Trước khi đi vào cách giải quyết của bài thì mình muốn các bạn phải đảm bảo biết qua điều này để về sau các bạn xem mới hiểu:

Với các hệ thống web chấm bài giới hạn thời gian chạy của bài tối đa chỉ được 1 giây thì các bạn sẽ rất dễ bị rơi vào lỗi Time Limit Exceeded (TLE: Chạy quá giới hạn thời gian cho phép). Thì với các hệ thống web chấm bài hiện nay nếu không muốn bị TLE thì tối đa số phép toán trên toàn bài của các bạn chỉ nên ở ngưỡng (3 đến 5)\*10^7 là ngưỡng an toàn với phần lớn hệ thống chấm bài, hoặc với một số hệ thống chấm server mạnh hơn thì có thể nâng ngưỡng lên 10^8 nhưng cũng hên xui nhé. Và thực ra ta không phải đếm chi tiết từng số phép toán trên toàn bài mà ta dựa theo số lần vòng lặp chạy bởi bên trong vòng lặp là nó xử lý lặp đi lặp lại nhiều phép toán nên cứ canh theo đó, các bạn cứ hiểu là tối đa toàn bài chỉ nên lặp (3 đến 5)\*10^7 lần là an toàn hoặc tệ nhất là 10^8 (lúc này thì hên xui nhé) còn vượt ngưỡng này là khả năng TLE rất cao. Lưu ý là ngưỡng mình vừa nói ở trên là với ngôn ngữ C/C++ thôi nhé chứ nếu với những ngôn ngữ khác như Python, C# hay Java nó chạy chậm hơn C/C++ thì ngưỡng đó phải thấp hơn, hoặc người ra đề sẽ chủ động đưa ra giới hạn nếu nộp với C/C++ thì thời gian chạy quy định tối đa 1 giây, các ngôn ngữ khác thì 2 giây. Và ngưỡng mình đưa ra ở trên nếu với đề cho thời gian 2 giây thì các bạn cứ tỷ lệ thuận nhân 2 lên với ngưỡng, 3 giây thì nhân 3, nếu 100 ms tức là 1/10 giây thì các bạn chia 10 cho ngưỡng đó nhé, cứ thế thôi.

Rồi thì như bài này đưa ra thời gian chạy giới hạn là 1 giây nên cứ như ở trên mình đã nói nhé. Giờ ta đi phân tích cách giải quyết cho bài này.

Khá dễ để nhìn ra tư tưởng xử lý cho bài này đó là ta hãy tìm ra số z đầu tiên (cũng đồng nghĩa là nhỏ nhất) thỏa là số siêu đối xứng sao cho z > x để rồi từ đó tìm ra được y = z - x thì lúc đó giá trị y này bảo đảm là nhỏ nhất. Nhưng chỗ này các bạn lưu ý nhé: Là tìm ra số z > x nha chứ không có z >= x tức là không có xét trường hợp z = x nha. Bởi vì nếu z = x thì khi đó y = z - x = 0. Như thế là không đúng với đề bài yêu cầu y phải là giá trị “nguyên dương”, bởi theo định nghĩa số nguyên dương là số > 0 nha các bạn, số 0 không phải là số nguyên dương (cũng không phải là số nguyên âm) nên theo sát định nghĩa toán học ta phải chú ý kỹ chỗ này.

Vậy nên nếu giả sử x = 22 thì đáp án z không thể chọn là 22 được mà phải chọn z = 33 từ đó y = z - x = 33 - 22 = 10.

Ta hãy thử đưa ra nhiều test để từ đó nhìn thấy rõ cách làm cho bài này:

Nếu x = 78 => giá trị z nhỏ nhất thỏa z > x mà z là số siêu đối xứng thì chọn z = 88. Từ đó tính y = z - x = 88 - 78 = 10.

Nếu x = 99 => giá trị z nhỏ nhất thỏa z > x mà z là só siêu đối xứng thì chọn z = 111. Từ đó tính y = z - x = 111 - 99 = 12.

