

# Bài tập về nhà buổi 3

## Basic

### Bài 1: Kiểm tra tính hợp lệ của chuỗi Latex

#### Đề bài:

Kiểm tra chuỗi Latex có hợp lệ không bằng cách đảm bảo các dấu ngoặc {}, [], () được đóng mở đúng thứ tự.

#### Đầu vào:

Chuỗi s ( $1 \leq |s| \leq 10^5$ ).

#### Đầu ra:

1 nếu chuỗi hợp lệ, ngược lại 0.

#### Lời giải:

Sử dụng ngăn xếp (stack) để kiểm tra tính cân bằng của các dấu ngoặc. Khi gặp dấu mở, đẩy vào ngăn xếp. Khi gặp dấu đóng, kiểm tra xem có khớp với dấu mở trên cùng của ngăn xếp không.

#### Mã nguồn:

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

int decode[256];

int main() {
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);

    string s; cin >> s;

    decode['{'] = -3;
    decode['}'] = 3;
    decode['['] = -2;
    decode[']'] = 2;
    decode['('] = -1;
    decode[')'] = 1;

    vector<int> stk;

    for (char c : s) if (decode[c] != 0) {
        if (decode[c] < 0) stk.push_back(decode[c]);
        else if (stk.empty() || stk.back() != -decode[c]) {
            cout << "0" << endl;
            return 0;
        }
        else {
            stk.pop_back();
        }
    }
}
```

```

        cout << 0;
        return 0;
    } else stk.pop_back();
}
cout << (stk.empty() ? 1 : 0);
return 0;
}

```

## Bài 2: Chèn vào danh sách liên kết đơn đã sắp xếp

### Đề bài:

Chèn giá trị  $x$  vào danh sách liên kết đơn đã sắp xếp sao cho danh sách vẫn giữ thứ tự tăng dần.

### Đầu vào:

- Số nguyên  $N$  ( $0 \leq N \leq 10^5$ ), số phần tử trong danh sách.
- $N$  số nguyên của danh sách liên kết.
- Số nguyên  $x$ , giá trị cần chèn.

### Đầu ra:

Danh sách liên kết sau khi chèn, cách nhau bởi dấu cách.

### Lời giải:

Duyệt danh sách liên kết để tìm vị trí thích hợp và chèn nút mới vào.

### Mã nguồn:

```

#include <iostream>
using namespace std;

class SinglyLinkedListNode {
public:
    int data;
    SinglyLinkedListNode *next;

    SinglyLinkedListNode(int node_data) {
        this->data = node_data;
        this->next = nullptr;
    }
};

SinglyLinkedListNode* insertSortedLinkedList(SinglyLinkedListNode* head, int x) {
    SinglyLinkedListNode *newNode = new SinglyLinkedListNode(x);
    if (head == NULL || head->data >= x) {
        newNode->next = head;
        head = newNode;
    } else {
        SinglyLinkedListNode *current = head;
        while (current->next != NULL && current->next->data < x) {
            current = current->next;
        }
        newNode->next = current->next;
        current->next = newNode;
    }
    return head;
}

```

```

        return newNode;
    }

    SinglyLinkedListNode *ptr = head;
    while (ptr->next != NULL && ptr->next->data < x) ptr = ptr->next;
    newNode->next = ptr->next;
    ptr->next = newNode;
    return head;
}

void printLinkedList(SinglyLinkedListNode* head) {
    while (head != NULL) {
        cout << head->data << ' ';
        head = head->next;
    }
}

int main() {
    int n, x;
    cin >> n;
    SinglyLinkedListNode* head = NULL;

