**TRIES – Cây từ điển**

# A.Code cơ bản

*//Cấu trúc một nút*

struct Tnode{

int stop; // Nút cuối một từ

int cnt; // Số lượng từ đi qua nút đó

int P; // Con trỏ đến đỉnh cha

int child[26]; // Con trỏ đến đỉnh con (26 chữ cái) - tùy đề

};

// *Khai báo cây (không dùng cấp phát động)*

Tnode Trie[maxT - sl chữ cái];

int nT=0; //đếm

int root;

// *Thêm một nút mới vào cây*

int AddNode() {

nT++;

for(int i=0;i<26;i++) Trie[nT].child[i]=0;

Trie[nT].P=0;

Trie[nT].cnt=Trie[nT].stop=0;

return nT;

}

// *Thêm một từ vào cây (từ input)*

void AddWord(char\* s) {

int r=root;

for(int i=0;s[i]!=0;i++) {

int k=s[i]-'a'; int l;

if (Trie[r].child[k]==0) {

l=AddNode();

Trie[r].child[k]=l;

Trie[l].P=r;

}

r=Trie[r].child[k];

}

Trie[r].stop++;

Trie[r].cnt++;

while (r!=root) {

r=Trie[r].P;

Trie[r].cnt++;

}

}

// *Đếm xem có bao nhiêu từ bằng s*

int CountWord(char\* s) {

int r=root;

for(int i=0;s[i]!=0;i++) {

int k=s[i]-'a';

if (Trie[r].child[k]==0) return 0;

r=Trie[r].child[k];

}

return Trie[r].stop;

}

// *Đếm xem có bao nhiêu từ có tiền tố bằng s*

int CountPrefix(char\* s) {

int r=root;

for(int i=0;s[i]!=0;i++) {

int k=s[i]-'a';

if (Trie[r].child[k]==0) return 0;

r=Trie[r].child[k];

}

return Trie[r].cnt;

}

// *Đếm xem có bao nhiêu từ nhỏ hơn hoặc bằng s*

int CountLE(char\* s) {

int r=root;

int ret=0;

for(int i=0;s[i]!=0;i++) {

int k=s[i]-'a';

for(int l=0;l<k;l++)

ret+=Trie[Trie[r].child[l]].cnt;

if (Trie[r].child[k]==0) return ret;

r=Trie[r].child[k];

}

ret+=Trie[r].stop;

return ret;

}

# B. Ứng dung quản lý tập hợp

**Bài toán:** Cho đa tập (muliset) với các phần tử là số nguyên dương trong phạm vi [1,109]. Thực hiện các thao tác:

1. Thêm một phần tử vào
2. Xóa phần tử khỏi
3. Tìm phần tử bé thứ trong (trả về nếu không tồn tại)
4. Đếm xem có bao nhiêu phần tử bé hơn
5. Tính tổng các phần tử bé hơn

// Cấu trúc nút

struct Tnode{

int stop;

int cnt;

int sum;

int P;

int child[2]; //bit

};

Tnode Trie[maxT];

int nT=0;

int root=0;

// Hàm thêm một nút trong Trie

int AddNode() {

++nT;

Trie[nT].P=Trie[nT].child[0]=Trie[nT].child[1]=0;

Trie[nT].stop=Trie[nT].cnt=Trie[r].sum=0;

return nT;

}

## 1. Thêm phần tử

void AddNum(int x) {

int r=root;

for(int k=31;k>=0;--k) {

int i=(x >> k) & 1;

if (Trie[r].child[i]==0) {

int l=AddNode();

Trie[r].child[i]=l;

Trie[l].P=r;

}

r=Trie[r].child[i];

}

// Them x vao S

++Trie[r].stop;

++Trie[r].cnt;

Trie[r].sum += x;

while (r!=root) {

r=Trie[r].P;

++Trie[r].cnt;

Trie[r].sum += x;

}

}

## 2. Xóa phần tử

void DelNum(int x) {

int r=root;

for(int k=31;k>=0;--k) {

int i=(x >> k) & 1;

if (Trie[r].child[i]==0) return; // x khong thuoc S

r=Trie[r].child[i];

