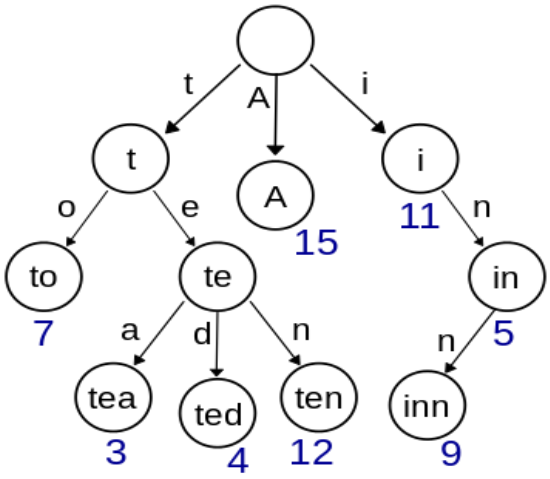
CHUYÊN ĐỀ

CÂY TRIE – CÂY TIỀN TỐ



MỤC LỤC

I. Mở đầu

1. Đặt vấn đề:

Việc quản lý, truy vấn trên các tập hợp xâu từ lâu đã là một đề tài được lựa chọn nhiều trong các kỳ thi lập trình thi đấu. Có rất nhiều các để giải quyết bài toán này bằng các cấu trúc dữ liệu mạnh mẽ liên quan đến các **cây nhị phân cân bằng** như: *Cây đỏ - đen (Red - black tree), Cây AVL (Adelson-Velsky Landis tree)*, … hoặc sử dụng kỹ thuật dùng **hàm băm** *(Hashing)*.

Tuy các **cây nhị phân cân bằng** có thời gian xử lý nhanh (**$\log_n$** – *với n là số lượng phần tử trong tập hợp*), và có nhiều ứng dụng mạnh mẽ, song việc cài đặt các loại cây này lại vô cùng khó khăn, phức tạp và nhiều khi cũng không thực sự cần thiết.

Còn đối với **hàm băm** *(Hashing)*, tuy thời gian thực thi rất nhanh, dễ cài đặt nhưng vẫn tồn tại khuyết điểm là độ chính xác không phải tuyệt đối. Điều này khiến cho **hàm băm** không được sử dụng nhiều ở các kỳ thi lập trình ACM – ICPC, khi thường có những test hiểm *(counter-test)* mà hàm băm không xử lý đúng được. Bên cạnh đó, hàm băm cũng có ứng dụng tương đối ít trong việc quản lý, truy vấn tập hợp xâu, khi khó, thậm chí nhiều khi là không xử lý các yếu tố liên quan đến tiền tố hay các yếu tố thứ tự từ điển.

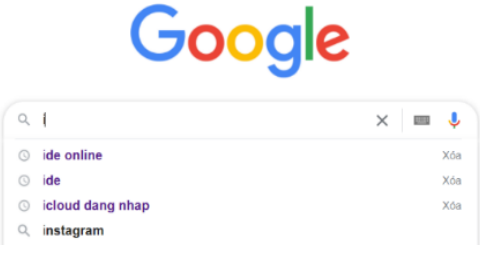
Thế nên, đối với việc quản lý, truy vấn trên tập hợp xâu, ta có thể sử dụng cây **Trie** như một giải pháp thay thế với những ưu điểm như tốc độ xử lý cao (**m** – với m là kích thước xâu của phần tử xét), dễ cài đặt và tiết kiệm bộ nhớ (bộ nhớ tối đa là tổng số lượng xâu – tuy nhiên thực tế nhỏ hơn nhiều vì sự trùng lặp tiền tố).

Vì cấu trúc dữ liệu **Trie** có dạng cây, nên ta có thể sự dụng kết hợp thêm nhiều thuật toán xử lý trên cây, làm hiệu năng của **Trie** tăng lên mạnh mẽ. Bởi thế mà ứng dụng của **Trie** là vô cùng lớn, nhất là khi được kết hợp với những kỹ thuật khác như quy hoạch động, tham lam, … **Trie** cũng có thể mở rộng trong việc quản lý được cả tập hợp số nguyên, tập hợp bit, …

Nói một cách khác, Cây **Trie** là một cấu trúc dữ liệu có thể được tìm kiếm ở tốc độ cao bằng cách thể hiện một tập hợp các chuỗi dưới dạng cấu trúc cây. Nó cũng được sử dụng như một nơi lưu trữ khóa-giá trị.

Một số ứng dụng thường gặp được xây dựng dựa trên cấu trúc cây Trie như:

- Công cụ tìm kiếm Google
- Định tuyến URL và địa chỉ IP
- Kiểm tra chính tả



- Morphology Parser
- Phát hiện thay đổi dữ liệu

2. Tóm tắt nội dung chuyên đề

Ở chuyên đề này, tôi sẽ nghiên cứu về các ứng dụng và cách cài đặt cây **Trie**. Bên cạnh đó, sẽ bổ sung thêm một số bài tập có lời giải cũng như một vài bài tập tự rèn luyện thêm.

II. Cơ sở lý thuyết:

1. Sơ lược về cây Trie - Cây tiền tố

1.1. Cấu trúc cây Trie

💻 Trong khoa học máy tính, **cây** là một cấu trúc dữ liệu được sử dụng rộng rãi gồm một tập hợp các nút được liên kết với nhau theo quan hệ cha-con. Và **Trie**, hay **cây tiền tố**, là một cấu trúc dữ liệu sử dụng cây có thứ tự, bắt nguồn từ từ "re**Trie**val" trong Tiếng Anh nghĩa là "thu hồi", chuyên sử dụng để giải quyết nhiều bài toán liên quan đến quản lý tập hợp, đặt biệt là liên quan đến các thao tác trên đoạn tiền tố.

💻 **Trie** là một cấu trúc dữ liệu sử dụng cây có thứ tự, dùng để lưu trữ một mảng liên kết của các giá trị (thường là các chữ cái trong một xâu ký tự). Không như cây tìm kiếm tìm kiếm nhị phân, mỗi nút trong cây không liên kết với một khóa trong mảng. Thay vào đó, mỗi nút liên kết với một xâu ký tự sao cho các xâu ký tự của tất cả các nút con của một nút đều có chung một tiền tố, chính là xâu ký tự của nút đó. Nút gốc tương ứng với xâu ký tự rỗng.

💻 **Ví dụ ta có bài toán sau:**

Giả sử bạn được đưa cho một cuốn từ điển với những từ sau:

algo

algea

also

tom

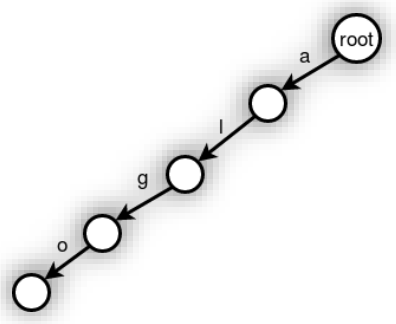
to

Yêu cầu : Bây giờ bạn hãy lưu trữ cuốn từ điển này trong bộ nhớ sao cho việc tìm thấy một từ đơn giản nhất.

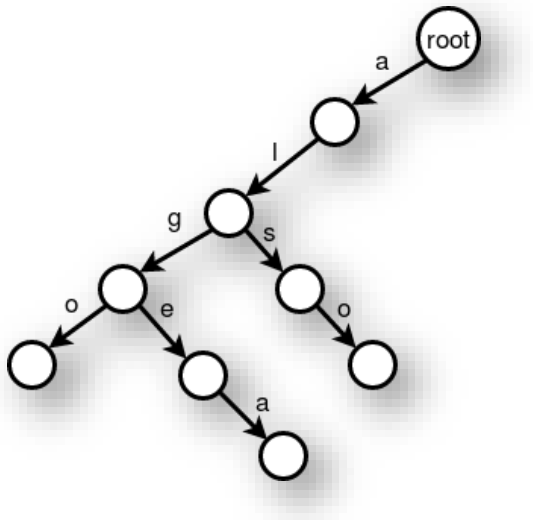
Một cách đơn giản là sắp xếp các từ được lưu trong từ điển giấy, sau đó chúng ta có thể tìm một từ bằng cách thực hiện tìm kiếm nhị phân. Một cách khác là sử dụng Cây tiền tố hay gọi tắt là Trie. Tiền tố có nghĩa là tạo một chuỗi mới với một vài ký tự từ đầu chuỗi. Ví dụ, tiền tố của ***blog*** là ***b***, ***bl***, ***blo*** và ***blog***.

Ban đầu chúng ta không có gì ngoài một nút gốc.

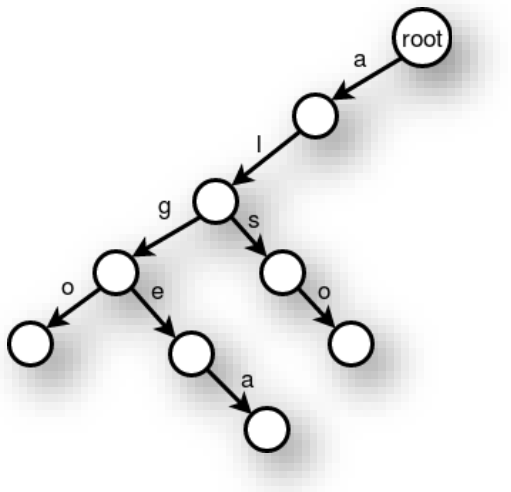
Bây giờ chúng ta sẽ thêm từ ***algo***. Xem cách từ được thêm vào như trong hình ảnh bên dưới. Từ nút gốc, chúng ta sẽ đưa ra một cạnh có tên là "***a***". Sau đó, tôi sẽ tạo một cạnh có tên "***l***" từ nút mới được tạo. Bằng cách này, tôi sẽ tạo cả hai cạnh "***g***" và "***o***". Lưu ý rằng chúng ta không đưa bất kỳ thông tin nào vào nút, chúng ta đang trích xuất cạnh từ nút trống.