    for (int i = 0; i < n; i++) {
        int value;
        cin >> value;
        head = insertSortedLinkedList(head, value);
    }
    cin >> x;
    head = insertSortedLinkedList(head, x);
    printLinkedList(head);
    return 0;
}

```

## Bài 3: Nhập và tính giá trị đa thức

### Đề bài:

Nhập một đa thức, hiển thị dạng chuẩn và tính giá trị tại  $x$ .

### Đầu vào:

- Số nguyên  $n$  ( $0 \leq n \leq 100$ ), số đơn thức.
- $n$  cặp số: hệ số (số thực) và số mũ (nguyên không âm).
- Số thực  $x$ .

### Đầu ra:

- Đa thức dạng chuẩn.
- Giá trị của đa thức tại  $x$ , làm tròn 2 chữ số thập phân.

**Lời giải:**

Sử dụng danh sách liên kết để lưu các đơn thức. Tính giá trị đa thức bằng cách duyệt qua danh sách và áp dụng công thức.

**Mã nguồn:**

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

struct DONTHUC {
    int somu;
    double heso;

    DONTHUC(double _heso = 0, int _somu = 0) {
        heso = _heso;
        somu = _somu;
    }
};

struct Node {
    DONTHUC* data;
    Node* next;

    Node(DONTHUC* _data = nullptr) {
        this->data = _data;
        this->next = nullptr;
    }
};

struct DATHUC {
    Node* head;
    Node* tail;
    DATHUC() {
        this->head = nullptr;
        this->tail = nullptr;
    }
};

void Nhap(DATHUC &B, double heso, int somu) {
    DONTHUC* p = new DONTHUC(heso, somu);
    Node* q = new Node(p);
    if (B.head == nullptr) {
        B.head = q;
        B.tail = q;
    } else {
        B.tail->next = q;
        B.tail = q;
    }
}

void Xuat(DATHUC B) {
    Node* p = B.head;
```

```

        bool first = true;
        while (p != nullptr) {
            if (!first && p->data->heso > 0) cout << "+";
            if (p->data->heso < 0) cout << "-";
            if (p->data->heso == 0) {
                p = p->next;
                continue;
            }
            first = false;
            if (abs(p->data->heso) != 1) cout << abs(p->data->heso);
            if (p->data->somu > 1) cout << "x^" << p->data->somu;
            else if (p->data->somu == 1) cout << "x";
            p = p->next;
        }
        if (first) cout << "0";
    }

    double TinhDaThuc(DATHUC B, double x) {
        double S = 0;
        Node* p = B.head;
        while (p != nullptr) {
            S += p->data->heso * pow(x, p->data->somu);
            p = p->next;
        }
        return S;
    }

    int main() {
        DATHUC B;
        int N;
        cin >> N;
        for (int i = 0; i < N; i++) {
            double heso; int somu;
            cin >> heso >> somu;
            Nhaph(B, heso, somu);
        }
        cout << "Da thuc vua nhap la: "; Xuat(B);
        double x; cin >> x;
        cout << "\nVoi x=" << x << ", gia tri da thuc la: "
            << setprecision(2) << fixed << TinhDaThuc(B, x);
        return 0;
    }
}

```

## Bài 4: Đảo ngược danh sách liên kết

### Đề bài:

Đảo ngược danh sách liên kết đơn và in kết quả.

**Đầu vào:**

- Số nguyên  $N$ , số phần tử trong danh sách.
- $N$  số nguyên của danh sách liên kết.

**Đầu ra:**

Danh sách liên kết sau khi đảo ngược, cách nhau bởi dấu cách.

**Lời giải:**

Duyệt danh sách và thay đổi con trỏ `next` của từng nút để đảo ngược danh sách.

**Mã nguồn:**

```
void reverseLinkedList(SinglyLinkedList* llist) {
    SinglyLinkedListNode* prev = nullptr;
    SinglyLinkedListNode* current = llist->head;
    SinglyLinkedListNode* next = nullptr;

    while (current != nullptr) {
        next = current->next;
        current->next = prev;
        prev = current;
        current = next;
    }

    llist->tail = llist->head;
    llist->head = prev;
}
```

**Bài 5: Chuyển đổi thập phân sang nhị phân****Đề bài:**

Chuyển đổi số nguyên từ hệ thập phân sang hệ nhị phân.

**Đầu vào:**

Số nguyên  $x$  ( $1 \leq x \leq 10^5$ ).

**Đầu ra:**

Chuỗi nhị phân tương ứng.

**Lời giải:**

Sử dụng ngăn xếp để lưu phần dư khi chia  $x$  cho 2, sau đó in ra theo thứ tự ngược lại.

**Mã nguồn:**

```

stack<int> s;

while (x > 0) {
    s.push(x % 2);
    x /= 2;
}

while (!s.empty()) {
    cout << s.top();
    s.pop();
}

```

## Bài 6: Trộn hai danh sách liên kết đã sắp xếp

### Đề bài:

Trộn hai danh sách liên kết đơn đã sắp xếp thành một danh sách cũng được sắp xếp.

