TỔ CHỨC THE GIFTED BATTLEFIELD
DỰ ÁN INTELLECTUAL SHOWDOWN

Educational Contest 1 - String
NĂM HỌC 2023 – 2024

Ngày thi: **29/10/2023**
Thời gian làm bài: **7 ngày**
Hướng dẫn giải gồm 07 trang, 11 bài

Tổng quan đề thi

	Tên bài	Hạn chế thời gian (s)	Hạn chế bộ nhớ (MB)
Bài 1	Radar biển	1	256
Bài 2	DNA	2	256
Bài 3	Xâu đối xứng	2	256
Bài 4	So sánh xâu	1	256
Bài 5	Baby String	2	256
Bài 6	Copy	1	256
Bài 7	Fibonacci	2	256
Bài 8	Xâu phân biệt	2	512
Bài 9	So sánh đuôi	2	512
Bài 10	Tổng lớn nhất	2	1024
Bài 11	Xâu ưng ý	2	512

Nhập xuất theo kiểu chuẩn - standardIO, không sử dụng nhập xuất file.

I. Hướng dẫn chung

1. Bài thi của thí sinh được chấm thi bằng phần mềm chấm thi trực tuyến (online judge) trên website https://oj.giftedbat.edu.vn/, sử dụng bộ test của Tổ chức The Gifted Battlefield – Ban Tin học, đúng với đáp án và biểu điểm được nêu trong hướng dẫn chấm thi.
2. Điểm bài thi được xuất từ phần mềm chấm thi.

Bài 1. Radar biển

Radar sẽ cần sửa khi: đã đi qua đầu/đuôi và cơ thể của sinh vật nhưng lại gặp đầu/đuôi. Vì vậy cần biết ta có đang ở trong một sinh vật biển hay không, và nếu có thì đã đi qua từ đầu hay đuôi của nó. Có thể làm được điều này bằng kỹ thuật đặt cờ hiệu.
Đi từ đầu đến cuối, gặp đầu hoặc đuôi thì xét biến "cờ" xem có đang ở trong thủy quái không. Nếu không, ta lưu biến cờ thành phần tử vừa đi qua; nếu có, phần tử hiện tại phải ngược lại với "cờ" (cờ là đầu/đuôi thì phần tử là đuôi/đầu) thì radar mới không cần sửa.

Bài 2. DNA

Prerequisite: Kỹ thuật Hashing (Rolling hash).
Ta có thể dễ dàng nhận thấy: nếu ta tìm được xâu có độ dài l thì cũng sẽ tìm được xâu có độ dài $l-1$ thỏa mãn yêu cầu đề bài. Vậy nên, ta có thể dùng tìm kiếm nhị phân để tìm độ dài xâu lớn nhất thỏa yêu cầu đề bài.
Mỗi bước chặt nhị phân, ta có được độ dài cần khảo sát là l, ta lưu lại mã hash của mọi xâu bắt đầu tại $i-l+1$ và kết thúc tại i ($\forall i : l \leq i \leq |s|$) và đếm xem có mã hash nào trùng nhau hay không.

Bài 3. Xâu đối xứng

Prerequisite: Hashing; kiểm tra Palindrome bằng hash.
Trong phần lời giải của bài này, tác giả quy ước cách gọi $T[u \dots v]$ $(u \leq v)$ là xâu con của T bắt đầu tại u và kết thúc tại v.
Để thêm số lượng kí tự ít nhất vào đuôi xâu S để tạo ra xâu S mới (tạm gọi là S') là một palindrome, trước tiên ta phải biết được hậu tố dài nhất của S là một palindrome. Nói cách khác, ta cần phải tính được số l sao cho xâu con của $S[n-l+1 \dots n]$ là palindrome và l là lớn nhất có thể.
Ví dụ, cho xâu $S =$ "abbcb".
Dễ dàng thấy được hậu tố palindrome dài nhất là "bcb".
Ta có thể tạo xâu S' mới là xâu S nối với nghịch đảo của tiền tố xâu S mà không chứa hậu tố là palindrome trên. Nói cách khác, xâu kết quả của bài toán sẽ là xâu mới $S' = S + reverse(S[1 \dots n-l])$ với l được định nghĩa như trên và hàm $reverse(x)$ là hàm đảo ngược xâu x.
$\Rightarrow S' =$ "$abbcb$" + "ba" = "$abbcbba$" là palindrome mới tạo được mà thêm số kí tự mới ít nhất (2 ký tự).
Thêm ví dụ từ test mẫu:

