# **Bài 4. Cài đặt thuật toán trên mảng**

**Mục tiêu:**

1. *Luyện tập cài đặt một số thuật toán đơn giản như sắp xếp, tìm kiếm, duyệt tổ hợp nhỏ*
2. *Luyện tập chuẩn bị dữ liệu test kiểm tra tính đúng đắn của chương trình.*

***Giới hạn:*** *chỉ được dùng mảng và string, không dùng thư viện stl (chẳng hạn vector, algorithm)*

***Yêu cầu:*** *hãy chọn một trong ba kiểu dưới đây:*

1. *phần A;* ***và*** *ít nhất 03 bài phần B*
2. *03 bài Password, Sherlock Array, và MysteryNumber.*
3. *02 bài Falling Rocks và TandemRepeats.*

## Thực hành

1. ***Test thuật toán sắp xếp***. Bạn đã có chương trình sắp xếp dãy số từ bài buổi trước. Bạn đã có đoạn chương trình sinh dữ liệu test ngẫu nhiên. Tuy nhiên việc dùng sinh ngẫu nhiên tuy tạo ra được nhiều test nhưng thực ra chỉ phủ được một số ít trường hợp cần test của thuật toán. Bạn hãy tách phần đó ra thành một chương trình riêng với nhiệm vụ in ra output chuẩn một dãy số ngẫu nhiên. Phần còn lại là một chương trình đọc từ input chuẩn một dãy số và in ra output chuẩn dãy đã được sắp xếp. Giả sử đề bài yêu cầu input không phải đúng 30 số mà là tối đa 100 số, với hình thức: giá trị đầu tiên được nhập là N - kich thước của dãy số cần sắp xếp, tiếp theo là N số, ràng buộc là 0 <= N <=100

Bây giờ bạn sẽ tạo một bộ test cho chương trình sắp xếp

* 1. Trường hợp cơ bản: một dãy số có kích thước nhỏ, thứ tự ngẫu nhiên.
  2. Trường hợp đặc biệt: dãy chỉ có 1 số.
  3. Trường hợp đặc biệt: dãy không có số nào.
  4. Trường hợp đặc biệt: dãy có 100 phần tử (dùng chương trình sinh ngẫu nhiên để tạo)
  5. Trường hợp đặc biệt: dãy đã sắp xếp tăng dần.
  6. Trường hợp đặc biệt: dãy đã sắp xếp giảm dần.

*Bạn có thể thấy là test ngẫu nhiên nhiều đến đâu cũng chỉ giải quyết được một trường hợp cơ bản. Còn theo kinh nghiệm thực tế thì chương trình lỗi chủ yếu ở các trường hợp đặc biệt (bạn có thấy là nhiều khi rất hay nhầm giữa >= và >, N và N-1, …và hay quên mất rằng dãy có thể rỗng?).*

*Bài này được giả định là input bao giờ cũng chuẩn, nên số test ít do bạn không cần phải test các trường hợp không nằm trong vùng giới hạn, chẳng hạn như N < 0 hay N > 100. Ở phần mềm ứng dụng thực, lập trình viên phải lường trước và test tất cả các tình huống có thể xảy ra.*

Như đã được hướng dẫn trong giờ thực hành tuần trước, bạn hãy tạo test trong các file text để tiện cho việc chạy chương trình. Hãy thử chạy chương trình của bạn bằng bộ test trên.

Nên làm cách nào để biết output của bạn đúng? Nhìn bằng mắt? Hay viết một đoạn code/chương trình kiểm tra output?

1. ***Test thuật toán tìm kiếm trong dãy chưa được sắp xếp***. Có thể bạn chưa viết chương trình tìm kiếm một số trong một dãy số cho trước, nhưng chưa cần quan tâm phải viết chương trình đó như thế nào, chỉ cần biết định dạng và các điều kiện ràng buộc input, bạn đã có thể chuẩn bị bộ test cho chương trình đó. Có những trường hợp nào?
   1. Cơ bản: Số cần tìm nằm ở khoảng giữa dãy. Dãy số có khoảng 5-6 phần tử
   2. Cơ bản: Số cần tìm không có mặt trong dãy. Dãy số có nhiều hơn một phần tử.
   3. Đặc biệt: Số cần tìm không có mặt trong dãy. Dãy số không có phần tử nào.
   4. Đặc biệt: Số cần tìm không có mặt trong dãy. Dãy số có 01 phần tử.
   5. Đặc biệt: Số cần tìm có mặt trong dãy. Dãy số có duy nhất một phần tử
   6. Đặc biệt: Số cần tìm là phần tử đầu dãy. Dãy số có nhiều hơn một phần tử
   7. Đặc biệt: Số cần tìm là phần tử cuối dãy. Dãy số có nhiều hơn một phần tử

Nếu biết thêm ràng buộc kích thước tối đa của dãy số là 3000, bạn sẽ cần thêm trường hợp nào?

