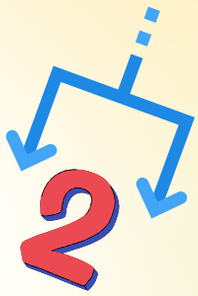


Chủ đề 2: CÁC KỸ THUẬT TÌM KIẾM CƠ BẢN

Đã bao giờ bạn đau đầu vì phải bỏ rất nhiều thời gian để tìm kiếm một thứ gì đó (bằng cách Vét cạn)? Vậy thì chủ đề tháng này sẽ giúp bạn tìm kiếm mọi thứ một cách "thông minh" hơn.

1. Kỹ thuật con trỏ (Pointers)

✏️ **Định nghĩa:** Kỹ thuật con trỏ là một phương pháp cực kỳ hiệu quả để tìm kiếm một (hay nhiều) cặp số thỏa mãn điều kiện nào đó trong dãy số.

✏️ **Ví dụ:** Cho dãy số nguyên **A** có **N** phần tử đã được **sắp xếp tăng dần** theo chiều từ trái sang phải. Tìm 2 phần tử **A[i]** và **A[j]** bất kì sao cho tổng của chúng bằng một số nguyên **X** cho trước. Nếu không tìm thấy, in ra -1.

- Dữ liệu: số N, X và dãy số A ($N \leqslant 10^6$).
- Kết quả: vị trí của hai phần tử thỏa mãn yêu cầu.

INPUT MẪU	OUTPUT MẪU
7 16 2 3 7 8 10 12 15	4 4

✏️ **Lời giải:**

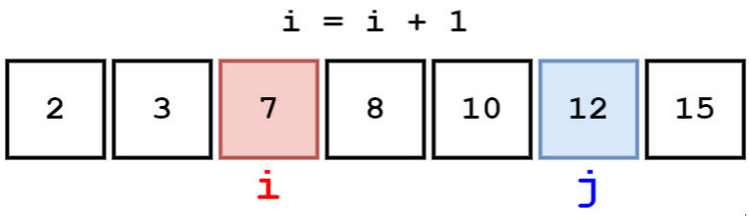
- **Dùng vét cạn**: ta phải cài đặt 2 vòng for lồng nhau để tìm kiếm 2 phần tử bất kì trong dãy sao cho tổng của chúng bằng X:

```
for (int i=1; i<=n; i++) {
    for (int j=i; j<=n; j++) {
        if (a[i] + a[j] == x) {
            cout << i << " " << j;
            return 0;
        }
    }
}
cout << -1;
```

Giải thích:
+ Vòng lặp (i) duyệt qua toàn bộ các phần tử trong mảng.
+ Vòng lặp (j) lần lượt ghép phần tử a[i] với n-i phần tử còn lại của mảng, nếu tìm thấy cặp số thỏa mãn điều kiện thì in ra vị trí của chúng và kết thúc chương trình.
+ Nếu sau khi duyệt qua hết các trường hợp vẫn không tìm thấy thì in -1.

- **Dùng 2 con trỏ:** Vì đây là mảng **đã được sắp xếp tăng dần** nên ta có thể làm như sau:

- Ban đầu, ta đặt **con trỏ L ở vị trí 1** (tức A[L] = A[1]) và **con trỏ R ở vị trí N** (A[R] = A[N]). Lúc này, ta có thể chắc chắn rằng **A[R] ⩾ A[L]**.
  - (Vòng lặp) so sánh **A[R] + A[L]** có bằng **X** không; nếu có thì ta in ra kết quả và kết thúc chương trình; nếu **A[R] + A[L] > X**, ta cho **R-1** để tổng của A[R] và A[L] **giảm dần**; nếu **A[R] + A[L] < X**, cho **L+1** để tổng của A[R] và A[L] **tăng dần**.
  - Vì mảng đã được sắp xếp tăng dần nên điều kiện lặp là L <= R, nếu không sẽ xét lại những cặp đã xét.
- **Xem GIF minh họa sau** (nếu GIF không chạy, bấm vào **ĐÂY**):



- **Cài đặt**:

```
int L = 1, R = n;
while (L <= R) {
   if (A[L] + A[R] == X) {
      cout << L << " " << R;
      return 0;
   }
   if (A[L] + A[R] > X) R--;
   else if (A[L] + A[R] < X) L++;
}
cout << -1;
```

✏️ **Nhận xét:** Khi giải quyết bài toán về xử lý mảng, việc sử dụng kĩ thuật con trỏ sẽ giúp bạn tối ưu hóa độ phức tạp. Trong bài toán nêu trên, ta đã giảm độ phức tạp từ $O(n^2)$ còn $O(n)$ so với phương pháp Vét cạn.