Xâu S	Hậu tố palindrome	Xâu S'
"aaaa"	"aaaa"	""+ "aaaa"
"educontest"	"t"	"educontest"+ "setnocude"
"xyz"	"z"	"xyz"+ "yx"
"abcdefghg"	"ghg"	"abcdefghg"+ "fedcba"

Để tìm hậu tố palindrome dài nhất của S, ta có 2 phương pháp như sau:

- Chặt nhị phân: với mỗi độ dài l, kiểm tra xem hậu tố bắt đầu lại vị trí $n-l+1$ có phải là 1 palindrome hay không bằng kỹ thuật 2 con trỏ, độ phức tạp là $O(n \log)$.
- Sử dụng **hash**: Tạo ra 2 mảng hash $head[]$, $tail[]$ lần lượt chứa mã hash của xâu S theo chiều thuận (từ đầu đến cuối) và chiều ngược (từ cuối về đầu). Để kiểm tra xâu con $S[l \dots r]$ có phải palindrome hay không, ta kiểm tra điều kiện head $(l, r) =$ tail(l, r).

Bài 4. So sánh xâu

Prerequisite: Hash, Segment Tree.
Dễ thấy nếu bài toán không có truy vấn 1, ta có thể giải bằng thuật toán Hashing thông thường.
Cách so sánh 2 xâu bằng hash:

- Nếu giá trị hash của đoạn [x, y] = [u, v] thì 2 xâu này bằng nhau
- Nếu 2 xâu này khác nhau, ta có thể chặt nhị phân để tìm ra vị trí p nhỏ nhất thỏa 2 điều kiện:
  - $x + p \leq y$ và $u + p \leq v$.
  - Giá trị hash của hash $[x, x + p] \neq$ hash $[u, u + p]$.
- Nếu không tồn tại giá trị p thỏa điều kiện, khi này chắc chắn 1 trong 2 xâu sẽ là tiền tố của xâu còn lại. Việc cần làm chỉ là so sánh độ dài của 2 xâu trên với nhau.
- Nếu tồn tại giá trị p thỏa, ta biết rằng đây là vị trí đầu tiên khác nhau của 2 xâu con. Do vậy nếu $s[x+p] < s[u+p]$ thì xâu con $S[x...y] < S[u...v]$ và ngược lại. Thực hiện các thao tác trên trong độ phức tạp $O(\log(n))$

Ở bài toán của đề, ta có thêm 1 thao tác thay đổi giá trị tại 1 điểm. Vậy nên ta phải kết hợp kĩ thuật Hash ở trên với 1 dạng CTDL có khả năng point update và lấy được giá trị của 1 đoạn hash tương ứng. Có thể sử dụng bằng Fenwick Tree, Segment Tree,...

Code mẫu sẽ được viết bằng Fenwick Tree. Sử dụng tính chất:

Với $pre[i]$ là một giá trị được tính trước với vị trí i. Dễ thấy base là constant, nên ta chỉ cần tìm cách update các giá trị pre này.
Trên cây Fenwick Tree, khi update vị trí pos, mọi vị trí id mà nó đi tới tiếp theo sẽ bị thay đổi với $base[id-pos]$ (có thể tự vẽ cây ra và hình dung, phần chứng minh xin nhường lại bạn đọc). Tương tự khi tính $pre[pos]$, mọi vị trí id mà nó đi ngược về sẽ bị ảnh hưởng bởi $base[pos-id]$.

Từ đây ta thu được 1 cách update các giá trị pre trong $O(\log(n))$.
Độ phức tạp tổng quát của cách làm này sẽ là $O(q \times \log(n))$.