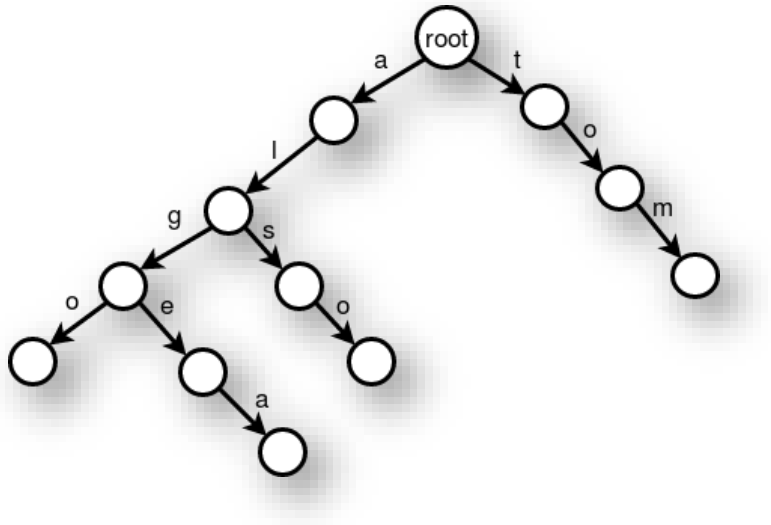
Bây giờ chúng ta muốn thêm từ ***algea***. Root cần một cạnh có tên là "***a***", nó đã ở đó rồi, không cần thêm mới nữa. Tương tự, có các cạnh từ ***a*** đến ***l*** và từ ***l*** đến ***g***. Vì vậy, "***alg***" đã được dùng thử, chúng ta sẽ chỉ thêm ***e*** và ***a***.



Tôi sẽ thêm từ cũng lần này. Tiền tố "***al***" từ gốc đã tồn tại, chỉ cần thêm "***so***".

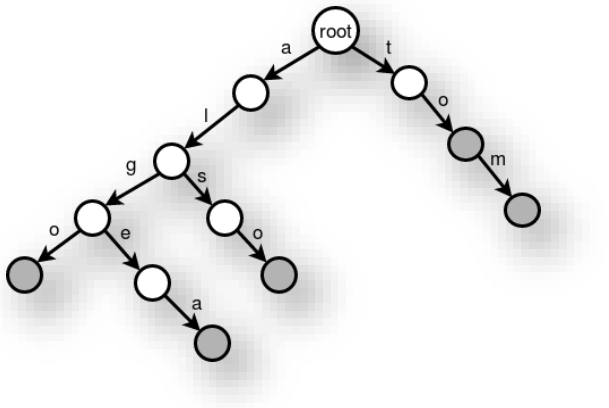


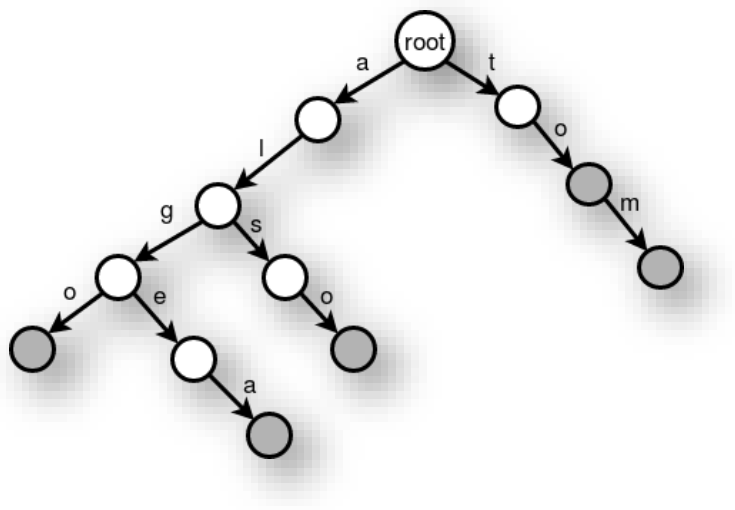
Hãy thêm "***tom***". Bây giờ chúng ta phải tạo một cạnh mới từ gốc vì không có tiền tố ***tom***



nào được thêm vào trước đó.

Bây giờ chúng ta phải làm thế nào để thêm từ "***to***" vào? Hoàn toàn là tiền tố của tom nên không cần thêm bất kỳ cạnh mới nào. Tất cả những gì chúng ta có thể làm là đặt một số dấu kết thúc vào các nút. Chúng ta đặt một dấu cuối trên các nút mà ít nhất một từ đã được hoàn thành, dấu cuối được biểu thị bằng màu xám trong hình. Đặt một dấu cuối cho tất cả các từ trước đó, cũng như từ mới "***to***", sẽ làm cho Trie như sau:


root
a
t
l
o
g
s
m
o
e
o
a



Việc đánh dấu như vậy có ích lợi gì? Giả sử bạn được hỏi liệu từ "***alice***" có trong từ điển hay không? Bạn tiếp tục cố gắng từ đầu. Đầu tiên hãy kiểm tra xem có cạnh nào có tên là ***a*** từ gốc không, sau đó kiểm tra xem có cạnh nào có tên là ***l*** từ ***a*** hay không. Sau đó, không thể tìm thấy cạnh có tên ***l*** đến ***i***, vì vậy bạn có thể nói rằng từ ***alice*** không tồn tại.

Nếu bạn tìm kiếm từ "***alg***", bạn sẽ tìm thấy tất cả các cạnh ***gốc-> a, a-> l và l-> g***, nhưng cuối cùng bạn sẽ không thể đi đến bất kỳ nút màu xám nào, vì vậy ***alg*** và không có trong từ điển. Nếu bạn tìm kiếm "***tom***", bạn sẽ thấy một nút màu xám như vậy từ này có trong từ điển.

1.2. Một số ưu điểm của cây Trie

- Cài đặt đơn giản, dễ nhớ
- Tiết kiệm bộ nhớ: Khi số lượng khóa lớn và các khóa có độ dài nhỏ, thông thường Trie tiết kiệm bộ nhớ hơn do các phần đầu giống nhau của các khoá chỉ được lưu 1 lần. Ưu điểm này có ứng dụng rất lớn, chẳng hạn trong từ điển.
- Thao tác tìm kiếm: O(m) với m là độ dài khóa. Với Binary search tree (cân bằng): là O(logN). Khi số lượng khóa cần tìm lớn và độ dài mỗi khóa tương đối nhỏ, logN xấp xỉ m, và như các bạn đã biết, để cài được Binary search tree cân bằng không phải là một việc đơn giản. Hơn nữa, các thao tác trên Trie rất đơn giản và thường chạy nhanh hơn trên thực tế.
- Dựa vào tính chất của cây Trie, có thể thực hiện một số liên quan đến thứ tự từ điển như sắp xếp, tìm một khóa có thứ tự từ điển nhỏ nhất và lớn hơn một khóa cho trước…; và một số thao tác liên quan đến tiền tố, hậu tố.

2. Cài đặt cây Trie – Cây tiền tố

Trie có 2 cách cài đặt khá phổ biến, đó là cài đặt Trie bằng mảng và bằng con trỏ. Trong nội dung bài viết này, Trong một số bài tập, tôi xin được trình bày bằng cả 2 cài đặt Trie, bằng con trỏ và bằng mảng.

Khi xét đến 1 nút trong cây Trie, chúng ta sẽ có 2 điều:

1. Một biến để giữ dấu kết thúc.

2. Các cạnh có tên a, b, c, …… .., x, y, z, ... có thể được hình thành từ mỗi nút. Chúng ta sẽ đặt một con trỏ cho mỗi ký tự. Một con trỏ sẽ kết nối một nút với một nút khác. Thêm một con trỏ có tên là *a* có nghĩa là có một cạnh có tên là *a* từ nút hiện tại. Lúc đầu tất cả các con trỏ sẽ là "*null*".

Đầu tiên chúng ta xây dựng cấu trúc của một nút và tạo nút

```
1  struct node {
2      bool endmark;
3      node* next[26 + 1];
4      node()
5      {
6          endmark = false;
7          for (int i = 0; i < 26; i++)
8              next[i] = NULL;
9      }
10 } * root;
11 int main()
12 {
13     root = new node();
14     return 0;
15 }
```

Next[] để chỉ mỗi phần tử của mảng trỏ đến một nút khác. Next[0] nếu nút mới được trỏ tới, tên của cạnh đó là "***a***"; Next [1] tên của cạnh là "***b***"; Next [25] tên cạnh là "***z***" . Ban đầu tất cả các con trỏ đều rỗng (NULL). Lưu ý rằng chúng ta đã tạo một phương thức khởi tạo "***node ()***" bên trong nút, để tạo các nút mới bất cứ khi nào bằng ***new node( )***. Nút gốc thực sự là một con trỏ, khi gọi ***root = new node ( ) ;***

```
void insert(char* str, int len)
{
    node* curr = root;
    for (int i = 0; i < len; i++) {
        int id = str[i] - 'a';
        if (curr->next[id] == NULL)
            curr->next[id] = new node();
        curr = curr->next[id];
    }
    curr->endmark = 1;
}
```

Xây dựng Hàm thêm một xâu vào Trie:

Đầu tiên chúng ta sẽ tạo một bản sao của root trong "***curr***". Vì tôi đang làm việc với con trỏ, nên việc tạo một cạnh mới từ gốc cũng giống như tạo một cạnh mới từ "***curr***". Bây giờ chúng ta chỉ làm việc với các ký tự ***a . . . z***, vì vậy hãy chuyển đổi các giá trị ASCII thành 0-25 bằng cách trừ các giá trị ASCII của '***a***'. Nó rất dễ chèn, chúng ta sẽ chỉ kiểm tra xem có bất kỳ cạnh nào của ký tự hiện tại từ nút hiện tại (***curr***) hay không? nếu không, chúng ta phải tạo một nút mới. Sau đó, chúng ta sẽ đến nút tiếp theo dọc theo cạnh đó. Tôi sẽ đánh dấu kết thúc đúng trong nút cuối cùng của tất cả.

```
bool search(char* str, int len)
{
    node* curr = root;
    for (int i = 0; i < len; i++) {
        int id = str[i] - 'a';
        if (curr->next[id] == NULL)
            return false;
        curr = curr->next[id];
    }
    return curr->endmark;
}
```

Xây dựng Hàm kiểm tra xem xâu s có trong Trie hay không?

