cases} u_0 \in \mathbb{F}, \\ u_n = f_n(n, u_0, u_1, \ldots, u_{n-1}),\ \forall n \in \mathbb{N}^\star, \end{cases} \tag{3}$$

if u_0 is the initial element, which is an element of the given field $\mathbb{F}$, of the sequence $\{u_n\}_{n=0}^\infty$, where $f_n : \mathbb{N}^\star \times \mathbb{F}^n \to \mathbb{F}$ is a scalar-valued function of $(n+1)$-dimensional-vector-valued argument, $\forall n \in \mathbb{N}^\star$; &

$$\begin{cases} u_1 \in \mathbb{F}, \\ u_n = f_n(n, u_1, \ldots, u_{n-1}),\ \forall n \in \mathbb{N},\ n \ge 2, \end{cases} \tag{4}$$

if u_1 is the initial element, which is an element of the given field $\mathbb{F}$, of the sequence $\{u_n\}_{n=1}^\infty$, where $f_n : \mathbb{N}_{\ge2} \times \mathbb{F}^{n-1} \to \mathbb{F}$ is a scalar-valued function of n-dimensional-vector-valued argument, $\forall n \in \mathbb{N}$, $n \ge 2$.

Question 1. *What define uniquely* (3)*?*

Answer. The solutions $\{u_n\}_{n=0}^\infty$ defined by (3), are uniquely determined in terms of $u_0 \in \mathbb{F}, \{f_n\}_{n=1}^\infty$. Analogously, the solutions $\{u_n\}_{n=0}^\infty$ defined by (4), are uniquely determined in terms of $u_1 \in \mathbb{F}, \{f_n\}_{n=2}^\infty$. □

Remark 1 (Starting index of a sequence)**.** *The starting index of a sequence* $\{u_n\}_{n\in\{0,1\}}^\infty$ *can be* 0*, which is commonly used in Computer Science & various programming languages, or* 1*, which is commonly used in Mathematics.*

Often, only k previous terms of the sequence appear in the equation, for a parameter k that is independent of n; this number k is called the *order* of the relation. If the values of the 1st k numbers in the sequence have been given, the rest of the sequence can be calculated by repeatedly applying the equation.

In *linear recurrences*, the nth term is equated to a linear function of the k previous terms. A famous example is the recurrence for the Fibonacci numbers

$$\begin{cases} F_0 = F_1 = 1, \\ F_n = F_{n-1} + F_{n-2},\ \forall n \in \mathbb{N},\ n \ge 2, \end{cases} \tag{Fib}$$

where the order $k = 2$ & the linear function merely adds the 2 previous terms. This example is a linear recurrence with constant coefficients, because the coefficients of the linear function (1 & 1) are constants that do not depend on n. For these recurrences, one can express the general term of the sequence as a closed-form expression of n. Linear recurrences with polynomial coefficients depending on n are also important, because many common [elementary functions] & special functions have a Taylor series whose coefficients satisfy such a recurrence relation (see Wikipedia/holonomic function).

Def: Solving a recurrence relation means obtaining a closed-form solution: a non-recursive function of n.

The concept of a recurrence relation can be extended to multidimensional arrays, i.e., indexed families that are indexed by tuples of naturals.

Definition 1 (Recurrence relation)**.** *A* recurrence relation *is an equation that expresses each element of a sequence as a function of preceding ones. More precisely, in the case where only the immediately preceding element is involved, a* 1st order recurrence relation *has the form*

$$\boxed{\begin{cases} u_0 \in X, \\ u_n = \varphi(n, u_{n-1}),\ \forall n \in \mathbb{N}^\star, \end{cases}} \tag{5}$$

where $\varphi : \mathbb{N} \times X \to X$ is a function, where X is a set to which the elements of a sequence must belong. For any $u_0 \in X$, this defines a unique sequence with u_0 as its 1st element, called the initial value*, which is easy to modify the definition for getting sequences starting from the term of index 1 or higher.*

A recurrence relation of order *$k \in \mathbb{N}^\star$ has the form*

$$\boxed{\begin{cases} u_0, u_1, \ldots, u_{k-1} \in X, \\ \qquad\qquad u_n = \varphi(n, k, u_{n-1}, u_{n-2}, \ldots, u_{n-k}), \ \forall n \in \mathbb{N}, \, n \geq k, \end{cases}} \tag{6}$$

where $\varphi : \mathbb{N}^2 \times X^k \to X$ is a function that involves k consecutive elements of the sequence. In this case, k initial values are needed for defining a sequence.