}

if (Trie[r].stop==0) return; // x không thuộc S do bị xóa từ trc

// Bo phan tu thuoc S

--Trie[r].stop;

--Trie[r].cnt;

Trie[r] -= x;

while (r!=root) {

r=Trie[r].P;

--Trie[r].cnt;

Trie[r].sum -= x;

}

}

## 3. Tìm phần tủ đứng thứ p

int P\_TH(int p) {

int r=root;

if (Trie[r].cnt<p) return INF; // Khong ton tai phan tu thu p

int x=0;

for(int k=31;k>=0;--k) {

int i=Trie[r].child[0]; // i la cay con trai

if (p>Trie[i].cnt) {

x |= (1<<k);

p -= Trie[i].cnt;

r=Trie[r].child[1];

} else r=Trie[r].child[0];

}

return x;

}

## 4. Số phần tử nhỏ hơn x

int CountLT(int x) {

int r=root;

int ret=0;

for(int k=31;k>=0;--k) {

int i=(x >> k) & 1;

if (i) ret += Trie[Trie[r].child[0]].cnt;

if (Trie[r].child[i]==0) return ret;

r=Trie[r].child[i];

}

return ret;

}

## 5. Tổng các phần tử nhỏ hơn x

int SumLT(int x) {

int r=root;

int ret=0;

for(int k=31;k>=0;--k) {

int i=(x >> k) & 1;

if (i) ret += Trie[Trie[r].child[0]].sum;

if (Trie[r].child[i]==0) return ret;

r=Trie[r].child[i];

}

return ret;

}

**Ghi chú:** Nút chỉ số 0 dùng làm nút rỗng, nên khi khới tạo Trie rỗng cần lệnh:

**root=AddNode();**

# 1. Số lượng [NPREFIX.\*]

Cho tập gồm xâu và truy vấn. Với mỗi truy vấn là một xâu , ta cần xác định số lượng xâu trong tập nhận là tiền tố.

Ta nói là tiền tố của xâu nếu như: và

Hãy giúp Mr X lập trình giải bài toán trên.

## Input:

* Dòng đầu chứa ba số nguyên dương .
* dòng tiếp, dòng thứ chứa xâu .
* Dòng tiếp theo ghi số nguyên dương
* dòng tiếp, mỗi dòng chứa một xâu thể hiện truy vấn

Output: dòng tương ứng là kết quả từng truy vấn.

## Example:

|  |  |
| --- | --- |
| **Input** | **Output** |
| **6**  **abc**  **abdh**  **acfi**  **bgjab**  **abe**  **bcf**  **5**  **a**  **ab**  **abc**  **b**  **bj** | **4**  **3**  **1**  **2**  **0** |

Các xâu chỉ gồm các chữ cái in thường trong tập “abcdefghij”. Tổng ký tự trong tất cả các xâu không quá 106.1.

# 2. Quản lý tập hợp [ORDSET.\*]

Bạn cần quản lý một tập hợp động S hỗ trợ hai thao tác cơ bản:

* INSERT(S,x): Nếu x không thuộc S, thêm x vào S
* DELETE(S,x): Nếu x thuộc S, xóa x khỏi S

và hai loại truy vấn:

* K-TH(S): Trả về số bé thứ k của S
* COUNT(S,x): Đếm số lượng số thuộc S bé hơn x

## Input:

* Dòng đầu tiên ghi Q ( - số thao tác
* Q dòng sau, đầu mỗi dòng chứa ký tự I, D, K hoặc C cho biết thao tác tương ứng là INSERT, DELETE, K-TH hay COUNT. Tiếp theo là một khoảng trắng và một số nguyên là tham số cho thao tác đó.