Bài 5. Baby String

Prerequisite: Hàm Z.
Khi gặp dạng bài prefix, suffix đối với xâu, ta thường nghĩ đến ngay Z-function.
Gọi mảng z là mảng Z-function trên xâu s.
Để một xâu bắt đầu tại i vừa là prefix vừa là suffix thì phải thỏa mãn $z[i]+i=n$ (1) với n là chiều dài xâu s.
Để đếm số lần lặp của một xâu ta có nhận xét như sau:

- Vì các xâu thỏa mãn cùng lúc hai điều kiện (vừa là suffix, prefix), nên các xâu thỏa mãn có cùng độ dài thì sẽ giống nhau.
- Vì các xâu cần đếm đều là các prefix của xâu $s \geq$ đếm các xâu với mỗi chiều dài $z[i]$. Tuy nhiên, ta nhận thấy rằng với mỗi xâu có chiều dài $z[i]$ thì sẽ có các prefix con có chiều dài từ $1 \rightarrow z[i]$. Ví dụ "abcabc"tại $i=3$ (đếm từ 0) có giá $z[i]=3$ tức là xâu "abc", thì ngoài ra còn xâu "a", "ab"cũng thỏa là một prefix.

$\Rightarrow$ Cách xử lí dùng mảng đếm các giá trị $z[i]$ sau đó cộng dồn từ cao xuống thấp.

Bài 6. Copy

Prerequisite: KMP, DP KMP.
Ta sẽ sử dụng quy hoạch động để xử lý bài toán. Định nghĩa $dp[i]$ $(i \leq n)$ là số xâu phân biệt độ dài i có thể tạo ra với thao tác như đã nói; $pi[i]$ là độ dài của tiền tố lớn nhất của xâu S trùng với hậu tố cùng độ dài kết thúc tại i.
Mảng $pi[]$ có thể được dựng bằng thuật toán KMP

Ta định nghĩa $S[i..j]$ là xâu con của S bắt đầu tại i và kết thúc tại j.

Nhận xét: Ta sẽ chỉ thêm các prefix có giá trị $pi[i]=0$ vào xâu s' có độ dài n' để tạo ra xâu mới có độ dài $n'+i+1$ để tránh bị trùng vì khi chọn 1 prefix có $pi[i] \neq 0$ thì cách chọn này sẽ bị trùng với cách chọn 2 prefix có độ dài $pi[i]$ và $i-pi[i]+1$ vì
$S[pi[i] \dots i] = S[0 \dots pi[i]-1]$
Do đó, ta sẽ dựng mảng $dp[]$ với công thức truy hồi như sau:

$$dp[n] = 1$$

$$dp[i] = \sum_{j=1}^{n} dp[i-j-1], \text{ nếu } pi[j-1] = 0.$$

Đáp án của bài toán là $dp[m]$.
Độ phức tạp thời gian: $O(m^2+n)$.

Bài 7. Fibonacci

Prerequisite: Automaton, KMP.
Đáp án là -1 khi và chỉ khi $F_1=0, F_2=0$ và $\forall i : S_i=0$. Ta xét các trường hợp còn lại.

- Nếu $F_1 = 0$ và $F_2 = 0$:

  Sử dụng các phép biến đổi để tạo đầy đủ dãy S (điền đủ dãy S), sau đó thực hiện điền suffix nhỏ nhất của S mà nó tạo với một suffix nào đó của S để tạo thêm một dãy S. Ví dụ $S = 1, 1, 2, 2, 1, 1$ thì cách điền sẽ là (112211) (2211) (2211) (2211)... cho tới khi hết số lượt biến đổi. Suffix này có thể tìm được bằng thuật toán KMP trên S.