### Đầu vào:

- Số nguyên  $T$  ( $T < 10$ ), số bộ test.
- Mỗi bộ test gồm:
  - Hai số nguyên  $n$  và  $m$ , kích thước của hai danh sách.
  - $n$  số nguyên của danh sách thứ nhất.
  - $m$  số nguyên của danh sách thứ hai.

### Đầu ra:

Danh sách liên kết sau khi trộn.

### Lời giải:

Sử dụng hai con trỏ để duyệt qua hai danh sách và trộn chúng lại.

### Mã nguồn:

```

SinglyLinkedListNode* mergeLists(SinglyLinkedListNode* head_list1,
SinglyLinkedListNode* head_list2) {
    SinglyLinkedListNode dummy(0);
    SinglyLinkedListNode* tail = &dummy;

    while (head_list1 && head_list2) {
        if (head_list1->data <= head_list2->data) {
            tail->next = head_list1;
            head_list1 = head_list1->next;
        } else {
            tail->next = head_list2;
            head_list2 = head_list2->next;
        }
    }
}

```

```

        tail = tail->next;
    }

    if (head_list1) tail->next = head_list1;
    else tail->next = head_list2;

    return dummy.next;
}

```

## Bài 7: Duyệt cây theo thứ tự NLR (Đệ quy)

### Đề bài:

Duyệt cây nhị phân theo thứ tự NLR (Node-Left-Right) bằng đệ quy.

### Đầu vào:

- Số nguyên  $N$ , số nút trong cây.
- $N$  số nguyên là giá trị các nút.

### Đầu ra:

Kết quả duyệt cây theo thứ tự NLR.

### Lời giải:

Sử dụng đệ quy để duyệt cây theo thứ tự NLR.

### Mã nguồn:

```

void preOrder(Node *root) {
    if (root == NULL) return;
    cout << root->data << " ";
    preOrder(root->left);
    preOrder(root->right);
}

```

## Bài 8: Duyệt cây theo thứ tự NLR (Không đệ quy)

### Đề bài:

Duyệt cây nhị phân theo thứ tự NLR (Node-Left-Right) không dùng đệ quy.

### Đầu vào:

Giống bài 7.

**Đầu ra:**

Giống bài 7.

**Lời giải:**

Sử dụng ngăn xếp để mô phỏng đệ quy và duyệt cây theo thứ tự NLR.

**Mã nguồn:**

```
void preOrder(Node *root) {
    stack<Node*> s;
    s.push(root);

    while (!s.empty()) {
        Node* ptr = s.top();
        s.pop();
        if (ptr == NULL) continue;

        cout << ptr->data << " ";

        if (ptr->right) s.push(ptr->right);
        if (ptr->left) s.push(ptr->left);
    }
}
```

**Bài 9: Tìm nút chung của hai danh sách liên kết****Đề bài:**

Tìm nút chung đầu tiên của hai danh sách liên kết đơn.

**Đầu vào:**

- Hai danh sách liên kết.
- Danh sách thứ ba chứa các nút chung.

**Đầu ra:**

Giá trị của nút chung đầu tiên, hoặc `NA` nếu không có.

**Lời giải:**

Tính độ dài của hai danh sách, căn chỉnh chúng và duyệt để tìm nút chung.

**Mã nguồn:**

```
SinglyLinkedListNode* findMergeNode(SinglyLinkedListNode* head1,
SinglyLinkedListNode* head2) {
    int len1 = 0, len2 = 0;
    SinglyLinkedListNode* temp1 = head1;
```

```

SinglyLinkedListNode* temp2 = head2;

while (temp1) { len1++; temp1 = temp1->next; }
while (temp2) { len2++; temp2 = temp2->next; }

temp1 = head1;
temp2 = head2;

if (len1 > len2) for (int i = 0; i < len1 - len2; i++) temp1 = temp1->next;
else for (int i = 0; i < len2 - len1; i++) temp2 = temp2->next;

while (temp1 && temp2) {
    if (temp1 == temp2) return temp1;
    temp1 = temp1->next;
    temp2 = temp2->next;
}

return nullptr;
}

```

## Bài 10: Nút tổ tiên thấp nhất trong cây BST

### Đề bài:

Tìm nút tổ tiên thấp nhất của hai nút trong cây nhị phân tìm kiếm (BST).

### Đầu vào:

- Số nguyên `N`, số nút trong cây.
- `N` số nguyên là giá trị các nút.
- Hai số nguyên `v1` và `v2`.