Vấn đề phát sinh: Nhiều khi chạy các bài có dữ liệu lớn, chúng ta thường hay gặp sự cố giới hạn bộ nhớ. Chính vì vậy cách an toàn nhất là xóa các ô nhớ được sử dụng sau mỗi trường hợp, chúng ta không chỉ phải xóa root mà còn phải xóa từng nút. Chúng ta có thể viết một hàm đệ quy để làm điều đó.

Xây dựng Hàm xóa nút

```
1 void del(node* cur)
2 {
3     for (int i = 0; i < 26; i++)
4         if (cur->next[i])
5             del(cur->next[i]);
6     delete (cur);
7 }
```

Độ phức tạp: Để tìm từng từ, bạn phải chạy một vòng cho đến hết độ dài của từ đó, độ phức tạp của việc tìm kiếm là O (độ dài). Độ phức tạp của việc chèn mọi từ là như nhau. Cần bao nhiêu bộ nhớ phụ thuộc vào việc triển khai và các tiền tố của các từ khớp với nhau như thế nào.

III. BÀI TẬP CÓ HƯỚNG DẪN

1. Bài toán 1 : Phone Number

1.1. Đề bài

Cho một danh sách các số điện thoại, hãy xác định danh sách này có số điện thoại nào là phần trước của số khác hay không? Nếu không thì danh sách này được gọi là nhất quán. Giả sử một danh sách có chứa các số điện thoại sau:

- Số khẩn cấp: 911
- Số của Alice: 97625999
- Số của Bob: 91125426

Trong trường hợp này, ta không thể gọi cho Bob vì tổng đài sẽ kết nối bạn với đường dây khẩn cấp ngay khi bạn quay 3 số đầu trong số của Bob, vì vậy danh sách này là không nhất quán.

Dữ liệu vào

- Dòng đầu tiên chứa một số nguyên $1 \le t \le 40$ là số lượng bộ test.
- Mỗi bộ test sẽ bắt đầu với số lượng số điện thoại n được ghi trên một dòng, $1 \le n \le 10000$.
- Sau đó là n dòng, mỗi dòng ghi duy nhất 1 số điện thoại. Một số điện thoại là một dãy không quá 10 chữ số.

Dữ liệu ra

- Với mỗi bộ dữ liệu vào, in ra "**YES**" nếu danh sách nhất quán và "**NO**" trong trường hợp ngược lại.

Ví dụ

Phone.inp	Phone.out
2 3 911 97625999	NO YES

91125426 5 113 12340 123440 12345 98346	

1.2. Phân tích thuật toán

Cách tiếp cận 1 :

Bài toán có thể được giải quyết bằng cách tổ chức lưu trữ dữ liệu đầu vào dưới dạng một danh sách sau đó sắp xếp tăng dần theo thứ tự từ điển.

Dùng hàm ***find*** trong C++ để tìm tiền tố, kết quả sẽ được đánh dấu có hay không có?

(Code mẫu có trong file chương trình)

Cách tiếp cận 2 :

Bài toán có thể được giải quyết bằng cách tổ chức lưu trữ dữ liệu đầu vào dưới dạng một danh sách sau đó sắp xếp tăng dần theo thứ tự từ điển.

Dùng kĩ thuật tìm kiếm nhị phân để tìm tiền tố, kết quả sẽ được trả về có hay không có?

Cách tiếp cận 3 : Sử dụng cấu trúc cây Trie để giải quyết bài toán này.

Đầu tiên, chúng ta tạo một cấu trúc Trie với một biến 'boolean` và một ' array` các nút Trie có chiều dài '10`. Biến boolean sẽ cho biết một chuỗi có kết thúc trên nút đó hay không. Với kích thước mảng ‘10’ sẽ là đủ cho các chữ số liền kề tiếp của số vì chữ số khác nhau `(0-9)`.

Cách tốt nhất để lấy đầu vào của số `n` cho mỗi testcase là dưới dạng chuỗi. Bởi vì Trie xử lý với tiền tố của một chuỗi. Nếu chúng ta muốn nhận số `n` dưới dạng số nguyên, chúng ta vẫn có thể nhưng điều này không hợp lý khi sử dụng Trie.

Vì vậy, chúng ta chỉ cần lấy các số đầu vào `n` dưới dạng chuỗi và chèn vào Trie. Khi bộ Trie

được hình thành, chúng ta tạo một hàm `***isPrefix ()***` sẽ kiểm tra xem có một tiền tố số duy nhất của một số khác hay không trong Trie. Chức năng sẽ kiểm tra như thế nào? Hàm sẽ đi qua Trie và nếu có một nút duy nhất có giá trị biến boolean `***true***` nhưng có nhiều nút Trie hơn được dựng ra từ nút đó, điều này xác nhận rằng nó là một tiền tố.

Ví dụ, nếu Trie của chúng ta bao gồm hai con số '123` và ' 12345` , tại nút cho chữ số '3` , có một số kết thúc nhưng vẫn còn nút được dựng từ nó. Vì vậy, điều này xác nhận rằng tập dữ liệu không nhất quán.

```
// Một bộ ba với hai chuỗi được chèn, "123" và "12345"


        1 (sai)
         \
          2 (sai)
           \
            3 (true) <- kết thúc của chuỗi "123" nhưng không phải là nút lá!
             \
              4 (sai)
               \
                5 (true) <- kết thúc của chuỗi "12345" và nó là một nút lá
```

1.3. Chương trình mẫu và test chấm

```
#include<bits/stdc++.h>
using namespace std;
///Trie
struct Trie
{
    bool endmark; //boolean variable to mark end of number
    Trie *arr[10]; //10length array of Trie nodes for 0-9
    //constructor
    Trie()
    {
```

```
        endmark = 0;
        for(int i=0; i<=9; i++)
        {
            arr[i] = NULL;
        }
    }
} * root; //globally declaring variable of the struct
void insert(char s[])
{
    int n = 0;
    for( ; s[n]; n++) {} //size()
    Trie *curr = root;
    for(int i=0; i<n; i++)
    {
        int x = int(s[i]-'0');
        if(curr->arr[x]==NULL) curr->arr[x] = new Trie();
        curr = curr->arr[x];
    }
    curr->endmark = 1;
}
void del(Trie* node)
{
    for(int i=0; i<10; i++)
    {
        if(node->arr[i]!=NULL) //recursive del!
            del(node->arr[i]);
    }
```

```
    delete(node);
}
bool isPrefix(Trie *node)
{
    for(int i=0; i<=9; i++)
    {
        if(node->arr[i]!=NULL)
        {
            if(node->endmark) return 1;
            if(isPrefix(node->arr[i])) return 1;
        }
    }
    return 0;
}
int main()
{
    int t=0,tc=0;
    scanf(" %d", &tc);
    for(t=1; t<=tc; t++) //testcase
    {
        int i=0, j=0;
        root = new Trie(); //creating the Trie for this testcase
        int n; //no. of numbers
        scanf(" %d", &n);
        char s[10];
        while(n--)
        {
```

```
            scanf(" %s",&s[0]); //taking number as string
            insert(s); //inserting string into the Trie
        }

        // The function determining the YES or NO as answer!
        isPrefix(root)? cout<<"NO" : cout<<"YES";cout<<endl;
        del(root); //destroying the Trie each time after a testcase ends to
not hold memory anymore
    }
    return 0;
}
```

Link chương trình mẫu và test chấm :

https://drive.google.com/drive/folders/1PFKUjokdCpMw5_-kApsNC4XP7nJxG7wd?usp=sharing

2. Bài toán 2: Chuỗi ADN

2.1. Đề bài

Cho một tập hợp n mẫu DNA, trong đó mỗi mẫu là một chuỗi chứa các ký tự từ {A, C, G, T}, chúng ta đang cố gắng tìm một tập hợp con các mẫu trong tập hợp, trong đó độ dài của tiền tố chung dài nhất nhân với số lượng mẫu trong tập con đó là tối đa.

Để cụ thể, hãy để các mẫu là:

1. ACGT
2. ACGTGCGT
3. ACCGTGC
4. ACGCCGT

Nếu lấy tập con {ACGT} thì kết quả là 4 (4 * 1), nếu lấy {ACGT, ACGTGCGT, ACGCCGT} thì kết quả là 3 * 3 = 9 (vì ACG là tiền tố chung), nếu lấy {ACGT, ACGTGCGT, ACCGTGC, ACGCCGT} thì kết quả là 2 * 4 = 8.

Bây giờ nhiệm vụ của bạn là báo cáo kết quả tối đa mà chúng ta có thể nhận được từ các mẫu.

Dữ liệu vào

Dòng đầu là số nguyên T (≤ 10), biểu thị số lượng trường hợp thử nghiệm.

Mỗi trường hợp bắt đầu bằng một dòng chứa số nguyên n (1 ≤ n ≤ 50000) biểu thị số lượng mẫu DNA.

Mỗi dòng trong số n dòng tiếp theo chứa một chuỗi không rỗng có độ dài không lớn hơn 50. Và các chuỗi chứa các ký tự từ {A, C, G, T}.

Dữ liệu ra

Đối với mỗi trường hợp, in số trường hợp và kết quả tối đa có thể nhận được.

Ví dụ

ADN.INP	**ADN.OUT**
3 4 ACGT ACGTGCGT ACCGTGC ACGCCGT 3 CGCGCGCGCGCCCCGCCCGCGC CGCGCGCGCGCCCCGCCCGCAC CGCGCGCGCGCCCCGCCCGCTC 2 CGCGCCGCGCGCGCGCGC GGCGCCGCGCGCGCGCTC	Case 1: 9 Case 2: 66 Case 3: 20

2.2. Phân tích thuật toán

Vấn đề này rõ ràng được thực hiện với Trie, sử dụng Trie để lưu trữ tất cả các trình tự DNA, và chúng ta có một sự thay đổi nhỏ ở đây, biến đánh dấu trong trường hợp này không còn cần thiết, và được thay bằng một biến ***count***, đại diện cho số lượng chuỗi DNA kết thúc tại nút đó.

