Solution Assignment 3

Võ Lê Ngọc Thịnh

Ngày 28 tháng 4 năm 2025

Mục lục

1 Basic **2**

1.1 Bài 1: Kiểm tra biểu thức toán học (Latex) 2

1.2 Bài 2: LinkedList-Insertion 3

1.3 Bài 3: LinkedList-NhapDaThuc 4

1.4 Bài 4: LinkedList-Reverse 6

1.5 Bài 5: Dec-to-Bin 7

1.6 Bài 6: MergeTwoSortedLinkedList 7

1.7 Bài 7: Tree - Preorder Traversal (NLR) 8

1.8 Bài 8: Tree - Preorder Traversal (NLR) không dùng đệ quy 9

1.9 Bài 9: LinkedList - Tìm node chung của hai SinglyLinkedList 9

1.10 Bài 10: Binary Search Tree - Nút tổ tiên thấp nhất 11

2 Advance **12**

2.1 Bài 1: Kiểm kê 2 12

2.2 Bài 2: Binary Search 2 14

2.3 Bài 3: DetectVirus2D 16

2.4 Bài 4: XepHang2 17

2.5 Bài 5: XepHang 19

1 Basic

1.1 Bài 1: Kiểm tra biểu thức toán học (Latex)

Tóm tắt bài toán: Bài toán cho một chuỗi Latex và yêu cầu kiểm tra tính hợp lệ của các dấu ngoặc **đóng và mở** như "{", "}", "(", ")", "[", "]".

Thuật toán:

- Quan sát: Vì số lượng và loại dấu ngoặc đóng và mở phải tương ứng nhau và thứ tự của chúng phải đúng.
- Ta duyệt qua toàn bộ ký tự của xâu Latex, chỉ quan tâm đến các kí tự dấu ngoặc.
- Đối với những dấu ngoặc mở, ta dùng một stack để lưu nó lại.
- Đối với những dấu ngoặc đóng, ta cần tìm một dấu ngoặc mở tương ứng cùng loại với nó. Nếu dấu ngoặc mở thêm vào sau cùng của stack không khớp với dấu ngoặc đóng này thì chuỗi Latex không hợp lệ. Ngược lại thì ta bỏ dấu ngoặc mở này ra khỏi stack.
- Khi đã xét hết xâu mà stack vẫn còn dấu ngoặc mở thì chuỗi Latex không hợp lệ vì không có dấu ngoặc đóng khớp với những dấu ngoặc mở này.

Source code:

```
const char dx[] = {'{', '}', '(', ')', '[', ']'};

bool validChar(char x){
    for (char i : dx)
        if (i == x)
            return true;
    return false;
}

int main(){
    string str; cin >> str;
    stack<char> mys;

    for (int i=0;i< (int)str.length();i++ ){
        if (!validChar(str[i])) continue;
        if (str[i] == dx[0] || str[i] == dx[2] || str[i] == dx
           [4])
            mys.push(str[i]);
        else{
            if (str[i] == '}'){
                if (mys.empty() || mys.top() != '{')
```

```
                    return void(cout << 0), 0;
                else mys.pop();
            }
            else if (str[i] == ')'){
                if (mys.empty() || mys.top() != '(')
                    return void(cout << 0), 0;
                else mys.pop();
            }
            else if (str[i] == ']'){
                if (mys.empty() || mys.top() != '[')
                    return void(cout << 0), 0;
                else mys.pop();
            }
        }
    }
    cout << (mys.empty() ? 1 : 0);
}
```

1.2 Bài 2: LinkedList-Insertion

Tóm tắt bài toán: Bài toán yêu cầu chèn một nút vào danh sách liên kết đã có thứ tự.

Thuật toán:

- Gọi *newNode* là phần tử cần chèn.
- Nếu phần tử cần chèn bé hơn *pHead* thì ta trỏ con trỏ *newNode.next* đến *pHead* và gán $pHead = newNode$.
- Ngược lại, ta đặt một con trỏ *current* duyệt qua LinkedList và dừng lại trước phần tử lớn hơn *newNode* và sau đó ta thực hiện việc chèn *newNode*.

Source Code:

```
SinglyLinkedListNode* newNode = new SinglyLinkedListNode(x){
    if (head == NULL || x < head->data){
        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;
}
```

1.3 Bài 3: LinkedList-NhapDaThuc

Tóm tắt bài toán: Bài toán yêu cầu nhập một đa thức với mỗi phần tử trong LinkedList là đơn thức, thực hiện việc tính toán đa thức này với một giá trị được cho sẵn và xuất ra đa thức này.

