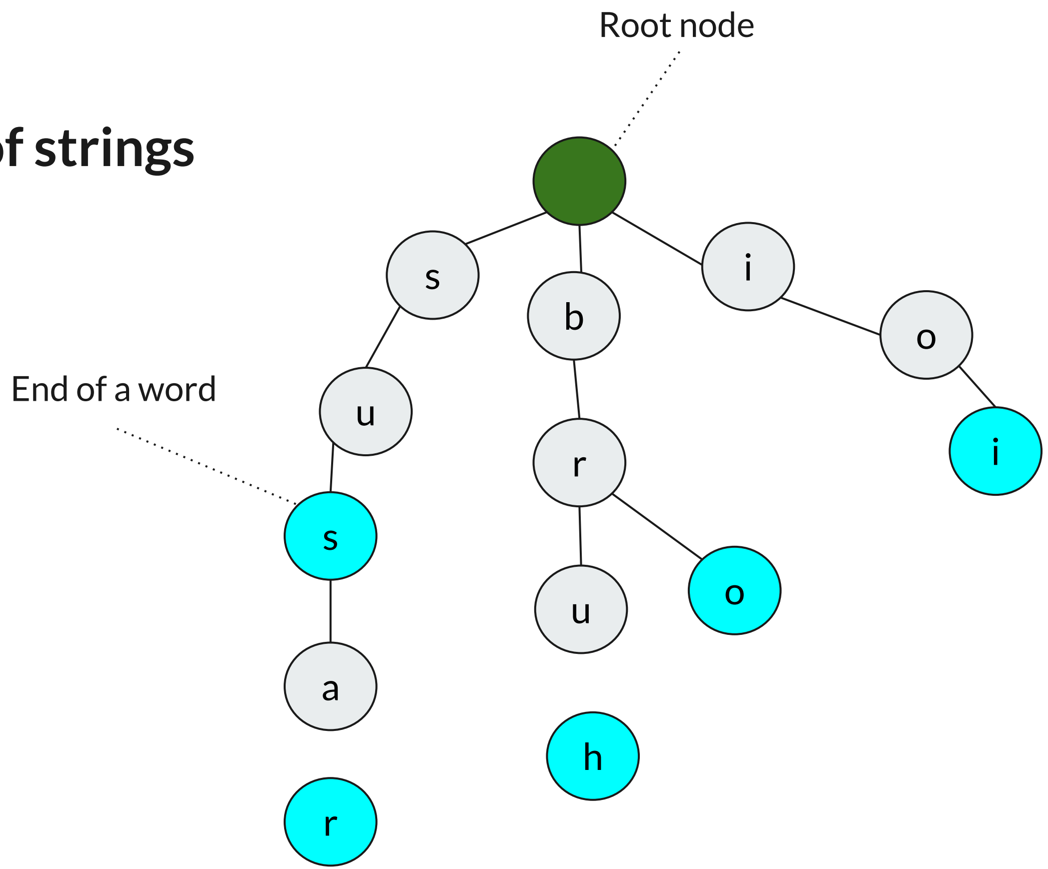
Prefix tree/Trie

Youkyum Kim

Used to store a collection of strings

<u>Example:</u>

sus
bruh
sussy
bro
ioi



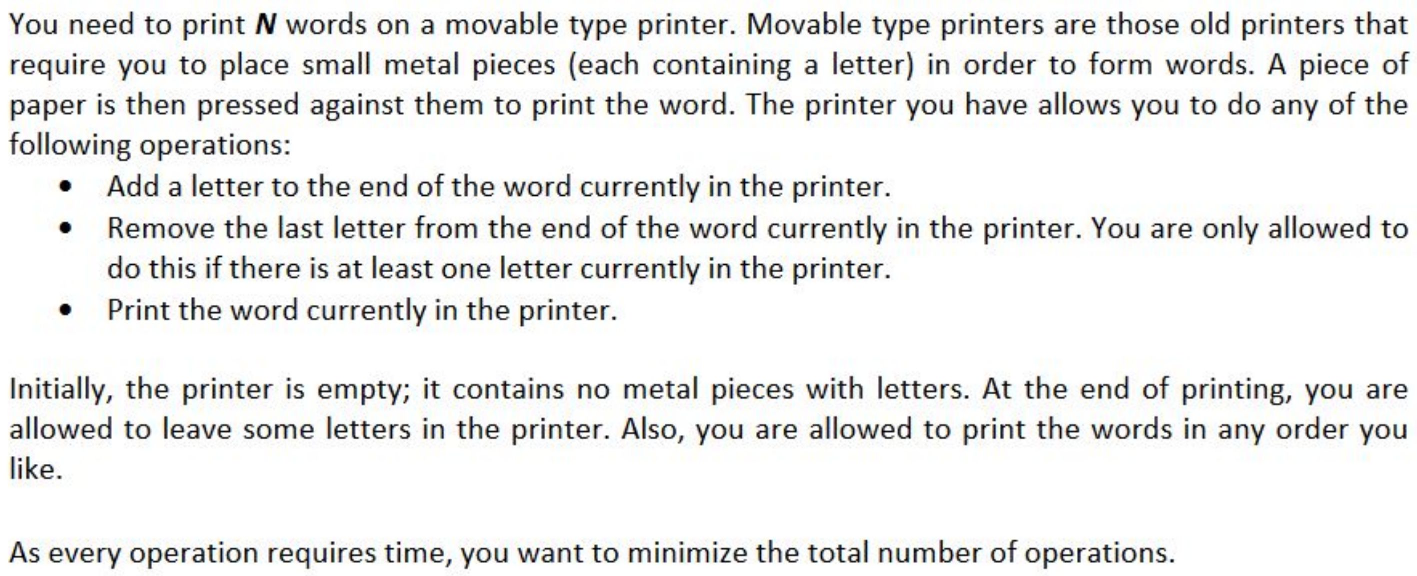
Implementation

2d array (ex. trie[n][26]) the 26 is for each letter in the alphabet (Does not have to be 26!)

An array endofword[n] for determining whether a node is the end of some string

Other features can easily be added depending on the problem

Direct Application: IOI 2008 Printer



You need to print ***N*** words on a movable type printer. Movable type printers are those old printers that require you to place small metal pieces (each containing a letter) in order to form words. A piece of paper is then pressed against them to print the word. The printer you have allows you to do any of the following operations:

- Add a letter to the end of the word currently in the printer.
- Remove the last letter from the end of the word currently in the printer. You are only allowed to do this if there is at least one letter currently in the printer.
- Print the word currently in the printer.

Initially, the printer is empty; it contains no metal pieces with letters. At the end of printing, you are allowed to leave some letters in the printer. Also, you are allowed to print the words in any order you like.

As every operation requires time, you want to minimize the total number of operations.

TASK

You must write a program that, given the ***N*** words you want to print, finds the minimum number of operations needed to print all the words in any order, and outputs one such sequence of operations.

CONSTRAINTS

1 <= ***N*** <= 25,000	The number of words you need to print.

INPUT

Your program must read from the standard input the following data:

- Line 1 contains the integer ***N***, the number of words you need to print.
- Each of the next ***N*** lines contains a word. Each word consists of lower case letters ('a' – 'z') only and has length between 1 and 20, inclusive.
  All words will be distinct.