### Đầu ra:

Giá trị của nút tổ tiên thấp nhất.

### Lời giải:

Sử dụng đệ quy để tìm điểm phân tách giữa hai nút.

### Mã nguồn:

```

Node* lca(Node* root, int v1, int v2) {
    if (root == NULL) return NULL;
    if (root->data == v1 || root->data == v2) return root;

    Node* left = lca(root->left, v1, v2);
    Node* right = lca(root->right, v1, v2);

    if (left && right) return root;
}

```

```

        return left ? left : right;
    }
}

```

## Advance

### Bài 1: Kiểm kê hàng hóa

#### Mô tả:

Cửa hàng X bán nhiều loại hàng, mỗi loại có mã riêng. Trong quá trình bán hàng, nhân viên chỉ ghi lại mã hàng mà không ghi số lượng. Hãy xác định số lượng bán ra của từng mã hàng.

#### Đầu vào:

- Dòng đầu tiên chứa số nguyên  $N$  ( $1 \leq N \leq 5 \times 10^4$ ), là số lượng mã hàng được ghi lại.
- $N$  dòng tiếp theo, mỗi dòng chứa một mã hàng (tối đa 100 ký tự).

#### Đầu ra:

- Với mỗi mã hàng, in ra mã hàng và số lượng bán được.
- Sắp xếp kết quả theo số lượng giảm dần. Nếu số lượng bằng nhau, sắp xếp theo mã hàng tăng dần.

#### Lời giải:

Sử dụng một hash map để đếm số lần xuất hiện của từng mã hàng. Sau đó, sắp xếp kết quả theo tần suất giảm dần và theo mã hàng tăng dần nếu tần suất bằng nhau.

#### Mã nguồn:

```

#include<bits/stdc++.h>
#define fi first
#define se second
using namespace std;
using ll = long long;

const int MOD = 1e9 + 7;
const int MAX = 5e4 + 5;

#define TASK ""

bool CompareStrings(const string &a, const string &b) {
    if (a.length() != b.length()) return a.length() < b.length();
    return a < b;
}

bool ComparePairs(const pair<int, string> &a, const pair<int, string> &b) {
    if (a.first != b.first) return a.first > b.first; // Tần suất giảm dần
}

```

```

        return CompareStrings(a.second, b.second); // Chuỗi tăng dần
    }

void Merge(vector<string> &vec, int left, int mid, int right) {
    vector<string> temp(right - left + 1);
    int i = left, j = mid + 1, k = 0;

    while (i <= mid && j <= right) {
        if (CompareStrings(vec[i], vec[j])) {
            temp[k++] = vec[i++];
        } else {
            temp[k++] = vec[j++];
        }
    }

    while (i <= mid) temp[k++] = vec[i++];
    while (j <= right) temp[k++] = vec[j++];

    for (int l = 0; l < temp.size(); l++) {
        vec[left + l] = temp[l];
    }
}

void MergeSort(vector<string> &vec, int left, int right) {
    if (left >= right) return;

    int mid = left + (right - left) / 2;
    MergeSort(vec, left, mid);
    MergeSort(vec, mid + 1, right);
    Merge(vec, left, mid, right);
}

void CustomSort(vector<string> &vec) {
    MergeSort(vec, 0, vec.size() - 1);
}

void Merge(vector<pair<int, string>> &vec, int left, int mid, int right) {
    vector<pair<int, string>> temp(right - left + 1);
    int i = left, j = mid + 1, k = 0;

    while (i <= mid && j <= right) {
        if (ComparePairs(vec[i], vec[j])) {
            temp[k++] = vec[i++];
        } else {
            temp[k++] = vec[j++];
        }
    }

    while (i <= mid) temp[k++] = vec[i++];
    while (j <= right) temp[k++] = vec[j++];

    for (int l = 0; l < temp.size(); l++) {
        vec[left + l] = temp[l];
    }
}

