**Q1. Auto Complete**

There is a dictionary **A** of **N** words, and **ith** word has a unique weight **Wi**.

Another string array **B** of size **M** contains the prefixes. For every prefix **B[i]**, output **atmost 5** words from the dictionary A that start with the same prefix.

Output the words in **decreasing order** of their weight.

**NOTE:** If there is no word that starts with the given prefix output **-1**.

**Logic -**

Create a prefix Trie, but the Node of a Trie will also contain an array that will denote the answer for each prefix autocomplete[i].

Sort the given dictionary in decreasing order of its weight. Now insert the word in the Trie and update the autocomplete array if its size is less than 5 (Since we need to output at most 5 words).

For each query, search for the prefix if no word exists in the trie with the given prefix output -1.

Else, the answer array at the end of the prefix will be the answer.

Print the answer in the given format.

**Code -**

public class Main {

public static class Sortbyweight implements Comparator<myNode> {

// Used for sorting in ascending order of

// weights

public int compare(myNode a, myNode b)

{

return b.weight - a.weight;

}

}

// Tries

public static class Node {

char data;

Node[] children = new Node[26];

int[] autoComplete = new int[5];

int pop = 0;

public void add(int d) {

if(pop <= 4) {

autoComplete[pop] = d;

pop+=1;

}

}

public Node(char a) {

data = a;

}

}

// Created to sort weight wise

public static class myNode {

int weight;

String word;

myNode(String data, int w) {

word = data;

weight = w;

}

}

public static void printAutoComplete(int n, int m, myNode[] dictionary, String[] prefixs) {

// Insertion

Node root = new Node('\_');

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

Node temp = root;

for(int j=0;j<dictionary[i].word.length();j++) {

char c = dictionary[i].word.charAt(j);

int idx = (int) c - (int)'a';

if(temp.children[idx] == null) {

temp.children[idx] = new Node(c);

}

temp = temp.children[idx];

temp.add(i);

}

}

// Printing autocompletes

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

Node temp = root;

boolean prefixFound = true;

for(int j=0;j<prefixs[i].length();j++) {

char c = prefixs[i].charAt(j);

int idx = (int) c - (int)'a';

if(temp.children[idx] == null) {

System.out.print(-1);

System.out.print(" ");

System.out.println("");

prefixFound = false;

break;

}

temp = temp.children[idx];

}

if(!prefixFound) continue;

for(int j=0;j<temp.pop;j++) {

System.out.print(dictionary[temp.autoComplete[j]].word);

System.out.print(" ");

}

System.out.println("");

}

}

public static void main(String[] args) {

// YOUR CODE GOES HERE

// Please take input and print output to standard input/output (stdin/stdout)

// DO NOT USE ARGUMENTS FOR INPUTS

// E.g. 'Scanner' for input & 'System.out' for output

// Take INPUTS

Scanner sc = new Scanner(System.in);

// Test cases

int t = sc.nextInt();

sc.nextLine();

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

// N and M values

String line2 = sc.nextLine();

String[] stArray = line2.split(" ");

// System.out.print(stArray[1]);

int n = Integer.parseInt(stArray[0]);

int m = Integer.parseInt(stArray[1]);

// Input Dictionary

String line3 = sc.nextLine();

String[] dictionary = line3.split(" ");

// Input weights

String line4 = sc.nextLine();

String[] temp = line4.split(" ");

int[] weights = new int[n];

for(int j=0;j<temp.length;j++) weights[j] = Integer.parseInt(temp[j]);

// Convert to dictionary to myNode

myNode[] wt\_dictionary = new myNode[n];

for(int x=0;x<n;x++) {

wt\_dictionary[x] = new myNode(dictionary[x], weights[x]);

}

// Sort weighted dictionary

Arrays.sort(wt\_dictionary, new Sortbyweight());

// Input Prefixes

String line5 = sc.nextLine();

String[] prefixs = line5.split(" ");

// Print auto complete

printAutoComplete(n, m, wt\_dictionary, prefixs);

}

}

}

**Q2. Shortest Unique Prefix**

Given a list of **N** words, find the shortest unique prefix to represent each word in the list.

**NOTE:** Assume that no word is the prefix of another. In other words, the representation is always possible

**Logic -**

Now we will build prefix tree and we will also store count of characters.

Now, for every leaf / word , we find the character nearest to the root with frequency as 1.   
The prefix that the path from root to this character corresponds to, is the representation of the word.

**Code -**

public class Solution {

class Node {

char data;

Node[] children = new Node[26];

int freq = 0;

public Node(char a) {

data = a;

}

}

public String[] prefix(String[] A) {

Node root = new Node('\_');

// insertion

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

Node temp = root;

for (int j=0;j<A[i].length();j++) {

char c = A[i].charAt(j);

int idx = (int)c - (int)'a';

if(temp.children[idx] == null) {

temp.children[idx] = new Node(c);

}

temp.freq += 1;

temp = temp.children[idx];

}

}

// finding prefix

String[] ans = new String[A.length];

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

Node temp = root;

StringBuilder temp2 = new StringBuilder();

for (int j=0;j<A[i].length();j++) {

char c = A[i].charAt(j);

int idx = (int)c - (int)'a';

if (temp.freq == 1) break;

temp2.append(c);

temp = temp.children[idx];

}

ans[i] = temp2.toString();

}

return ans;

}

}

**Q3. Modified Search**

Given two arrays of strings **A** of size **N** and **B** of size **M**.

