In a school, the students are given an array of strings words[]. Students have

to find the longest string in words[] such that every prefix of it is also in words.

For example, let words = ["a", "app", "ap","appl", "apply"]. The string "apply"

has prefixes "ap","app","appl" and "a", all of which are in words.

Your task is the find and return the longest string in words as described above.

If there is more than one string with the same length, return the lexicographically

smallest one, and if no string exists, return "".

Input Format

-------------

Line1: string separated by spaces

Output Format

--------------

string

Sample Input-1:

---------------

k kmi km kmit

Sample Output-1:

----------------

kmit

Explanation:

------------

"kmit" has prefixes "kmi", "km", "k" and all of them appear in words.

Sample Input-2:

---------------

t tanker tup tupl tu tuple tupla

Sample Output-2:

----------------

tupla

Explanation:

------------

Both "tuple" and "tupla" have all their prefixes in words.

However, "tupla" is lexicographically smaller, so we return that.

Sample Input-3:

---------------

abc bc ab abcd

Sample Output-3:

----------------

""

Only 83.3% testcases passed

import java.util.\*;

public class Main{

public static void main(String[] args){

Scanner sc=new Scanner(System.in);

String s=sc.nextLine();

String[] arr=s.split(" ");

for(int i=0;i<arr.length;i++){

insert(arr[i]);

}

int max=0;

String ans="";

for(int i=0;i<arr.length;i++){

if(search(arr[i])){

if(arr[i].length()==max){

String[] arrr=new String[2];

arrr[0]=arr[i];

arrr[1]=ans;

Arrays.sort(arrr);

ans=arrr[0];

}

else if(arr[i].length()>max){

max=arr[i].length();

ans=arr[i];

}

}

}

System.out.println(ans);

}

public static boolean search(String s){

Trie curr=root;

for(int i=0;i<s.length();i++){

char cur=s.charAt(i);

if(curr.children[s.charAt(i)-'a']==null||!curr.children[cur-'a'].isend){

return false;

}

curr=curr.children[s.charAt(i)-'a'];

}

return true;

}

static class Trie{

Trie children[]=new Trie[26];

boolean isend;

// int count;

Trie(){

isend=false;

// count=0;

}

}

static Trie root=new Trie();

public static void insert(String s){

Trie curr=root;

for(int i=0;i<s.length();i++){

if(curr.children[s.charAt(i)-'a']==null){

Trie node=new Trie();

curr.children[s.charAt(i)-'a']=node;

// curr.count+=1;

}

curr=curr.children[s.charAt(i)-'a'];

}

curr.isend=true;

// curr.count+=1;

}

}

Malika taught a new fun time program practice for Engineering Students.

As a part of this she has given set of N numbers, and asked the students

to perform the operations listed below:

First, build the segment tree and then do the following,

1. sumRange(s1, s2) - return the sum of numbers between the indices

s1 and s2, both are inclusive and 0<=s1<=s2<N.

2. update(ind, val) - update the value at the index 'ind' to 'val'.

Your task is to solve this problem using Segment Tree concept.

Input Format:

-------------

Line-1: Two integers N and Q, size of the array(set of numbers) and query count.

Line-2: N space separated integers.

next Q lines: Three integers option, start/ind and end/val.

Output Format:

--------------

An integer result, for every sumRange query.

Sample Input:

-------------

8 5

1 2 13 4 25 16 17 8

1 2 6 //sumRange

1 0 7 //sumRange

2 2 18 //update

2 4 17 //update

1 2 7 //sumRange

Sample Output:

--------------

75

86

80

import java.util.\*;

public class Solution{

// Segment Tree Node structure

class SegmentTreeNode {

int start, end;

SegmentTreeNode left, right;

int sum;

public SegmentTreeNode(int start, int end) {

this.start = start;

this.end = end;

this.left = null;

this.right = null;

this.sum = 0;

}

}

SegmentTreeNode root = null;

public Solution(int[] nums) {

root = buildTree(nums, 0, nums.length - 1);

}

private SegmentTreeNode buildTree(int[] nums, int start, int end){

if (start > end) return null;

SegmentTreeNode node = new SegmentTreeNode(start, end);

if (start == end) {

node.sum = nums[start];

} else {

int mid = start + (end - start) / 2;

node.left = buildTree(nums, start, mid);

node.right = buildTree(nums, mid + 1, end);

node.sum = node.left.sum + node.right.sum;

}

return node;

}

void update(int ind, int val){

update(root, ind, val);

}

private void update(SegmentTreeNode node, int index, int val) {

if (node.start == node.end){

node.sum = val;

return;

}

int mid = node.start + (node.end - node.start) / 2;

if (index <= mid){

update(node.left, index, val);

} else {

update(node.right, index, val);

}

node.sum = node.left.sum + node.right.sum;

}

public int sumRange(int i, int j) {

return sumRange(root, i, j);

}

private int sumRange(SegmentTreeNode node, int start, int end) {

if (node.start == start && node.end == end) {

return node.sum;

}

int mid = node.start + (node.end - node.start) / 2;

if (end <= mid) {

return sumRange(node.left, start, end);

} else if (start > mid) {

return sumRange(node.right, start, end);

} else {

return sumRange(node.left, start, mid) + sumRange(node.right, mid + 1, end);

}

}

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

int N = scanner.nextInt();

int Q = scanner.nextInt();

int[] nums = new int[N];

for (int i = 0; i < N; i++) {

nums[i] = scanner.nextInt();

}

Solution solution = new Solution(nums);

for (int i = 0; i < Q; i++) {

int option = scanner.nextInt();

int start = scanner.nextInt();

int end = scanner.nextInt();

if (option == 1) {

System.out.println(solution.sumRange(start, end));

} else if (option == 2) {

solution.update(start, end);

}

}

    }

}