Remark 2 (Explicit- vs. implicit recurrence relations)**.** *The explicit recurrence relations are the recurrence relations that can be given as (3) or (4); meanwhile the implicit recurrence relations are the recurrence relations that can be given as*

$$\begin{cases} u_0 \in \mathbb{F}, \\ f_n(n, u_0, u_1, \ldots, u_{n-1}, u_n) = 0, \ \forall n \in \mathbb{N}^\star, \end{cases} \tag{7}$$

if u_0 is the initial element, which is an element of the given field $\mathbb{F}$, of the sequence $\{u_n\}_{n=0}^\infty$, where $f_n : \mathbb{N}^\star \times \mathbb{F}^{n+1} \to \mathbb{F}$ is a scalar-valued function of $(n+1)$-dimensional-vector-valued argument, $\forall n \in \mathbb{N}^\star$; &

$$\begin{cases} u_1 \in \mathbb{F}, \\ f_n(n, u_1, \ldots, u_{n-1}, u_n) = 0, \ \forall n \in \mathbb{N}, \ n \geq 2, \end{cases} \tag{8}$$

if u_1 is the initial element, which is an element of the given field $\mathbb{F}$, of the sequence $\{u_n\}_{n=1}^\infty$, where $f_n : \mathbb{N}_{\geq 2} \times \mathbb{F}^n \to \mathbb{F}$ is a scalar-valued function of n-dimensional-vector-valued argument, $\forall n \in \mathbb{N}$, $n \geq 2$. The wellposednesses of (7) & (8) require that the corresponding recurrent equation has a unique solution to be able to define u_n uniquely.

Example 1 (Factorial)**.** *The factorial is defined by the recurrence relation $n! = n\cdot(n-1)!$, which is (5) with $X = \mathbb{N}^\star$, $u_0 = 0! = 1$, $\varphi(x, y) = xy$, $\forall x, y \in X = \mathbb{N}^\star$ so that $u_n = \varphi(n, u_{n-1}) = nu_{n-1} = n(n-1)! = n!$, $\forall n \in \mathbb{N}^\star$. This is an example of a* linear recurrence with polynomial coefficients *of order 1, with the simple polynomial (in n) n as its only coefficient.*

Example 2 (Logistic map)**.** *An example of a recurrence relation is the logistic map defined by*

$$\begin{cases} x_0 \in \mathbb{R}, \\ x_{n+1} = rx_n(1 - x_n), \end{cases} \tag{lgt}$$

for a given constant r. The behavior of the sequence depends dramatically on r, but is stable when the initial condition x_0 varies (proofs?)

4.1 Linear recurrence with constant coefficients – Hồi quy tuyến tính với hệ số hằng

See, e.g., Wikipedia/linear recurrence with constant coefficients. In mathematics (including combinatorics, linear algebra, & dynamical system), a *linear recurrence with constant coefficients* (also known as a *linear recurrence relation* or *linear difference equation*) sets equal to 0 a polynomial that is linear in the various iterates of a variable – i.e., in the values of the elements of a sequence. The polynomial's linearity means that each of its terms has degree 0 or 1. A linear recurrence denotes the evolution of some variable over time, with the current time period or discrete moment in time denoted as t, 1 period earlier denoted as $t-1$, 1 period later as $t+1$, etc.

The solution of such an equation is a function of t, & not of any iterate values, giving the value of the iterate at any time. To find the solution, it is necessary to know the specific values (known as *initial conditions*) of n of the iterates, & normally these are the n iterates that are oldest. The equation or its variable is said to be *stable* if from any set of initial conditions the variable's limit as time goes to ∞ exists; this limit is called the *steady state*.