Return a binary string **C** where **C[i]** = **'1'** if **B[i]** can be found in dictionary **A** using **exactly** one modification in **B[i]**, Else **C[i]** = **'0'**.

**NOTE:** modification is defined as converting a character into another character.

**Logic -**

First insert all the strings of array A in the Trie.

Now, For each query we need to check if there is any string in the dictionary such that we need to change only a single character in B[i].

Think of running a DFS(Depth First search) on Trie. Will it be efficient?

Take a flag variable, make flag = 1 if we have to change the character. Now we cannot do any further change so we will not traverse the part of trie which required more than 1 changes.

Return True if there exist any string such that we need to change a single character.

**Code -**

public class Solution {

public class Node {

char data;

boolean isEnd = false;

Node[] children = new Node[26];

public Node(char x) {

data = x;

}

}

public boolean search(Node root, String A, int flag, int x) {

// If flag > 1 this indicates there are more than 1 changes

if(flag > 1) return false;

// This indicates that the word search has ended

// If the word is completed and modifications = 1 then return true

if (A.length() == x) return flag==1 && root.isEnd;

char c = A.charAt(x);

int idx = (int)c - (int)'a';

boolean ans = false;

for(int i=0;i<26;i++) {

if(root.children[i] == null) continue;

if(i == idx) ans |= search(root.children[i], A, flag, x+1); // If you found that character

else ans |= search(root.children[i], A, flag+1, x+1); // If you did not find that character and made a modification

}

return ans;

}

public String solve(ArrayList<String> A, ArrayList<String> B) {

// Create Trie

Node root = new Node('\_');

for(int i=0;i<A.size();i++) {

String word = A.get(i);

Node temp = root;

for(int j=0;j<word.length();j++) {

char c = word.charAt(j);

int idx = (int)c - (int)'a';

if(temp.children[idx] == null) temp.children[idx] = new Node(c);

temp = temp.children[idx];

}

temp.isEnd = true;

}

// Search

StringBuilder ans = new StringBuilder();

for(int i=0;i<B.size();i++) {

boolean x = search(root, B.get(i), 0, 0);

if(x==true) ans.append("1");

else ans.append("0");

}

return ans.toString();

}

}

**Q4. Maximum XOR Subarray**

Given an array, **A** of integers of size **N**. Find the subarray **AL, AL+1, AL+2, ... AR** with **1**<=**L**<=**R**<=**N,** which has maximum **XOR** value.

**NOTE:** If there are multiple subarrays with the same maximum value, return the subarray with **minimum length**. If the length is the same, return the subarray with the **minimum starting index**.

**Logic -**

Build a **prefXor** array in which the **ith** element represents the xor of all elements from **0** to **i**. To find the xor of any subarray[l..r], we can just take the xor of prefXor[r] and prefXor[l-1].

To find the maximum xor subarray ending at the index **i**, insert the **bit representation**(starting from most significant bit) of all the elements of **prefXor** array upto **i-1** into the trie data structure.

Using the trie Data Structure we can find a pair in prefXor which has the maximum XOR value.  
We have two possible cases at **ith** index.

1. The prefix itself has maximum xor.
2. We need to remove some prefix (ending at index from **0** to **i-1**).Try to have most significant bit to be set bit i.e. **1**. As we have maintained the trie data structure of bit representation of **i-1** elements of prefXor array, we can find the maximum xor in O(logm) where m is the maximum number present in the given array.

We can find the maximum subarray ending at every index and return the subarray, which has the maximum XOR value.

**Code -**

public class Solution {

public class Node {

int data;

Node[] children = new Node[2];

public Node(int data) {

this.data = data;

}

}

public int getMaxXor(ArrayList<Integer> A) {

Node root = new Node(-1);

// Insertion

for(int i=0;i<A.size();i++) {

Node temp = root;

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

int idx = (A.get(i) >> j) & 1;

if(temp.children[idx] == null) temp.children[idx] = new Node(idx);

temp = temp.children[idx];

}

}

// finding maxXor

int ans = Integer.MIN\_VALUE;

for(int i=0;i<A.size();i++) {

Node temp = root;

int xor = 0;

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

int idx = (A.get(i) >> j) & 1;

if(temp.children[1-idx] == null) temp = temp.children[idx];

else {

xor |= (1<<j);

temp = temp.children[1-idx];

}

}

ans = Math.max(ans, xor);

}

return ans;

}

public ArrayList<Integer> solve(ArrayList<Integer> A) {

ArrayList<Integer> pref = new ArrayList<Integer>();

pref.add(0);

for(int i=1;i<A.size()+1;i++) pref.add(A.get(i-1) ^ pref.get(i-1));

int maxXor = getMaxXor(pref);