- Nếu $F_1 \neq 0$ hoặc $F_2 \neq 0$:

  Dãy tăng nhanh, do đó sẽ có một F_L nào đó mà các phần tử của F từ F_L trở đi không thể biến đổi về một phần tử thuộc S. Dưới giới hạn của đề bài, chỉ có tối đa khoảng 90 phần tử đầu của F có thể đưa về một phần tử thuộc $S : L < 90$. Do đó nếu n lớn hơn L, đáp án là 0. Xét quy hoạch động $dp(i, j, c)$ với trạng thái (i, j, c) mang ý nghĩa: "suffix dài nhất kết thúc tại i và trùng với một prefix nào đó của S" đang có độ dài là j, đã thực hiện c phép biến đổi và chỉ biến đổi các phần tử không đứng sau i; giá trị của $dp(i, j, c)$ là số lần xuất hiện tối đa của S có thể đạt được (hoặc -1 nếu không thể thỏa được điều kiện). Ta thử biến đổi phần tử thứ $i + 1$ của F, hoặc là để yên, hoặc là đưa về một phần tử nào đó của S và tính lại KMP cho vị trí này rồi chuyển trạng thái trên KMP thu được (có thể làm nhanh nếu tính trước **KMP automaton**: $next(i, j) = pi(i)$ nếu vị trí i có giá trị j).

Có $O\left(k \times L^2\right)$ trạng thái và mất $O(L)$ mỗi lần chuyển trạng thái. Độ phức tạp tổng là $O(k \times L^3)$.

Bài 8. Xâu phân biệt

Prerequisite: KMP.
Tham khảo VNOI Wiki: Thuật toán KMP - Đếm số xâu con phân biệt trong một xâu.

Bài 9. So sánh đuôi

Prerequisite: Trie.
Ta sẽ giải quyết bài toán này bằng CTDL Trie kết hợp với thuật toán quy hoạch động. Sử dụng Trie, ta lưu trũ được tất cả xâu kí tự trong mảng mà đề bài cho để mỗi đỉnh trong cây tương ứng với một prefix nào đó của các xâu trong mảng. Tại mỗi đỉnh, ta sẽ lưu thêm hai giá trị:

- $dp[v][c]$: đỉnh nào sẽ trở nên quan trọng khi có truy vấn $(v + 1, c)$ mà v là kí tự cuối của 1 prefix của xâu S. Ở đây, khái niệm đỉnh quan trọng là đỉnh chứa vị trí cuối cùng của prefix dài nhất của xâu S có trong cây tiền tố.

- $ans[v][c]$: Bao nhiêu xâu trong cây tiền tố có thứ tự từ điển nhỏ hơn prefix và kết thúc bằng v + "tất cả kì tự sau là c".

Cả hai giá trị trên đều có thể được tính bằng cách sử dụng kỹ thuật DFS trên cây Trie.
Ở thao tác truy vấn sửa đổi, ta có thể xét hai trường hợp:

- Nếu truy vấn thay đổi không nằm tại prefix dài nhất đang có hiện tại và kí tự phía sau nó thì ta có thể bỏ qua vì nó không ảnh hưởng gì tới kết quả. Giả sử ta có xâu S là "aabdd" và một xâu trong mảng là "aabcd", ta có prefix dài nhất là "aab" nên kí tự phía sau ('d' và 'c') là kí tự quan trọng ảnh hưởng tới thứ tự từ điển của hai xâu. Ngoài ra thì phần còn lại sau đó không còn quan trọng tới kết quả bài toán nữa.

- Trong trường hợp truy vấn nằm trong prefix dài nhất đang có hiện tại, ta sử dụng thuật toán Binary Lift (có thể xây dựng mảng chứa đỉnh cha cùng lúc trong hàm DFS ở trên) để tìm đỉnh V mới là prefix chung của các xâu con trong cây tiền tố và xâu S chưa tính đến phần thay đổi. Sau đó ta có thể sử dụng mảng dp để tìm ra được đỉnh quan trọng.

Phân tích độ phức tạp :

- Xây dựng cây Trie tốn $O(D)$ với D là tổng số kí tự có trong các xâu trong mảng mà đề bài cho.

- Hàm DFS có độ phức tạp là $O(D \times (K + \log D))$ với D là tổng độ dài các xâu trong mảng và K là số kí tự viết thường trong bảng chữ cái Latin.

