BÀI 1. ƯỚC SỐ CHUNG LỚN NHẤT VÀ BỘI SỐ CHUNG NHỎ NHẤT

Input: Dòng đầu ghi số bộ test. Mỗi bộ test ghi trên một dòng 2 số nguyên a và b không quá 9 chữ số.

Output: Mỗi bộ test ghi trên 1 dòng, lần lượt là USCLN, sau đó đến BSCNN.

Phân tích bài toán:

Đầu tiên, chúng ta đọc số lượng test cases từ đầu vào (biến t).

Sau đó, chúng ta sử dụng một vòng lặp while để xử lý từng test case. Trong mỗi test case, chúng ta đọc hai số nguyên dương a và b từ đầu vào.

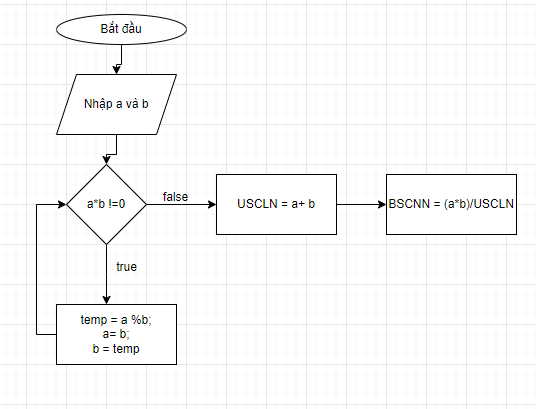
Chúng ta sử dụng hai hàm ucln và bcnn để tính ước số chung lớn nhất và bội số chung nhỏ nhất.

Hàm ucln (ước số chung lớn nhất) sử dụng thuật toán Euclid để tìm ước số chung lớn nhất của a và b. Trong vòng lặp, chúng ta lấy phần dư của a chia cho b và cập nhật giá trị của a và b cho đến khi a\*b = 0. Kết quả là giá trị của a + b.

Hàm bcnn (bội số chung nhỏ nhất) sử dụng công thức a\*b / ucln(a, b) để tính bội số chung nhỏ nhất.

Cuối cùng, chúng ta in ra ước số chung lớn nhất và bội số chung nhỏ nhất của a và b.

Quay lại bước 2 để xử lý test case tiếp theo.



- Độ phức tạp của thuật toán ở bài này: O(t \* log(min(a, b)))

BÀI 2. BẮT ĐẦU VÀ KẾT THÚC

Viết chương trình kiểm tra một số nguyên dương bất kỳ (2 chữ số trở lên, không quá 9 chữ số) có chữ số bắt đầu và kết thúc bằng nhau hay không.

Input: Dòng đầu tiên ghi số bộ test. Mỗi bộ test viết trên một dòng số nguyên dương tương ứng cần kiểm tra.

Output: Mỗi bộ test viết ra YES hoặc NO, tương ứng với Input

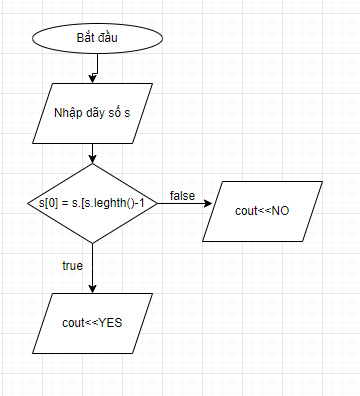
Phân tích bài toán:

· Đầu tiên, chúng ta đọc số lượng test cases từ đầu vào (biến t).

· Sau đó, chúng ta sử dụng một vòng lặp while để xử lý từng test case. Trong mỗi test case, chúng ta đọc một chuỗi số nguyên từ đầu vào (biến s).

· Chúng ta kiểm tra xem chữ số đầu tiên (s[0]) có bằng chữ số cuối cùng (s[s.length()-1]) hay không. Nếu chữ số đầu tiên và chữ số cuối cùng bằng nhau, chúng ta in ra "YES". Ngược lại, in ra "NO".