OUTPUT

Your program must write to the standard output the following data:

- Line 1 must contain an integer ***M*** that represents the minimum number of operations required to print the ***N*** words.
- Each of the next ***M*** lines must contain one character each. These characters describe the sequence of operations done. Each operation must be described as follows:
  - Adding a letter is represented by the letter itself in lowercase
  - Removing the last letter is represented by the character '-' (minus, ASCII code 45)
  - Printing the current word is represented by the character 'P' (uppercase letter P)

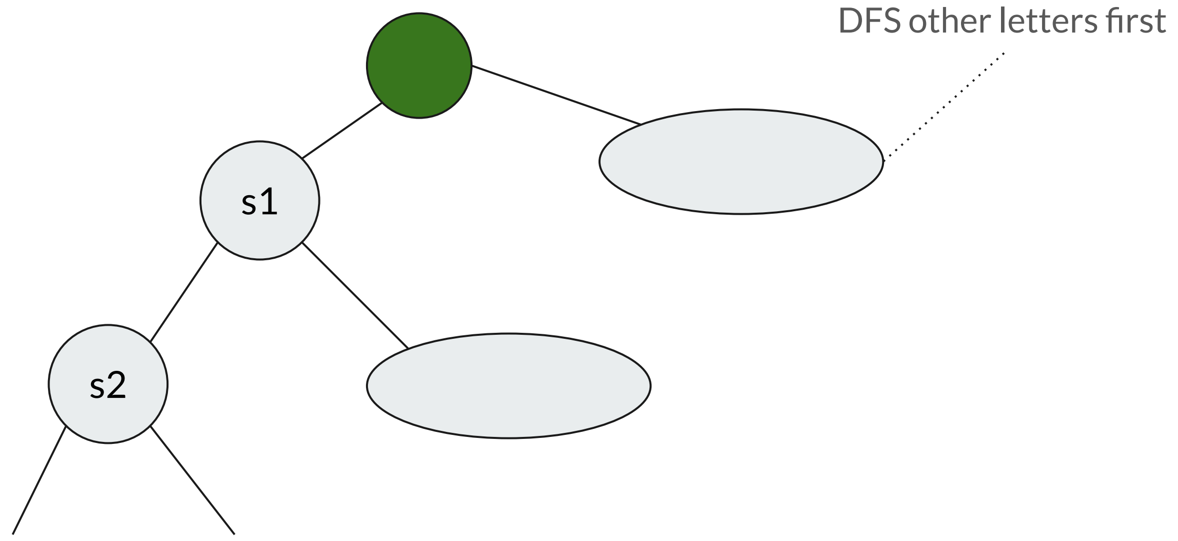
1st observation: includes a collection of strings, thus we can use trie

2nd observation: notice that DFS of the trie starting at root node will be able to print every word once.

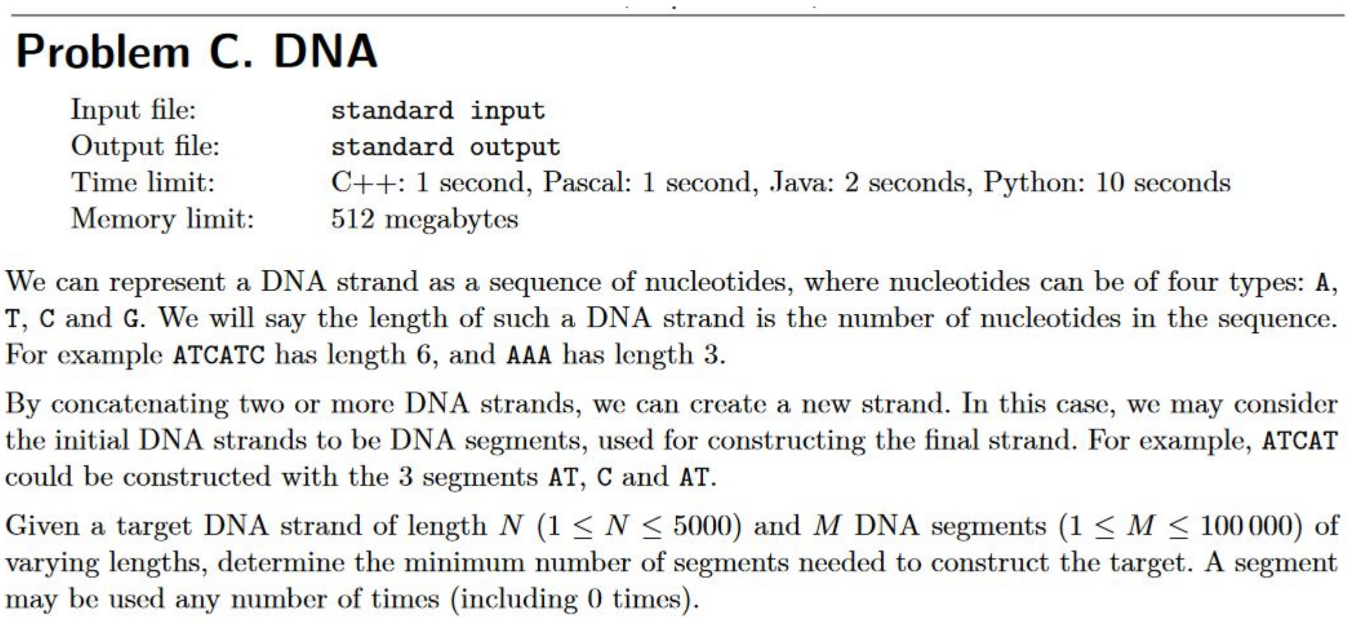
-When we move "forwards" in the trie, we print the letter.
-When we move "backwards" we print '-'
-When we visit a node that is the end of a string, we print 'P'.
(keep a visited array so that we print P at most once for a node)