Để dễ hình dung hơn, chúng ta cùng xem ví dụ sau:

```
//A trie with two DNA strings inserted, "ACGACG" and "ACGTACG"

            A(count=2)
             \
              C(count=2)
               \
                G(count=2) //value = 3*2 = 6; because 03 length prefix is shared by 2 DNA strings
               / \
     (count=1)A   T(count=1)
             /     \
   (count=1)C       A(count=1)
           /         \
  (count=1)G          C(count=1)
                       \
                        G(count=1) //value = 7*1=7; 7 length prefix is shared by 1 DNA strings
```

2.3. Chương trình mẫu

```
#include<bits/stdc++.h>
using namespace std;
///Trie
struct Trie
{
    //bool endmark; //we dont need endmark of string for this problem! :)
    int count;
    Trie *arr[4]; //04 length array of Trie nodes for {A,C,G,T}
    //constructor
    Trie()
    {
        count = 0; //to count prefix occurences
```

```
        //endmark = 0;
        for(int i=0; i<=3; i++)
        {
            arr[i] = NULL;
        }
    }
} * root; //globally declaring variable of the struct
void insert(char s[])
{
    int n = 0;
    for( ; s[n]; n++) {} //n is size of the string

    Trie *curr = root;
    for(int i=0; i<n; i++)
    {
        int x = 0; //default for 'A'
        if(s[i]=='C') x=1;
        else if(s[i]=='G') x=2;
        else if(s[i]=='T') x=3;
        if(curr->arr[x]==NULL) curr->arr[x] = new Trie();
        curr = curr->arr[x];
        curr->count++;
    }
}
void del(Trie* node)
{
    for(int i=0; i<4; i++)
```

```
    {
        if(node->arr[i]!=NULL) //recursively deleting!
            del(node->arr[i]);
    }
    delete(node);
}
int compute(Trie *node, int level)
{
    int ret = 0;
    ret = (node->count * level);
    for(int i=0; i<4; i++)
    {
        if(node->arr[i]!=NULL)
        {
            ret = max(ret , compute(node->arr[i], level+1));
        }
    }
    return ret;
}
int main(void)
{
    int t=0,tc=0;
    scanf(" %d", &tc);
    for(t=1; t<=tc; t++) //testcase
    {
        int i=0, j=0;
        root = new Trie(); //creating the Trie for this testcase
```

```
        int n; //no. of DNA strings
        scanf(" %d", &n);
        char s[50];
        while(n--)
        {
            scanf(" %s",&s[0]); //taking DNA string
            insert(s); //inserting string into the Trie
        }
        // The function determining the maximum answer!!
        printf("Case %d: %d\n", t, compute(root, 0));
        del(root);
    }
    return 0;
}
```

Link chương trình mẫu và test chấm :

https://drive.google.com/drive/folders/1PFKUjokdCpMw5_-kApsNC4XP7nJxG7wd?usp=sharing

3. Bài toán 3: Tin mật

3.1. Đề bài (nguồn bài : SEC - SPOJ)

Bessie định dẫn đàn bò đi trốn. Để đảm bảo bí mật, đàn bò liên lạc với nhau bằng cách tin nhắn nhị phân.Từng là một nhân viên phản gián thông minh, John đã thu được M $(1 \le M \le 50{,}000)$ tin nhắn mật, tuy nhiên với tin nhắn i John chỉ thu được b_i $(1 \le b_i \le 10{,}000)$ bit đầu tiên.

John đã biên soạn ra 1 danh sách N $(1 \le N \le 50{,}000)$ các từ mã hóa mà đàn bò có khả năng đang sử dụng. Thật không may, John chỉ biết được c_j $(1 \le c_j \le 10{,}000)$ bit đầu tiên của từ mã hóa thứ j.

Với mỗi từ mã hóa j, John muốn biết số lượng tin nhắn mà John thu được có khả năng là từ mã hóa j này. Tức là với từ mã hóa j, có bao nhiêu tin nhắn thu được có phần đầu giống với từ mã hóa j này. Việc của bạn là phải tính số lượng này.

Tổng số lượng các bit trong dữ liệu đầu vào (tổng các b_i và c_j) không quá 500,000.

Dữ liệu vào

- Dòng đầu tiên chứa hai số nguyên *N* và *M*.
- Mỗi dòng trong số *M* dòng sau đó mô tả tin nhắn thứ *i* thu được, đầu tiên là b_i sau đó là b_i bit cách nhau bởi dấu cách, các bit có giá trị *0* hoặc *1*.
- Mỗi dòng trong số *N* dòng sau mô tả mã hóa thứ *j*, đầu tiên là c_i sau đó là c_i bit cách nhau bởi dấu cách, các bit có giá trị *0* hoặc *1*.

Dữ liệu ra

- In ra *M* dòng, mỗi dòng là một số nguyên cho biết số lượng tin nhắn có thể từ mã hóa thứ *j*.

Ví dụ

Dữ liệu	Kết quả
4 5 3 0 1 0 1 1 3 1 0 0 3 1 1 0 1 0 1 1 2 0 1 5 0 1 0 0 1 0 2 1 1	1 3 1 1 2

Giải thích

Có *4* tin nhắn và *5* từ mã hóa. Các tin nhắn thu được có phần đầu là *010, 1, 100* và *110*. Các từ mã hóa có phần đầu là *0, 1, 01, 01001,* và *11*.

0 chỉ có khả năng là *010* → *1* tin nhắn. *1* chỉ có khả năng là *1, 100*, hoặc *110* → *3* tin nhắn. *01*

chỉ có thể là *010* → *1* tin nhắn. *010010* chỉ có thể là *010* → *1* tin nhắn. *11* chỉ có thể là *1* hoặc *110* → *2* tin nhắn.

3.2. Phân tích thuật toán

- ***Cách tiếp cận thứ 1 :***

Ta sử dụng một vector để lưu trữ các xâu ban đầu. Đồng thời, ta cũng sử dụng quy hoạch động trên cây Trie, bằng một mảng quy hoạch động. Với mảng quy hoạch động f[u] cho biết số lượng xâu tối đa làm tiền tố cho xâu kết thúc tại đỉnh u.

Bằng cách này, ta có thể dễ dàng xây dựng công thức quy hoạch động như sau:

f[u]=f[u]+ f[v], với u là đỉnh lưu toàn bộ xâu S, v là đỉnh kề với u.

- ***Cách tiếp cận thứ 2 :*** Cài đặt theo con trỏ, ta xây dựng các hàm Insert và hàm Search, tại mỗi bước xây dựng cây, ta cập nhật biến res.

3.3. Chương trình minh họa:

```
#include<bits/stdc++.h>
using namespace std;
struct Trienode
{
    int leaf, cross;
    Trienode *child[2];
    Trienode()
    {
        child[0] = child[1] = NULL;
        cross = leaf = 0;
    }
};
Trienode *root = new Trienode();
char s[500005];
void insert(const int &len)
```

```
{
    Trienode *p = root; p->cross++;
    for (int i = 1; i <= len; i++)
    {
        int t = s[i]-'0';
        if (p->child[t] == NULL) p->child[t] = new Trienode();
        p = p->child[t];
        ++p->cross;
     }
    ++p->leaf;
}
int search(const int &len)
{
    int res = 0;
    Trienode *p = root;
    bool check = false;
    for (int i = 1; i <= len; i++)
    {
        int t = s[i] - '0';
        if (p->child[t] == NULL)
        {
            check = true; break;
        }
        p = p->child[t];
        res += p->leaf;
    }
```

```
        //cout <<res <<endl;
        if (!check) res += p->cross - p->leaf;
        return res;
    }
    int main()
    {
        //freopen("tinmat.inp", "r", stdin);
        //freopen("tinmat.out", "w", stdout);
        ios_base::sync_with_stdio(false);
        cin.tie(0);
        cout.tie(0);
        int n, m;
        cin >> n >> m;
        while (n--)
        {
            int k;
            cin >> k;
            for (int i = 1; i<= k; i++) cin >> s[i];
            insert(k);
        }
        while(m--)
        {
            int k;
            cin >> k;
            for (int i = 1; i <= k; i++) cin >> s[i];
            cout << search(k) <<"\n";
```

```
} }
```

Link chương trình minh họa và test chấm

https://drive.google.com/drive/folders/1PFKUjokdCpMw5_-kApsNC4XP7nJxG7wd?usp=sharing

Link nộp bài trực tiếp tại : Tin mật - VNOJ: VNOI Online Judge

4. Bài toán 4 : Chuỗi từ

4.1. Đề bài (nguồn bài : CHAIN2 - SPOJ)

Chuỗi từ có độ dài n là các chuỗi gồm các từ $w_1, w_2, w_3, \ldots w_{n-1}, w_n$ sao cho với mọi $1 \le n \le 10^5$, từ w_i là tiền tố của từ w_{i+1}.

Nhắc lại, từ u có độ dài k là tiền tố của từ v với độ dài l nếu $l > k$ và k ký tự đầu tiên của v trùng với u.

Cho tập hợp các từ $S = \{S_1, S_2, \ldots S_m\}$. Tìm chuỗi từ dài nhất có thể xây dựng được bằng cách dùng các từ trong tập hợp S (có thể không sử dụng hết các từ).

Dữ liệu vào

- Dòng đầu tiên chứa số nguyên m $(1 \le m \le 250000)$. Mỗi dòng trong số m dòng sau chứa một từ trong tập S.
- Biết rằng mỗi từ có độ dài không quá *250000* ký tự và tổng độ dài của các từ không vượt quá *250000* ký tự.

Dữ liệu ra

- In ra một số duy nhất là độ dài của chuỗi từ dài nhất xây dựng được từ các từ trong tập đã cho.