· In xuống dòng và quay lại bước 2 để xử lý test case tiếp theo.



Độ phức tạp của thuật toán ở bài này: O(1);

BÀI 3. MẢNG ĐỐI XỨNG

Nhập một dãy số nguyên có n phần tử (n không quá 100, các phần tử trong dãy không quá 109). Hãy viết chương trình kiểm tra xem dãy có phải đối xứng hay không. Nếu đúng in ra YES, nếu sai in ra NO.

Input: Dòng đầu ghi số bộ test, mỗi bộ test gồm hai dòng. Dòng đầu là số phần tử của dãy, dòng sau ghi ra dãy đó, mỗi số cách nhau một khoảng trống.

Output: Ghi ra YES hoặc NO trên một dòng.

Phân tích bài toán:

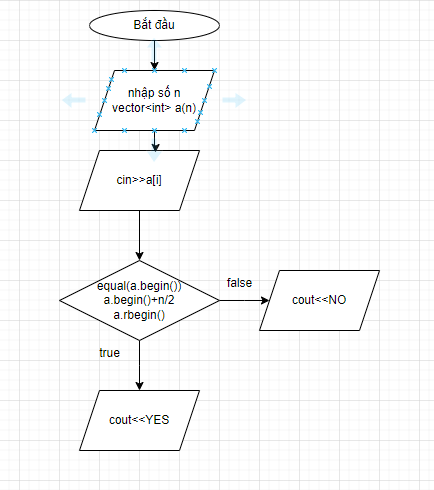
· Đầu tiên, chúng ta đọc số lượng test cases từ đầu vào (biến t).

· Sau đó, chúng ta sử dụng một vòng lặp while để xử lý từng test case. Trong mỗi test case, chúng ta đọc một số nguyên n từ đầu vào (biến n), đại diện cho số phần tử trong dãy.

· Chúng ta tạo một vector a có kích thước n để lưu trữ dãy số nguyên. Chúng ta sử dụng một vòng lặp for để đọc n số nguyên từ đầu vào và lưu trữ chúng vào vector a.

· Sử dụng hàm equal của thư viện algorithm để kiểm tra xem nửa đầu và nửa cuối của vector a có đối xứng nhau hay không. Hàm equal so sánh từng phần tử tương ứng của hai dãy. Nếu nửa đầu và nửa cuối của dãy đối xứng nhau, chúng ta in ra "YES". Ngược lại, in ra "NO".

· Quay lại bước 2 để xử lý test case tiếp theo.



- Độ phức tạp thuật toán của bài này là O(n)

BÀI 4. PHÂN TÍCH THỪA SỐ NGUYÊN TỐ

Hãy phân tích một số nguyên dương thành tích các thừa số nguyên tố.

Input: Dòng đầu tiên ghi số bộ test. Mỗi bộ test viết trên một dòng số nguyên dương n không quá 9 chữ số.

Output: Mỗi bộ test viết ra thứ tự bộ test, sau đó lần lượt là các số nguyên tố khác nhau có trong tích, với mỗi số viết thêm số lượng số đó.

Phân tích bài toán:

· Đầu tiên, chúng ta đọc số lượng test cases từ đầu vào (biến t).

· Sau đó, chúng ta sử dụng một vòng lặp for để xử lý từng test case. Trong mỗi test case, chúng ta đọc một số nguyên dương từ đầu vào (biến n).

