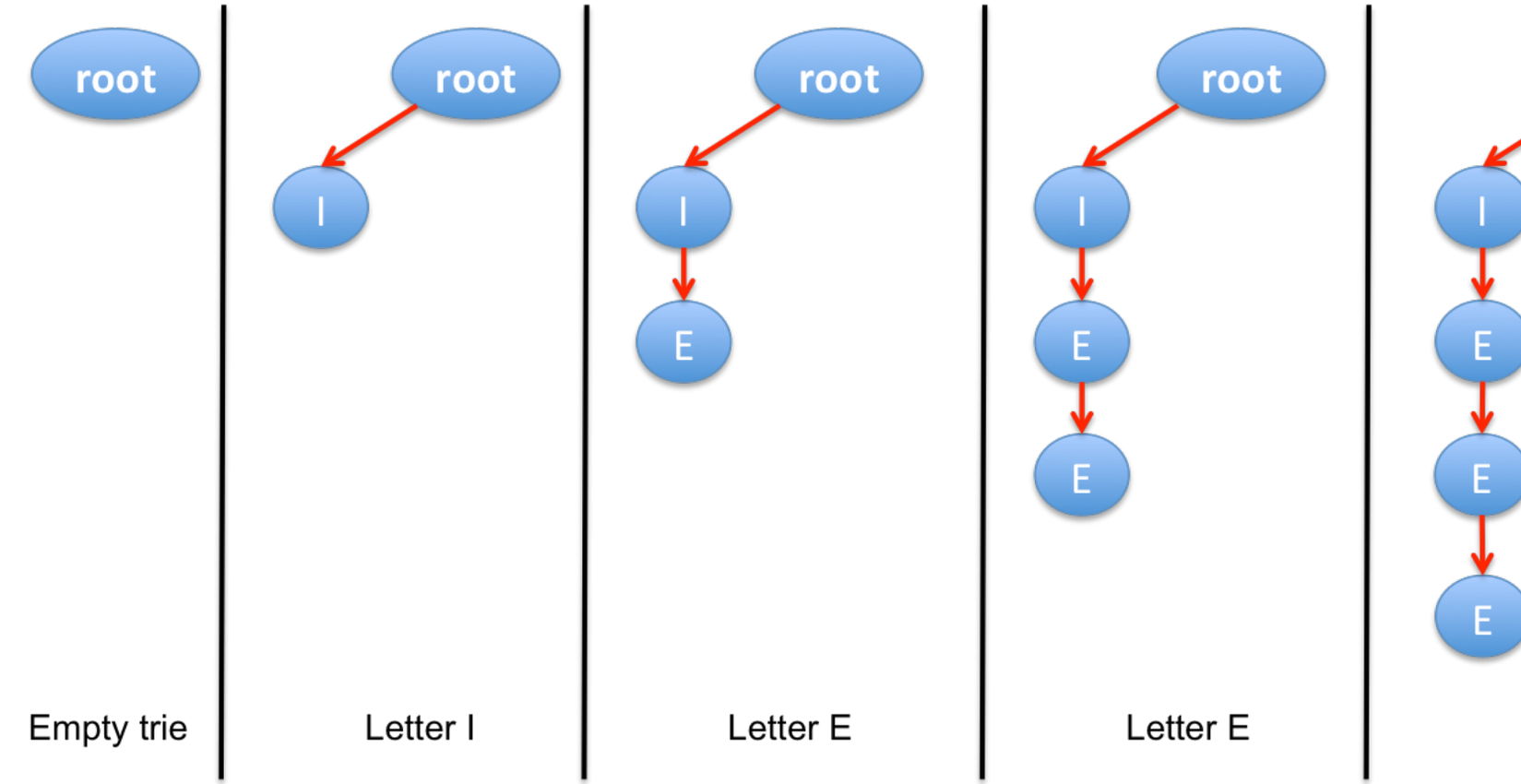
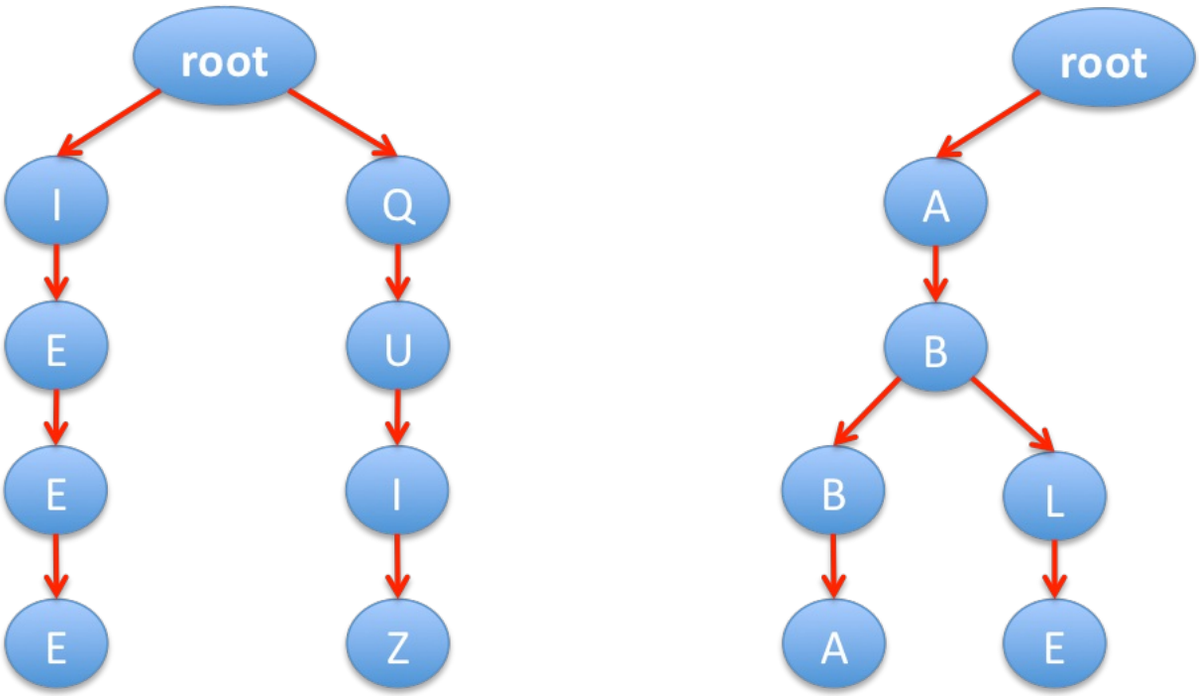
Playing with Tries