Thuật toán:

- Do các đơn thức được nhập vào với số mũ tăng dần nên mỗi lần nhập đơn thức ta chỉ cần chèn vào cuối LinkedList.
- Ta xây dựng các hàm $Nhap$, $TinhDaThuc$, $Xuat$. Riêng hàm $Xuat$ cài đặt khá phức tạp, nên khi code cẩn thận và chú ý các trường hợp đặc biệt mà yêu cầu nêu.

Source code:

```
void Nhap(DATHUC &B, double heso, double somu){
    DONTHUC* newDONTHUC = new DONTHUC(heso, somu);
    Node* newNode = new Node(newDONTHUC);

    if (B.head == NULL){
        B.head = B.tail = newNode;
    }
    else{
        B.tail->next = newNode;
        B.tail = newNode;
    }
}

double TinhDaThuc(DATHUC B, double x){
    double res = 0;
    Node* current = B.head;

    while (current != NULL){
        double heso = current->data->heso;
        int somu = current->data->somu;
        res += (double)heso*pow(x, somu);
        current = current->next;
    }
```

```
    return res;
}

void Xuat(DATHUC B){
    Node* current = B.head;
    bool first = true;

    while (current != NULL){
        double heso = current->data->heso;
        int somu = current->data->somu;

        if (heso == 0)
            current = current->next;
        else if (heso > 0){
            if (heso != 1 || somu == 0){
                if (!first)  cout << "+";
                cout << heso;
                first = false;
            }
            else if (heso == 1 && somu > 0){
                if (!first)  cout << "+";
                first = false;
            }
            if (somu > 0){
                cout << "x";
                first = false;
                if (somu > 1)
                    cout << "^" << somu;
            }
            current = current->next;
        }
        else if (heso < 0){
            if (heso != -1 || somu == 0) {
                cout << heso;
                first = false;
            }
            else if (heso == -1 && somu > 0){
                cout << "-";
                first = false;
            }
            if (somu > 0){
                cout << "x";
                first = false;
```

```
                if (somu > 1)
                    cout << "^" << somu;
            }
            current = current->next;
        }

    }
    if (first)
        cout << 0;

}
```

1.4 Bài 4: LinkedList-Reverse

Tóm tắt bài toán: Bài toán cho một SinglyLinkedList và yêu cầu đảo ngược LinkedList này.

Thuật toán:

- Tạo một SinglyLinkedList mới.
- SỬ dụng đệ quy để đi ngược tử cuối dãy LinkedList ban đầu về, với mỗi phần tử ta chèn vào vào cuối LinkedList vừa được tạo.

Source code:

```
void insert_node(SinglyLinkedList* list, int x){
    SinglyLinkedListNode* newNode = new SinglyLinkedListNode(x);
    if (list->head == NULL){
        list->head = list->tail = newNode;
    }
    else{
        list->tail->next = newNode;
        list->tail = newNode;
    }
}

void reverse(SinglyLinkedListNode* current, SinglyLinkedList*
   list){
    if (current->next != NULL)
        reverse(current->next, list);
    insert_node(list, current->data);
}

void reverseLinkedList(SinglyLinkedList* &list){
```

```
    SinglyLinkedList* newList = new SinglyLinkedList();
    reverse(list->head, newList);
    list = newList;
}
```

1.5 Bài 5: Dec-to-Bin

Tóm tắt bài toán: Bài toán cho một số ở hệ thập phân và yêu cầu chúng ta chuyển từ hệ thập phân sang nhị phân.

Thuật toán:

- Nếu số Dec hiện tại là số lẻ thì ta thêm 1 vào đầu dãy số Bin, ngược lại Dec là số chẵn thì ta thêm 0 vào đầu dãy số Bin. Sau đó, ta chia số Dec cho 2.

Source code:

```
int main(){
    int n;  cin >> n;
    string str = "";
    while (n > 0){
        str = ((n & 1) ? "1" : "0") + str;
        n /= 2;
    }
    cout << str;
}
```

1.6 Bài 6: MergeTwoSortedLinkedList

Tóm tắt bài toán: Bài toán cho ta hai LinkedlList đã được sắp xếp sẵn và yêu cầu merge hai LinkedList này thành một LinkedList được sắp xếp.

Thuật toán:

- Gọi *newLinkedList* là LinkedList sau khi merge. Ta duy trì hai con trỏ *First* và *Second* duyệt qua hai LinkedList A và B theo thứ tự từ đầu về cuối.