Nếu x = 123 => giá trị z nhỏ nhất thỏa z > x mà z là só siêu đối xứng thì chọn z = 222. Từ đó tính y = z - x = 222 - 123 = 99.

Nếu x = 9031 => giá trị z nhỏ nhất thỏa z > x mà z là só siêu đối xứng thì chọn z = 9999. Từ đó tính y = z - x = 9999 - 9031 = 968.

Nếu x = 9998 => giá trị z nhỏ nhất thỏa z > x mà z là só siêu đối xứng thì chọn z = 9999. Từ đó tính y = z - x = 9999 - 9998 = 1.

Nếu x = 9999 => giá trị z nhỏ nhất thỏa z > x mà z là só siêu đối xứng thì chọn z = 11111. Từ đó tính y = z - x = 11111 - 9999 = 1112.

Vậy đã quá rõ ràng rồi các bạn hen, mấu chốt quyết định là phải tìm ra số z nhỏ nhất thỏa là số siêu đối xứng sao cho z > x từ đó đáp án chính là z - x.

Vậy đến đây tuy nhìn ra cách giải nhưng mình đã thấy nhiều bạn chọn cách giải không khéo là rất phức tạp trong việc hiện thực và còn bị nhiều lỗi khó kiểm soát. Chưa kể có thể còn bị quá thời gian chạy nếu chọn hướng giải quyết không tối ưu.

Vậy một hướng giải quyết nhẹ đầu nhất mà nhiều bạn chắc chắn nhìn ra đó là cứ lần lượt tăng dần đều z lên 1 đơn vị rồi mỗi lần xong kiểm tra z có là số siêu nguyên tố không? Cứ thế đến khi z thỏa là số siêu nguyên tố thì xuất kết quả z tại thời điểm đó - x. Đây có thể xem là cách làm vét cạn dễ dàng cài đặt. Nếu bạn nào chưa nghĩ ra cách giải quyết tối ưu khác thì mình luôn khuyên trước tiên cứ cài đặt theo cách vét cạn này để ta có thể dùng nó làm công cụ kiểm tra tính chính xác cho một cách làm tối ưu sau này của chúng ta.

Source code để các bạn tham khảo nếu làm theo cách vét cạn:

#include <iostream>

using namespace std;

bool isSupersymmetryNumber(long long x)

{

int num = x % 10;

x /= 10;

while(x)

{

if(x % 10 != num)

{

return false;

}

x /= 10;

}

return true;

}

int main()

{

long long x;

cin >> x;

long long z = x;

while(!isSupersymmetryNumber(++z));

cout << z - x;

return 0;

}

Giải thích một số chỗ trong code ở trên cho bạn nào có không hiểu:

1/ Vì đề bài đã cho giới hạn của số x tối đa có thể lên đến 10^16 nên ta phải chọn kiểu số nguyên 8 byte (long long) cho biến x để tránh bị tràn số nha các bạn.

2/ while(x) <=> while(x != 0) nha các bạn, trong lập trình khi gọi 1 biến ra so sánh mà không chỉ ra rõ so sánh gì thì đồng nghĩa là so sánh có tồn tại (khác 0). Tương tự nếu là: while(!x) <=> while(x == 0). Trong lập trình 0 đại diện cho false nên nếu giả sử x là biến trả về kiểu bool (true/false) thì while(x) ⇔ while(x != false) ⇔ while(x == true) còn while(!x) <=> while(x == false) nha các bạn.

3/ Hàm isSupersymmetryNumber nhận vào tham số là biến số nguyên x và kiểm tra xem x có là số siêu đối xứng không? Bằng cách ban đầu tính num = x % 10; để lấy ra chữ số cuối cùng của x hiện tại làm mốc so sánh, sau đó x /= 10; để loại bỏ đi chữ số cuối cùng vừa lấy ra rồi bắt đầu quy trình lấy từng chữ số của số x ra bằng cách liên tục % 10 và / 10 đến khi nào x = 0 thì dừng, cứ mỗi chữ số lấy ra so sánh với num là chữ số cuối cùng trước đó nếu chỉ cần tìm thấy 1 chữ số khác num là return false; ngay vì đó không phải là số siêu đối xứng, còn nếu sau cùng khi vòng lặp kết thúc hàm vẫn còn chạy xuống dưới tức là không có trường hợp nào khiến return false; thì ta return true; vì lúc này mọi chữ số đều giống nhau.