Difference equations are used in a variety of contexts, e.g. in economics to model the evolution through time of variables e.g. gross domestic product, the inflation rate, the exchange rate, etc. They are used in modeling such time series because values of these variables are only measured at discrete intervals. In econometric applications, linear difference equations are modeled with stochastic terms in the form of autoregressive (AR) models & in models e.g. vector autoregression (VAR) & autoregressive moving average (ARMA) models that combine AR with other features.

Definition 2 (Linear recurrence with constant coefficients)**.** *A* linear recurrence with constant coefficients *is an equation of the following form, written in terms of parameters $a_1, \ldots, a_n, b$:*

$$y_n = \sum_{i=1}^k a_i y_{n-i} + b, \tag{9}$$

or equivalently as

$$y_{n+k} = \sum_{i=1}^n a_i y_{n+k-i} + b, \tag{10}$$

5 CSES Problem Set

Link: https://cses.fi/problemset/.

5.1 Introductory Problems

Problem 1 (CSES). *There are n concert tickets available, each with a certain price. Then, m customers arrive, one after another. Each customer announces the maximum price they are willing to pay for a ticket, & after this, they will get a ticket with nearest possible price such that it does not exceed the maximum price.*

Input. *1st input line contains $n, m \in \mathbb{N}$: number of tickets & number of customers. The next line contains n integers $h_1, h_2, \ldots, h_n$: the price of each ticket. The last line contains m integers $t_1, t_2, \ldots, t_m$: the maximum price for each customer in the order they arrive.*

Output. *Print, for each customer, the price that they will pay for their ticket. After this, ticket cannot be purchased again. If a customer cannot get any ticket, print -1.*

Constraints. $1 \le m, n \le 2 \cdot 10^5$, $1 \le h_i, t_i \le 10^9$.

5.2 Dynamic Programming

Resources – Tài nguyên.

1. [Ber05; Ber17]. Dimitri P. Bertsekas. *Dynamic Programming & Optimal Control. Vol. I.* 3e. 4e (can't download yet).

2. [Ber07; Ber12] Dimitri P. Bertsekas. *Dynamic Programming & Optimal Control. Vol. II.* 3e. 4e (can't download yet).

5.3 Graph Algorithms

5.4 Range Queries

5.5 Mathematics

Problem 2 (CSES/Josephus Queries, https://cses.fi/problemset/task/2164)**.** *Consider a game where there are $n \in \mathbb{N}^\star$ children, numbered $1, 2, \ldots, n$, in a circle. During the game, every 2nd child is removed from circle, until there are no children left. Task: process q queries of the form: "when there are n children, who is the kth child that will be removed?"*

- Input. *The 1st input line has an integer q: the number of queries. After this, there are q lines that describe the queries. Each line has 2 integers n, k: the number of children & the position of the child.*

- Output. *Print q integers: the answer for each query.*

It seems to me that Jack97 (nickname: `abortion_grandmaster`) proposed this problem.
Codes:

- C++: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/C%2B%2B/gcd_Pascal_triangle.cpp.

Problem 3 (CSES/Dice Probability, https://cses.fi/problemset/task/1725)**.** *Throw a dice $n \in \mathbb{N}^\star$ times, & every throw produces an outcome between 1 & 6. What is the probability that the sum of outcomes is between $a, b \in \mathbb{Z}$?*

- Input. *The only input line contains 3 integers $n, a, b \in \mathbb{N}^\star$.*

- Output. *Print probability rounded to 6 decimal places (rounding half to even).*

- Constraints. $1 \le n \le 100, 1 \le a \le b \le 6n$.

- Example. *Input:* `2 9 10`. *Output:* `0.194444`.

Phân tích. Gọi n outcomes là $a_1, \ldots, a_n \in \{1, \ldots, 6\}$. Sum of outcomes: $S \coloneqq \sum_{i=1}^n a_i \in \{n, \ldots, 6n\}$.