Ví dụ

Dữ liệu	Kết quả	Dữ liệu	Kết quả
3 a ab abc	3	5 a ab bc bcd	2

		add	

4.2. Phân tích thuật toán

- ***Cách tiếp cận thứ 1:***

Để giải bài này, ta cần sử dụng cấu trúc cây tiền tố **Trie**, lưu tất cả các xâu ban đầu. Đồng thời, ta cũng sử dụng quy hoạch động trên cây **Trie**, bằng một mảng quy hoạch động. Với mảng quy hoạch động *f[u]* cho biết số lượng xâu tối đa làm tiền tố cho xâu kết thúc tại đỉnh *u*.

Bằng cách này, ta có thể dễ dàng xây dựng công thức quy hoạch động như sau cho từng xâu *S:*

f[u]=max(f[u], f[v]+1), với *u* là đỉnh lưu toàn bộ xâu *S*, *v* là các đỉnh trên đường đi từ *1* đến *u* (nói cách khác, *v* là các đỉnh lưu các tiền tố của sâu *S*).

Lưu ý một vấn đề là ta chỉ cập nhật tại đỉnh u thôi.

```
#include <bits/stdc++.h>
using namespace std
int nodes[250007][26];
vector <int> adj[250007];
int f[250007]; int curNode = 1;

void initTrie(const string &s)
{
      int cur = 1;
      for (int i = 0; i <= (int)s.size()-1; ++i) {
      int nextNode = nodes[cur][s[i]-'a'];
      if (!nextNode) adj[cur].push_back(++curNode);
      if (!nextNode) nextNode = curNode;
```

```
        cur = nodes[cur][s[i]-'a'] = nextNode;
        }

return;
}

void DFS(const string &s)
{
        int cur = 1;
        int res = 0;
        for (int i = 0; i <= (int)s.size()-1; ++i) {
        int nextNode = nodes[cur][s[i]-'a']; res = max(res, f[cur]+1);
        cur = nextNode;
        if (i == (int)s.size()-1) f[nextNode] = res;
        }
return;
}
int main() {
int n; scanf("%d", &n);
vector <pair <int, string> > vc;
for (int i = 1; i <= n; ++i) {
        string s; cin >> s;
        initTrie(s);
        vc.push_back(make_pair((int)s.size(), s));
        }
sort(vc.begin(), vc.end());
```

```
for (int i = 0; i <= n-1; ++i) DFS(vc[i].second);
int ans = 0;
for (int i = 1; i <= 250000; ++i) ans = max(ans, f[i]);

printf("%d\n", ans); return 0;
}
```

- ***Cách tiếp cận thứ 2 :*** Cài đặt theo con trỏ, ta xây dựng các hàm Insert và hàm Search, tại mỗi bước xây dựng cây, và cập nhật biến res.

Chương trình minh họa:

```
#include<bits/stdc++.h>
using namespace std;
struct Trienode
{
    int leaf;
    Trienode *child[26];
    Trienode()
    {
        for (int i = 0; i < 26; i++)
            child[i] = NULL;
        leaf = 0;
    }
};
Trienode *root = new Trienode();
int res = 0;
void insert(const string &s)
{
```

```
        int cnt = 0;
        int n = (int )s.size();
        Trienode *p = root;
        for (int i = 0; i < n; i++)
        {
            int x = s[i] - 'a';
            if (p->child[x] == NULL) p->child[x] = new Trienode();
            cnt += p->leaf;
            p = p->child[x];
        }
        ++p->leaf; cnt += p->leaf; res = max(res, cnt);
    }
    int main()
    {
        #ifndef ONLINE_JUDGE
        freopen("chan2.inp", "r", stdin);
        freopen("chan2.out", "w", stdout);
        #else
        //online submission
        #endif
        ios_base::sync_with_stdio(false);
        cin.tie(0);
        cout.tie(0);
        int n; string s;
        cin >> n;
        for (int i = 1; i <= n; i++)
```

```
        {
            cin >> s;
            insert(s);
        }
        cout << res;
    }
```

4.3. Chương trình minh họa:

Link chương trình minh họa và test chấm

https://drive.google.com/drive/folders/1PFKUjokdCpMw5_-kApsNC4XP7nJxG7wd?usp=sharing

Link nộp bài trực tiếp tại : CHAIN2 - VNOI

5. Bài toán 5 : Trò chơi – STR2N

5.1. Đề bài

Khi học về xâu kí tự, để luyện tập thêm về nội dung này, An và Bình cùng nhau chơi mộ trò chơi với các xâu kí tự như sau:

- An tạo ra xâu kí tự ngẫu nhiên, sau đó, mỗi xâu ban đầu tạo ra một xâu mới bằng cách sao chép một đoạn đầu (hoặc toàn bộ) của xâu đó để tạo thêm được xâu.

- Với xâu mà An tạo ra và được đánh số theo thứ ngẫu nhiên từ 1 đến , Bình cần đưa ra một phương án để giải thích cách tạo xâu của An.

Yêu cầu: Cho xâu, hãy chia xâu thành nhóm, mỗi nhóm gồm hai xâu mà xâu này là đoạn đầu (hoặc toàn bộ) của xâu kia.

Dữ liệu vào

- Dòng đầu chứa số nguyên dương ;

- Tiếp theo là dòng, mỗi dòng là một xâu chỉ gồm các kí tự ‘a’ đến ‘z’.

Tổng số kí tự trong file không vượt quá 10^6

Dữ liệu ra

- Gồm dòng, mỗi dòng chứa hai số là chỉ số xâu gốc và chỉ số xâu được tạo ra.

Ví dụ

STR2N.INP	STR2N.OUT
2 ab adc a adce	1 3 4 2

5.2. Phân tích thuật toán

- Hướng giải quyết dễ thấy nhất của bài toán trên là xét tất cả các trường hợp nghiệm bằng đệ quy. Với mỗi nghiệm, kiểm tra xem các cặp xâu mà nghiệm đó đã chia có thỏa mãn với yêu cầu đề bài hay không bằng vòng for:

+ Nếu tất cả đều thỏa mãn, thì xuất kết quả và thoát khỏi đệ quy.

+ Nếu không thì TH nghiệm đang xét không phải là nghiệm của bài toán, bỏ TH nghiệm này và tiếp tục xét các TH khác.

- Tuy dễ nhận thấy và cài đặt cũng không quá phức tạp nhưng độ phức tạp của cách làm trên là cực kì lớn, do vậy cách làm này là không hợp lí, ta cần tìm một hướng giải quyết tốt hơn.

- Bài toán trên yêu: cầu hãy chia bộ xâu đã cho thành các nhóm, mỗi nhóm gồm hai xâu **mà xâu này là đoạn đầu (hoặc toàn bộ) của xâu kia**, dễ dàng nhận thấy rằng bài toán trên có tính tiền tố, vì vậy chúng ta nghĩ ngay đến cây Trie.

-Áp dụng cây Trie vào bài toán, chúng ta có thể làm như sau:

+ Nhồi tất cả các xâu vào cây Trie, sau đó lưu lại các cạnh của cây Trie bằng danh sách kề, lưu lại đỉnh kết thúc của tất cả các xâu, và ứng với mỗi đỉnh của cây Trie lưu lại chỉ số của xâu kết thúc tại đỉnh đó.

+ Vì bài toán đảm bảo luôn tồn tại kết quả nên, ta DFS từ gốc cây Trie, tại mỗi đỉnh ta push vào stack chỉ số của những xâu kết thúc tại đỉnh này. Đến khi DFS tới nút lá, ta xuất tất cả các chỉ số đã lưu trong stack theo từng cặp ta sẽ được kết quả.

5.3. Chương trình minh họa

```
#include <bits/stdc++.h>
using namespace std;
```

```
struct node {
    int nxt[26]; vector <int> id;

    node() {
        memset(nxt, 0, sizeof nxt);
    }
};
vector <node> tr;
vector <int> st;
vector <int> own;
void insert(const string &s, int i) {
    int pt = 0;
    for (char c : s) {
        int t = c - 'a';
        if (!tr[pt].nxt[t]) {
            tr[pt].nxt[t] = tr.size();
            tr.emplace_back();
        }
        pt = tr[pt].nxt[t];
    }
    tr[own[i] = pt].id.push_back(i);
}
void dfs(int pt) {
    while (tr[pt].id.size()) {
        st.push_back(tr[pt].id.back());
        tr[pt].id.pop_back();
    }
    for (int t = 0; t < 26; t++)
```

```
            if (tr[pt].nxt[t])
                dfs(tr[pt].nxt[t]);
        while (st.size() > 1 &&
        own[st.back()] == pt) {
            cout << st.back() << ' ';
            st.pop_back();
            cout << st.back() << '\n';
            st.pop_back();
        }
    }
    int main() {
        freopen("str2n.inp", "r", stdin);
        freopen("str2n.out", "w", stdout);
        cin.tie(0)->sync_with_stdio(0);
        int n; cin >> n;
        tr.emplace_back();
        own.resize(2 * n + 1);
        for (int i = 1; i <= 2 * n; i++) {
            string s; cin >> s;
            insert(s, i);
        }
        dfs(0); return 0;
    }
```

Độ phức tạp thuật toán:

O(n*maxlen(|a|)) (với maxlen(|a|) là độ dài xâu lớn nhất trong mảng)

Link chương trình minh họa và test chấm

https://drive.google.com/drive/folders/1PFKUjokdCpMw5_-kApsNC4XP7nJxG7wd?usp=sharing

6. Bài toán 6 : Phép XOR

6.1. Đề bài

Cho dãy n số nguyên không âm a_1, a_2, a_3, ...a_N . Gọi giá trị hòa hợp của một cặp hai số (a_i , a_j) với i<j được tính bằng a_i XOR a_j

Yêu cầu: Hãy tìm giá trị hòa hợp lớn nhất trong tất cả các cặp.