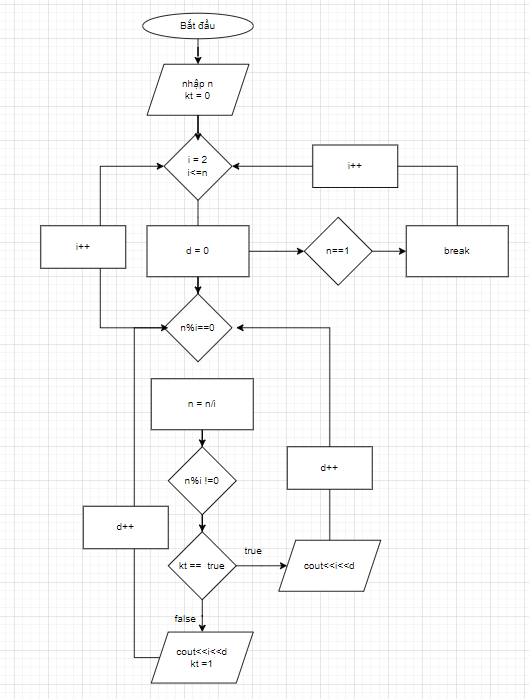
· Chúng ta tạo một biến kt và đặt giá trị ban đầu của nó là 0. Biến kt sẽ đóng vai trò để xác định xem đã in ra thừa số nguyên tố đầu tiên hay chưa.

· Chúng ta bắt đầu từ i = 2 và duyệt qua các giá trị i từ 2 đến n. Trong vòng lặp, chúng ta sử dụng một biến đếm d và vòng lặp while để tìm các thừa số nguyên tố của n. Trong vòng lặp while, chúng ta kiểm tra n có chia hết cho i hay không (n % i == 0). Nếu có, chúng ta tăng biến đếm d lên 1 và chia n cho i (n = n/i).

· Nếu n không chia hết cho i (n % i != 0), chúng ta kiểm tra xem đã in ra thừa số nguyên tố trước đó hay chưa (biến kt). Nếu đã in ra, chúng ta in ra thừa số nguyên tố hiện tại và giá trị của d. Ngược lại, chúng ta in ra thừa số nguyên tố hiện tại và giá trị của d, đồng thời đặt giá trị của kt là 1.

· Tiếp theo, chúng ta kiểm tra xem n có bằng 1 hay không. Nếu bằng 1, chúng ta thoát khỏi vòng lặp for. Ngược lại, tiếp tục vòng lặp để tìm các thừa số nguyên tố tiếp theo.

· Cuối cùng, chúng ta in xuống dòng và quay lại bước 2 để xử lý test case tiếp theo.



* Độ phức tạp của thuật toán này là O(n \* log n)

BÀI 5. PHÉP CỘNG

Cho một phép toán có dạng a + b = c với a,b,c chỉ là các số nguyên dương có một chữ số. Hãy kiểm tra xem phép toán đó có đúng hay không.

Input: Chỉ có một dòng ghi ra phép toán (gồm đúng 9 ký tự)

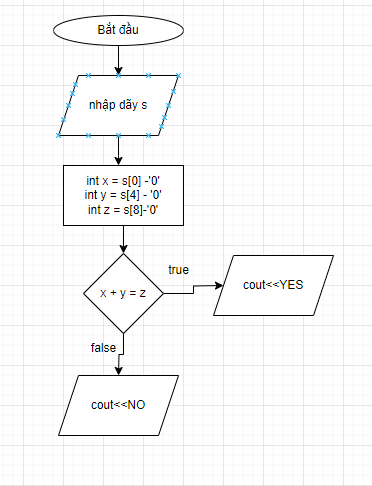
Output: Ghi ra YES nếu phép toán đó đúng. Ghi ra NO nếu sai.

Phân tích bài toán:

· Đầu tiên, chúng ta đọc một chuỗi từ đầu vào (biến s) để lưu trữ phép toán.

· Sau đó, chúng ta lấy giá trị của các chữ số a, b và c từ chuỗi s. Chữ số a được lấy từ vị trí thứ 0 (s[0]), chữ số b được lấy từ vị trí thứ 4 (s[4]) và chữ số c được lấy từ vị trí thứ 8 (s[8]). Chúng ta chuyển đổi các kí tự số sang giá trị số nguyên bằng cách trừ đi giá trị ASCII của kí tự '0' (s[0] - '0').

