问题

【题目】 第一行一个整数 n。 接下来有 n 行文本：每行可能为空行；也可能以空格开头；也可能包含连续空格。你必须“原样”保留（包括行首空格与空行）。 然后输入一个整数 q，接下来 q 行查询：

- LEN i：输出第 i 行文本的长度（空格也算）
- HASH i：输出第 i 行文本的 FNV-1a 64-bit 哈希（unsigned long long），对每个字符按字节参与
- JOIN l r：输出把第 l..r 行用字符 '|' 连接后的结果（不额外添加/删除任何空格）

约束：1 ≤ n,q ≤ 2e5，总文本长度 ≤ 2e6。 要求：C++20，不能用不安全输入函数。

【原代码】

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

unsigned long long fnv(const char* s){
    unsigned long long h = 14695981039346656037ULL;
    for(int i=0; s[i]; ++i){
        h ^= (unsigned char)s[i];
        h *= 1099511628211ULL;
    }
    return h;
}

char* readLine(){
    char buf[1000000];
    gets(buf);
    return buf;
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n; cin >> n;
    vector<string> a;
    a.reserve(n);
    for(int i=0;i<n;i++){
        string s;
        getline(cin >> ws, s);
        a[i] = s;
    }

    int q; cin >> q;
```

```
    while(q--){
        string op; cin >> op;
        if(op=="LEN"){
            int i; cin >> i;
            cout << a[i-1].size() << "\n";
        }else if(op=="HASH"){
            int i; cin >> i;
            cout << fnv(readLine()) << "\n";
        }else{
            int l,r; cin >> l >> r;
            for(int i=l;i<=r;i++){
                if(i>l) cout << "|";
                cout << a[i];
            }
            cout << "\n";
        }
    }
}
```

回答

1) 题意一句话

读入 n 行“原样文本”（允许空行/行首空格），再处理 q 个 LEN/HASH/JOIN 查询。

2) 致命点清单（≥5条）

1. `` `** 吞行首空格 + 跳过空行**： `` ws` 会吃掉所有前导空白（含换行），导致“必须原样保留”直接违题；现象：空行变没、行首空格丢失、行号整体错位。
2. `** 不等于 **` ：`a.reserve(n)` 只扩容量不扩 size，`a[i]=...` 越界写（UB）；现象：偶尔 RE/漏行/数据乱。
3. `` `** 不安全 + 读到的根本不是第 i 行文本**： `` HASH i 里你没用 i`，而是又读了一行输入（大概率读到查询行剩余或空行）；现象：HASH 随机/永远不对。
4. `` `** 返回局部数组地址（悬垂指针 UB）**： `` buf` 在函数返回就失效；现象：HASH 可能“有时对、有时乱”。
5. **FNV offset basis 常量写错**：64-bit FNV-1a offset basis 应是 `14695981039346656037ULL`（你少了一个 7）；现象：哪怕读对字符串，哈希也对不上标准。
6. **JOIN 下标体系错（1-based vs 0-based）**：循环 `i=l..r` 却 `cout<<a[i]`，应是 `a[i-1]`；现象：越界/多输出一行/少一行。
7. `** 以 **` **终止**：如果文本含 `\0`（一般不出现，但题意“按字节”更稳），用 `string` 长度遍历更一致。

3) 修复方案 A（最小修复，贴近原写法，可编译）

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

static inline unsigned long long fnv1a64(const string& s){
    unsigned long long h = 14695981039346656037ULL; // 正确 offset basis
    for(unsigned char c : s){
        h ^= (unsigned long long)c;
        h *= 1099511628211ULL;
    }
    return h;
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    if(!(cin >> n)) return 0;
    string dummy;
    getline(cin, dummy);              // 吃掉 n 后面的换行

    vector<string> a;
    a.reserve(n);
    for(int i=0;i<n;i++){
        string s;
        getline(cin, s);              // 不要 ws：保留空行/行首空格
        a.push_back(std::move(s));
    }

    int q; cin >> q;
    while(q--){
        string op; cin >> op;
        if(op=="LEN"){
            int i; cin >> i;
            cout << a[i-1].size() << "\n";
        }else if(op=="HASH"){
            int i; cin >> i;
            cout << fnv1a64(a[i-1]) << "\n";
        }else if(op=="JOIN"){
            int l,r; cin >> l >> r;
            for(int k=l;k<=r;k++){
                if(k>l) cout << '|';
                cout << a[k-1];
            }
```

```
            cout << "\n";
        }
    }
    return 0;
}
```

4) 修复方案 B（现代 C++ 推荐写法：更稳、更清晰，可编译）

- 核心差异：显式 `resize` + 统一 0-based，FNV 写成"按 bytes/长度"版本，避免任何 `char*`。

```
#include <iostream>
#include <string>
#include <vector>
using namespace std;

static inline unsigned long long fnv1a64_bytes(const string& s){
    unsigned long long h = 14695981039346656037ULL;
    for(size_t i=0;i<s.size();++i){
        h ^= (unsigned long long)(unsigned char)s[i];
        h *= 1099511628211ULL;
    }
    return h;
}

int main(){
    ios::sync_with_stdio(false);
    cin.tie(nullptr);

    int n;
    if(!(cin >> n)) return 0;

    string line;
    getline(cin, line); // consume endline

    vector<string> a(n);
    for(int i=0;i<n;i++) getline(cin, a[i]);

    int q; cin >> q;
    while(q--){
        string op; cin >> op;
        if(op=="LEN"){
            int idx; cin >> idx;
            cout << a[idx-1].size() << '\n';
        }else if(op=="HASH"){
            int idx; cin >> idx;
            cout << fnv1a64_bytes(a[idx-1]) << '\n';
```

```
        }else if(op=="JOIN"){
            int l,r; cin >> l >> r;
            for(int idx=l-1; idx<=r-1; ++idx){
                if(idx>l-1) cout << '|';
                cout << a[idx];
            }
            cout << '\n';
        }
    }
    return 0;
}
```

5) 复杂度

- 时间：读取与哈希总计 **O(总文本长度 + 总输出长度)**；每次 HASH 为 O(该行长度)，JOIN 为 O(拼接输出长度)。
- 空间：存储文本 **O(总文本长度)**。

6) 至少 6 个边界测试（输入要点→期望现象）

1. n=1，文本为 **空行**，查询 `LEN 1`、`HASH 1`、`JOIN 1 1` → LEN=0；HASH 为“空串的标准 FNV-1a”；JOIN 输出空行。
2. 文本行首有 3 个空格：`"   hi"`，`LEN` → 5；`JOIN` 结果保留行首空格。
3. 文本含连续空格：`"a  b   c"` → LEN 计入所有空格。
4. 最后一行是空行 + 前一行非空，`JOIN 1 2` → 末尾仍出现 `|` 后接空（即输出以 `|` 结尾）。
5. JOIN 边界：`JOIN 1 1`、`JOIN n n`、`JOIN 1 n` → 不越界、不多/漏行。
6. 大数据：总长度接近 2e6，q=2e5，交替 LEN/HASH/JOIN → 不超时、不爆内存、输出不乱序。
7. 非 ASCII（UTF-8 中文/emoji）→ LEN 按字节数（`string::size()`）符合“按字节参与哈希”的一致性。

7) 1 个可复用小模板（≤6行）

```
int n; cin >> n;
string dummy; getline(cin, dummy);      // 吃掉数字后的换行
vector<string> a(n);
for(int i=0;i<n;i++) getline(cin, a[i]); // 原样保留空行/行首空格
```