✏️ **Lưu ý:** Ở một số bài toán phức tạp, có thể bạn cần sử dụng nhiều con trỏ hơn.

2. Bài tập vận dụng kỹ thuật Con trỏ: ✨

Bài toán 1: EASY

Cho **2 dãy không giảm A và B** lần lượt có **N** và **M** phần tử. Hãy ghép chúng thành dãy **C** có **N+M** phần tử được sắp xếp **không giảm (***không dùng* **sort)**.

- Dữ liệu:
  - Dòng đầu tiên là 2 số N và M (N, M $\leqslant 10^6$).
  - Dòng thứ 2 là dãy số A (A[i] $\leqslant 100$).
  - Dòng thứ 3 là dãy số B (B[i] $\leqslant 100$).
- Kết quả: in ra dãy C.

INPUT	OUTPUT
5 6 1 3 6 8 10 2 6 7 12 14 15	1 2 3 6 6 7 8 10 12 14 15

Bài toán 2: **MEDIUM**

Cho dãy số nguyên dương **A** có **N** phần tử. Hãy tìm độ dài đoạn con dài nhất trong dãy sao cho tổng các phần tử trong đoạnnày không quá **S**.

- Dữ liệu:
  - Dòng đầu tiên là hai số dương N và S ($N \leqslant 10^6$; $S \leqslant 10^{18}$).
  - Dòng thứ hai là dãy số nguyên dương A ($A[i] \leqslant 10^9$).
- Kết quả: độ dài của dãy con tìm được.

INPUT	OUTPUT
7 20 2 6 5 3 6 8 9	4

Bài toán 3: **HARD** (p/s: tham khảo lời giải tại VNOI)

Cho 3 số **a, b, c** ($a \leqslant 10^4$; $b, c \leqslant 10^{14}$) và một dãy số X được tạo theo quy luật:
- X[0] = 1.
- X[i+1] = **(**a*****X[i] **+** X[i] **div** b**) mod** c.

Tìm **n** sao cho **m<n** và **x[n] == x[m]**. Dữ liệu đảm bảo $n \leqslant 2*10^7$.

- Dữ liệu: 3 số nguyên dương a, b, c.
- Kết quả: số n tìm được.

INPUT	OUTPUT
2 2 32	12

>> Nguồn tham khảo <<:
- VNOI: vnoi.info/wiki/algo/basic/two-pointers.md
- Geeks for Geeks: geeksforgeeks.org/two-pointers-technique

3. Tìm kiếm nhị phân (Binary search)

❓ Khi tra từ điển, bạn làm cách nào để lật đến trang có từ mình muốn tra nhanh và chính xác nhất? Hãy cùng tìm hiểu tư tưởng **tìm kiếm nhị phân** nào!

✏️ **Định nghĩa:** Tìm kiếm nhị phân là một thuật toán được sử dụng để tìm kiếm trong một mảng **đã được sắp xếp**. Thuật toán này hoạt động bằng cách *so sánh phần tử cần tìm với phần tử ở giữa mảng*.

- Nếu **phần tử ở giữa mảng là phần tử cần tìm**, ta dừng tìm kiếm và xác định đáp án.

- Nếu **phần tử ở giữa mảng nhỏ hơn phần tử cần tìm**, thì tìm kiếm được tiếp tục trên mảng con **bên phải**.
- Ngược lại, **nếu phần tử ở giữa mảng lớn hơn phần tử cần tìm**, thì tìm kiếm được tiếp tục trên mảng con **bên trái**.

Lặp lại quá trình này cho đến khi tìm thấy phần tử cần tìm hoặc khi cả mảng đã được duyệt hết. Trong tìm kiếm nhị phân, mỗi lần so sánh phần tử ở giữa và phần tử cần tìm, ta giảm được một nửa số phần tử so với lần trước, do đó thuật toán này có hiệu suất tìm kiếm cao hơn so với tìm kiếm tuyến tính (Vét cạn).