- Nếu giá trị con trỏ $First \leq Second$ thì ta thêm phần tử *First* vào *newLinkedList* và chuyển con trỏ *First* tới phần tử kế tiếp nếu danh sách đó còn phần tử. Tương tự cho trường hợp $First > Second$.

- Nếu một trong hai LinkedList A và B được duyệt qua hết, ta sẽ thêm hết các phần tử của LinkedList còn lại vào *newLinkedList*.

Source code:

```
SinglyLinkedList* newList = new SinglyLinkedList(){
    while (head_list1 != NULL && head_list2 != NULL){
        if (head_list1->data <= head_list2->data){
            newList->insert_node(head_list1->data);
            head_list1 = head_list1->next;
        }
        else{
            newList->insert_node(head_list2->data);
            head_list2 = head_list2->next;
        }
    }

    while (head_list1 != NULL){
        newList->insert_node(head_list1->data);
        head_list1 = head_list1->next;
    }

    while (head_list2 != NULL){
        newList->insert_node(head_list2->data);
        head_list2 = head_list2->next;
    }

    return newList->head;
}
```

1.7 Bài 7: Tree - Preorder Traversal (NLR)

Tóm tắt bài toán: Bài toán cho ta một cây nhị phân và yêu cầu cài đặt hàm preOrder duyệt cây theo thứ tự NLR.

Thuật toán:

- Ta sẽ dùng đệ quy để duyệt cây.
- Khi ở một nút ở cây, gọi là *current*, ta sẽ in ra giá trị của node *current*, sau đó nếu *current* tồn tại node con bên trái thì ta sẽ gọi đệ quy đến node con bên trái, tương tự với node con bên phải (nếu node con bên phải tồn tại).

Source code:

```
void preOrder(Node *root){
    cout << root->data << ' ';
    if (root->left != nullptr)
        preOrder(root->left);
```

```
    if (root->right != nullptr)
        preOrder(root->right);
}
```

1.8 Bài 8: Tree - Preorder Traversal (NLR) không dùng đệ quy

Tóm tắt bài toán: Bài toán cho ta một cây nhị phân và yêu cầu cài đặt hàm preOrder duyệt cây theo thứ tự NLR nhưng không được dùng đệ quy.

Thuật toán:

- Ta sẽ dùng stack để lưu được trạng thái duyệt cây.
- Ban đầu, ta push node root của cây vào stack. Ta lặp cho tới khi stack rỗng.
- Khi đang ở node *current* của cây, ta xuất ra giá tri của node *current*. Nếu node *current* này có node con bên trải thì ta push node con này stack trước. Sau đó xét node con bên trái của *current*, nếu tồn tại thì ta push vào stack.

Source code:

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

    while (!st.empty()){
        Node* current = st.top();
        cout << current->data << ' ';
        st.pop();
        if (current->right != nullptr)
            st.push(current->right);
        if (current->left != nullptr)
            st.push(current->left);
    }
}
```

1.9 Bài 9: LinkedList - Tìm node chung của hai SinglyLinkedList

Tóm tắt bài toán: Cho hai danh sách liên kết đơn, bài toán yêu cầu cài đặt hàm findMergeNode trả về node chung của danh sách liên kết này.

Thuật toán:

- Vì hai danh sách có thể có độ dài khác nhau. Nên nếu ta đi qua từng danh sách tuần tự thì con trỏ sẽ không bao giờ cùng đến nút chung cùng lúc.

- Vì vậy, ta tính độ dài từng danh sách, rồi dịch con trỏ của danh sách dài hơn trước một khoảng bằng hiệu độ dài 2 danh sách.
- Khi đó, cả hai con trỏ sẽ cách nút chung về phía cuối danh sách cùng một khoảng, ta di chuyển đồng thời đến khi trỏ trỏ tới cùng một nút thì đó chính là nút chung.
- Nếu không có nút chung, cuối cùng cả 2 con trỏ đều tới *nullptr*.

Source code:

```
SinglyLinkedListNode* findMergeNode(SinglyLinkedListNode* head1,
   SinglyLinkedListNode* head2){
    SinglyLinkedListNode* current_list_1 = head1;
    SinglyLinkedListNode* current_list_2 = head2;

    int length_1 = 0, length_2 = 0;
    while (current_list_1 != nullptr){
        length_1++;
        current_list_1 = current_list_1->next;
    }

    while (current_list_2 != nullptr){
        length_2++;
        current_list_2 = current_list_2->next;
    }

    current_list_1 = head1;
    current_list_2 = head2;
    if (length_1 > length_2){
        int diff = length_1 - length_2;
        while (diff){
            current_list_1 = current_list_1->next;
            diff--;
        }
    }
    else if (length_2 > length_1){
        int diff = length_2 - length_1;
        while (diff){
            current_list_2 = current_list_2->next;
            diff--;
        }
    }

    while (current_list_1 != nullptr && current_list_2 != nullptr
       ){
```

```
        if (current_list_1 == current_list_2)
            return current_list_1;
        current_list_1 = current_list_1->next;
        current_list_2 = current_list_2->next;
    }
    return nullptr;
}
```

1.10 Bài 10: Binary Search Tree - Nút tổ tiên thấp nhất

Tóm tắt bài toán: Cho một cây nhị phân tìm kiếm, yêu cầu cài đặt hàm lca tìm tổ tiên chung gần nhất của node $v1$ và $v2$.

Thuật toán:

- Ta bắt đầu từ node *root* của cây BST.
- Nếu root->data lớn hơn cả v1 và v2, thì nghĩa là cả 2 nút nằm ở cây con bên trái, ta di chuyển root xuống root->left.
- Nếu root->data nhỏ hơn cả v1 và v2, thì cả 2 nút nằm ở cây con bên phải, ta di chuyển root xuống root->right.
- Nếu root->data nằm giữa v1 và v2, thì root chính là nút tổ tiên chung thấp nhất.

Source code:

```
Node *lca(Node *root, int v1,int v2) {
    if (v1 > root->data && v2 > root->data) {
        return lca(root->right, v1, v2);
    }

    if (v1 < root->data && v2 < root->data) {
        return lca(root->left, v1, v2);
    }
    return root;
}
```

2 Advance

2.1 Bài 1: Kiểm kê 2

Tóm tắt bài toán: Bài toán N mã hàng, yêu cầu tính số lượng của mỗi loại mã hàng và xuất ra theo thứ tự giảm dần theo số lượng mã hàng, nếu số lượng bằng nhau thì in theo mã số tăng dần (điều kiện không được dùng các hàm sort, các CTDL của thư viện STL).

Thuật toán:

- Vì không được dùng các hàm sort được xây dựng sẵn, ta xây dựng thuật toán QuickSort vì đây là thuật toán sort đủ nhanh cho giới hạn của bài toán, bên cạnh đó ta phải xây dựng toán tử so sánh giữa hai chuỗi theo thứ tự từ điển.
- Sau khi sort lại, các mã giống nhau sẽ nằm cạnh nhau và ta có thể có số lượng của mỗi loại mã dễ dàng.
- Để xuất ra đúng định dạng theo yêu cầu của đề, ta có thể xây dựng thêm một hàm sort để sort theo số lượng giảm dần, nếu số lượng bằng nhau thì tăng dần theo mã hàng.

Source code:

```
const int N = 5e4;
struct Item{
    string str;
    Item (string _str = ""){
        str = _str;
    }
};

bool operator < (const Item &lhs, const Item &rhs){
    string l = lhs.str;
    string r = rhs.str;
    if (l.length() != r.length())
        return l.length() < r.length();
    else{
        for (int i=0;i < (int)l.length();i++ ){
            if (l[i] != r[i])
                return (l[i] < r[i]);
            }
    }
    return false;
}

```

```
23 struct Package{
24     int cnt;
25     Item it;
26     bool operator < (const Package &other)  const{
27         return (cnt == other.cnt ? it < other.it : cnt > other.
               cnt);
28     }
29 };
30 
31 template<class T>
32 void QuickSort(T *arr, int _left, int _right){
33     if (_left >= _right) return;
34     int mid = (_left + _right) >> 1;
35     T pivot = arr[mid];
36 
37     int i = _left, j = _right;
38     while (i < j){
39         while (arr[i] < pivot)  i++;
40         while (pivot < arr[j])  j--;
41 
42         if (i <= j){
43             swap(arr[i], arr[j]);
44             i++, j--;
45         }
46     }
47 
48     if (_left < j)
49         QuickSort(arr, _left, j);
50     if (i < _right)
51         QuickSort(arr, i, _right);
52 }
53 
54 Item items[N + 5];
55 Package pack[N + 5];
56 
57 int main(){
58     int n;  cin >> n;
59     for (int i=0;i< n;i++ )
60         cin >> items[i].str;
61     QuickSort(items, 0, n-1);
62 
63     int numPack = 0;
64     for (int i=0;i< n;i++ ){
```