```

```

    }

}

void MergeSort(vector<pair<int, string>> &vec, int left, int right) {
    if (left >= right) return;

    int mid = left + (right - left) / 2;
    MergeSort(vec, left, mid);
    MergeSort(vec, mid + 1, right);
    Merge(vec, left, mid, right);
}

void CustomSort(vector<pair<int, string>> &vec) {
    MergeSort(vec, 0, vec.size() - 1);
}

int main() {
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    if (fopen(TASK".inp", "r")) {
        freopen(TASK".inp", "r", stdin);
        freopen(TASK".out", "w", stdout);
    }

    int n; cin >> n;
    vector<string> arr(n);
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
    }

    CustomSort(arr); // Sắp xếp mảng

    vector<pair<int, string>> freqArr;
    int count = 1;

    for (int i = 1; i < n; i++) {
        if (arr[i] == arr[i - 1]) {
            count++;
        } else {
            freqArr.emplace_back(count, arr[i - 1]);
            count = 1;
        }
    }
    freqArr.emplace_back(count, arr[n - 1]); // Thêm phần tử cuối cùng

    CustomSort(freqArr); // Sắp xếp theo tần suất và chuỗi

    for (const auto &p : freqArr) {
        cout << p.second << ' ' << p.first << '\n';
    }

    return 0;
}

```

## Bài 2: Truy vấn tìm kiếm nhị phân

### Mô tả:

Cho một mảng số nguyên `A` và `Q` truy vấn. Với mỗi truy vấn:

- Nếu `type = 1`, tìm vị trí đầu tiên xuất hiện của `y` trong `A`.
  - Nếu `type = 2`, tìm vị trí cuối cùng xuất hiện của `y` trong `A`.
- Nếu không tìm thấy, trả về `-1`.

### Đầu vào:

- Dòng đầu tiên chứa hai số nguyên `N` và `Q` ( $1 \leq N \leq 10^5, 1 \leq Q \leq 5 \times 10^5$ ).
- Dòng thứ hai chứa `N` số nguyên `A[i]` ( $-10^9 \leq A[i] \leq 10^9$ ).
- `Q` dòng tiếp theo, mỗi dòng chứa một truy vấn dạng `type x y`.

### Đầu ra:

- Với mỗi truy vấn, in ra kết quả trên một dòng.

### Lời giải:

Sắp xếp mảng kèm theo chỉ số ban đầu. Sử dụng tìm kiếm nhị phân để tìm vị trí đầu tiên và cuối cùng của giá trị cần tìm.

### Mã nguồn:

```
#include<iostream>
#include<vector>
using namespace std;
using ll = long long;
using ii = pair<int, int>

const int MOD = 1e9 + 7;
const int MAX = 1e5 + 5;

#define TASK "" // Retain only relevant defines

int n, q;
int a[MAX];
pair<int, int> res[MAX * 2]; // Adjust size to handle negative values if needed
vector<pair<int, int>> processedArray;
vector<int> discreteValues; // Store discrete values globally

void CombineSegments(vector<pair<int, int>>& arr, int left, int mid, int right) { // Renamed from Merge
    vector<pair<int, int>> temp(right - left + 1);
    int i = left, j = mid + 1, k = 0;

    while (i <= mid && j <= right) {
        if (arr[i].first < arr[j].first) {
            temp[k] = arr[i];
            i++;
        } else {
            temp[k] = arr[j];
            j++;
        }
        k++;
    }

    while (i <= mid) {
        temp[k] = arr[i];
        i++;
        k++;
    }

    while (j <= right) {
        temp[k] = arr[j];
        j++;
        k++;
    }

    for (int l = left; l <= right; l++) {
        arr[l] = temp[l - left];
    }
}
```

```

        if (arr[i].first <= arr[j].first) {
            temp[k++] = arr[i++];
        } else {
            temp[k++] = arr[j++];
        }
    }

    while (i <= mid) temp[k++] = arr[i++];
    while (j <= right) temp[k++] = arr[j++];

    for (int p = 0; p < k; p++) {
        arr[left + p] = temp[p];
    }
}

void CustomArrange(vector<pair<int, int>>& arr, int left, int right) {
    if (left < right) {
        int mid = left + (right - left) / 2;
        CustomArrange(arr, left, mid);
        CustomArrange(arr, mid + 1, right);
        CombineSegments(arr, left, mid, right); // Updated reference
    }
}

void Prepare() {
    processedArray.clear();
    for (int i = 1; i <= n; i++) {
        processedArray.push_back({a[i], i});
    }

    CustomArrange(processedArray, 0, processedArray.size() - 1);
}

int CustomLowerBound(const vector<pair<int, int>>& arr, pair<int, int> val) {
    int left = 0, right = arr.size();
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] < val) {
            left = mid + 1;
        } else {
            right = mid;
        }
    }
    return left;
}

int CustomUpperBound(const vector<pair<int, int>>& arr, pair<int, int> val) {
    int left = 0, right = arr.size();
    while (left < right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] <= val) {
            left = mid + 1;
        } else {
    }
}