- Trả lời truy vấn có độ phức tạp là $O(q \times \log D)$.

Vậy tổng thể độ phức tạp của bài toán là $O(D \times (K + \log D + q \times \log D)$.

Bài 10. Tổng lớn nhất

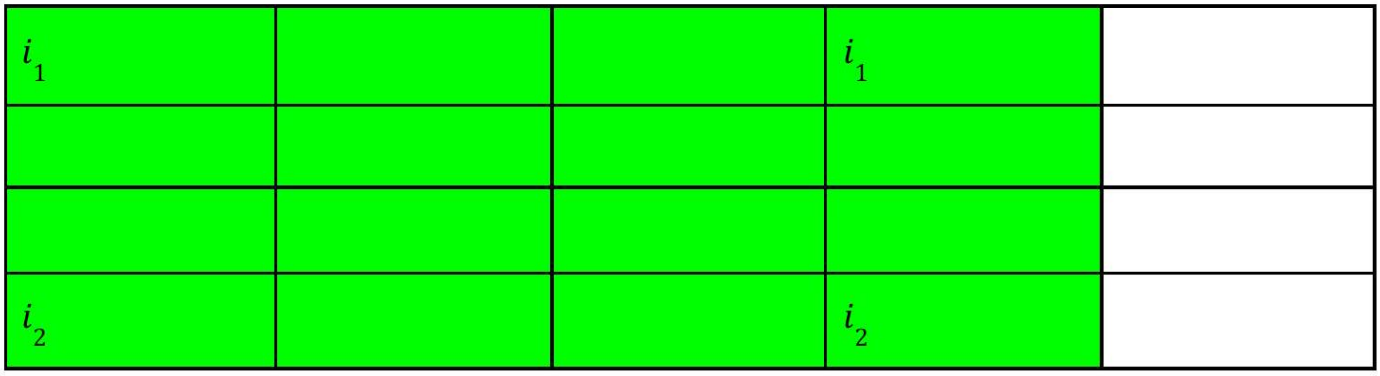
Prerequisite: Trie

Đầu tiên, chúng ta sẽ giải bài toán con sau:
Cho mảng A gồm N phần tử, hãy tìm một đoạn con liên tiếp sao cho tổng XOR ($\oplus$) các phần tử được chọn là lớn nhất.
Giới hạn: $1 \leq N \leq 10^5, 0 \leq A_i < 2^{25}$.
Với bài toán này, chúng ta sẽ sử dụng cấu trúc dữ liệu Trie để giải quyết:

- Gọi $pref\_xor[i] = a_1 \oplus a_2 \oplus \ldots \oplus a_i = pref\_xor[i-1] \oplus a_i$. Lúc này, giả sử ta đang xét vị trí thứ i. Ta sẽ tìm một prefix j $(j < i)$ sao cho $pref\_xor[i] \oplus pref\_xor[j]$ là lớn nhất. Với điều kiện này, bài toán đã trở thành: Cho dãy số B và số x. Tìm B_i sao cho $B_i \oplus x$ là lớn nhất. Tham khảo: VNOI Wiki - Trie .
- Độ phức tạp: $O\left(N \times \log_2\left(A_i\right)\right)$.

Bây giờ, ta sẽ mở rộng bài toán sang hai chiều:

- Thuật toán "ngây thơ"sẽ là sử dụng 4 vòng *for* lồng nhau để xét các tọa độ trái trên, phải dưới cũng như kết hợp với kỹ thuật mảng tiền tố để xét tất cả các bảng con. Độ phức tạp: $O(n^2 \times m^2)$.
- Lúc này, ta cần một nhận xét để giảm bớt độ phức tạp.
- Gọi i_1, i_2 là chỉ số hàng của ô trái trên và phải dưới ta xét. Ta sẽ cố gắng cố định hai chỉ số này và tìm cách tối ưu.