Playing with Tries

Mike has always been fond of playing with words. Recently, he learned programming and started to write programs to find all sorts of things related with texts. However, his programs started to run very slowly when the number of words was very high. Luckily, he found a beautiful data structure to help him called Trie.
Mike uses a Trie like a tree where each edge is labeled with a letter. To build a Trie, Mike first creates a root node. Then, for each word he wants to store in the Trie, he descends from the root node creating nodes whenever necessary. For example, if Mike wants to add “IEEE” to an empty Trie, he follows the steps in the following picture:



Each blue oval represents a node. Each node contains a link to each letter of the alphabet. These nodes can be reused for words with the same prefix. Take “ABBA” and “ABLE” for example. They have the same prefix, so they share the same first two nodes in the Trie as shown the following picture.



On the left, a Trie with the words “IEEE” and “QUIZ”. On the right, a Trie with “ABBA” and “ABLE”
The Tries shown above use the 26 letters of the English alphabet. So, each node contains 26 links despite using just one or two of them (e.g. the root node on the left Trie and first node labeled with B on the right Trie use 2 links each, whereas all the remaining nodes on both Tries use only a single link out of the 26 available letters). Each link needs 4 bytes. So, the Trie on the left consumes 9 nodes times 26 links times 4 bytes = 936 bytes. The Trie on the right uses 7 nodes, so it consumes 728 bytes.
Mike is going to play in many words, but hes worried that his Trie will consume too much memory.

Task

Given the size of the alphabet **A** , the maximum length of the words **L** and the number **N** of words to consider, your task is to find what is the maximum amount of bytes necessary to store **N** words in a Trie.

Input

The first line of input contains an integer **T** (1 <= T <= 10^5) the number of test cases.
Then, T lines follow, each containing three integers **A** (1 <= A <= 26), **L** (1 <= L <= 10^5) and **N** (1 <= N <= 10^18), separated by a single space.

Output

For each test case, your program should print one line with a single integer: the worst-case amount of memory necessary.
Note1: The result is guaranteed to fit in a signed 64-bit integer.
Note2: There is a newline character at the end of the last line of the output.