· Chúng ta kiểm tra xem tổng của a và b có bằng c hay không. Nếu có, chúng ta in ra "YES", ngược lại in ra "NO".



Độ phức tạp của thuật toán này là O(1);

BÀI 6. SỐ TĂNG GIẢM

Một số được gọi là số tăng giảm nếu số đó có các chữ số thỏa mãn hoặc không giảm, hoặc không tăng từ trái qua phải. Hãy kiểm tra xem một số có phải số tăng giảm hay không.

Input: Dòng đầu tiên ghi số bộ test. Mỗi bộ test viết trên một dòng một số nguyên dương cần kiểm tra, không quá 500 chữ số.

Output: Mỗi bộ test viết ra chữ YES nếu đó đúng là số tăng giảm, chữ NO nếu ngược lại.

Phân tích bài toán:

· Đầu tiên, chúng ta đọc số lượng test cases từ đầu vào (biến test\_cases).

· Sau đó, chúng ta sử dụng một vòng lặp while để xử lý từng test case. Trong mỗi test case, chúng ta đọc một chuỗi số từ đầu vào (biến num). Chúng ta tạo hai biến boolean increase và decrease, ban đầu đặt giá trị của cả hai biến là true. Biến increase sẽ đại diện cho điều kiện không giảm, trong khi biến decrease đại diện cho điều kiện không tăng.

· Tiếp theo, chúng ta duyệt qua từng chữ số trong chuỗi num bắt đầu từ vị trí thứ hai (i = 1). Đối với mỗi chữ số, chúng ta kiểm tra hai điều kiện:

o Nếu chữ số trước đó lớn hơn chữ số hiện tại, tức là chữ số giảm, chúng ta đặt giá trị của increase là false.

o Nếu chữ số trước đó nhỏ hơn chữ số hiện tại, tức là chữ số tăng, chúng ta đặt giá trị của decrease là false.

· Sau khi đã duyệt qua tất cả các chữ số, chúng ta trả về giá trị của biểu thức increase || decrease, đại diện cho việc kiểm tra xem số có thỏa mãn điều kiện không giảm hoặc không tăng từ trái qua phải hay không. Nếu giá trị trả về là true, chúng ta in ra "YES", ngược lại in ra "NO".

· Quay lại bước 2 để xử lý test case tiếp theo.  
  
Độ phức tạp của thuật toán trên là: O(n)

BÀI 7. CHUẨN HÓA 1

Các xâu họ tên trong Tiếng Việt được viết theo dạng chuẩn như sau:

- Chữ cái đầu mỗi từ viết hoa. Các chữ cái sau viết thường

- Các từ cách nhau đúng 1 khoảng trống

Hãy viết chương trình đưa danh sách các xâu họ tên về dạng chuẩn

Input: Dòng 1 ghi số N là xâu họ tên trong danh sách. N dòng tiếp theo ghi lần lượt các xâu họ tên (không quá 80 ký tự).

Output: Ghi ra các xâu chuẩn.

Phân tích bài toán:

· Đầu tiên, chúng ta đọc số lượng test cases từ đầu vào (biến t).

· Sau đó, chúng ta sử dụng một vòng lặp while để xử lý từng test case. Trong mỗi test case, chúng ta đọc một chuỗi họ tên từ đầu vào (biến s). Độ dài của chuỗi s được lưu vào biến n. Chúng ta tạo một chuỗi kết quả rỗng (biến result) để lưu trữ chuỗi họ tên sau khi được chuyển đổi.

· Tiếp theo, chúng ta duyệt qua từng kí tự trong chuỗi s. Đối với mỗi kí tự, chúng ta kiểm tra xem kí tự có nằm trong khoảng chữ cái in hoa ('A' đến 'Z') hay không. Nếu có, chúng ta chuyển đổi kí tự đó thành chữ cái thường bằng cách cộng 32 vào giá trị ASCII của nó.