```

```

        right = mid;
    }
}
return left;
}

int FindFirst(int val) {
    int idx = CustomLowerBound(processedArray, make_pair(val, 0));
    if (idx == processedArray.size() || processedArray[idx].first != val)
return -1;
    return processedArray[idx].second;
}

int FindLast(int val) {
    int idx = CustomUpperBound(processedArray, make_pair(val, 0x3f3f3f3f));
    if (idx == 0 || processedArray[idx - 1].first != val) return -1;
    return processedArray[idx - 1].second;
}

int main() {
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    if (fopen(TASK".inp", "r")) {
        freopen(TASK".inp", "r", stdin);
        freopen(TASK".out", "w", stdout);
    }

    cin >> n >> q;
    for (int i = 1; i <= n; i++) {
        cin >> a[i];
    }

    Prepare();

    while (q --) {
        string s;
        int type, x;

        cin >> s >> type >> x;

        if (type == 1) cout << FindFirst(x) << '\n';
        if (type == 2) cout << FindLast(x) << '\n';
    }

    return 0;
}

```

---

## Bài 3: Phát hiện virus trong tệp

**Mô tả:**

Cho một tệp hình chữ nhật kích thước  $N \times M$  chứa các ký tự thường. Xác định xem mỗi mẫu virus có xuất hiện trong tệp hay không (theo chiều ngang hoặc dọc).

**Đầu vào:**

- Dòng đầu tiên chứa ba số nguyên  $N$ ,  $M$ , và  $Q$  ( $1 \leq N \times M \leq 10^5$ ,  $1 \leq Q \leq 10^4$ ).
- $N$  dòng tiếp theo, mỗi dòng chứa  $M$  ký tự.
- $Q$  dòng tiếp theo, mỗi dòng chứa một mẫu virus (độ dài từ 2 đến 10).

**Đầu ra:**

- Một dòng duy nhất gồm  $Q$  ký tự, ký tự thứ  $i$  là  $1$  nếu mẫu virus thứ  $i$  xuất hiện, ngược lại là  $0$ .

**Lời giải:**

Tính trước hash cho các mẫu virus. Duyệt qua tất cả các chuỗi con trong tệp (theo chiều ngang và dọc) để kiểm tra khớp.

**Mã nguồn:**

```
#include <iostream>
#include <vector>
#include <string>
#include <unordered_set>

using namespace std;

// Global variables
int N, M, Q;
vector<string> file;
vector<string> viruses;
unordered_set<size_t> virusHashes;
unordered_set<string> foundSubstrings;

// Function to compute hash for a string
size_t computeHash(const string &s) {
    const int p = 31; // A prime number
    const int m = 1e9 + 9; // A large prime modulus
    size_t hash_value = 0;
    size_t p_pow = 1;

    for (char c : s) {
        hash_value = (hash_value + (c - 'a' + 1) * p_pow) % m;
        p_pow = (p_pow * p) % m;
    }

    return hash_value;
}

// Function to check for virus patterns
```

```

string detectVirus() {
    virusHashes.clear();
    foundSubstrings.clear();

    // Precompute hashes for all virus patterns
    for (const string &virus : viruses) {
        virusHashes.insert(computeHash(virus));
    }

    // Check horizontally
    for (int i = 0; i < N; ++i) {
        for (int len = 2; len <= 10; ++len) {
            for (int j = 0; j + len - 1 < M; ++j) {
                string substring = file[i].substr(j, len);
                if (virusHashes.count(computeHash(substring))) {
                    foundSubstrings.insert(substring);
                }
            }
        }
    }

    // Check vertically
    for (int j = 0; j < M; ++j) {
        for (int len = 2; len <= 10; ++len) {
            for (int i = 0; i + len - 1 < N; ++i) {
                string substring;
                for (int k = 0; k < len; ++k) {
                    substring += file[i + k][j];
                }
                if (virusHashes.count(computeHash(substring))) {
                    foundSubstrings.insert(substring);
                }
            }
        }
    }

    // Build result string
    string result(viruses.size(), '0');
    for (int k = 0; k < viruses.size(); ++k) {
        if (foundSubstrings.count(viruses[k])) {
            result[k] = '1';
        }
    }

    return result;
}

int main() {
    ios::sync_with_stdio(0); cin.tie(0); cout.tie(0);
    if (fopen(".inp", "r")) {
        freopen(".inp", "r", stdin);
        freopen(".out", "w", stdout);
    }
}