4/ while(!isSupersymmetryNumber(++z)); vòng lặp sẽ liên tục lặp, luôn tăng giá trị biến z lên 1 đơn vị trước (z ban đầu được sao chép từ giá trị x sang). Sau đó gọi hàm kiểm tra xem giá trị z mới sau khi tăng đó có thỏa là số siêu đối xứng không? Nếu không thỏa thì vòng lặp sẽ tiếp tục lặp, vòng lặp chỉ dừng khi giá trị z thỏa. Sau cùng thì ta in ra kết quả z - x là đáp án.

Đánh giá độ phức tạp của cách làm này:

+ Độ phức tạp không gian (Space Complexity): O(1) do trong bài này chỉ dùng các biến đơn lẻ không dùng mảng gì để lưu trữ cả.

+ Độ phức tạp thời gian (Time Complexity): Ta thấy số lần lặp của vòng while ở hàm main sẽ là Z - X với Z là số siêu đối xứng nhỏ nhất thỏa > X. Nhưng bên trong mỗi lần lặp đó ta phải gọi hàm kiểm tra xem z có thỏa số siêu đối xứng không? Thì lúc này phải làm vòng lặp while lấy ra từng chữ số của Z. Cho rằng Z tối đa cũng chỉ là 16 chữ số như đề bài công bố, nên ta có thể xem độ phức tạp là O(16 \* (Z - X)). Thử nhẩm xem trường hợp xấu nhất là khi nào? Là khi Z - X ra kết quả thật lớn, rất dễ nhìn ra với trường hợp khi X là số siêu đối xứng thì Z sẽ là số siêu đối xứng tiếp theo, giả sử X là số siêu đối xứng có 16 chữ số như X = 1111111111111111 thì lúc này Z muốn thỏa phải là Z = 2222222222222222 và lúc này tính ra số lần lặp Z - X = 1,111,111,111,111,111 (hơn 1 triệu tỷ lần lặp) chưa kể còn phải nhân thêm 16 thành ra hơn 16 triệu tỷ lần lặp. Tức là 16 \* 10^15. Nhẩm tính thử nếu 1 giây chạy được 10^8 thì lúc này cần 16 \* 10^7 giây tức là 160 triệu giây <=> hơn 5 năm mới chạy ra kết quả nha các bạn. Trong khi đó đề bài chỉ đưa ra giới hạn chạy trong 1 giây, nên code này chắc chắn sẽ bị TLE nha các bạn. Nhưng vẫn nên viết ra để dùng nó làm công cụ kiểm tra tính chính xác cho cách làm tối ưu hồi đầu bài mình có nói.

============

Giờ ta cần tìm ra một cách làm tốt hơn cho bài này. Ta không cần phải cực như thế khi cứ liên tục lặp tăng z lên dần đều 1 đơn vị. Rõ ràng các bạn thấy ví dụ khi x = 11111 là số siêu đối xứng thì z = 22222 là số siêu đối xứng tiếp theo thỏa mãn, cái này quá rõ để nhìn ra, thì tại sao phải mất công lặp tăng dần đều 1 đơn vị?

Dễ nhất cho bài này là áp dụng chuỗi (string) nha các bạn. Cụ thể ngay từ đầu bài đọc biến x ở định dạng string luôn. Ví dụ string x = “102”. Và các bạn sẽ nhìn ra ta sẽ tạo ra biến string z muốn thỏa là số siêu đối xứng nhỏ nhất > x thì trước tiên z sẽ có độ dài (số lượng chữ số) cũng bằng với x luôn đúng không các bạn? Tức là trong trường hợp này z sẽ có độ dài = 3. Chứ nếu z có độ dài nhỏ hơn độ dài x thì chắc chắn z đã nhỏ hơn x rồi và như vậy là không đúng với hướng giải quyết chúng ta đề ra.