Hình 1: Minh họa về chỉ số i_1, i_2

- Lúc này, gọi $pref\_row[i][j]$ là tổng XOR các phần tử $a[i][1], a[i][2], \ldots, a[i][j]$. Ta có thể tính tổng XOR của các hàng trong bảng ta đang xét một cách dễ dàng.
- Gọi $b[j] = a\left[i_1\right]\left[j\right] \oplus a\left[i_1+1\right]\left[j\right] \oplus \ldots \oplus a\left[i_2\right]\left[j\right]$. Với mảng $pref\_row[][]$ đã tính ở trên, ta có thể dễ dàng tính mảng $B[]$ trong $O(m)$.
- Ta đã chuyển bài toán từ hai chiều sang một chiều, lúc này ta sẽ giải như bài toán con mà ta đã giải ở trên.
- Phân tích độ phức tạp:
  - Tiền xử lý mảng $pref\_col[][]$ trong $O(m \times n)$.
  - 2 vòng *for* để xét hết các chỉ số i_1, i_2: $O(n^2)$.
  - Tính mảng $B[]$ trong $O(m)$.
  - Giải bài toán con trong $O(m \log_2 B_i)$.

$\rightarrow$ Độ phức tạp: $O\left(m \times n + n^2 \times m \log_2 B_i\right) = O\left(n^2 \times m \log_2 B_i\right)$.

Bài 11. Xâu ưng ý

Prerequisite: Suffix Array.
Từ dữ kiện đề bài và sử dụng tính chất của dãy tỉ số bằng nhau thì ta đưa đề bài thành tìm số k lớn nhất thỏa mãn điều kiện sau:

$$\frac{D_{A_i}}{D_{B_j}} = \frac{D_{A_{i+1}}}{D_{B_{j+1}}} = \frac{D_{A_{i+2}}}{D_{B_{j+2}}} = \dots \frac{D_{A_{i+k-1}}}{D_{B_{j+k-1}}}$$

Hay, tìm k lớn nhất mà có i,j thỏa mãn điều kiện sau:

$$\frac{D_{A_i}}{D_{B_j}} = \frac{D_{A_{i+1}}}{D_{B_{j+1}}}$$

$$\frac{D_{A_{i+1}}}{D_{B_{j+1}}} = \frac{D_{A_{i+2}}}{D_{B_{j+2}}}$$

$$\frac{D_{A_{i+k-2}}}{D_{B_{j+k-2}}} = \frac{D_{A_{i+k-1}}}{D_{B_{j+k-1}}}$$

hay:

$$\frac{D_{A_i}}{D_{A_{i+1}}} = \frac{D_{B_j}}{D_{B_{j+1}}}$$

$$\frac{D_{A_{i+1}}}{D_{A_{i+2}}} = \frac{D_{B_{j+1}}}{D_{B_{j+2}}}$$

$$\frac{D_{A_{i+k-2}}}{D_{A_{i+k-1}}} = \frac{D_{B_{j+k-2}}}{D_{B_{j+k-1}}}$$

Nếu đặt

$$u_i = \frac{D_{A_i}}{D_{A_{i+1}}}$$

$$v_j = \frac{D_{B_j}}{D_{B_{j+1}}}$$

thì bài toán sẽ trở về tìm k lớn nhất thỏa mãn các điều kiện sau:

$$u_i = v_j$$

$$u_{i+1} = v_{j+1}$$

$$u_{i+k-1} = v_{j+k-1}$$

Đến đây, bài toán trở về tìm mảng con chung dài nhất của hai mảng, chúng ta có thể sử dụng Suffix Array và để ứng dụng tìm LCP (Longest Common Prefix) vào bài toán trên.

Tài liệu

[1] Trịnh Quang Anh. Thuật toán KMP - Đếm số xâu con phân biệt trong một xâu - *https://vnoi.info/wiki/algo/string/kmp.md#đếm-số-xâu-con-phân-biệt-trong-một-xâu.*

[2] Ngô Nhật Quang. Trie - Xử lí truy vấn tìm XOR lớn nhất với giá trị được cho - *https://vnoi.info/wiki/algo/data-structures/trie.md#xử-lí-truy-vấn-tìm-xor-lớn-nhất-với-giá-trị-được-cho.*