5.6 String Algorithms

5.7 Geometry

5.8 Advanced Techniques

5.9 Additional Problems

6 OLP

7 ICPC

1. [WW16]. Yonghui Wu, Jiande Wang. *Data Structure Practice for Collegiate Programming Contests & Education.*

2. [WW18]. Yonghui Wu, Jiande Wang. *Algorithm Design Practice for Collegiate Programming Contests & Education.*

Problem 4 ([WW16], p. 4: financial management). Larry *graduated this year & finally has a job. He's making a lot of money, but somehow never seems to have enough.* Larry *has decided that he needs to get a hold of his financial portfolio & solve his financial problems. The 1st step is to figure out what's been going on with his money.* Larry *has his bank account statements & wants to see how much money he has. Help* Larry *by writing a program to take his closing balance from each of the past 12 months & calculate his average account balance.*

Input. *The input will be 12 lines. Each line will contain the closing balance of his bank account for a particular month. Each number will be positive & displayed to the penny. No dollar sign will be included.*

Output. *The output will be a single number, the average (mean) of the closing balances for the 12 months. It will be rounded to the nearest penny, preceded immediately by a dollar sign, & followed by the end of the line. There will be no other spaces or characters in the output.*

Source. *ACM Mid-Atlantic United States 2001.*

IDs for online judges. *POJ 1004, ZOJ 1048, UVA 2362.*

Math Analysis. Let $(a_i)_{i=1}^{12} \subset [0,\infty)$ be monthly incomes of 12 months. Compute their average by the formula $\overline{a} = \frac{1}{12}\sum_{i=1}^{12} a_i$. This can be generalized to $n \in \mathbb{N}^\star$ months with a sequence of monthly incomes $(a_i)_{i=1}^{n} \subset [0,\infty)$ with its average value given by the formula $\overline{a} \coloneqq \frac{1}{n}\sum_{i=1}^{n} a_i$.

CS Analysis. The income of 12 months `a[0..11]` is input by a `for` statement & the total income `sum` $\coloneqq \sum_{i=0}^{11} a[i]$ is calculated. Then the average monthly income `avg = sum/12` is calculated. Finally, `avg` is output in accordance with the problem's output format by utilizing `printf`'s format functionalities via `printf("$%.2f", avg)`.

- Input: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/input/financial_management.inp.
- Output: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/output/financial_management.out.
- C++: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/C++/financial_management.cpp.

```
#include <iostream>
using namespace std;
int main() {
    double avg, sum = 0.0, a[12] = {0};
    for (int i = 0; i < 12; ++i) { // input income of 12 motnhs a[0..11] & summation
        cin >> a[i];
        sum += a[i];
    }
    avg = sum/12; // compute average monthly
    printf("$%.2f", avg); // output average monthly
    return 0;
}
```

Remark 3 (array of 0s). *The technique* `a[12] = {0}` *initializes an array* a *with all zero elements.*

- Python: https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/Python/financial_management.py.

```
sum = 0
for __ in range(12):
    a = float(input())
    sum += a
print("${:.2f}".format(sum / 12))
```

5 (Basic statistical data sample – mẫu dữ liệu thống kê cơ bản). *Cho 1 mẫu dữ liệu* $(a_i)_{i=1}^{n}$. *Tính trung bình, độ lệch chuẩn, phương sai của mẫu.*

Problem 5 ([WW16], pp. 5–6: doubles). *As part of an arithmetic competency program, your students will be given randomly generated lists of 2–15 unique positive integers & asked to determine how many items in each list are twice some other item in same list. You will need a program to help you with the grading. This program should be able to scan the lists & output the correct answer for each one. E.g., given the list* `1 4 3 2 9 7 18 22` *your program should answer* 3, *as* 2 *is twice* 1, 4 *is twice* 2, *&* 18 *is twice* 9.