· Chúng ta kiểm tra các điều kiện để xác định xem kí tự hiện tại có được thêm vào chuỗi kết quả hay không:

o Nếu kí tự không phải là khoảng trắng hoặc (kí tự là khoảng trắng và kí tự tiếp theo không phải khoảng trắng và không phải kết thúc chuỗi), chúng ta thêm kí tự hiện tại vào chuỗi kết quả.

o Nếu vị trí hiện tại là kí tự đầu tiên hoặc kí tự trước đó là khoảng trắng và kí tự hiện tại không phải là khoảng trắng, chúng ta chuyển đổi kí tự hiện tại thành chữ cái in hoa bằng cách trừ 32 từ giá trị ASCII của nó.

· Sau khi đã duyệt qua tất cả các kí tự trong chuỗi, chúng ta in ra chuỗi kết quả.

· In xuống dòng và quay lại bước 2 để xử lý test case tiếp theo.

Độ phức tạp của thuật toán trên là O(n)

BÀI 8. CHUẨN HÓA 2

Một trong các cách viết họ tên theo dạng chuẩn trong Tiếng Anh là phần họ được viết sau cùng, phân tách với phần tên đệm và tên bởi dấu phẩy. Các chữ cái của phần họ đều viết hoa. Cho trước các xâu họ tên. Hãy đưa về dạng chuẩn tương ứng.

Input:

- Dòng 1 ghi số N là xâu họ tên trong danh sách

- N dòng tiếp theo ghi lần lượt các xâu họ tên (không quá 50 ký tự)

Output: Ghi ra các xâu chuẩn.

Phân tích bài toán:

· Đầu tiên, chúng ta đọc số lượng test cases từ đầu vào (biến t).

· Sau đó, chúng ta sử dụng một vòng lặp while để xử lý từng test case. Trong mỗi test case, chúng ta đọc một chuỗi họ tên từ đầu vào (biến s). Độ dài của chuỗi s được lưu vào biến n.

· Chúng ta tạo một vector k có kích thước n và khởi tạo tất cả các phần tử của nó bằng false. Vector k được sử dụng để đánh dấu các vị trí có chứa kí tự hợp lệ trong chuỗi.

· Tiếp theo, chúng ta duyệt qua từng kí tự trong chuỗi s. Nếu kí tự là một chữ cái in hoa (kiểm tra bằng cách so sánh ký tự với 'A' và 'Z'), chúng ta chuyển đổi nó thành chữ cái thường bằng cách cộng 32. Chúng ta kiểm tra các điều kiện để đánh dấu các vị trí hợp lệ trong vector k:

o Nếu kí tự không phải là khoảng trắng hoặc (kí tự là khoảng trắng và kí tự tiếp theo không phải khoảng trắng và kí tự tiếp theo không phải kết thúc chuỗi), chúng ta đánh dấu vị trí hiện tại trong vector k là true.

o Nếu vị trí hiện tại là kí tự đầu tiên hoặc kí tự trước đó là khoảng trắng và kí tự hiện tại không phải là khoảng trắng, chúng ta chuyển đổi kí tự hiện tại thành chữ cái in hoa bằng cách trừ đi 32.

· Chúng ta tìm vị trí dầu tiên của khoảng trắng trong chuỗi s sau vị trí hiện tại (vị trí đầu tiên của phần họ). Chúng ta in ra phần họ (phần sau vị trí d) bằng cách duyệt từ vị trí d+1 đến cuối chuỗi và in ra các kí tự hợp lệ (có đánh dấu true trong vector k). Sau đó, chúng ta in ra dấu phẩy và khoảng trắng. Cuối cùng, chúng ta in ra phần tên đệm và tên (phần trước vị trí d) bằng cách duyệt từ đầu chuỗi đến vị trí d và in ra các kí tự hợp lệ (có đánh dấu true trong vector k).

· In xuống dòng và quay lại bước 2 để xử lý test case tiếp theo.