✏️ **Ví dụ 1:** Bạn cần tra từ "program" trong quyển từ điển (giả sử mỗi trang một chữ cái và quyển từ điển có 26 trang). Theo bạn, đâu là cách tra nhanh nhất?

- **Lời giải theo Tìm kiếm tuyến tính (Linear Search):** Một phương pháp đơn giản và đảm bảo tìm thấy kết quả (Vét cạn) đó là lật từng trang một đến khi tìm được từ "program". Nhưng nếu phạm vi tìm kiếm quá rộng thì cách này sẽ tốn rất nhiều thời gian. Do đó, một giải thuật tìm kiếm thần thánh đã ra đời.

- **Lời giải áp dụng Tìm kiếm nhị phân:** Đặt **MID = (L+R)/2**

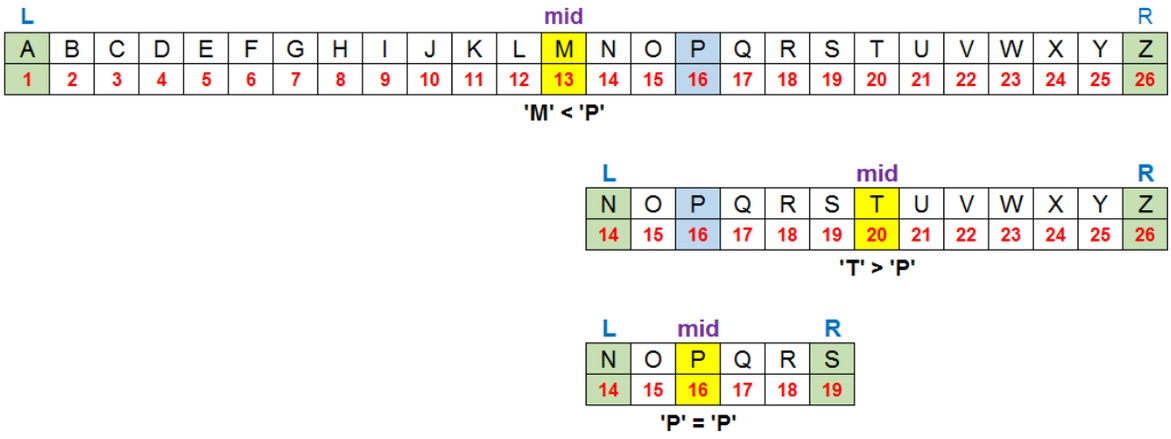
  - Bước 1: Lật ra giữa (MID) của phạm vi tìm kiếm. (Phạm vi tìm kiếm **ban đầu**: **L → R = trang đầu → trang cuối**).

  - Bước 2: Kiểm tra trang hiện tại có phải chữ "p" hay không? Nếu có thì ta **dừng tìm kiếm** và tra từ "program"; nếu không thì đến bước tiếp theo.

  - Bước 3 (TH1): Nếu chữ cái ở trang đó là một trong các chữ cái **nằm trước "p"** trong bảng alphabet, ta thu hẹp phạm vi tìm kiếm thành: **L → R = trang giữa → trang cuối**. Lặp lại 3 bước **với phạm vi tìm kiếm mới**.

  - Bước 3 (TH2): Nếu chữ cái ở trang đó là một trong các chữ cái **nằm sau "p"** trong bảng alphabet, ta thu hẹp phạm vi tìm kiếm thành: **L → R = trang đầu → trang giữa**. Lặp lại 3 bước **với phạm vi tìm kiếm mới**.

  - Xem ảnh minh họa sau:



✏️ **Ví dụ 2:** Cho dãy số nguyên **A** có **N** phần tử được **sắp xếp không giảm**. Tìm trong dãy A có phần tử nào có giá trị bằng **X** cho trước hay không?

- Dữ liệu:
  - Dòng đầu là số nguyên dương N và số nguyên X ($N \leqslant 10^9$; $X \leqslant 1000$).
  - Dòng thứ hai là dãy số nguyên A ($A[i] \leqslant 1000$).
- Kết quả: vị trí của phần tử có giá trị bằng X; nếu không tìm thấy, in ra -1.

INPUT	OUTPUT
17 37 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59	12

- **Nhận xét:** giới nhạn của N là 10^9, ta không thể dùng phương pháp tìm kiếm tuyến tính để giải. Vì với độ phức tạp chương trình là O(n) mà n > 10^8 sẽ có khả năng gây lỗi TLE (chạy quá thời gian).