Input. *The input file will consist of 1 or more lists of numbers. There will be 1 list of numbers per line. Each list will contain from 2–15 unique positive integers. No integer will be* > 99. *Each line will be terminated with the integer* 0, *which is not considered part of the list. A line with the single number* -1 *will mark the end of the file. Some lists may not contain any doubles.*

Output. *The output will consist of 1 line per input list, containing a count of the items that are double some other item.*

Source. *ACM Mid-Central United States 2003.*

IDs for online judges. *POJ 1552, ZOJ 1760, UVA 2787.*

Remark 4 (Multiple test cases – đa bộ test)**.** *For any problem with multiple test cases, a loop is used to deal with multiple test cases. The loop enumerates every test case.*

– Đối với bất kỳ vấn đề nào có nhiều trường hợp thử nghiệm, một vòng lặp được sử dụng để xử lý nhiều trường hợp thử nghiệm. Vòng lặp liệt kê mọi trường hợp thử nghiệm.

- Input: `https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/input/double.inp`.
- Output: `https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/output/double.out`.
- C++:
  - `https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/C++/double.cpp`.

```
#include <iostream>
using namespace std;
int main() {
    int i, j, n, count, a[20];
    cin >> a[0]; // input 1st element
    while (a[0] != -1) { // if it is not end of input, input a new test case
        n = 1;
        for ( ; ; ++n) {
            cin >> a[n];
            if (a[n] == 0) break;
        }
        count = 0; // determine how many items in each list are twice some other item
        for (i = 0; i < n - 1; ++i) { // enumerate all pairs
            for (j = i + 1; j < n; ++j) {
                if (a[i]*2 == a[j] || a[j]*2 == a[i]) // accumulation
                    ++count;
            }
        }
        cout << count << endl; // output result
        cin >> a[0]; // input 1st element of next test case
    }
    return 0;
}
```

  - `https://github.com/NQBH/advanced_STEM_beyond/blob/main/OLP_ICPC/C++/double_DPAK.cpp`: use map & vector.

Bài toán có thể mở rộng từ double thành triple, multiple, or just equal.

Problem 6 ([WW16], pp. 7–8: sum of consecutive prime numbers)**.** *Some positive integers can be represented by a sum of 1 or more consecutive prime numbers. How many such representations does a give positive integer have? E.g., the integer* 53 *has 2 representations* $5+7+11+13+17$ *&* 53*. The integer* 41 *has 3 representations:* $2+3+5+7+11+13, 11+13+17$*, &* 41*. The integer* 3 *has only 1 representation, which is* 3*. The integer* 20 *has no such representations. Note: summands must be consecutive prime numbers, so neither* $7+13$ *nor* $3+5+5+7$ *is a valid representation for the integer* 20*. Your mission is to write a program that reports the number of representations for the given positive integer.*

Input. *The input is a sequence of positive integers, each in a separate line. The integers are between* 2 *&* 10000*, inclusive. The end of the input is indicated by a zero.*

Output. *The output should be composed of lines each corresponding to an input line, except the last zero. An output line includes the number of representations for the input integer as the sum of 1 or more consecutive prime numbers. No other characters should be inserted in the output.*

Source. *ACM Japan 2005.*

IDs for online judges. *POJ 2739, UVA 3399.*

Problem 7 ([WW16], pp. 9–10: I think I need a houseboat)**.** *FRED MAPPER is considering purchasing some land in Louisiana to build his house on. In the process of investigating the land, he learned that the state of Louisiana is actually shrinking by* 50 *square miles each year, due to erosion caused by the Mississippi River. Since FRED is hoping to live in this house for the rest of his life, he needs to know if his land is going to be lost to erosion.*

After doing more research, FRED has learned that the land that is being lost forms a semicircle. This semicircle is part of a circle centered at $(0,0)$*, with the line that bisects the circle being the* x *axis. Locations below the* x *axis are in the water. The semicircle has an area of* 0 *at the beginning of year* 1*.*

Input. *The 1st line of input will be a positive integer indicating how many data sets will be included N. Each of the next N lines will contain the x, y Cartesian coordinates of the land* FRED *is considering. These will be floating-point numbers measured in miles. The y coordinate will be nonnegative.* $(0,0)$ *will not be given.*