However notice the last word's letters are not removed from the printer.

Thus, we need to make sure to print the longest word last.
(This can be done by simply getting the longest word $w_1w_2w_3 \ldots w_k$ and making sure those letters are chosen last in DFS)


DFS other letters first
s1
s2

Trie used for optimisation: SAPO 2019 dna



Problem C. DNA

Input file:	`standard input`
Output file:	`standard output`
Time limit:	C++: 1 second, Pascal: 1 second, Java: 2 seconds, Python: 10 seconds
Memory limit:	512 megabytes

We can represent a DNA strand as a sequence of nucleotides, where nucleotides can be of four types: `A`, `T`, `C` and `G`. We will say the length of such a DNA strand is the number of nucleotides in the sequence. For example `ATCATC` has length 6, and `AAA` has length 3.

By concatenating two or more DNA strands, we can create a new strand. In this case, we may consider the initial DNA strands to be DNA segments, used for constructing the final strand. For example, `ATCAT` could be constructed with the 3 segments `AT`, `C` and `AT`.

Given a target DNA strand of length N ($1 \leq N \leq 5000$) and M DNA segments ($1 \leq M \leq 100\,000$) of varying lengths, determine the minimum number of segments needed to construct the target. A segment may be used any number of times (including 0 times).

Input

The first line contains the 2 integers N and M, separated by a space.

The second line contains the target DNA strand consisting of N characters, where each character is one of `A`, `T`, `C` or `G`.

For each segment i ($1 \leq i \leq M$), an additional 2 lines follow. The first of these contains the integer L_i (the length of segment i). The second line contains a single DNA segment consisting of L_i characters, where each character is one of `A`, `T`, `C` or `G`. It is guaranteed that for each segment, $1 \leq L_i < N$. It is guarenteed that the sum of the lengths of all segments is no greater than 1 000 000.

Output

Output a single integer, the minimum number of segments required to construct the target strand. If it is not possible to construct the target strand, output -1 instead.

1st step: DP
Set dp[i]=min no. of segments needed for first i letters (dp[i]=-1 if there are no such segments)

dp[k]=min(dp[c] where c<k and substring sc…sk is a segment)
If there are no such values, then dp[k]=-1

Now for a value k, we need to find all the possible substrings of the dna ending on sk that is also a segment (in O(n) time)

However, we can then simply store all the segments in a trie and traverse down the trie according to the letters sk, sk-1, …, s1 and check whether a node is the end of a segment.

Thus the problem is done