Giả sử input có định dạng: số đầu tiên là X (số cần tìm), thứ hai là N (kích thước dãy số), tiếp theo là N số của dãy số mà chương trình cần tìm trong đó. Hãy viết bộ test bạn đã thiết kế vào các file text. Nhớ rằng bạn đã có chương trình tạo dãy số ngẫu nhiên.

Hãy viết chương trình tìm kiếm theo thuật toán duyệt từ đầu dãy cho đến khi gặp số cần tìm hoặc gặp cuối dãy.

1. ***Test thuật toán tìm kiếm trong dãy đã được sắp xếp***. Tương tự bài trên, nhưng có thêm ràng buộc là dãy số input đã được sắp xếp. Nhớ là bạn đã có một chương trình tạo dãy ngẫu nhiên, và một chương trình sắp xếp tăng dần (giả sử là đã test kĩ và sửa hết lỗi), để hỗ trợ việc tạo test.
2. ***Thử tải thuật toán sắp xếp*** *(không bắt buộc)* Bài 1 chỉ yêu cầu kích thước chuỗi số tối đa là 100, nên chương trình chạy quá nhanh với tất cả các test trong bài trên. Thực ra sắp xếp nổi bọt là một thuật toán chạy rất chậm với dữ liệu lớn và nhanh chậm tùy đặc điểm của dữ liệu. Bạn hãy thử xem dữ liệu lớn đến đâu thì bạn bắt đầu nhận ra sự khác biệt.
   1. Trường hợp thử tải: dãy cực lớn thứ tự ngẫu nhiên
   2. Trường hợp tò mò: dãy cực lớn đã sắp xếp tăng dần
   3. Trường hợp tò mò: dãy cực lớn đã sắp xếp giảm dần

Bạn có muốn đo thời gian chạy một cách chính xác hơn là nhìn đồng hồ bấm giờ? Hàm clock() trong thư viện ctime trả về thời gian tính bằng nhịp đồng hồ tại thời điểm gọi hàm. Hãy gọi clock() trước và sau đoạn lệnh bạn muốn đo thời gian chạy.

clock\_t begin = clock();   
// code to time   
clock\_t end = clock();  
double elapsedSecs = double(end - begin) / CLOCKS\_PER\_SEC;

Sau khi đo thì bạn có nhận xét gì về thời gian chạy các test trên? Từ đó có thể nhận xét gì về thuật toán?

## Bài tập

1. ***Tổng đôi*** (*bỏ qua nếu bạn thấy bài này dễ, nhưng hãy đọc cẩn thận nếu bạn chưa quen duyệt tổ hợp*). Hãy viết một chương trình nhập một chuỗi số nguyên tối đa 10000 phần tử, tìm xem trong đó có cặp số nào có tổng bằng 0 hay không. In ra cặp số đó. Input là chuỗi các số, cách nhau bởi dấu cách. Output không quan trọng thứ tự in, nếu có nhiều hơn 1 cặp số tổng bằng 0 thì in cặp nào ra cũng được.

Gợi ý: Một cách tiếp cận rất trực quan là duyệt tổ hợp tất cả các cặp hai phần tử trong dãy.  
 *Duyệt mọi khả năng lựa chọn phần tử thứ nhất trong cặp,   
 Với mỗi lựa chọn của phần tử thử nhất, duyệt mọi khả năng lựa chọn phần tử thứ hai*

*Với mỗi lựa chọn phần tử thứ hai, ta có một lựa chọn cặp hai phần tử,   
 nếu cặp số này có thỏa mãn điều kiện thì kết luận và dừng.*

*Khi đã duyệt hết mà chưa tìm được thì có nghĩa không tồn tại cặp số tổng bằng 0.*

Cách cài đặt thô nhất cho chiến lược duyệt tổ hợp hai phần tử là:

for each i from 0 to n-1:

for each j from 0 to n-1:  
 Xét a[i][j]

Cách duyệt thô trên mất tối đa n\*n lần xét (bạn có thấy hai vòng lặp lồng nhau, mỗi vòng chạy n lần?).

Bạn có thấy là nó hơi thừa thãi, xét cả a, b lẫn b, a, thậm chí xét cả a,a?

