Alice took File Compression as her term project. Supervisor suggested her to do everything on her own. She devised her own algorithm for compression. Her algorithm converts any file to its string representation S. Then she encodes the string with series of operations. She starts processing the string from its end. At any step she does any of the following two operations:

1. Encode the last character of the remaining non-encoded string. This last character encoding needs x unit of memory.

   Encode$(S[1, 2, 3, \ldots, n])$ = **Encode**$(S[1, 2, 3, \ldots, n-1])$ + **information of** $S[n]$.

   Where n is the length of string S.

2. Encode a suffix of the remaining non-encoded string. She can encode any suffix that satisfies the folding property. Suffix starting at index i has the folding property if it's the mirror of a substring ending at index $i-1$. For example, both **suffix**("`aabbaabba`", 8) and **suffix**("`aabbaabba`", 6) have the folding property. Any possible suffix encoding needs y unit of memory.

   Here, **suffix**(S, i) is the suffix of string S starting at index i.

   Encode$(S[1, 2, 3, \ldots, i-1, i, \ldots, n-1, n])$ = **Encode**$(S[1, 2, 3, \ldots, i-1])$ + **information of suffix**(S, i).

   Where, **suffix**(S, i) satisfies the folding property.

Alice wants to know the performance of her compression algorithm. She has collection of files for performance testing. She converted each file to its corresponding string representation. Now, needs your help in determining the total memory unit each file needs after compression.

Input

Input starts with a line with number of test cases T $(1 \leq T \leq 25)$. Each of the following T lines has information about a single file. Each line has two integer x, y $(1 \leq x, y \leq 1000)$ and a string S $(1 \leq |S| \leq 1000000)$. S comprises only of lowercase letter.

Output

Output contains T lines. Each of them is in format '`Case` t`:` c'. t is the test case number and c is the amount of memory needed by the compressed string.

Test Case Analysis:

Optimal compression steps of case 1 are "`aabbaa`" `->` "`aab`", "`aab`" `->` "`aa`", "`aa`" `->` "`a`", "`a`" `->` "". First and third step require cost 1 because of suffix encoding. Second and fourth step require cost 2. So, the total cost is $1 + 2 + 1 + 2 = 6$.

Sample Input

```
2
2 1
aabbaa
1 1
aabbaabba
```

Sample Output

```
Case 1: 6
Case 2: 5
```