Output. *For each data set, a single line of output should appear. This line should take the form of*

Property N: This property will begin eroding in year Z.

where N is the data set (counting from 1*) & Z is the 1st year (start from* 1*) this property will be within the semicircle AT THE END OF YEAR Z. Z must be an integer. After the last data set, this should print out "END OF OUTPUT."*

Source. *ACM Mid-Atlantic United States 2001.*

Note. *No property will appear exactly on the semicircle boundary: it will be either inside or outside. This problem will be judged automatically. Your answer must match exactly, including the capitalization, punctuation, & white space. This includes the periods at the ends of the lines. All locations are given in miles.*

IDs for online judges. *POJ 1005, ZOJ 1049, UVA 2363.*

Problem 8 ([WW16], p. 12: Hangover)**.** *How far can you make a stack of cards overhang a table? If you have 1 card, you can create a maximum overhang of half a card length. (We're assuming that the cards must be perpendicular to the table.) With 2 cards, you can make the top card overhang the bottom one by half a card length, & the bottom one overhang the table by a third of a card length, for a total maximum overhang of $\frac{1}{2} + \frac{1}{3} = \frac{5}{6}$ card lengths. In general, you can make n cards overhang by $\sum_{i=2}^{n+1} \frac{1}{i} = \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n+1}$ card lengths, where the top card overhangs the 2nd by $\frac{1}{2}$, the 2nd overhangs the 3rd by $\frac{1}{3}$, the 3rd overhangs the 4th by $\frac{1}{4}$, & so on, & the bottom card overhangs the table by $\frac{1}{n+1}$.*

Input. *The input consists of 1 or more test cases, followed by a line containing the number* 0.00 *that signals the end of the input. Each test case is a single line containing a positive floating-point number c whose value is at least* 0.01 *& at most* 5.20*; c will contain exactly 3 digits.*

Output. *For each test case, output the minimum number of cards necessary to achieve an overhang of at least c card lengths. Use the exact output format shown in the examples.*

Source. *ACM Mid-Central United States 2001. item* IDs for online judges. *POJ 1003, UVA 2294.*

Problem 9 ([WW16], p. 17: Sum)**.** *Your task is to find the sum of all integer numbers lying between* 1 *& N inclusive.*

Input. *The input consists of a single integer N that is not greater than* 10000 *by its absolute value.*

Output. *Write a single integer number that is the sum of all integer numbers lying between* 1 *& N inclusive.*

Source. *Source: ACM 2000, Northeastern European Regional Programming Contest (test tour).*

ID for online judge. *Ural 1068.*

8 Miscellaneous

8.1 Contributors

1. Võ Ngọc Trâm Anh. C++ codes.
2. Đặng Phúc An Khang. C++ codes.
   - Đặng Phúc An Khang. *Combinatorics & Number Theory in Competitive Programming – Tổ Hợp & Lý Thuyết Số trong Lập Trình Thi Đấu.*
   - Đặng Phúc An Khang. *Hướng Đến Kỳ Thi Olympic Tin học Sinh Viên Toàn Quốc & ICPC 2025.* URL: https://github.com/GrootTheDeveloper/OLP-ICPC/blob/master/2025/COMPETITIVE_REPORT.pdf.
3. Nguyễn Lê Anh Khoa. C++ codes.
4. Phan Vĩnh Tiến. C++ codes.

8.2 Donate or Buy Me Coffee

Donate (but do not donut) or buy me some coffee via NQBH's bank account information at https://github.com/NQBH/publication/blob/master/bank/NQBH_bank_account_information.

8.3 See also

1. *Vietnamese Mathematical Olympiad for High School- & College Students (VMC) – Olympic Toán Học Học Sinh & Sinh Viên Toàn Quốc.*

   PDF: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/VMC/NQBH_VMC.pdf.

   TEX: URL: https://github.com/NQBH/advanced_STEM_beyond/blob/main/VMC/NQBH_VMC.tex.

