Alpha
(Решение)

Задачата може да се подходи по много начини. Да започнем с лесните тестове – тези, с дължина на имената до 3. Лесна за смятане горна граница на отговора би била 3^{26} = 2,541,865,828,329, тъй като на всяка от 26-те позиции имаме избор от до 3 букви. Трябва, обаче, да забележим, че най-късно на 9-тия стринг ще имаме поне една буква, която вече сме ползвали (по принципа на Дирихле – 9 * 3 = 27 > 26), тоест до 9-та позиция гарантирано вече ще имаме поне едно множимо 2 (или по-малко!) вместо 3. Можем да забележим, че колкото повече букви сме ползвали, толкова по-често ще се случва да не можем да ползваме всичките 3 букви от текущия стринг.

Реално се оказва, че това драстично намалява броя възможни отговори (и съответно, колко бавно би вървяло решение, базирано на backtrack). Оказва се, че максималният отговор е под 1,000,000 – тоест решение, базирано на бектрек, без проблем хваща всички тестове с дължина на стринговете по-малка или равна на 3 (даже малко повече – това просто решение хваща 8 от 25-те теста, тоест 32 точки).

Когато дължината на стринговете стане 4, най-баналният бектрек вече не е достатъчно бърз. Можем, обаче, да направим по-ефективна имплементация, базирана на битови трикове. За целта ще подаваме на рекурсията два аргумента: текущ индекс (до кой стринг сме стигнали) и битова маска на буквите, които вече сме ползвали. За всеки стринг ще пазим друга битова маска – буквите, които можем да ползваме от съответния стринг. Така, например, стрингът "ELLY" ще има битова маска $2113538_{(10)} = 00001000000100000000000010_{(2)}$

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
00001000000100000000000010
```

Да кажем, че вече сме ползвали буквите {'A', 'C', 'D', 'F', 'H', 'I', 'L', 'N', 'O', 'P', 'V', 'Z'}, тоест маската, която подаваме на рекурсията е:

```
ABCDEFGHIJKLMNOPQRSTUVWXYZ
10110101100101110000010001
```

Нека маските, кои букви се съдържат във всеки от стринговете, пазим в масив `wordMask[26]`, а текущата маска пазим в променливата `usedMask`. Можем да намерим кои букви можем да ползваме от текущата дума (с индекс *idx*) ползвайки следната побитова операция:

```
int options = wordMask[idx] & ~usedMask;
```

Наистина, изразът `~usedMask` дава всички букви, които все още не са използвани, а операцията `&` (bitwise and) оставя само тези от тях, които можем да ползваме от текущата дума.

Имаме бърз начин, по който можем да намерим и най-младшия сетнат бит в дадено двоично число:

```
int bit = options & -options;
```

Така за всяка маска и всяка дума ни трябват точно толкова операции, колкото са неизползваните от нея букви (при това, всичко случвайки се с побитови операции), което е няколко пъти по-ефективно, от това да ползваме цикли и `if`-ове.

Дори с тези оптимизации, обаче, бектрекът хваща едва още 1-2 теста. Което можем да направим, обаче, е динамично оптимиране по маската на използваните букви! Наистина, броят сетнати битове в маската ни дава до коя дума сме стигнали, а самите битове - кои букви можем да ползваме нататък.

Тук, обаче, забелязваме, че паметта ни не стига – масив с размер `long long dyn[1 << 26]` (който ще ни стигне да пазим отговора със сигурност) би заемал 512 мегабайта. Дори масив от тип `unsigned dyn[1 << 26]` заема 256 мегабайта – повече от ограничението за задачата, а и няма да е може да пази някои от стоиностите, които можем да получим. Все пак, за по-малките тестове (тези с дължина 3 и 4) очакваме отговорите да са малки – тоест и броят валидни маски, които можем да получим, да са сравнително малко. Това наистина е така, и съответно можем да exploit-нем, като ползваме хештаблица за динамична таблица. Така хващаме вече 48 точки (а можем и малко повече, ползвайки няколко оптимизации).

Какво в крайна сметка е решението за 100 точки? Видяхме, че отговорите никога не са твърде големи (реално, най-големият, който успях да постигна, е малко над 5 милиарда). Това е твърде много за да го преброим с бектрек, но не е прекалено много (примерно, за да го искаме по модул). Често в такива случаи на помощ идва техниката "meet-in-the-middle" – точно това трябва да направим и тук.

Нека намерим всички маски, които можем да получим сред първите 13 стринга (и по колко начина можем да получим всяка от тях). Нека ги запишем в масив. Нека след това намерим всички възможни маски сред вторите 13 числа. Имайки маска от втората половина, тя ни дава еднозначно с коя маска от първата половина може да бъде комбинирана (там, където първата има нули, втората трябва да има единици и обратно). Така, за всяка маска от втората половина, ще добавяме към отговора броя на маската с инвъртнати битове, който сме записали от първата половина.

Отново идва проблемът с паметта. Макар и броят начини, по които можем да изберем 13 бита от 26 (C(26, 13)) да не е твърде голям – малко над 10 милиона – нямаме лесен (константен) начин, по който да превръщаме маска в комбинация и обратно. Отново можем да ползваме хештаблица (като така точките ни скачат до 56), но можем и да ползваме масив – просто трябва да сме хитри!

Трябва да забележим, че маската винаги има 13 сетнати бита – тоест ако я разделим на 2 (премахнем най-младшия й бит) и тя стане с 12 сетнати бита, то най-младшият бит е бил 1. Ако пък все още е с 13, то най-младшият е бил 0. Така еднозначно можем да пазим маски с фиксиран брой битове, ползвайки двойно по-малко памет. Такава оптимизация трябваше да бъде приложена и в първата

задача за А група, която дадох на състезание: задачата "Разстояние" (http://www.math.bas.bg/infos/Shumen_2007.html) от есенния турнир през далечната 2007-ма година.
Задачата можете да решите тук: https://action.informatika.bg/problems/57

Така сложността на цялата задача е $O(|S_i|^{13})$, но както казахме в началото, реално е значително по-малко, тъй като няма как винаги да имаме по пет възможности за буква. Най-големият брой, който успях да постигна, е под 200,000,000 операции (които, за съжаление, са индексиране в масив, съответно са относително бавни).

Решението на задачата е учудващо кратко:

```
int arrOffset;
int wordMask[26];
int maskCount[1 << 25];

long long answer = 0;
void recurse(int idx, int usedMask) {
    if (idx >= 13) {
        !arrOffset ? maskCount[usedMask >> 1]++
                   : answer += maskCount[(((1 << 26) - 1) ^ usedMask) >> 1];
    } else {
        int options = wordMask[arrOffset + idx] & ~usedMask;
        while (options) {
            recurse(idx + 1, usedMask | (options & -options));
            options ^= (options & -options);
        }
    }
}

void solve() {
    recurse(0, 0);
    arrOffset = 13;
    recurse(0, 0);
}
```

Тук вече сме прочели в wordMask[i] кои букви се съдържат във всяка от думите.

Забележете, че макар и максималният отговор да е по-голям от 2^{31}, то максималният брой пъти, в която дадена маска се среща сред първите 13 (или вторите 13, for that matter) стринга е 5^{13} = 1,220,703,125 (реално много по-малко), което се събира в `int`.

Автор: Александър Георгиев