Ta có thể cải tiến để bỏ đi những lần xét trùng, sẽ giảm đi được khoảng một nửa số lần xét:  
Không mất tổng quát, ta chỉ xét các cặp số (Ai, Aj) mà i < j (xét cả a,b và b,a xem có tổng bằng 0 hay không là thừa, đúng không?)  
Như vậy, với i<j, ta có N-1 lựa chọn cho chỉ số i (0...N-2, chỉ trừ phần tử cuối cùng, An-1, vì nếu Ai là phần tử cuối dãy thì không có phần tử Aj nào đứng sau nữa để thành cặp),  
Với mỗi lựa chọn của i, ta có các lựa chọn cho j là nửa mảng nằm bên phải của i. Nghĩa là:

for each i from 0 to n-2:

for each j from i+1 to n-1:  
 Xét a[i][j]

Thử xem, bạn sẽ nhìn thấy được hiệu quả cải tiến khi chạy với dữ liệu lớn.

1. ***Tổng ba.*** Tương tự bài trên nhưng là **ba** số có tổng bằng 0.   
   Có thể bắt đầu bằng thuật toán đơn giản: duyệt tất cả các bộ ba phần tử trong dãy cho đến khi tìm được. Hãy test để chạy đúng với dãy ngắn.   
   Mở rộng (*không bắt buộc*): dùng test kích thước lớn xem thời gian chạy như thế nào. Thử cải tiến thuật toán bằng cách áp dụng một số thuật toán đã học (xem chi tiết tại bài giảng).
2. ***Liệt kê từ***. Cho một bảng chữ cái của ngôn ngữ X với thứ tự từ điển là thứ tự cho trong input. Hãy liệt kê tất cả các từ độ dài 2,3 theo thứ tự từ điển của ngôn ngữ X. Ví dụ  
   Input:   
   wq   
   Output:  
   ww www wwq wq wqw wqq qw qww qwq qq qqw qqq  
   Giới hạn: có tối đa 25 chữ cái là các chữ cái a..z hoặc chữ số 0..9.
3. ***Khiêu vũ***. Có N cặp khiêu vũ (1 <= N <= 10000), mỗi cặp gồm một bạn nam và một bạn nữ. Cần sắp xếp lại để có đội hình đẹp, nghĩa là tất cả các bạn nam đều cao hơn bạn nữ cùng cặp. Hãy kiểm tra xem có tồn tại cách xếp đội hình đẹp hay không.   
   Input: Một cách xếp gồm N dòng, mỗi dòng gồm một cặp hai số thực lần lượt là chiều cao tính bằng mét của bạn nam và bạn nữ  
   Output: ‘Yes’ nếu có cách xếp, ‘No' nếu không có cách xếp.   
   Lưu ý là đội hình cho trong input có thể đã đẹp sẵn rồi, khi đó có thể kết luận yes mà không nhất thiết phải tìm cách khác.  
   *Gợi ý: Cách đơn giản nhất là dùng sắp xếp tăng dần.   
   Hỏi thêm: Ngoài ra có cách nào nhanh hơn không?*
4. ***Đấu hậu***. Trên một bàn cờ vua kích thước NxN (N <=10000) có N con hậu. Hãy viết chương trình kiểm tra xem có hai con hậu đang đe dọa nhau (cùng hàng/cột/đường chéo) hay không. Input có dạng: dòng đầu tiên là số N. N dòng sau, dòng thứ i chứa 2 số x và y là tọa độ của con hậu thứ i. Output: ‘yes’ hoặc ‘no’ tùy theo có hay không có hai con hậu như vậy.

Tất nhiên bạn sẽ dùng mảng, vì bài thực hành này yêu cầu dùng mảng. Nhưng bạn định dùng mảng loại gì và mô hình hóa dữ liệu như thế nào cho thuật toán kiểm tra các con hậu?

1. Lựa chọn đầu tiên rất trực quan, là dùng mảng hai chiều để mô hình hóa bàn cờ NxN, mỗi ô trong mảng đại diện cho một ô tọa độ tương ứng trên bàn cờ. Có thể mỗi ô nhận giá trị true nghĩa là có hậu, false nghĩa là không có.
2. Lựa chọn thứ hai là dùng hai mảng một chiều song song. Một mảng X kích thước N để lưu tọa độ x của N con hậu, một mảng Y cùng kích thước để lưu tọa độ y của N con hậu. Tọa độ của con hậu thứ i sẽ được lưu bởi X[i], Y[i]
3. Lựa chọn thứ ba: dùng mảng 1 chiều A kích thước N, trong đó ô A[i] cho biết có một con hậu nằm tại tọa độ (i, A[i]) hoặc không có con hậu nào có tọa độ x bằng i……..

Mỗi lựa chọn dẫn đến một thuật toán xử lý rất khác. Hãy viết ba chương trình theo cả ba cách trên.