   - Codes:
     - C++ code: https://github.com/NQBH/advanced_STEM_beyond/tree/main/VMC/C++.
     - Python code: https://github.com/NQBH/advanced_STEM_beyond/tree/main/VMC/Python.
   - Resource: https://github.com/NQBH/advanced_STEM_beyond/tree/main/VMC/resource.
   - Figures: https://github.com/NQBH/advanced_STEM_beyond/tree/main/VMC/figure.

Tài liệu

[AB20] Dorin Andrica and Ovidiu Bagdasar. *Recurrent sequences—key results, applications, and problems*. Problem Books in Mathematics. Springer, Cham, [2020] ©2020, pp. xiv+402. ISBN: 978-3-030-51502-7; 978-3-030-51501-0. DOI: 10.1007/978-3-030-51502-7. URL: https://doi.org/10.1007/978-3-030-51502-7.

[Ber05] Dimitri P. Bertsekas. *Dynamic programming and optimal control. Vol. I.* Third. Athena Scientific, Belmont, MA, 2005, pp. xvi+543. ISBN: 1-886529-26-4.

[Ber07] Dimitri P. Bertsekas. *Dynamic programming and optimal control. Vol. II.* Third. Athena Scientific, Belmont, MA, 2007, p. 445.

[Ber12] Dimitri P. Bertsekas. *Dynamic programming and optimal control. Vol. II. Approximate dynamic programming.* Fourth. Athena Scientific, Belmont, MA, 2012, pp. xvii+694. ISBN: 978-1-886529-44-1; 1-886529-44-2; 1-886529-08-6.

[Ber17] Dimitri P. Bertsekas. *Dynamic programming and optimal control. Vol. I.* Fourth. Athena Scientific, Belmont, MA, 2017, pp. xix+555. ISBN: 978-1-886529-43-4; 1-886529-43-4; 1-886529-08-6.

[Đức22] Nguyễn Tiến Đức. *Tuyển Tập 200 Bài Tập Lập Trình Bằng Ngôn Ngữ Python.* Nhà Xuất Bản Đại Học Thái Nguyên, 2022, p. 327.

[Laa20] Antti Laaksonen. *Guide to Competitive Programming: Learning & Improving Algorithms Through Contests.* 2nd edition. Undergraduate Topics in Computer Science. Springer, 2020, pp. xv+309.

[Laa24] Antti Laaksonen. *Guide to Competitive Programming: Learning & Improving Algorithms Through Contests.* 3rd edition. Undergraduate Topics in Computer Science. Springer, 2024, pp. xviii+349.

[Thư+21a] Trần Đan Thư, Nguyễn Thanh Phương, Đinh Bá Tiến, and Trần Minh Triết. *Nhập Môn Lập Trình.* Nhà Xuất Bản Khoa Học & Kỹ Thuật, 2021, p. 427.

[Thư+21b] Trần Đan Thư, Nguyễn Thanh Phương, Đinh Bá Tiến, Trần Minh Triết, and Đặng Bình Phương. *Kỹ Thuật Lập Trình.* Nhà Xuất Bản Khoa Học & Kỹ Thuật, 2021, p. 526.

[Tru23] Vương Thành Trung. *Tuyển Tập Đề Thi Học Sinh Giỏi Cấp Tỉnh Trung Học Phổ Thông Tin Học.* Tài liệu lưu hành nội bộ, 2023, p. 235.

[TTK21] Trần Đan Thư, Đinh Bá Tiến, and Nguyễn Tấn Trần Minh Khang. *Phương Pháp Lập Trình Hướng Đối Tượng.* Nhà Xuất Bản Khoa Học & Kỹ Thuật, 2021, p. 401.

[WW16] Yonghui Wu and Jiande Wang. *Data Structure Practice for Collegiate Programming Contests & Education.* 1st edition. CRC Press, 2016, p. 496.

[WW18] Yonghui Wu and Jiande Wang. *Algorithm Design Practice for Collegiate Programming Contests & Education.* 1st edition. CRC Press, 2018, p. 692.