ĐỘ phức tạp của thuật toán trên là: O(n)

BÀI 9. VÒNG TRÒN

Tí viết bảng chữ cái 2 lần lên trên một vòng tròn, mỗi kí tự xuất hiện đúng 2 lần. Sau đó nối lần lượt các kí tự giống nhau lại. Tổng cộng có 26 đoạn thẳng. Hình vẽ quá chằng chịt, Tí muốn đố các bạn xem có tất cả bao nhiêu giao điểm? Một giao điểm được tính khi hai đường thẳng của một cặp kí tự cắt nhau.

Input: Gồm một xâu có đúng 52 kí tự in hoa. Mỗi kí tự xuất hiện đúng 2 lần.

Output: In ra đáp án tìm được.

Phân tích bài toán:

· Đầu tiên, chúng ta đọc một chuỗi kí tự từ đầu vào, đại diện cho việc lặp lại bảng chữ cái hai lần. Chuỗi này có tổng cộng 52 kí tự.

· Tiếp theo, chúng ta sử dụng một vòng lặp for để duyệt qua từng kí tự trong chuỗi. Đối với mỗi kí tự, chúng ta kiểm tra xem đã gặp kí tự đó trước đó chưa (a[c[i]] == 0). Nếu chưa, chúng ta đánh dấu kí tự đó đã được gặp và tiến hành tìm cặp kí tự giống nhau.

· Từ vị trí hiện tại của kí tự (ind1), chúng ta sử dụng hàm search để tìm vị trí xuất hiện đầu tiên của kí tự đó sau vị trí hiện tại (ind2). Hàm search sẽ trả về vị trí xuất hiện đầu tiên của kí tự đó trong khoảng từ i+1 đến 51.

· Sau đó, chúng ta sử dụng một mảng check để đánh dấu các kí tự đã được gặp trước đó trong khoảng từ ind1 + 1 đến ind2 - 1. Nếu kí tự đó chưa được đánh dấu (check[c[j]] == 0), chúng ta kiểm tra xem kí tự đó có xuất hiện sau vị trí hiện tại và trước vị trí ind2 - 1 không. Để làm điều này, chúng ta sử dụng hàm search và kiểm tra kết quả trả về. Nếu kết quả trả về là -1, tức là kí tự đó không xuất hiện sau vị trí hiện tại và trước vị trí ind2 - 1, chúng ta tăng biến đếm res lên 1.

· Cuối cùng, chúng ta in ra kết quả res chia 2, vì mỗi giao điểm được tính hai lần khi chúng ta duyệt qua các cặp kí tự.

Độ phức tạp của thuật toán trên là: O(n^2)

BÀI 10. CHIA HẾT CHO 2

Cho số nguyên dương N. Nhiệm vụ của bạn là hãy xác định xem có bao nhiêu ước số của N chia hết cho 2.

Input: Dòng đầu tiên là số lượng bộ test T (T ≤ 100). Mỗi bộ test gồm một số nguyên N (1 ≤ N ≤ 109)

Output: Với mỗi test, in ra đáp án tìm được trên một dòng.

Phân tích bài toán:

· Đầu tiên, chúng ta đọc số nguyên dương N từ đầu vào.

· Tiếp theo, chúng ta sử dụng vòng lặp for để tìm các ước số của N. Vòng lặp này chạy từ 1 đến căn bậc hai của N, và kiểm tra xem i có phải là ước số của N hay không bằng cách kiểm tra xem N chia hết cho i hay không (n % i == 0).

· Trong vòng lặp, nếu i chia hết cho 2 (i % 2 == 0), chúng ta tăng biến đếm d lên 1. Đồng thời, nếu phần nguyên của phép chia N cho i chia hết cho 2 (n / i % 2 == 0), chúng ta cũng tăng biến đếm d lên 1. Lý do là nếu i là một ước số của N chia hết cho 2, thì N/i cũng là một ước số chia hết cho 2.

· Cuối cùng, chúng ta in ra giá trị của biến đếm d, đại diện cho số ước số của N chia hết cho 2.

Độ phức tạp của thuật toán trên là: O(sqrt(n))