```
        string last = items[i].str;
        int j = i;
        while (j + 1 < n && items[j + 1].str == last)
            j++;
        pack[numPack++] = {j-i+1, Item(last)};
        i = j;
    }

    QuickSort(pack, 0, numPack-1);
    for (int i=0;i < numPack;i++ )
        cout << pack[i].it.str << ' ' << pack[i].cnt << '\n';
}
```

2.2 Bài 2: Binary Search 2

Tóm tắt thuật toán: Cho một dãy A gồm N số nguyên, thực hiện các dạng truy vấn *type x y*:

- Nếu x = 1, xuất ra vị trí đầu tiên xuất hiện của y. Nếu không có thì in ra -1.
- Nếu x = 2, xuất ra vị trí cuối cùng xuất hiện của y. Nếu không có thì in ra -1.

Thuật toán:

- Ta xây dựng hai mảng maxPos, minPos với maxPos[i], minPos[i] lần lượt là giá trị chỉ số nhỏ nhất và lớn nhất mà giá trị i xuất hiện.
- Để có thể xây dựng được hai mảng maxPos và minPos, ta cần phải thực hiện rời rạc hóa giá trị của dãy A.

Source code:

```
const int N = 5e5;

template<class T>
void QuickSort(T *arr, int _left, int _right){
    if (_left >= _right)    return;
    int mid = (_left + _right) >> 1;
    T pivot = arr[mid];

    int i = _left, j = _right;
    while (i < j){
        while (arr[i] < pivot) i++;
        while (pivot < arr[j]) j--;

```

```
14          if (i <= j){
15              swap(arr[i], arr[j]);
16              i++, j--;
17          }
18      }
19
20      if (_left < j)  QuickSort(arr, _left, j);
21      if (i < _right) QuickSort(arr, i, _right);
22  }
23
24  int main(){
25      int n, q;  cin >> n >> q;
26      int *arr = new int[n];
27      int *copy = new int[n];
28      for (int i=0;i< n;i++ ){
29          cin >> arr[i];
30          copy[i] = arr[i];
31      }
32
33      QuickSort(copy, 0, n-1);
34      vector<int> value;
35      for (int i=0;i< n;i++ )
36          value.push_back(copy[i]);
37      UNIQUE(value);
38
39      for (int i=0;i< n;i++ )
40          arr[i] = lower_bound(ALL(value), arr[i]) - value.begin();
41
42      int MAX_INT = 1e9, MIN_INT = -1e9;
43      int *maxPos = new int[n];
44      for (int i=0;i< n;i++ )
45          maxPos[i] = MIN_INT;
46
47      int *minPos = new int[n];
48      for (int i=0;i< n;i++ )
49          minPos[i] = MAX_INT;
50
51      for (int i=0;i< n;i++){
52          minPos[arr[i]] = min(minPos[arr[i]], i);
53          maxPos[arr[i]] = max(maxPos[arr[i]], i);
54      }
55
56      while (q-- ){
```

```
        string s;    cin >> s;
        int type, p;     cin >> type >> p;
        int idx = lower_bound(ALL(value), p) - value.begin();
        if (value[idx] != p)
            cout << -1 << '\n';
        else{
            if (type == 1)
                cout << minPos[idx] + 1 << '\n';
            else
                cout << maxPos[idx] + 1 << '\n';
        }
    }
}
```

2.3 Bài 3: DetectVirus2D

Tóm tắt bài toán: Cho một bảng hình chữ gồm $N * M$ ô, mỗi ô chứa ký tự thường $a...z$. CHo Q truy vấn, mỗi truy vấn gồm một chuỗi có độ dài từ 2 đến 10. Hãy trả lời xem chuỗi này có xuất hiện trong bảng không, có thể là theo chiều ngang hoặc chiều dọc.

Thuật toán:

- Ta thực hiện hash hết tất cả các xâu có độ dài từ 2 đến 10 trong bảng và lưu các giá trị hash này vào một unordered_map để thực hiện truy vấn.
- Chú ý việc chọn $base = 311$ và $mod = 1e15 + 37$, giá trị mod này vừa lớn hơn số lượng các xâu có thể tạo ra trong bảng để hạn chế việc bị trùng giá trị hash.

Source code:

```
const long long base = 311;
const long long mod = 1e15 + 37;

ll getHash(const string &s){
    int n = (int)s.length();
    ll hashVal = 0;
    for (int i=0;i< n;i++ )
        hashVal = (1LL*hashVal*base + (s[i] - 'a' + 1)) % mod;

    return hashVal;
}

string str[N + 5];
unordered_map<ll, bool> HashMap;

```

```
int main(){
    int n, m, q;    cin >> n >> m >> q;
    for (int i=1;i<= n;i++ )
        cin >> str[i];

    for (int i=1;i<= n;i++){
        for (int j=0;j< m;j++ ){
            string cur = "";
            for (int k=j;k < min(m, j + 10);k++ ){
                cur += str[i][k];
                ll hashVal = getHash(cur);
                HashMap[hashVal] = true;
            }
        }
    }

    for (int j=0;j< m;j++ ){
        for (int i=1;i<= n;i++ ){
            string cur = "";
            for (int k=i;k <= min(n,i + 9);k++ ){
                cur += str[k][j];
                ll hashVal = getHash(cur);
                HashMap[hashVal] = true;
            }
        }
    }


    for (int i=1;i<= q;i++ ){
        string query;   cin >> query;
        ll hashVal = getHash(query);
        cout << (HashMap[hashVal] ? 1 : 0);
    }

}
```

2.4 Bài 4: XepHang2

Tóm tắt bài toán: Cho một lớp có n sinh viên xếp hàng theo thứ tự từ 1 đến n. Mỗi lần thầy giáo gọi một bạn học sinh i thì bạn học sinh đó phải lên đứng ở đầu hàng và thầy giáo cũng muốn biết học sinh đứng cuối hàng sau mỗi lần gọi.

Thuật toán:

- Ta sử dụng một deque chứa các học sinh và lưu một mảng cnt với cnt[i] là số lần xuất hiện của i trong deque.
- Ban đầu, ta đẩy các học sinh theo thứ tự từ 1 đến n và tăng cnt của tất cả lên 1.
- Sau mỗi lần thầy giáo gọi, ta đẩy học sinh được gọi vào đầu deque và tăng cnt của học sinh lên 1.
- Để tìm được học sinh đứng cuối deque, ta liên tục loại bỏ các học sinh ở cuối deque nếu cnt của học sinh đó lớn hơn 1.
- Vì chỉ có tối đa n bước thêm học sinh vào hàng đợi nên cũng chỉ có tối đa n học sinh được loại bỏ khỏi hàng đợi. Do đó thuật toán có độ phức tạp O(n).

Source code:

```
const int N = 1e5;

int n, m;
deque<int> dq;
int cnt[N + 5];

int main(){
    cin >> n >> m;
    while (!dq.empty())     dq.pop_back();
    for (int i=1;i<= n;i++ ){
        dq.push_back(i);
        cnt[i] = 1;
    }

    while (m-- ){
        int p;  cin >> p;
        dq.push_front(p);
        cnt[p]++;
        while (!dq.empty() && cnt[dq.back()] > 1){
            cnt[dq.back()] -= 1;
            dq.pop_back();
        }
        cout << dq.back() << ' ';
    }
}
```

2.5 Bài 5: XepHang

Tóm tắt bài toán: Dữ liệu bài toán được cho giống như bài 4 nhưng không có các truy vấn, yêu cầu chỉ xuất ra dãy sau khi thầy giáo thực hiện xong việc gọi học sinh.

Thuật toán:

- Vì các học sinh đứng ở đầu dãy chỉ có thể là do các học sinh được gọi ở những lần cuối cùng. Tức những học sinh đứng ở vị trí càng đầu là do những học sinh này được gọi ở những lần càng cuối.
- Ta duyệt ngược dãy số các học sinh được gọi, các học sinh này theo thứ tự lời gọi càng cuối.
- Những học sinh còn lại không được gọi thì vẫn đứng theo thứ tự, ta duyệt từ 1 đến n và xuất ra những học sinh không được gọi.

Source code:

```
const int N = 1e5;

int n, m;
bool check[N + 5];
int arr[N + 5];

int main(){
    cin >> n >> m;
    for (int i=1;i<= m;i++ )    cin >> arr[i];

    for (int i=m;i>= 1;i-- ){
        int p = arr[i];
        if (!check[p]){
            cout << p << ' ';
            check[p] = true;
        }
    }

    for (int i=1;i<= n;i++ )
        if (!check[i])
            cout << i << ' ';
}
```