I-查字典

当妮妮确定自己想要查找的单词在字典的第 l 个到第 r 个单词之间时，妮妮会查找第 $f_{l,r}$ $(l \le f_{l,r} \le r)$ 个单词，若第 $f_{l,r}$ 个单词 s 已经是想要查找的单词 t，则会停止查字典。若 s 的字典序小于 t，则妮妮可以确定想要查找的单词 t 一定在第 $f_{l,r}+1$ 到第 r 个单词之间。同理若 s 的字典序大于 t，则妮妮可以确定想要查找的单词 t 一定在第 l 到第 $f_{l,r}-1$ 个单词之间。如此继续查找字典直到查到单词为止。

整个查找流程就是二叉树上的查找，$f_{l,r}$就是二叉树的根节点，查找的值小于$f_{l,r}$在左子树上查找，等于$f_{l,r}$终止，大于$f_{l,r}$在右子树上查找。每个单词都是树上节点，需要确定一棵二叉树，使得总代价最小。

对于深度为d_i的节点，查找的代价为d_i+1，节点i共需查找p_i次，即字符串s_i共出现p_i次

总的查找代价为$\Sigma(d_i+1)p_i$.

可以统计每个字符串的出现次数，然后按照字典序分配节点和权值，最后再求出最小代价的最小代价的二叉搜索树即可。

其中，关于最小代价树的求法可以参考H题的思路。

由于每个单词只有两个字母，单词个数也比较少，可以使用哈希，也可以使用stl的map来统计频率。

```
#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

map<string, int> cnt;
int n;
ll p[400];
ll e[400][400], w[400][400];
string word;

int main() {
    cin.tie(NULL);
    ios::sync_with_stdio(false);

    cin >> n;
    for (int i = 0; i < n; ++i) {
        cin >> word;
        cnt[word]++;
    }

    n = 0;

    for (auto ent : cnt) {
        p[++n] = ent.second;
        w[n][n - 1] = 0;
        e[n][n - 1] = 0;
    }

    for (int l = 1; l <= n; ++l) {
        for (int i = 1; i <= n - l + 1; ++i) {
            int j = i + l - 1;
            e[i][j] = 1e15;
            w[i][j] = w[i][j - 1] + p[j];
            for (int r = i; r <= j; ++r) {
```

```
                ll t = e[i][r - 1] + e[r + 1][j] + w[i][j];
                if (t < e[i][j]) {
                    e[i][j] = t;
                }
            }
        }
    }

    cout << e[1][n] << endl;

    return 0;
}
```