Rồi ta khởi đầu sẽ cho toàn bộ chuỗi z sẽ chính là ký tự chữ số đầu tiên của x, nên khi x = “102” thì z = “111”. Lúc này rồi ta so sánh chuỗi z có thỏa lớn hơn x không? Việc so sánh này class string trong C++ đã hỗ trợ sẵn các operator so sánh nên ta chỉ việc gọi if(z > x) là được, lúc này nó sẽ làm theo nguyên tắc so sánh chuỗi trong C++ là đi từ trái qua phải lần lượt từng ký tự tại vị trí index tương ứng so sánh mã ascii với nhau xem ký tự nào mà lớn hơn thì kết luận ngay chuỗi chứa ký tự đó sẽ lớn hơn, nếu bằng nhau thì tiếp tục lặp lại quy trình trên qua ký tự tiếp theo. Và vô hình là cách so sánh như vậy cũng chính là cách so sánh 2 số nguyên có cùng số lượng chữ số (độ dài) coi số nào lớn hơn cũng là đi từ trái qua phải và so sánh từ hàng cao nhất về hàng thấp nhất, ví dụ với số có 3 chữ số thì so hàng trăm với nhau, so hàng chục với nhau, sau cùng là so hàng đơn vị. Thì nếu tại hàng trăm đã có sự khác biệt là đã kết luận luôn được rồi không cần so hàng chục, hàng đơn vị. Nên nếu lúc này z = “111” và x = “103” so sánh thấy z > x nên ta sẽ tiến hành đổi chuỗi z và x sang số nguyên tương ứng là 111 và 103 rồi lấy 2 số đó trừ nhau để được 111 - 103 = 8.

Nhưng hãy thử với ví dụ chuỗi x = “135” thì sao? Lúc này tạo ra chuỗi z = “111” và so sánh thấy chuỗi z không thỏa lớn hơn chuỗi x, lúc này thì sao? Lúc này ta phải biến chuỗi z thành số siêu đối xứng tiếp theo tức là z = “222” rồi thì giờ đây chắc chắn z > x nên ta chỉ việc đổi sang số nguyên tương ứng rồi trừ nhau: 222 - 135 = 87. Vậy việc biến chuỗi z thành số siêu đối xứng tiếp theo lúc này chỉ là ban đầu cho toàn bộ chuỗi s chính là ký tự x[0] thì giờ sẽ cho toàn bộ chuỗi z là ký tự x[0] + 1 là được.

Tuy nhiên sẽ có tình huống như sau, ví dụ chuỗi x = “999” thì sao? Lúc này tạo ra z = “999” so sánh thấy z không thỏa lớn hơn x, thì lúc này z phải biến thành “1111” rồi thì 1111 - 999 = 112. Vậy lúc này ta hiểu khi so sánh thấy z không lớn hơn x thì biến z thành toàn bộ ký tự x[0] + 1 nhưng nếu bản thân x[0] đang là ký tự ‘9’, lúc này phải tạo ra chuỗi z có độ dài length(x) + 1 tức là dài hơn chuỗi x 1 đơn vị, rồi thì toàn bộ ký tự trong chuỗi z sẽ toàn là ký tự ‘1’ nha các bạn. Rồi thì chuyển chuỗi z và x qua số tương ứng rồi lấy z - x như bình thường. Như lúc này là 1111 - 999 = 112.