Dữ liệu vào

- Dòng đầu chứa số nguyên T (T<10) là số bộ dữ liệu;

- Tiếp theo là T dòng, mỗi dòng tương ứng với một bộ dữ liệu, số đầu tiên là số n (n<=10^5), tiếp theo là n số nguyên không âm a_1, a_2, a_3, ...a_N (0<=a_i<=10^9)

Dữ liệu ra

Gồm T dòng, mỗi dòng chứa một số là giá trị hòa hợp lớn nhất tìm được tương ứng với bộ dữ liệu vào.

Ví dụ

XOR.inp	XOR.out
2 3 1 2 3 3 2 4 6	3 6

6.2. Phân tích thuật toán

- Thuật toán đơn giản và dễ cài đặt nhất là chúng ta đi xét giá trị XOR tất cả các cặp (i,j<=n , i!=j) trong dãy số và lấy max.

- Mặc dù dễ cài đặt và dễ nhận thấy nhưng thuật toán trên có độ phức tạp rất lớn O(n^2), nên với các test có n lớn, chúng ta không thể áp dụng thuật toán trên.

- Tuy nhiên, nếu để ý chúng ta có thể nhận thấy rằng phép XOR có tính chất sau: 1^1=0, 1^0=1, 0^1=1, 0^0=0 và tính giao hoán: a^b=b^a.

- Dựa vào các tính chất trên ta có được hai nhận xét:

  + Khi xét đến một số a_i (i<=n) trong dãy số, để chọn một số aj có XOR với a_i đạt giá trị lớn nhất có thể, ta nên chọn những số a_j có bit thứ t nghịch với bit thứ t của a_i và cần phải xét từ bit quan trọng nhất (bit lớn nhất) về bit kém quan trọng nhất (bit nhỏ nhất).

  + Không quan trọng về mặt thứ tự của các số được xét.

- Từ việc tất cả các số phải xét từ bit quan trọng nhất (bit lớn nhất) về bit kém quan trọng nhất (bit nhỏ nhất) để lấy kết quả và việc không quan trọng thứ tự của các số được xét, ta thấy được bài toán trên có tính tiền tố và hoàn toàn có thể vận dụng hướng giải quyết bằng cây Trie vào bài toán này.

Ta có thể làm như sau:

-Với mỗi số a[i] (i<=n), chuyển a[i] sang hệ nhị phân, khởi tạo biến kết quả kq=0, xét các bit từ trái sang phải, với một bit thứ j của số a[i], ta tìm vị trí và đi đến nút lưu bit thứ j nghịch lại so với bit thứ j của a[i] trên cây Trie, chèn thêm bit 1 vào bên phải biến kq (nhằm tìm được kết quả lớn nhất), nếu không tìm thấy thì ta buộc phải chọn đi đến một bit tương đồng với bit thứ j của a[i] và chèn thêm bit 0 vào bên phải biến kq.

-Sau đó ta chèn các bit của số b[i] này theo thứ tự từ trái sang phải vào cây Trie và xét tiếp phần tử tiếp theo.

-Lấy max các biến kq tìm được trong suốt quả trình ta sẽ được kết quả cặp số có XOR lớn nhất.

Lưu ý: Độ dài của các phần tử trong mảng khi chuyển sang hệ nhị phân phải bằng nhau.

Ví dụ:

Dãy thập phân

8

1

3

Dãy nhị phân tương ứng

1000

0001

0011

6.3. Chương trình minh họa và test chấm

```
#include<bits/stdc++.h>
using namespace std;
const int N=1e5+5;
int Trie[N*32][2],a[N],m,max1,res,Nn,n,t;
void build(int s)
{
    int c=1;
    for (int i=res-1;i>=0;i--)
    {
        if (((s >> i)&1)==0)
        {
            if (Trie[c][0]==-1)
```

```
                {
                    ++Nn;
                    Trie[c][0]=Nn;
                }
                c=Trie[c][0];
            }
            else
            {
                if (Trie[c][1]==-1)
                {
                    ++Nn;
                    Trie[c][1]=Nn;
                }
                c=Trie[c][1];
            }
        }
    }
    int check(int s)
    {
        int c=1,point=0;
        for (int i=res-1;i>=0;i--)
        {
            if (((s >> i)&1)==0)
            {
                if (Trie[c][1]!=-1)
                {
                    c=Trie[c][1];
                    point=((point << 1)|1);
                }
                else {c=Trie[c][0];point=(point << 1);}
            }
            else
            {
                if (Trie[c][0]!=-1)
                {
                    c=Trie[c][0];
                    point=((point <<1 )|1);
                }
```

```
                else {c=Trie[c][1];point=(point<<1);}
            }
        }
        return point;
    }
    int main()
    {
        ios_base::sync_with_stdio(0);
        cin.tie(0);cout.tie(0);
        freopen("XOR.inp","r",stdin);
        freopen("XOR.out","w",stdout);
        cin>>t;
        for (int j=1;j<=t;j++)
        {
            memset(Trie,-1,sizeof(Trie));
            Nn=1;res=0;m=0;
            cin>>n;
            max1=0;
            for (int i=1;i<=n;i++)
            {
                cin>>a[i];max1=max(max1,a[i]);
            }
            while (max1!=0) {++res;max1/=2;}
            for (int i=1;i<=n;i++)
            {
                build(a[i]);
                if (i>1)
                {
                    m=max(m,check(a[i]));
                }
            }
            cout<<m<<"\n";
        }
    }
```

Độ phức tạp thuật toán:

O(n*maxlenbit(a)) (với maxlenbit(a) là độ dài lớn nhất của một số a[i] trong dãy khi chuyển sang hệ nhị phân).

Link Chương trình minh họa và test chấm :

https://drive.google.com/drive/folders/1PFKUjokdCpMw5_-kApsNC4XP7nJxG7wd?usp=sharing

7. Bài toán 7 : LRXOR

7.1. Đề bài

Cho dãy n số nguyên không âm a_1, a_2, a_3, ...a_N . Gọi giá trị hòa hợp của một đoạn từ vị trí L đến vị trí R là a_L XOR a_{L+1} XOR . . .XOR a_R

Yêu cầu: Hãy tìm giá trị hòa hợp lớn nhất trong tất cả các đoạn.

Dữ liệu vào

- Dòng đầu chứa số nguyên T (T<10) là số bộ dữ liệu;

- Tiếp theo là T dòng, mỗi dòng tương ứng với một bộ dữ liệu, số đầu tiên là số n ($n<=10^5$), tiếp theo là n số nguyên không âm a_1, a_2, a_3, ...a_N ($0<=a_i<=10^9$)

Dữ liệu ra

- Gồm T dòng, mỗi dòng chứa một số là giá trị hòa hợp lớn nhất tìm được tương ứng với bộ dữ liệu vào.

Ví dụ

LRXOR.inp	LRXOR.out
1 3 1 2 3	3

7.2. Phân tích thuật toán

- Thuật toán dễ cài đặt nhất là xét từng cặp (L,R) sau đó tính XOR trong từng đoạn và lấy giá trị MAX nhưng độ phức tạp của thuật toán này rất lớn ($O(n^3)$), vì vậy với những test có n lớn chúng ta không thể áp dụng thuật toán trên.

- Tuy nhiên, ta nhận thấy rằng phép toán XOR có tính chất: a^a=0, suy ra (a_l ^a_{l+1}^a_{l+1}^…^a_r)^(a_l ^a_{l+1}^a_{l+1}^…^a_k)= (a_k ^a_{k+1}^a_{k+1}^…^a_r) (với k<=r và ^ là kí hiệu của phép XOR)

- Dựa vào tính chất trên, ta lập một mảng mới là mảng XOR dồn các giá trị trong mảng đề cho và đưa bài toán về dạng tìm 2 số a_i , a_j (i!=j) trong mảng có XOR là lớn nhất và giải

quyết tương tự bài **XOR**. Ta có thể giải quyết bài toán này bằng cây Trie như sau:

+ Lập mảng b là XOR dồn của n phần tử trong mảng đầu tiên (tạm kí hiệu là mảng a), với vị trí i thì $b_i=(a_1 \wedge a_2 \wedge a_3 \wedge \ldots \wedge a_i)$ ➔ ta hoàn toàn có thể lập mảng trên với độ phức tạp O(n) với công thức $b_i=b_{i-1} \wedge a_i$.

+ Với mỗi số b[i] (i<=n), chuyển b[i] sang hệ nhị phân, khởi tạo biến kết quả kq=0, xét các bit từ trái sang phải, với một bit thứ j của số b[i], ta tìm vị trí và đi đến nút lưu bit thứ j nghịch lại so với bit thứ j của b[i] có trên cây Trie, chèn thêm bit 1 vào bên phải biến kq (nhằm tìm được kết quả lớn nhất), nếu không tìm thấy thì ta buộc phải chọn đi đến một bit tương đồng với bit thứ j của b[i] và chèn thêm bit 0 vào bên phải biến kq.

+ Sau đó ta chèn các bit của số b[i] này theo thứ tự từ trái sang phải vào cây Trie và xét tiếp phần tử tiếp theo.

+ Lấy max các biến kq tìm được trong suốt quá trình ta sẽ được kết quả đoạn L,R lớn nhất.

