Diverse Strings

locked

by kevinsogo

Editorial by kevinsogo

There are a few things we can immediately get from the fact that we're looking for the shortest and lexicographically smallest such string:

- The first k lowercase letters of the alphabet will be used.
- The string begins and ends with `aa` (for $n > 1$).
- The letter `a` appears more times than any other letter.
- In fact, each letter appears strictly more times than the subsequent letter in the alphabet.

The first task is to determine whether (n, k) is possible; we can then try to solve for the lexicographically smallest one later.

One helpful observation is the following; if (n, k) is possible, then (n', k) is also possible; we can simply add $n' - n$ `a`s in front. Given this, we can simply try to compute, for each k, the smallest n_k such that (n_k, k) is possible. Then (n, k) is possible if and only if $n \geq n_k$.

Now, one common first guess on the answer is the following:

```
k n_k string
1 1   a
2 5   aabaa
3 11  aaabbcbbaaa
4 19  aaaabbbccdccbbbaaaa
...
```

Unfortunately, this doesn't work; in fact, the strings for $k \geq 3$ are not even diverse!

One could try to adjust it by making the runs fatter, like so:

```
k n_k string
1 1   a
2 5   aabaa
3 15  aaaaabbcbbaaaaa
4 37  aaaaaaaaaaabbbbbccdccbbbbbaaaaaaaaaaa
...
```

Now, the strings are diverse; unfortunately, they're not optimal! For example, for $k = 3$, we have the following shorter string ($n = 13$): `aaababcbabaaa`.

To proceed, we need to get some more insights. One important insight is the following: if we have a diverse string with k distinct letters, and we ignore one of the letters, then we get a diverse string with $k - 1$ distinct letters. (Can you see why?) For example, given

Statistics

Difficulty: Hard
Time Complexity: $O(n)$ Required
Knowledge: Greedy algorithms
Publish Date: Jan 17 2018

`aaababcbabaaa` which has 3 distinct letters, if we ignore `a`, then we get `bbcbb` which indeed has 2 distinct letters. This allows us to prove some lower bounds on n_k.

The most natural one is that $n_k > n_{k-1}$, but this is a little weak. To get a tighter bound, let's try ignoring the *most occurring* letter (which will be `a`).

More formally, let's define $f(k)$ is the fewest number of times the most occurring letter appears in *any* diverse string with k distinct letters. Now, consider our diverse string with n_k letters and k distinct letters. If we ignore `a`, then we get a diverse string with $k-1$ distinct letters. The most occurrence of any letter here would be $\geq f(k-1)$. That letter, `b`, would be the second most-occurring letter in our original string. By continuing this, we see that the rth most occurring letter in our string appears $\geq f(k-r+1)$ times. Thus, the shortest diverse string contains at least $f(1) + f(2) + \ldots + f(k)$ letters, so we have the inequality

$$n_k \geq f(1) + f(2) + \ldots + f(k).$$

Now, for our original string to be assorted, there must be $f(k-1)+1$ occurrences of `a` before the last occurrence of `b`. Hence, we have the inequality $f(k) \geq f(k-1) + 1$. However, we also assert that the string must end with `aa`, hence there are two more appearances of `a`, and we get the inequality

$$f(k) \geq f(k-1) + 3.$$

Now, since $f(1) = 1$, we get by induction that $f(k) \geq 3k - 2$. Thus, we get the following inequality:

$$n_k \geq \sum_{r=1}^{k} f(r) \geq \sum_{r=1}^{k} (3r-2) = \frac{k(3k-1)}{2}.$$

We still haven't proven that n_k is actually equal to this bound or perhaps higher for some k. So let's attempt to construct a diverse string with these lengths.

Luckily, we can! Consider $f(k) \geq f(k-1) + 3$. We would like to arrange `a`'s and `b`'s such that `a` appears exactly three more times than `b`, and the resulting string is diverse. An arrangement like this would be sufficient:

```
aabababababababababababaa
```

By ensuring that consecutive letters appear like this in our string, we can construct a diverse string where each letter appears exactly three more times than the next letter. For example,

```
aababacbacbadcbadcbaedcbaedcbafedcbaedcbadcbacbabaa
```

If we consider only consecutive pairs of letters, we get:

```
a & b: aababababababaebaebaebaebababababaa
b & c: bbcbcbcbcbecbecbecbecbcbcbb
c & d: ccdcdcedcedcedcedcdcc
d & e: ddedededededd
e & f: eefee
```

This construction tightens the bound $f(k) = f(k-1) + 3$ and proves that:

$$n = \frac{k(3k-1)}{2}$$

Tabulating the values for n_k, we get:

k	1	2	3	4	5	6	7	8	9	10	...
n_k	1	5	12	22	35	51	70	92	117	145	...

The optimal strings look like this:

```
k n_k string
1 1   a
2 5   aabaa
3 12  aababacbabaa
4 22  aababacbacbadcbacbabaa
...
```

As a bonus, we can also see that these are the *lexicographically smallest* ones for each k since each consecutive pair of letters can actually only be arranged in one way: `aababababababababababaa`, thus drastically limiting our choices.