HashMap<Integer, Integer> map = new HashMap<Integer, Integer>();

int l = -1;

int r = -1;

for(int i=0;i<pref.size();i++) {

// Check if maxXor ^ pref.get(i) is present in map

if(map.containsKey(maxXor ^ pref.get(i))) {

int x = map.get(maxXor ^ pref.get(i));

if(l==-1 && r==-1 || Math.abs(l-r) > Math.abs(i-x)) {

l = Math.min(x,i);

r = Math.max(x,i);

}

}

// Insert current element in map

if(!map.containsKey(pref.get(i))) map.put(pref.get(i), i);

map.put(pref.get(i), i);

}

ArrayList<Integer> ansArr = new ArrayList<Integer>();

ansArr.add(l+1);

ansArr.add(r);

return ansArr;

}

}

**Q5. Map Sum Pairs!**

Rishabh was sitting ideally in his office and then suddenly his boss gave him some operations to perform.

You being his friend tried to help him in finishing the task fast.

So you have to perform **Q** operation of two types:

* **Operation 1: INSERT :** You are given an pair of (string, integer).The string represents the key and the integer represents the value. Insert the **key-value** pair in the hash and If the key already exists in hash, then the original key-value pair will be overridden to the new one.
* **Operation 2: SUM :** you'll be given an pair of (string, -1) where string representing the prefix, and you need to return the sum of all the pairs' value in the hash whose key starts with the prefix.

**Logic – Use Trie data structure with a value and sum**

1. Inserting of key value pairs ->
   1. Search the key if it is already added before. Ie if user adds apple 5 and then again wants to add apple 25 then we must reduce the sum of nodes a p p l e by 5, then we should insert apple 25 in the trie.
   2. If it is not already added we simply insert the new key
2. Find sum -> We search for the key in trie and return the sum associated with it.

**Code -**

public class Solution {

class Node {

char data;

Node[] children = new Node[26];

boolean isEnd = false;

int sum = 0;

int value = 0;

public Node(char d) {

data = d;

}

}

// search in the trie for the given word

public int search(Node root, String word) {

Node temp = root;

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

char c = word.charAt(i);

int idx = (int)c - (int)'a';

if(temp.children[idx] == null) return -1;

temp = temp.children[idx];

}

if(!temp.isEnd) return -1;

return temp.value;

}

// Insert word in trie

public void insert(Node root, String word, int value) {

Node temp = root;

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

char c = word.charAt(i);

int idx = (int)c - (int)'a';

if(temp.children[idx] == null) temp.children[idx] = new Node(c);

temp = temp.children[idx];

temp.value = value;

temp.sum += value;

}

temp.isEnd = true;

}

// Edit the sum of the trie

public void delete(Node root, String word, int value, int x) {

Node temp = root;

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

char c = word.charAt(i);

int idx = (int)c - (int)'a';

temp = temp.children[idx];

temp.sum -= x;

}

}

// Return the sum of the prefix

public int sum(Node root, String word) {

Node temp = root;

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

char c = word.charAt(i);

int idx = (int)c - (int)'a';

if(temp.children[idx] == null) return 0;

temp = temp.children[idx];

}

return temp.sum;

}

public ArrayList<Integer> solve(ArrayList<String> A, ArrayList<Integer> B) {

ArrayList<Integer> ans = new ArrayList<Integer>();

Node root = new Node('\_');

for(int i=0;i<A.size();i++) {

// return sum of the prefix

if(B.get(i) == -1) ans.add(sum(root, A.get(i)));

else {

// Else search word in the trie

int x = search(root, A.get(i));

// If word is found change its sum value then insert

if(x!=-1) delete(root, A.get(i),B.get(i), x);

// If word is not found simply insert it

insert(root, A.get(i), B.get(i));

}

}

return ans;

}

}

**Q6. Valid Phone Directory!**

Given a phone directory in the form of string array **A** containing **N** numeric strings.

If any phone number is prefix of another phone number then phone directory is **invalid** else it is **valid**.

You need to check whether the given phone directory is valid or not if it is valid then return **1** else return **0**.

**Logic – Use trie data structure with a frequency variable**

1. While inserting the strings in the trie, check if the frequency is less than 1.
2. If it becomes 2 it means a number with same prefix is found. So. return 0 in that case.

**Code -**

public class Solution {

public class Node {

char data;

Node[] children = new Node[10];

int freq = 0;

public Node(char d) {

data = d;

}

}

public int solve(ArrayList<String> A) {

Node root = new Node('\_');

for(int i=0;i<A.size();i++) {

Node temp = root;

for(int j=0;j<A.get(i).length();j++) {

char c = A.get(i).charAt(j);

int idx = Character.getNumericValue(c);

if(temp.children[idx] == null) temp.children[idx] = new Node(c);

temp = temp.children[idx];

temp.freq += 1;

if(temp.freq > 1) return 0;

}

}

return 1;

}

}