7.3. Chương trình minh họa và test chấm

```
#include <bits/stdc++.h>
using namespace std;
int T,n;
int f[100012];
struct Trie
{
    int maxnode=1;
    struct Node
    {
        int child[2];
        int size;
        Node()
        {
            child[0]=child[1]=0;
```

```
            size=0;
        }
    };
    vector <Node> node;
    Trie(int n)
    {
        node.resize(32*n+5);
    }
    void insert(int val)
    {
        int cur=1;
        for(int i=31; i>=0; i--)
        {
            int bit=((val>>i)&1);
            node[cur].size++;
            if (node[cur].child[bit]==0)
            {
                node[cur].child[bit]=++maxnode;
            }
            cur=node[cur].child[bit];
        }
    }
    int query(int val)
    {
        int res=0;
```

```
            int cur=1;
            for(int i=31; i>=0; i--)
            {
                int bit=((val>>i)&1);
                bit^=1;
                if (node[cur].child[bit]==0)
                {
                    cur=node[cur].child[bit^1];
                }
                else
                {
                    res+=(1<<i);
                    cur=node[cur].child[bit];
                }
            }
            return res;
        }
    };
    int main()
    {
        freopen("LRXOR.INP","r",stdin);
        freopen("LRXOR.OUT","w",stdout);
        cin>>T;
        while (T--)
        {
```

```
            cin>>n;
            Trie trie(n);
            int res=0;
            trie.insert(0);
            f[0]=0;
            for(int i=1; i<=n; i++)
            {
                int x;
                cin>>x;
                f[i]=f[i-1]^x;
                trie.insert(f[i]);
            }
            for(int i=1; i<=n; i++)
                res=max(res,trie.query(f[i-1]));
            cout<<res<<"\n";
        }
    }
```

Độ phức tạp thuật toán:

O(n*maxlenbit(a)) (với maxlenbit(a) là độ dài lớn nhất của một số a[i] trong dãy khi chuyển sang hệ nhị phân)

Link chương trình mẫu và test chấm

https://drive.google.com/drive/folders/1PFKUjokdCpMw5_-kApsNC4XP7nJxG7wd?usp=sharing

IV. BÀI TẬP TỰ LÀM

1. Bài toán 8 : Tìm kiếm mẫu - Pattern

1.1. Đề bài

Cho xâu ***t*** và tập các xâu mẫu $s_1, s_2, \ldots, s_n$. Hãy đếm số lần xuất hiện của các xâu mẫu trong xâu **t**.

Dữ liệu vào:

- Dòng đầu tiên chứa xâu ***t*** có độ dài không vượt quá 5×10^5.
- Dòng thứ hai ghi số nguyên ***n*** ($1 \le n \le 100$).
- Dòng thứ ***i*** trong ***n*** dòng tiếp theo chứa xâu s_i có độ dài không vượt quá 15. Các xâu s_i chỉ chứa các chữ cái Latin thường 'a'..'z' và hoa 'A'..'Z', các chữ số '0'..'9'. Xâu ***t*** cũng chứa các ký tự đó và thêm ký tự dấu cách. Chú ý rằng có thể có một số xâu mẫu giống nhau và mỗi vị trí nó xuất hiện trong xâu ***t*** ta chỉ đếm 1 lần.

Dữ liệu ra

- Ghi ra số lần xuất hiện của các xâu mẫu $s_1, s_2, \ldots, s_n$ trong xâu ***t***.

Ví dụ:

Pattern.inp	Pattern.out	Giải thích
shers 5 he she his he hers	3	Các xâu mẫu xuất hiện 3 lần trong xâu t là: he, she, hers.

1.2. Phân tích thuật toán

Chúng ta đã biết một số thuật toán tìm kiếm một mẫu trong một xâu như thuật toán RabinKarp (RK), thuật toán Knuth-Moris-Pratt (KMP). Trong bài này, chúng ta sẽ xét thuật toán tìm kiếm nhiều mẫu trong một xâu.

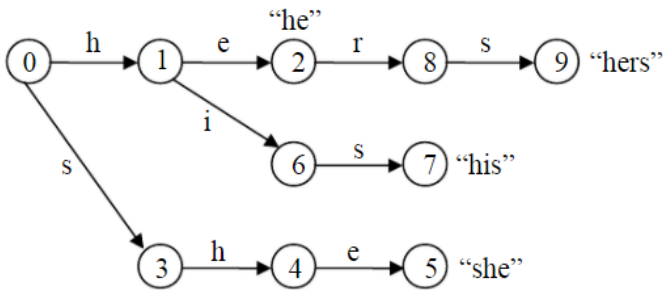
Rõ ràng một phương pháp đơn giản để giải bài toán trên là ta áp dụng thuật toán Rabin-Karp

hoặc thuật toán Knuth-Moris-Pratt với lần lượt từng xâu mẫu.Thuật toán theo cách này có độ phức tạp là O(n×m), ở đó n là số xâu mẫu và m là độ dài của xâu t.

Một thuật toán hiệu quả để giải quyết bài toán tìm kiếm nhiều mẫu dựa trên cây Trie

Đầu tiên chúng ta tạo một Trie biểu diễn tập các xâu mẫu:

- Mỗi cạnh của Trie có nhãn là một ký tự.
- Hai cạnh đi ra từ một nút có nhãn khác nhau.
- Nhãn của một nút v là ghép các nhãn của các cạnh trên đường đi từ nút gốc tới nút v và kí hiệu là L(v). Khi đó với mỗi xâu mẫu si, tồn tại một nút v với L(v) = si.
- Nhãn L(v) của một nút lá v bất kỳ là bằng một xâu mẫu si nào đó.

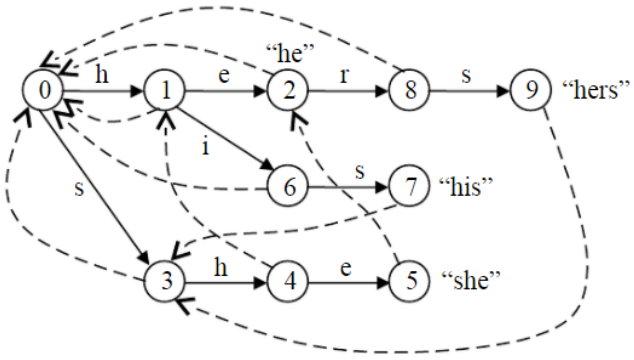


Ví dụ cây Trie với các xâu mẫu là "he", "she", "his", "hers" như sau:

Bây giờ giả sử ta cần tìm kiếm các mẫu trong xâu "shers". Ta bắt đầu từ nút 0 và đi theo cạnh 's' tới nút 3, tiếp theo từ nút 3 đi theo cạnh 'h' tới nút 4, rồi từ nút 4 đi theo cạnh 'e' tới nút 5.

Vì nút 5 là nút lá nên ta ghi nhận có một vị trí xuất hiện mẫu (chú ý đây là vị trí cuối của mẫu xuất hiện). Nhưng từ nút 5 lại không có cạnh 'r' đi ra, vì vậy ta cần phải quay lui lại phía trước xem có mẫu nào khác xuất hiện không. Việc quay lui lại là không phải từ đầu mà chỉ quay lui lại nút có nhãn là phần hậu tố dài nhất của nút 5, tức là nút 2. Nhưng nút 2 lại là nút lá nên ta lại ghi nhận thêm một vị trí nữa mẫu xuất hiện. Sau đó từ nút 2 ta lại đi theo cạnh 'r' tới nút 8, rồi từ nút 8 đi theo cạnh 's' tới nút 9. Nút 9 là nút lá nên ta lại ghi nhận thêm một vị trí nữa mẫu xuất hiện. Đến đây ta dừng việc tìm kiếm vì đã xét đến cuối xâu.

Như vậy với mỗi nút u ta cần biết nút v sao cho nhãn của nút v là phần hậu tố dài nhất của nhãn nút u và ta đặt link[u] = v. Các đường nét đứt trong Trie dưới đây là link của mỗi nút.



1.3. Chương trình minh họa và test chấm

Link chương trình minh họa:

https://drive.google.com/drive/folders/1PFKUjokdCpMw5_-kApsNC4XP7nJxG7wd?usp=sharing

2. Bài toán 9 : Ngôn ngữ - Language

2.1. Đề bài

Sự xuất hiện của dấu câu muộn hơn sự xuất hiện của từ, do đó các ngôn ngữ trước đây không có dấu câu. Bây giờ tất cả những gì bạn phải đối phó là một bài báo không có dấu chấm câu.

Một đoạn văn T gồm một số chữ cái viết thường. Một từ W cũng bao gồm một số chữ cái viết thường. Từ điển D là một tập hợp của một số từ. Chúng ta nói rằng một phần của bài báo T có thể được hiểu theo một từ điển D nhất định, có nghĩa là nếu bài báo T có thể được chia thành nhiều phần và mỗi phần là một từ trong từ điển D.

Ví dụ, từ điển D bao gồm các từ ***{'is', 'name', 'what', 'your'}***, thì bài viết ***'whatisyourname'*** có thể được hiểu trong từ điển D vì nó có thể được chia thành 4 từ: ***'what' , 'is', 'your', 'name'*** và mỗi từ thuộc từ điển D, và bài viết ***'whatisyouname'*** không thể hiểu được trong từ điển D, nhưng nó có thể được sử dụng trong từ điển ***D' = D + {'you'}*** . Tiền tố ***'whatis'*** trong bài viết này cũng có thể được hiểu theo từ điển D, và nó là tiền tố dài nhất có thể hiểu theo từ điển D.

Với một từ điển D, chương trình của bạn cần xác định xem một số đoạn văn bản có thể được hiểu trong từ điển D. Và đưa ra vị trí của tiền tố dài nhất có thể hiểu được theo từ điển D.

Dữ liệu vào

- Dòng đầu tiên là hai số nguyên dương n và m, nghĩa là có n từ trong từ điển D và m đoạn của bài báo cần được xử lý. Mỗi dòng trong số n dòng tiếp theo mô tả một từ và m dòng tiếp theo, mỗi dòng mô tả một đoạn văn bản.

Trong đó, 1 <= n, m <= 20, độ dài mỗi từ không quá 10 và độ dài mỗi bài không quá 10^6.

Dữ liệu ra

- Đối với mỗi đoạn văn bản bạn nhập, bạn cần xuất ra vị trí của tiền tố dài nhất mà đoạn văn bản này có thể hiểu được trong từ điển D.

Ví dụ

Language.inp	Language.out
4 3 is name what your whatisyourname whatisyouname whaisyourname	14 6 0

2.2. Phân tích thuật toán

Tóm tắt bài toán : Chúng ta nói rằng một phần của bài báo T có thể được hiểu theo một từ điển D nào đó, có nghĩa là nếu bài báo T có thể được chia thành nhiều phần và mỗi phần là một từ trong từ điển D.

Với một từ điển D, bạn cần xác định xem một số đoạn văn bản có thể được hiểu trong từ điển D. Và đưa ra vị trí của tiền tố dài nhất có thể hiểu được theo từ điển D.

