Iterables and Iterators

The problem text is attempting to lead you to look at itertools. It's good to be familar with this module, as it has many functions that are useful for creating and managing iterators.

The relevant function in this case is `itertools.combinations()`, which yields all combinations of a given length chosen from the items of an input iterable. This is actually a generator, which means that we don't need to allot memory to store all possible combinations. Instead, we generate them one at a time and see which contain an *a*. Since we choose combinations uniformly, the probably of getting one with at least one *a* is just the ratio of counts.

```
import itertools

def calc_prob(ltrs, k):
    count = total = 0
    for comb in itertools.combinations(ltrs, k):
        total += 1
        if 'a' in comb:
            count += 1

    return count / total

if __name__ == '__main__':
    import sys
    _ = sys.stdin.readline()
    ltrs = sys.stdin.readline().strip().replace(' ', '')
    k = int(sys.stdin.readline())
    print(calc_prob(ltrs, k))
```

Math is Delicious

We don't *actually* need to work out all the combinations; all we need to come up with is the total number of them. The total (unordered) combinations of *k* elements from *n* choices is *k* choose *n*. It's easier to work out the number of combinations that do *not* have an *a* in them: If *a* shows up *m* times, there are (*n*-*m*) choose *k* combinations without an *a*, and a little subtraction gives us the number of combinations with an *a*.

```
import math
```

```
def nCk(n, k):
    if k > n:
        return 0
    return math.factorial(n) / math.factorial(k) / math.factorial(n - k)

def calc_prob(ltrs, k):
    num = len(ltrs)
    not_a = len([l for l in ltrs if l != 'a'])
    return 1 - nCk(not_a, k) / nCk(num, k)
```

And less filling

Another method for dealing with this is to dodge the problem. In particular, computing with factorials gets expensive as *n* or *k* get large (but it's fine for the input sizes used on hackerrank). We can instead compute the probability of drawing up *k* values that aren't *a* directly: out of *n* choices, if *notA* of them aren't an *a*, the probability of choosing one that's not *a* is (notA)/n, then the probability of choosing a second one if we did get not an *a* is (notA - 1)/(n - 1), and so on. The probability is then the product of these, and taking that from *1* gives us the probability we're looking for.

```
def calc_prob(ltrs, k):
    n = len(ltrs)
    not_a = len([letter for letter in ltrs if letter != 'a'])
    bad_prob = 1
    for i in range(0,k):
        bad_prob *= (not_a - i) / (n - i)

    return 1 - bad_prob
```