技術室奥プログラミングコンテスト#5 Day1-N 数え上げを愛したい(文字列編) 解説

kaage(@ageprocpp)

お疲れ様でした！

コンテスト参加ありがとうございます。
コンテスト結果は次のようになりました。

AC Count: 17
First AC: HIR180 (04:07)
Shortest Code:
Fastest Code:

原案は **sanadayukimura73** が作りましたが、**kaage** が制約を上げて今の問題になりました。

問題概要

英小文字からなる文字列 S がある。

S から任意の文字を抜き出して並べ替えることにより得られる文字列の総数を求めよ。

$1 \leq |S| \leq 3 \cdot 10^5$

$O(S!2^{|S|})$ 解法

使う文字と並べ方を全探索する

$|S| = 8$ くらいまではこれで解ける

全然本題ではないので飛ばします

基本的な考察

文字は自由に選べるし自由に並び替えられるので、もとの文字列の並び順は一切関係ない
気にするべきなのは各文字の出現数だけ

たとえば、サンプル 1 ‘yuu’ なら、‘y’ が 1 個と ‘u’ が 2 個

このように、高々 26 種類の文字の個数だけまとめて考えると、自ずから次の解法が思いつく

$O(|S|^2)$ 解法

先に述べたように、それぞれの種類の文字の個数だけで通り数を決定できるので、

すでに生成された distinct な文字列に、1 種類ずつ文字をいくつか挿入して新たな文字列を作ることを考えることができる

もともと存在する文字列は distinct であり、かつ挿入方法も同じ文字列に対しては異なるようにすれば、帰納法から、この操作をいくら行っても生成される文字列が distinct であることは示せる

よって、$dp[i][j] =$ ラテン文字の $i-1$文字目まで使って、長さ j の文字列を作る方法の数 として

DPをすれば、$O(|S|^2)$ でこの問題が解ける

$O(|S|^2)$ 解法

遷移式は簡単だが、次の考察で使うのでここでも考えてみる

$dp[i][j]$ に対して、文字 $i+1$ を k 個挿入する方法は、どこに挿入できるかを考えれば、

${}_{j+1}\mathrm{H}_k = {}_{j+k}\mathrm{C}_k$ 通りある

よって、

$dp[i+1][j+k]$ には $dp[i][j]$ から ${}_{j+k}\mathrm{C}_k$ 通りの遷移が考えられる

DPの高速化

実はこのDPを高速化することでこの問題は解けるが、遷移の係数の計算や遷移元の計算に時間がかかるわけではないので、DPで解くならDP配列そのものをデータ構造などに載せることを考えなければいけない（このような考察はこの問題以外でもたまに重要になる）
先ほどの遷移式を変形してみる

$$dp[i+1][j+k] \leftarrow {}_{j+k}\mathrm{C}_{k} dp[i][j] = \frac{(j+k)!}{j!k!} dp[i][j]$$

係数は $j, k, j+k$ にしか依存しないので、実はもとの文字数と挿入する文字数だけから高速に計算できる

DPの高速化

ここで、i 番目の文字の数を c_i として、

$$f_i(x) = \sum_{j=0}^{|S|} \frac{1}{j!} dp[i][j] x^j, g_i(x) = \sum_{j=0}^{c_i} \frac{1}{j!} x^j$$

としてやれば、i 番目の文字の挿入についてのDP配列の遷移を再現できて、多項式 $P(x)$ について、その j 次の係数を $\mathrm{coeff}_j P$ とでも表すこととすれば、

$$f_{i+1}(x) = \sum_{j=0}^{|S|} j! \mathrm{coeff}_j(f_i(x) g_i(x))$$

とできる

DPの高速化

これは多項式乗算なので、NTT(Number Theoretic Transform) を使って $O(|S|\log|S|)$ で行うことができ、この問題が解けた