Nếu tham số là x, dữ liệu đảm bảo |x|≤109. Nếu tham số là chỉ số k dữ liệu đảm bảo 1≤k≤109

Output: Với mỗi truy vấn in kết quả trên một dòng. Với truy vấn K-TH, nếu k lớn hơn số phần tử của S, in ra '**invalid'**

## Example:

|  |  |
| --- | --- |
| **Input** | **Output** |
| **8**  **I -1**  **I -1**  **I 2**  **C 0**  **K 2**  **D -1**  **K 1**  **K 2** | **1**  **2**  **2**  **invalid** |

# 3. Tách xâu [SEPARATE.\*]

Có một xâu ký tự cần phải chia ra thành các đoạn con (gồm các ký tự liên tiếp của xâu) sao cho mỗi đoạn con này là một từ trong tập danh sách các từ cho trước.

*Yêu cầu:* Viết chương trình đếm số cách chia khác nhau có thể có. Hai cách chia được gọi là khác nhau nếu có ít nhất một vị trí cắt khác nhau. Vì con số này có thể rất lớn nên bạn chỉ cần in phần dư của nó khi chia cho 1337377.

## Input:

* Dòng 1: Chứa xâu ký tự cần chia có độ dài không quá ký tự.
* Dòng 2: Chứa số nguyên dương - số từ có trong danh sách
* Dòng 3...: Mỗi dòng chứa một từ trong danh sách. Tất cả các từ có độ dài không vượt quá 100 và chỉ chứa các chữ cái tiếng Anh in thường.

Output: Một số nguyên duy nhất - kết quả tìm được.

## Example:

|  |  |
| --- | --- |
| **Input** | **Output** |
| **abcd**  **4**  **a**  **b**  **cd**  **ab** | **2** |

# 4. Mật khẩu [PASS.\*]

Người quản trị đang cố gắng xây dựng hệ thống bảo vệ dữ liệu cho công ty. Họ biên soạn một cuốn từ điển gồm *M* (1 ≤ *M* ≤ 300) từ khác nhau, mỗi từ chỉ chứa các chữ cái Latin in thường và có độ dài không vượt quá 300. Mật khẩu được tạo ra như bằng cách móc nối các từ trong từ điển, một từ có thể xuất hiện nhiều lần. Để giữ bí mật, người dùng có thể thêm các chữ cái in thường vào trước, sau hoặc giữa các từ trong từ điển, tạo thành mật khẩu *P* là từ có chiều dài *N* (5 ≤ *N* ≤ 300.000).

Bạn hãy viết chương trình đọc vào từ *P* có chiều dài *N*, xác định số lượng tối thiểu các chữ cái cần loại bỏ khỏi *P* để có được một từ mới là từ ghép của một số từ trong từ điển.

Ví dụ với từ “a**f**bach**tdspy**a”, cần loại bỏ các chữ cái ở vị trí in đậm để nhận được từ “abacha” là ghép của các từ trong từ điển: “a”, “bach”, “a”.

## Input:

* Dòng 1: Ghi hai số nguyên *N* và *M*.
* Dòng 2: Ghi từ *P*.
* Dòng 3...M+2: Mỗi dòng ghi một từ trong từ điển.

Output: Một số nguyên - số lượng chữ cái tối thiểu cần xóa

## Example:

|  |  |
| --- | --- |
| **Input** | **Output** |
| **12 5**  **afbachtdspya**  **aba**  **a**  **bach**  **dsy**  **zero** | **6** |

# 5. Xâu tiền tố [PREFIX.\*]

Một xâu được gọi là xâu tiền tố của một xâu khác nếu nó xuất hiện ở vị trí đầu tiên của xâu này. Ví dụ xâu 'ab' là tiền tố của xâu 'abcd'; 'aa' là tiền tố của 'aa'

*Yêu cầu:* Cho xâu ký tự, hãy đếm số cặp xâu mà xâu này là tiền tố của xâu còn lại

## Input:

* Dòng đầu tiên ghi số nguyên dương
* dòng tiếp theo, mỗi dòng ghi một xâu ký tự chỉ gồm các chữ cái tiếng Anh in thường với độ dài của mỗi xâu không vượt quá 10

Output: In ra một số nguyên duy nhất là số lượng xâu tìm được

## Example:

|  |  |
| --- | --- |
| **Input** | **Output** |
| **4**  **abc**  **aa**  **aab**  **aa** | **3** |