Đến đây chúng ta thấy thuật toán đặc trưng để giải quyết vấn đề này là Cây Trie.

2.3. Chương trình minh họa và test chấm

https://drive.google.com/drive/folders/1PFKUjokdCpMw5_-

kApsNC4XP7nJxG7wd?usp=sharing

3. Bài toán 10 : Giải mã - Codes

3.1. Đề bài

Bảng mã của một tập hợp các ký hiệu được cho là có thể giải mã ngay lập tức nếu không có mã nào cho một ký hiệu là tiền tố của mã cho một ký hiệu khác. Chúng ta sẽ giả định cho vấn đề này rằng tất cả các mã đều ở dạng nhị phân, không có hai mã nào trong một bộ mã giống nhau, mỗi mã có ít nhất một bit và không nhiều hơn mười bit và mỗi bộ có ít nhất hai mã và không quá tám.

Ví dụ: Giả sử một bảng chữ cái có các ký hiệu {A, B, C, D}

Với các chuỗi Bit tương ứng :

A: 01 B: 10 C: 0010 D: 0000 → kết quả là **YES**

A: 01 B: 10 C: 010 D: 0000 → kết quả là **NO**

(Vì A là tiền tố của C)

Dữ liệu vào

- Đầu vào một loạt các nhóm bản ghi. Mỗi bản ghi trong một nhóm chứa một tập hợp các số 0 và số 1 đại diện cho mã nhị phân cho một ký hiệu khác nhau, mỗi nhóm được kết thúc bởi số 9. Mỗi nhóm độc lập với các nhóm khác; các mã trong một nhóm không liên quan đến các mã trong bất kỳ nhóm nào khác (nghĩa là mỗi nhóm sẽ được xử lý độc lập).

Dữ liệu ra

- Đối với mỗi nhóm, hãy in kết quả là **YES** nếu các mã trong nhóm đó có thể giải mã được và in ra **NO** nếu không.

Ví dụ

Codes.in	Codes.out
01 10 0010 0000	YES NO

9 01 10 010 0000 9	

3.2. Phân tích thuật toán

Tóm tắt bài toán : Cho trước một số chuỗi chữ số, hãy xác định xem chuỗi một chữ số có phải là tiền tố của một chuỗi khác hay không.

Chuỗi số chỉ chứa 0, 1. Nhớ độ dài của mỗi chuỗi số là d thì 1 <= d <= 10. Mỗi nhóm dữ liệu có ít nhất 2 chuỗi chữ số và nhiều nhất 8 chuỗi chữ số.

Về thuật toán, bài toán này có thể giải theo thuật toán duyệt cơ bản, sử dụng 2 vòng for để duyệt qua các đoạn [i,j] sau đó so sánh kết quả với bảng mã code.

3.3. Chương trình minh họa và test chấm

https://drive.google.com/drive/folders/1PFKUjokdCpMw5_-kApsNC4XP7nJxG7wd?usp=sharing

4. Bài toán 11 : Phần thưởng - BONUS (Bài 4 - Đề thi HSG Quốc gia 2021)

4.1. Đề bài

Nam vừa đoạt giải quán quân trong một kỳ thi lập trình danh giá. Ban tổ chức trao thưởng thông qua một trò chơi như sau. Có n thẻ xếp trên một hàng, được đánh số thứ tự từ 1 đến n từ trái sang phải, trên thẻ thứ i ($1 \le i \le n$) có hai thông tin:

- Nhãn ci là một xâu kí tự gồm các kí tự chữ cái la tinh in hoa (từ A đến Z);
- Số may mắn pi, là một số nguyên dương.

Ban tổ chức cũng công bố m cặp số may mắn và cho phép Nam thực hiện một hoặc nhiều bước chọn các thẻ. Ban đầu, Nam chọn một thẻ bất kì, điểm nhận được là số may mắn trên thẻ đó. Mỗi bước tiếp theo, Nam thực hiện theo một trong hai cách sau:

- Trong các thẻ ở phía bên phải thẻ hiện tại, chọn một thẻ có nhãn là một xâu mà có thể

nhận được bằng cách xóa từ xâu là nhãn của thẻ hiện tại một số kí tự cuối cùng (ít nhất một kí tự). Số điểm nhận được thêm bằng số may mắn trên thẻ mới chọn;

- Trong các thẻ ở phía bên phải thẻ hiện tại, chọn một thẻ mà số may mắn trên thẻ này và số may mắn trên thẻ hiện tại là một cặp số may mắn. Cụ thể, nếu số may mắn trên thẻ hiện tại là pi và số may mắn trên thẻ mới chọn là pj thì cặp số (pi,pj) hoặc (pj,pi) phải là một trong các cặp số may mắn mà Ban tổ chức đã công bố. Số điểm nhận được thêm bằng số may mắn trên thẻ mới chọn.

Sau một số bước chọn, nếu Nam không chọn được nữa thì số tiền thưởng bằng tổng điểm tích lũy được.

Yêu cầu: Hãy giúp Nam tìm cách chọn các thẻ để đạt được tổng số tiền thưởng là lớn nhất.

Dữ liệu vào

- Dòng thứ nhất chứa số nguyên dương n và số nguyên không âm m.
- Tiếp theo là n cặp dòng mô tả thông tin trên thẻ. Cặp dòng thứ i ($1 \le i \le n$) có dòng thứ nhất chứa nhãn và dòng thứ hai chứa số may mắn của thẻ thứ i.
- Mỗi dòng trong số m dòng cuối cùng chứa hai số nguyên dương mô tả một cặp số may mắn.

Các thẻ có thể có nhãn giống nhau và tổng số các kí tự của các nhãn trên các thẻ không vượt quá 10^6. Các số may mắn có giá trị không vượt quá 10^5. Các số trên cùng một dòng cách nhau bởi dấu cách.

Dữ liệu ra

- Ghi ra một số nguyên duy nhất là tổng tiền thưởng lớn nhất chọn được.

Ràng buộc

- *Có 40% số test tương ứng với 40% số điểm của bài thỏa mãn: $n \le 100$, nhãn trên các thẻ có độ dài không vượt quá 100 và $m \le 100$;*
- *40% số test khác ứng với 40% số điểm của bài thảo mãn: $n \le 3 \times 10^5$ và $m=0$.*
- *20% số test còn lại ứng với 20% số điểm của bài thỏa mãn: $n \le 3 \times 10^5$ và $m \le 3 \times 10^5$.*

4.2. Phân tích thuật toán

Tóm tắt đề : Cho n thẻ xếp trên một hàng, thẻ thứ i có nhãn c_i (là một xâu kí tự) và giá trị p_i.

Thực hiện các bước sau:

- Chọn một thẻ bất kì.
- Giả sử thẻ vừa được chọn là i, có thể tiếp tục chọn thẻ j (j > i) thỏa mãn: c_j là tiền tố của c_i hoặc (p_i, p_j) là một cặp số may mắn. Lặp lại như vậy cho đến khi không tìm được j thỏa mãn.
- Tổng điểm nhận được là tổng các p_i trên các thẻ được chọn.

Yêu cầu : Tìm cách chọn các thẻ sao cho tổng điểm nhận được là lớn nhất.

- ***Subtask 1: n<=100, m<=100, \|ci\|<=100.***

Hướng giải quyết: Quy hoạch động

+ dp(i) = điểm thưởng lớn nhất tính tới thẻ thứ i

+ dp(i) = pi + max(0, max{dp(j)}) (với mọi j<i)

+ Duyệt tất cả các thẻ j (j<i), kiểm tra xâu c_i có phải tiền tố của c_j hoặc cặp {p_i,p_j} có phải là cặp số may mắn hay không.

*Độ phức tạpO(n2*max(|c|));*

- ***Subtask 2: m=0.***

- m=0 => Không có các cặp số đẹp. Mặc dù vậy không thể duyệt qua tất cả các j(j<i) và kiểm tra tiền tố trong O(|c|) (vì giới hạn của n và |c| là lớn).

- Hướng giải quyết: Sử dụng cây tiền tố (Trie).

+ Dùng cây Trie để lưu lại xâu tiền tố của xâu ci, với mỗi đỉnh của cây Trie lưu thêm giá trị lớn nhất trong các thẻ có tiền tố là xâu từ đầu tính đến nút đó.

+ Xét lần lượt tất cả các thẻ i (i<=n), với mỗi thẻ i ta kiểm tra giá trị lớn nhất trong tất cả các xâu c_j (j<i) nhận xâu c_i là tiền tố. Cập nhật xâu c_i với giá trị (w+p_i) (w là giá trị lớn nhất vừa tìm được) vào cây Trie.

*Độ phức tạp O(n*max|c|).*

- ***Subtask 3: n<=3e5, m<=3e5.***

Hướng giải quyết: Sử dụng cây tiền tố (Trie), quy hoạch động.

+ Xét lần lượt tất cả các thẻ I (i<=n), với mỗi thẻ i ta tìm w2 là giá trị lớn nhất trong tất cả các xâu c_j (j<i) nhận xâu c_i là tiền tố. Tìm w1 là giá trị lớn nhất trong các thẻ là cặp số may mắn

với thẻ i. Cập nhật xâu c_i với giá trị $(w+p_i)$ $(w=max(w1,s2))$ vào cây Trie.

+ Để tìm nhanh s2 ta sử dụng mảng f[i] để lưu giá trị lớn nhất của các thẻ có số may mắn là i. Và liên kết các cặp số may mắn bằng vector.

4.3. Chương trình minh họa và test chấm

https://drive.google.com/drive/folders/1PFKUjokdCpMw5_-kApsNC4XP7nJxG7wd?usp=sharing

Test chấm : Link Chấm trực tiếp tại trang https://oj.vnoi.info/problem/voi21_bonus

V. Tài liệu tham khảo:

[1] VNOI.INFO – TRIE

[2] Tutorial on Trie and example problems

[3] https://leduythuccs.github.io/2018-05-23-c-u-tr-c-d-li-u-trie

---------- *Xin cảm ơn !* ----------