- **Áp dụng Binary Search:**

  - Xét một đoạn trong mảng **A[L...R]**. Lúc này giá trị của **L** và **R** lần lượt là **1** và **số phần tử của mảng**.

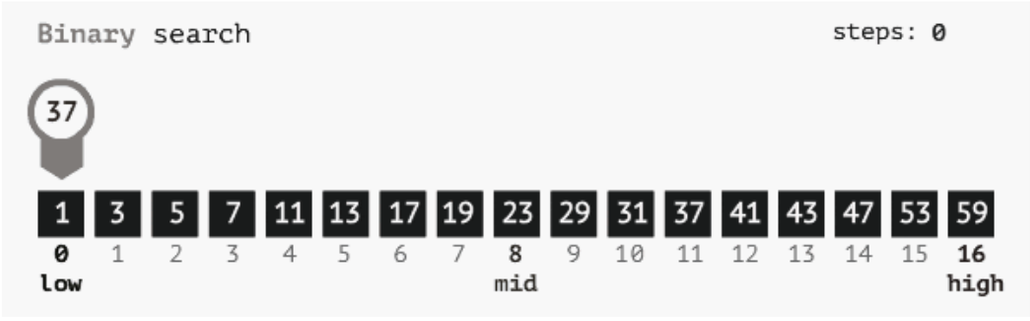
  - So sánh **X** với phần tử nằm chính giữ mảng **(MID = (left + right) /2)**. Nếu **X** bằng **A[mid]** thì trả về vị trí và kết thúc chương trình.

  - Nếu **X < A[mid]** thì chắc chắn **X** sẽ nằm ở phía bên trái, tức là **A[L ... MID-1]**. Lúc này, ta **giữ nguyên L** và đặt lại **R = MID - 1**.

  - Nếu **X > A[MID]** thì chắc chắn **X** sẽ nằm ở phía bên phải, tức là **A[MID+1 ... R]**. Lúc này, ta đặt lại **L = MID + 1** và **giữ nguyên R**.

  - Tiếp tục thực hiện chia đôi các khoảng tìm kiếm tới khi nào tìm thấy được vị trí của X trong mảng hoặc khi đã duyệt hết mảng.

- Xem GIF minh họa (Nếu GIF không chạy, bấm vào **ĐÂY**).



- **Cài đặt:**

Binary Search	Linear Search (TLE)

`int L = 1, R = n;`
`while (L <= R) {`
`  int mid = (L+R) / 2;`
`  if (a[mid] == x) {`
`  cout << mid;`
`  return 0;`
`  }`
`  if (a[mid] < x) L = mid+1;`
`  else R = mid-1;`
`}`
`cout << -1;`
	`for (int i=1; i<=n; i++) {`
	`  if (a[i] == x) {`
	`  cout << i;`
	`  return 0;`
	`  }`
	`}`
	`cout << -1;`

Độ phức tạp: **O(log n)**	Độ phức tạp: **O(n)**

4. Bài tập vận dụng Tìm kiếm nhị phân:

Bài tập 1: Lớn nhưng nhỏ: EASY

Cho một dãy số nguyên **A** gồm **N** phần tử được sắp xếp **tăng dần**. Tìm phẩn tử lớn nhất nhỏ hơn hoặc bằng ***X*** cho trước. Dữ liệu đảm bảo có kết quả.

- Dữ liệu:
  - Dòng đầu là số dương N và số nguyên X ($N \leqslant 10^9$; $X \leqslant 1000$).
  - Dòng thứ hai là dãy số nguyên A ($A[i] \leqslant 1000$).
- Kết quả: in ra vị trí của phần tử thỏa mãn yêu cầu.

INPUT	OUTPUT
6 8 1 3 6 7 9 12	4

Bài tập 2: Sushi MEDIUM *(nguồn: codeforces.com/problemset/problem/1138/A)*

Ngày nọ, Arkady mời Anna đi ăn sushi. Nhưng quy tắc bán sushi của nhà hàng này rất lạ: họ cung cấp **N** miếng sushi trên một hàng dọc, và người mua phải chọn ra một đoạn con sushi liên tiếp để mua. Nhân sushi chia làm 2 loại: lươn và cá ngừ. Người ta ký hiệu **t[i] = 1** cho nhân cá ngừ và **t[i] = 2** cho nhân lươn.