Finally, for $n > n_k$, we simply greedily append $n - n_k$ `a`s in front of the optimal one for (n_k, k). Proving that this is the lexicographically smallest one requires more tedious argumentation but it should not be difficult to believe it. We will leave it up to you to convince yourself why that works. To guide you towards the proof, try proving the following in sequence:

- The optimal string for (n, k) begins with at least $n - n_k + 2$ `a`s.
- There are at least $f(k-1)+1$ occurrence of `a` after the first occurrence of `b`.
- `a` appears exactly $n - n_k + f(k)$ times.
- The rth most occurring letter appears exactly $f(k-r+1)$ times.
- The optimal string is $n - n_k$ `a`s followed by the optimal string for (n_k, k).

 Set by kevinsogo

Problem Setter's code :

```
from string import lowercase
def realize(src, seq):
    return ''.join(src[v] for v in seq)

construct = [[], [0]]
for n in xrange(2, 26 + 1):
    res = [n-1, n-1]
    for v in construct[-1]:
        res.append(v)
        if v == n-2:
            res.append(n-1)

    construct.append(res + [n-1])

construct = [realize(lowercase[:n][::-1], seq) for n, seq in enumerate(cons
truct)]

for cas in xrange(input()):
    n, k = map(int, raw_input().strip().split())
    d = n - len(construct[k])
    if d >= 0:
        print 'a' * d + construct[k]
    else:
        print 'NONE'
```

 Tested by Xellos0

Problem Tester's code :

```
#include <bits/stdc++.h>
// iostream is too mainstream
#include <cstdio>
// bitch please
#include <iostream>
#include <algorithm>
#include <cstdlib>
#include <vector>
#include <set>
#include <map>
#include <queue>
#include <stack>
#include <list>
#include <cmath>
#include <iomanip>
#include <time.h>
#define dibs reserve
#define OVER9000 1234567890
#define ALL_THE(CAKE,LIE) for(auto LIE =CAKE.begin(); LIE != CAKE.end(); LI
E++)
#define tisic 47
#define soclose 1e-8
#define chocolate win
// so much chocolate
#define patkan 9
#define ff first
#define ss second
#define abs(x) ((x < 0)?-(x):x)
#define uint unsigned int
#define dbl long double
#define pi 3.14159265358979323846
using namespace std;
// mylittledoge

typedef long long cat;

#ifdef DONLINE_JUDGE
        // palindromic tree is better than splay tree!
        #define lld I64d
#endif

int main() {
        cin.sync_with_stdio(0);
        cin.tie(0);
        cout << fixed << setprecision(10);
        int T;
        cin >> T;
        string ans0[100];
        ans0[1] ="z";
        for(int k =2; k <= 26; k++) {
                int L =ans0[k-1].length(), D =3*k-4;
                string newch;
                newch +=('z'-k+1);
                vector< vector<string> > dp[2];
                for(int i =0; i < 2; i++) dp[i].resize(L+1,vector<string>(D
+1));
                for(int i =0; i <= L; i++) for(int j =0; j <= D; j++) {
                        if(j < D && !(dp[0][i][j] == "" && i+j > 0)) {
                                string s =dp[0][i][j]+newch;
                                if(dp[1][i][j+1] == "") dp[1][i][j+1] =s;
                                dp[1][i][j+1] =min(dp[1][i][j+1],s);
                        }
                        if(i < L && !(dp[1][i][j] == "" && i+j > 0)) {
                                if(ans0[k-1][i] == 'z'-k+2) {
                                        string s =dp[1][i][j]+ans0[k-1][i];
                                        if(dp[0][i+1][j] == "") dp[0][i+1][
j] =s;
                                        dp[0][i+1][j] =min(dp[0][i+1][j],s
);
                                }
                                else {
                                        string s =dp[1][i][j]+ans0[k-1][i];
                                        if(dp[1][i+1][j] == "") dp[1][i+1][
```

```
j] =s;
                                        dp[1][i+1][j] =min(dp[1][i+1][j],s
);
                                }
                        }
                        if(i < L && !(dp[0][i][j] == "" && i+j > 0) && ans0
[k-1][i] != 'z'-k+2) {
                                string s =dp[0][i][j]+ans0[k-1][i];
                                if(dp[0][i+1][j] == "") dp[0][i+1][j] =s;
                                dp[0][i+1][j] =min(dp[0][i+1][j],s);
                        }
                }
                ans0[k] =newch+dp[1][L][D]+newch;
        }
        while(T--) {
                int N,K;
                cin >> N >> K;
                string S =ans0[K];
                if((int)S.length() > N) {
                        cout << "NONE\n";
                        continue;
                }
                string ans;
                for(int i =0; i < N-(int)S.length(); i++) ans +='a';
                for(int i =0; i < (int)S.length(); i++) ans +=S[i]-(26-K);
                cout << ans << "\n";
        }
        return 0;}

// look at my code
// my code is amazing
```