文字数の回数ぶん行う必要があるので時間がかかりそうに思えるが、先に文字ごとに個数を昇順ソートしておけば、計算量は十分落ちる

また、次数を 2 冪に合わせようとすると、場合によって次数が指数関数的に増えて大変なことになるため、高次で係数が 0 の項を消す、などの工夫も必要になる

再帰 NTT だと問答無用で落ちますが、これは重要な定数倍改善なので、許して

C++ なら、非再帰 NTT & 可変 mod で通ることは確認しています（mod を定数で固定すれば余裕があるはずです）

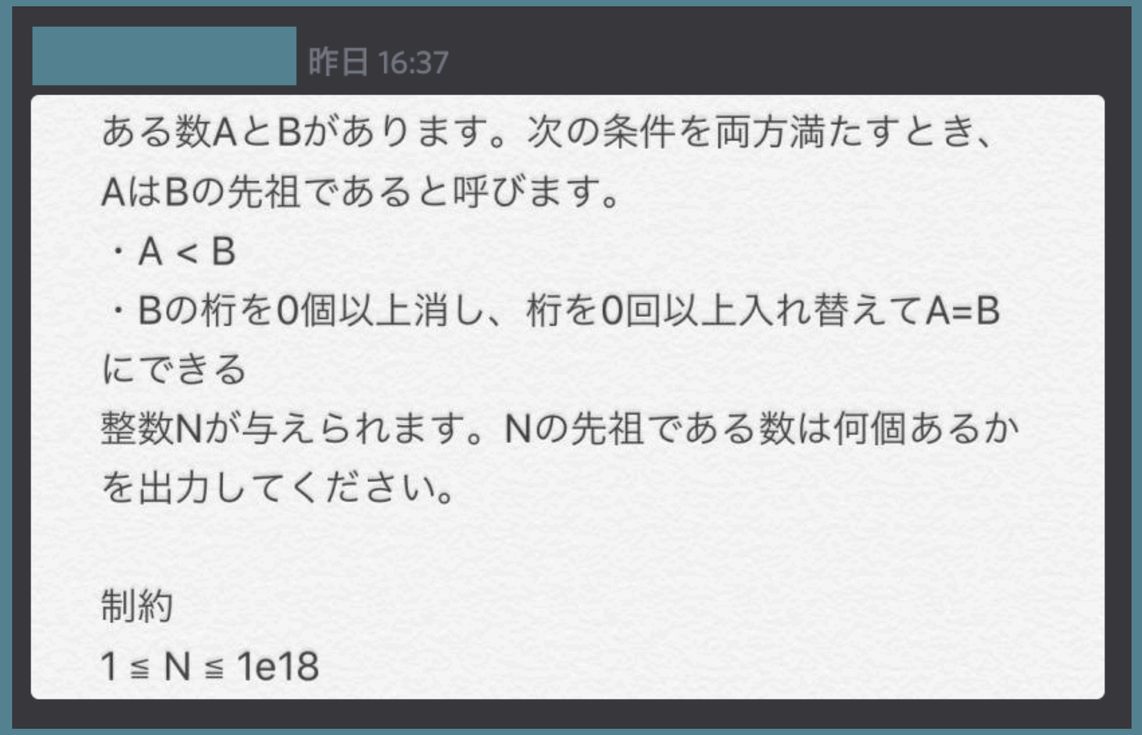
DPの高速化

アルゴリズム本体を除けば実装は簡単です

```
std::string s;
int a[26];
ModInt fact[20010],inv[20010];
int main(){
    std::cin>>s;
    rep(i,s.size())a[s[i]-'a']++;
    std::sort(a,a+26);
    fact[0]=1;
    REP(i,s.size())fact[i]=fact[i-1]*i;
    inv[s.size()]=ModInt(1)/fact[s.size()];
    for(int i=s.size()-1;i>=0;i--)inv[i]=inv[i+1]*(i+1);
    std::vector<ModInt> ans={1};
    rep(i,26){
        if(!a[i])continue;
        std::vector<ModInt> vec(a[i]+1,1);
        rep(j,ans.size())ans[j]*=inv[j];
        rep(j,vec.size())vec[j]*=inv[j];
        ans=NumberTheoreticTransform::multiply(ans,vec);
        while(!ans.back())ans.pop_back();
        rep(j,ans.size())ans[j]*=fact[j];
    }
    ModInt res=0;
    REP(i,ans.size()-1)res+=ans[i];
    std::cout<<res<<std::endl;
}
```

余談

昨日爆破されかけました（画像は競プロer集会所より）



参加ありがとうございました