```

```

    cin >> N >> M >> Q;

    file.resize(N);
    for (int i = 0; i < N; ++i) {
        cin >> file[i];
    }

    viruses.resize(Q);
    for (int i = 0; i < Q; ++i) {
        cin >> viruses[i];
    }

    cout << detectVirus() << endl;

    return 0;
}

```

## Bài 4: Sinh viên cuối hàng

### Mô tả:

Ban đầu, các sinh viên xếp hàng theo thứ tự mã số sinh viên từ  $1$  đến  $n$ . Khi một sinh viên được gọi, họ sẽ lên đầu hàng. Hãy xác định sinh viên cuối hàng sau mỗi lần gọi.

### Đầu vào:

- Dòng đầu tiên chứa hai số nguyên  $n$  và  $m$  ( $1 \leq n, m \leq 10^5$ ).
- Dòng thứ hai chứa  $m$  số nguyên, là mã số sinh viên được gọi.

### Đầu ra:

- $m$  số nguyên, là mã số sinh viên cuối hàng sau mỗi lần gọi.

### Lời giải:

Sử dụng một map để theo dõi thứ tự của các sinh viên. Cập nhật map sau mỗi lần gọi và in ra sinh viên cuối hàng.

### Mã nguồn:

```

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

const int MAX = 1e5 + 7;

int n, m;

int val[MAX];

```

```

map<int, int> mp;

int main() {
    ios_base ::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    if (fopen(".inp", "r")) {
        freopen(".inp", "r", stdin);
        freopen(".out", "w", stdout);
    }

    cin >> n >> m;

    for (int i = n; i >= 1; i --) {
        val[i] = -i + 1;
        mp[val[i]] = i;
    }

    for (int i = 1; i <= m; i++) {
        int student; cin >> student;

        mp.erase(val[student]);
        mp[i] = student;
        val[student] = i;

        map<int, int>::iterator it = mp.begin();
        cout << it->second << " "; // Print the last element in the map
    }

    return 0;
}

```

## Bài 5: Thứ tự cuối cùng của sinh viên

### Mô tả:

Ban đầu, các sinh viên xếp hàng theo thứ tự mã số sinh viên từ `1` đến `n`. Khi một sinh viên được gọi, họ sẽ lên đầu hàng. Hãy xác định thứ tự cuối cùng của các sinh viên sau tất cả các lần gọi.

### Đầu vào:

- Dòng đầu tiên chứa hai số nguyên `n` và `m` ( $1 \leq n, m \leq 10^5$ ).
- Dòng thứ hai chứa `m` số nguyên, là mã số sinh viên được gọi.

### Đầu ra:

- Một dòng duy nhất chứa thứ tự cuối cùng của các sinh viên.

**Lời giải:**

Sử dụng một tập hợp để theo dõi các sinh viên đã được gọi. Duyệt ngược danh sách các lần gọi, sau đó thêm các sinh viên còn lại vào danh sách kết quả.

**Mã nguồn:**

```
#include <iostream>
#include <vector>
#include <set>
#include <algorithm>
using namespace std;

int main() {
    // Redirect input and output
    if (fopen(".inp", "r")) {
        freopen(".inp", "r", stdin);
        freopen(".out", "w", stdout);
    }

    int n, m;
    cin >> n >> m;

    vector<int> calls(m);
    for (int i = 0; i < m; ++i) {
        cin >> calls[i];
    }

    set<int> called; // To track unique students called
    vector<int> result;

    // Traverse calls in reverse order
    for (int i = m - 1; i >= 0; --i) {
        if (called.find(calls[i]) == called.end()) {
            result.push_back(calls[i]);
            called.insert(calls[i]);
        }
    }

    // Add remaining students in ascending order
    for (int i = 1; i <= n; ++i) {
        if (called.find(i) == called.end()) {
            result.push_back(i);
        }
    }

    // Output the result
    for (int student : result) {
        cout << student << " ";
    }
}
```

```
    return 0;  
}
```