*Gợi ý: Có nhiều cách, dưới đây chỉ là một*

* *Đối với mô hình mảng hai chiều, có thể duyệt từng hàng/cột/đường chéo và đếm số hậu trên đó xem có nhiều hơn 1 con hay không.*
* *Với hai mảng song song, có thể duyệt từng cặp hai con hậu để kiểm tra xem chúng có đe dọa nhau hay không.*
* *Với một mảng một chiều, khi đọc input ta thả từng con hậu vào mảng. Khi hai con hậu được thả vào cùng một phần tử mảng nghĩa là chúng có tọa độ x giống nhau…...*

Câu hỏi mở rộng: hãy so sánh ba lựa chọn trên, cách nào chạy nhanh nhất?

1. ***Sắp hậu***. Trên một bàn cờ vua kích thước 5x5, hãy tìm cách xếp 5 con hậu sao cho không có hai con hậu nào đe dọa nhau (cùng hàng/cột/đường chéo).   
   Input: không có.   
   Output: in một cách xếp ra màn hình.  
   *Gợi ý*: *bạn có thể làm bài này không cần đến đệ quy. Cứ dùng 5 vòng lặp lồng nhau nếu muốn. Hãy tận dụng kiến thức về cấu trúc dữ liệu bạn đã thu được tại bài Đấu hậu.  
   Nếu bạn đã biết dùng đệ quy để duyệt thì hãy dùng đệ quy và mở rộng bài toán để input là kích thước bàn cờ.*
2. ***Password.*** Danny có một danh sách các từ có thể là password của tài khoản facebook của Manny. Tất cả các password đều có độ dài lẻ. Nhưng Danny biết rằng Manny rất thích sự đối xứng. Do đó password và chuỗi kí tự đảo ngược của password sẽ cùng có mặt trong danh sách.  
   Bạn cần in ra độ dài của password của Manny và kí tự đứng giữa password.  
   *Lưu ý:* Lời giải là duy nhất, chỉ có đúng một cặp từ trong danh sách là đảo ngược của nhau  
   Input:Dữ liệu đọc từ input chuẩn (bàn phím). Dòng đầu mỗi file chứa số tự nhiên N là số từ trong danh sách, N dòng tiếp theo, mỗi dòng chứa một từ.  
   Output: ghi ra output chuẩn (màn hình). Output gồm duy nhất một dòng chứa hai giá trị: độ dài của password đúng và kí tự nằm giữa password đó.  
   Ràng buộc: 1 <= N <= 100  
   Ví dụ:

|  |  |
| --- | --- |
| Input | Output |
| 4  abc  defgh  Fegi  cba | 3 b |

Chạy test tại <http://www.hackerearth.com/problem/algorithm/password-1/>

1. ***Eratosthenes sieve.*** Sàng Eratosthenes là một phương pháp hiệu quả để liệt kê các số nguyên tố trong khoảng từ 1 đến N.   
   Ý tưởng thuật toán rất đơn giản: bắt đầu với các số nguyên từ 2 đến N. Duyệt từ đầu dãy, nếu một số vẫn còn trong dãy thì ghi nhận đó chính là một số nguyên tố và xoá khỏi dãy tất cả các bội số của nó. Xem minh hoạ tại [Sieve of Eratosthenes - Wikipedia, the free encyclopedia](https://en.wikipedia.org/wiki/Sieve_of_Eratosthenes).  
   Hãy viết một chương trình với input là số N và output là chuỗi tất cả các số nguyên tố trong khoảng từ 1 đến N.
2. ***Sherlock and Array***. Watson cho Sherloc0k một mảng A gồm N số và đề nghị Sherlock kiểm tra xem trong mảng có tồn tại phần tử nào thoả mãn tính chất tổng tất cả các phần tử bên trái nó đúng bằng tổng tất cả các phần tử bên phải nó hay không. Nếu phần bên phải hoặc bên trái không có phần tử nào thì coi như tổng đó bằng 0.   
   Input: Dòng đầu chứa T là số test. Từ dòng sau, mỗi test chiếm hai dòng, dòng thứ nhất là N - kích thước mảng A, dòng thứ hai là mảng A.  
   Output: Với mỗi test, in ra ‘YES' hoặc ‘NO' trên một dòng. YES nghĩa là tồn tại phần tử mà Watson muốn tìm, NO nghĩa là không tồn tại.   
   Ràng buộc: 1<=T <=10, 1<=N<=100000, 1<=Ai<=20000.   
   Ví dụ:

|  |  |
| --- | --- |
| Input | Output |
| 2  3  1 2 3  4  1 2 3 3 | NO  YES |

Chạy test tại <https://www.hackerrank.com/challenges/sherlock-and-array>

***14. Mystery Number***. https://www.hackerrank.com/contests/101hack31/challenges/naughty-number

15. ***Falling Rocks.*** https://www.hackerrank.com/contests/101hack31/challenges/falling-rocks

16. ***Tandem Repeats***. Đề bài và test tại: https://www.hackerrank.com/contests/101hack31/challenges/tandem-repeats