Vậy ta chuẩn hóa lại quy trình như sau: Đầu tiên tạo ra chuỗi z có độ dài bằng độ dài chuỗi x và chuỗi z có toàn bộ các ký tự là ký tự đầu tiên của chuỗi x (x[0]). Rồi so sánh nếu thấy chuỗi z thỏa mãn lớn hơn chuỗi x thì đổi chuỗi z và x sang số nguyên rồi lấy z - x. Còn nếu chuỗi z không thỏa mãn lớn hơn chuỗi x lúc này kiểm tra xem ký tự đầu tiên của chuỗi z lúc này (hoặc chuỗi x cũng được, như nhau nha các bạn) có phải ký tự ‘9’ không? Nếu không phải thì tiến hành đổi toàn bộ ký tự chuỗi z sang ký tự z[0] + 1 (hoặc x[0] + 1 cũng như nhau nha các bạn). Sau đó đổi chuỗi z và chuỗi x ra số nguyên rồi lấy z - x. Còn nếu ký tự đầu tiên của chuỗi z là ký tự ‘9’ lúc này ta sẽ tạo ra chuỗi z có thêm 1 ký tự mới (độ dài hơn chuỗi x 1 đơn vị) và toàn bộ các ký tự của chuỗi z sẽ là ký tự ‘1’. Rồi thì đổi chuỗi z và x sang số nguyên rồi lấy z - x. Tóm lại quy trình là vậy thôi nha các bạn.

Source code để các bạn tham khảo với hướng làm tối ưu này:

#include <iostream>

#include <iomanip>

#include <string>

using namespace std;

int main()

{

string x;

cin >> x;

string z(x.length(), x[0]);

if(z > x)

{

cout << stoll(z) - stoll(x);

}

else

{

if(z[0] < '9')

{

fill(z.begin(), z.end(), z[0] + 1);

cout << stoll(z) - stoll(x);

}

else

{

cout << setfill('1') << setw(z.length()) << 2;

}

}

return 0;

}

Giải thích một số chỗ trong code ở trên cho bạn nào có không hiểu:

1/ string z(x.length(), x[0]); nghĩa là dùng constructor with parameters (hàm khởi tạo có tham số) của class string để tạo ra sẵn chuỗi có độ dài là x.length(); và toàn bộ chuỗi đều là ký tự x[0]. Bạn nào không biết cú pháp này thì sẽ phải code dài hơn lắm, biết nó sẽ rất tiện.

2/ stoll(z) là chuyển chuỗi z sang số nguyên tương ứng kiểu long long. Tương tự stoll(x) là chuyển chuỗi x sang số nguyên tương ứng kiểu long long. Kết quả trả về của 2 hàm này là trả về số nguyên và ta có thể lấy 2 kết quả đó trừ nhau. Tương tự ta có hàm stoi là trả về số nguyên kiểu int nha các bạn.

3/ Các bạn sẽ thấy chỗ nếu z[0] < ‘9’ thì ta sẽ gọi hàm fill(z.begin(), z.end(), z[0] + 1); ý nghĩa là thay thế toàn bộ các ký tự từ đầu chuỗi z (z.begin()) đến cuối chuỗi (z.end()) và thay thế bằng ký tự chỉ định là z[0] + 1. Rồi thì sau đó đổi z và x sang số nguyên rồi trừ đi.

4/ Cuối cùng là trong trường hợp nếu chuỗi z được tạo ra ban đầu không thỏa > x lúc này kiểm tra thấy z[0] lại là ký tự ‘9’. Lúc này ta hiểu nó chính là trường hợp x cũng là số siêu đối xứng và đều là toàn bộ ký tự ‘9’. Thì lúc này theo như mô tả mình nói là ta sẽ tạo lại chuỗi z có độ dài tăng thêm 1 và toàn bộ là ký tự ‘1’ sau đó đổi sang số và lấy z - x như bình thường. Thì ở đây nếu các bạn để ý sẽ nhìn thấy quy luật sau. Nếu x = 9 thì lúc này kết quả = 11 - 9 = 2. Nếu x = 99 thì lúc này kết quả = 111 - 99 = 12, x = 999 => 112, x = 9999 => 1112. Các bạn thấy quy luật gì không? Lúc này đáp án sẽ luôn là length\_x - 1 các chữ số 1 và sau đó là duy nhất chữ số 2. Như ví dụ x = 9999 (4 số 9) thì đáp án sẽ là 3 chữ số 1 sau đó là 1 chữ số 2. Vậy nên lúc này ta có thể theo quy luật đó mà in ra luôn kết quả khỏi phải mất công tạo lại chuỗi z rồi lại đổi chuỗi z và x sang số nguyên rồi trừ nhau làm gì. Ta để luôn: cout << setfill('1') << setw(z.length()) << 2; nghĩa là in ra số 2 và các ký tự ‘1’ đằng trước nó sao cho tất cả (tính luôn cả số 2) đều có độ dài đúng bằng z.length().