Arkady thích lươn còn Anna thích cá ngừ nên họ quyết định chọn một đoạn sushi sao cho mỗi nửa đoạn chỉ chứa một loại. Ví dụ: **{2, 2, 2, 1, 1, 1}** thì được nhưng **{1, 2, 1, 2, 1, 2}** thì không. Hãy xác định độ dài của đoạn sushi dài nhất mà họ có thể chọn mua.

- Dữ liệu: (đảm bảo tìm được kết quả)
  - Dòng đầu là số dương N ($N \leqslant 10^5$).
  - Dòng thứ hai là dãy số nguyên t ($0 < t[i] \leqslant 2$).
- Kết quả: độ dài cua đoạn sushi dài nhất mà Arkady và Anna có thể chọn.

INPUT	OUTPUT
7 2 2 2 1 1 2 2	4

Bài tập 3: Ván bay kỳ diệu: HARD

Có một hang động chứa **N** chướng ngại vật (măng đá nhô từ dưới lên và nhũ đâm từ trên xuống) nằm **xen kẻ** nhau. Đội thám hiểm có một chiếc ván bay có khả năng chứa đủ các thành viên cùng ngồi. Vấn đề là họ cần điều chỉnh nó ở một độ cao phù hợp để số lần va chạm với chướng ngại vật là **ít nhất**.

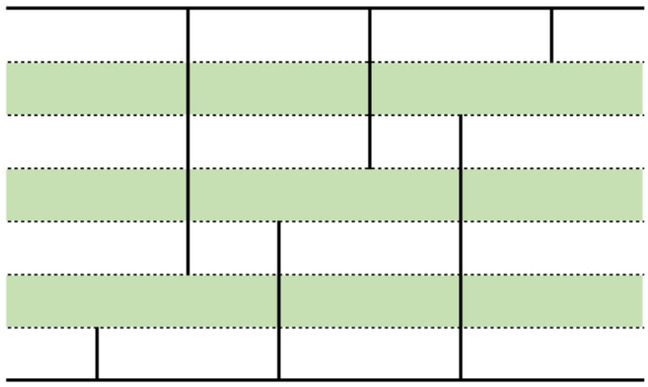
- Dữ liệu:
  - Dòng đầu là số dương N ($N \leqslant 2.10^5$) và số X ($X \leqslant 10^5$) là chiều cao của hang.

- Dòng thứ hai là dãy số nguyên H (H ⩽ X) là kích thước của các chướng ngại vật. Lưu ý: vị trí **lẻ** là của măng đá; vị trí **chẵn** là của nhũ đá.

- Kết quả: số chướng ngại vật cần phá hủy sau khi chỉnh độ cao ván phù hợp.

INPUT	OUTPUT
6 7 1 5 3 3 5 1	2

- Giải thích: khi tấm ván ở các đường màu xanh thì cần phá 2 chướng ngại vật.



>> *Nguồn tham khảo* <<:

- VNOI: *vnoi.info/wiki/algo/basic/binary-search.md*
- Freetuts: freetuts.net/thuat-toan-tim-kiem-nhi-phan-2634.html
- @Code Mely: Facebook post

5. Dặn dò:

✏️ Kỹ thuật Con trỏ và Tìm kiếm nhị phân là những giải thuật tìm kiếm rất được ưa chuộng trong giới lập trình thi đấu.

✏️ Binary Search có thể được áp dụng để giải rất nhiều bài toán trong các kỳ thi lớn, hay thậm chí là trong đời sống của bạn. Do đó, hãy tiếp thu kiến thức và làm quen với tư tưởng này thật tốt nhé ❤️

✏️ Đón chờ contest luyện tập sắp tới tại **GDOJ**.

✏️ Để theo dõi dự án, vui lòng tham gia nhóm **Guidelines for Grade 9**.

✏️ Tham gia đoạn chat cộng đồng: m.me/ch/Aba9cpzNk_Td6_fl/ để giao lưu cùng các bạn đồng môn và các anh chị học sinh chuyên Tin trên Tp.HCM.

✏️ Thông tin liên hệ:
- Page: **Căng TIN 2326**
- Group: **Guidelines for Grade 9**
- Gmail: cangtin2326@gmail.com
- Confession: forms.gle/dvByc9UQ2bRsPQHn8