Sample Input

```
4
26 4 2
3 3 10
3 4 3
4 5 101
```

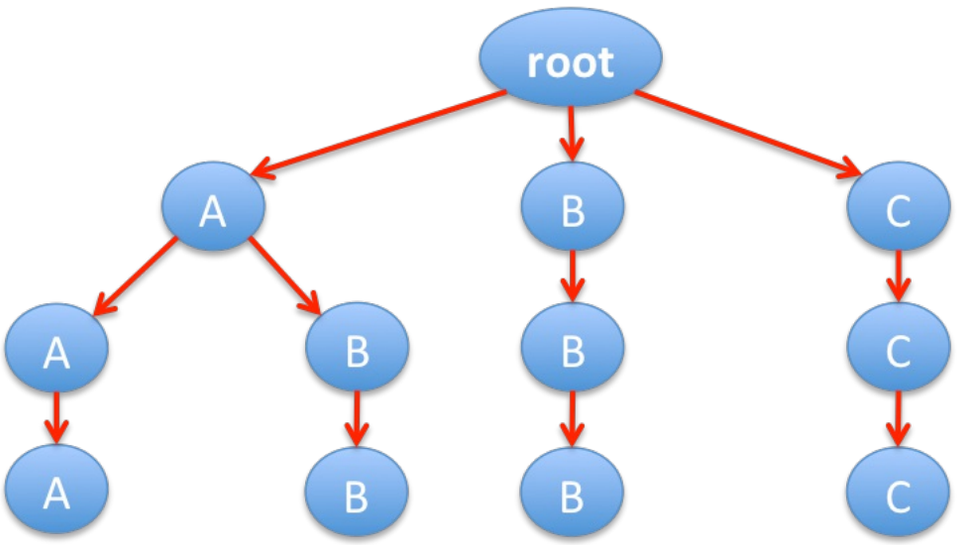
Sample Output

```
936
276
156
4592
```

The first sample input corresponds to the example given in the problem statement.
In the second sample, we have only 3 letters in the Alphabet (e.g. [A, B, C]), 10 words (**N** =10) and a maximum size of 3 letters per word (**L** =3). A possible set of words fulfilling the aforementioned criteria could be the following:

Word 1: A
Word 2: AA
Word 3: AAA
Word 4: B
Word 5: BB
Word 6: BBB
Word 7: C
Word 8: CC
Word 9: CCC
Word 10: ABB

For this example, a Trie that would be suitable for storing these words would look as follows:

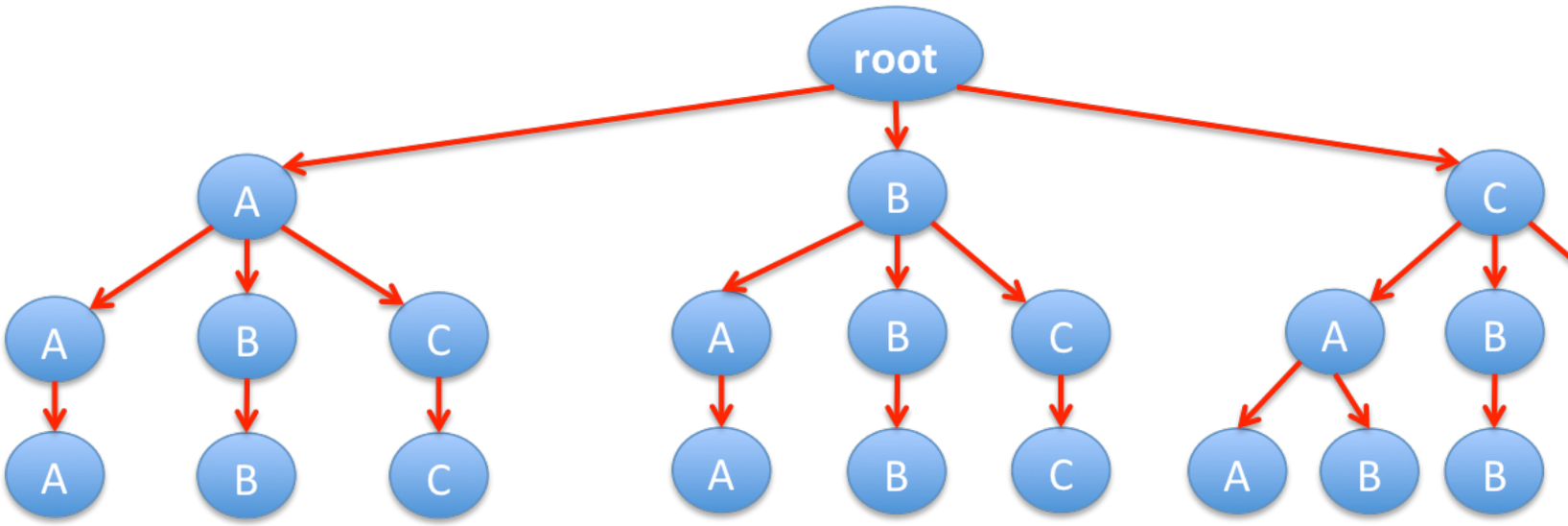


This Trie has 12 nodes so it consumes 12 * 3 * 4 = 144 bytes. However, there might be another set of words which again fulfill the defined criteria for which a bigger Trie would be required to store those words. For example, if the set of 10 words were as follows:

Word 1: AAA
Word 2: ABB
Word 3: ACC
Word 4: BAA

Word 5: BBB
Word 6: BCC
Word 7: CAA
Word 8: CAB
Word 9: CBB
Word 10: CCC

Then, the following Trie would be required, consuming 23 * 3 * 4 = 276 bytes.



Although other Tries would be required for different sets of words, they would all consume less than or equal to 276 bytes. Therefore, the Trie displayed above is one of the worst-case scenarios (i.e. this Trie and possibly several would require at maximum 276 bytes), and for this reason the reported result is 276 bytes.