Đánh giá độ phức tạp của cách làm này:

+ Độ phức tạp không gian (Space Complexity): O(1) do trong bài này chỉ dùng các biến đơn lẻ không dùng mảng gì để lưu trữ cả.

+ Độ phức tạp thời gian (Time Complexity): Toàn bộ các thao tác như tạo ra chuỗi z ban đầu, rồi đổi chuỗi z và x sang số nguyên rồi trừ nhau, hay cập nhật lại toàn bộ chuỗi z với giá trị mới, hay in ra các ký tự ‘1’ sau đó in ra 2. Thì tất cả đều có độ phức tạp là O(length\_x) và dù có nhiều thành phần cộng lại với nhau thì tổng quát theo quy tắc cộng BigO chỉ lấy theo thành phần lớn nhất nên chốt lại có thể kết luận là O(length\_x). Mà nếu như vậy thì các bạn xem số x tối đa cũng chỉ có 10^16 tức là 16 chữ số vì vậy nếu trong ngữ cảnh bài này thì ta thậm chí có thể kết luận độ phức tạp thời gian là hằng số vì nó quá nhỏ. Nhưng vẫn cứ để là O(length\_x) để nếu bài này mà nâng giới hạn số x có thể tối đa đến 10^1000 hay 10^100,000 thì lại khác.

Vậy các bạn xem với độ phức tạp hằng số có thể nói là ra kết quả ngay lập tức luôn nha các bạn không cần phải chờ đợi gì hết. Thử chạy ngay với trường hợp xấu nhất mà mình lấy ở cách làm trước đó là X = 111… (16 chữ số 1) thì ngay lập tức nó trả ra được đáp án luôn. Còn cách làm trước đó ước chừng phải hơn 5 năm mới ra được kết quả. Các bạn đã thấy được cái hay của giải thuật chưa, chịu khó suy luận tối ưu lại 1 chút mà chạy nhanh hơn rất nhiều nè

==========

Đến đây thì xong rồi nhưng sẵn mình cũng cho các bạn xem luôn nếu làm theo bài viết hướng dẫn mình để đầu bài dùng cách làm vét cạn làm công cụ đi kiểm tra tính chính xác cho cách làm tối ưu xem có trường hợp nào sai không thì làm sao nhé

#include <iostream>

#include <ctime>

#include <string>

using namespace std;

bool isSupersymmetryNumber(long long x)

{

int num = x % 10;

x /= 10;

while(x)

{

if(x % 10 != num)

{

return false;

}

x /= 10;

}

return true;

}

long long solve1(long long x)

{

long long z = x;

while(!isSupersymmetryNumber(++z));

return z - x;

}

long long solve2(long long a)

{

string x = to\_string(a);

string z(x.length(), x[0]);

if(z > x)

{

return stoll(z) - stoll(x);

}

if(z[0] < '9')

{

fill(z.begin(), z.end(), z[0] + 1);

return stoll(z) - stoll(x);

}

fill(z.begin(), z.end(), '1');

z.back() = '2';

return stoll(z);

}

int main()

{

srand(time(0));

while(1)

{

long long x = 1 + rand() % 1000000;

long long res1 = solve1(x);

long long res2 = solve2(x);

if(res1 == res2)

{

cout << "OK\n";

}

else

{

cout << "FAIL\n";

cout << "x = " << x << endl;

cout << "res1 = " << res1 << endl;

cout << "res2 = " << res2;

break;

}

}

return 0;

}

Chạy chương trình lên nếu treo 10 phút mà nhìn lại toàn thấy chữ OK hết thì yên tâm là code mình ổn, nếu nhìn mà thấy chữ FAIL to đùng xuất hiện và chương trình dừng ngay đó rồi có xuất ra các giá trị x, res1, res2 là ta đã biết bị sai tại trường hợp x đó và ta có thể kiểm tra lại bài làm với trường hợp đó.