1220：单词接龙

题目描述

单词接龙是一个与我们经常玩的成语接龙相类似的游戏，现在我们已知一组单词，且给定一个开头的字母，要求出以这个字母开头的最长的“龙”（每个单词都最多在“龙”中出现两次），在两个单词相连时，其重合部分合为一部分，例如beast和astonish，如果接成一条龙则变为beastonish，另外相邻的两部分不能存在包含关系，例如at和atide间不能相连。

输入

输入的第一行为一个单独的整数n(n<=20)表示单词数，以下n行每行有一个单词（只含有大写或小写字母，长度不超过20），输入的最后一行为一个单个字符，表示“龙”开头的字母。你可以假定以此字母开头的“龙”一定存在。

输出

只需输出以此字母开头的最长的“龙”的长度。

输入样例

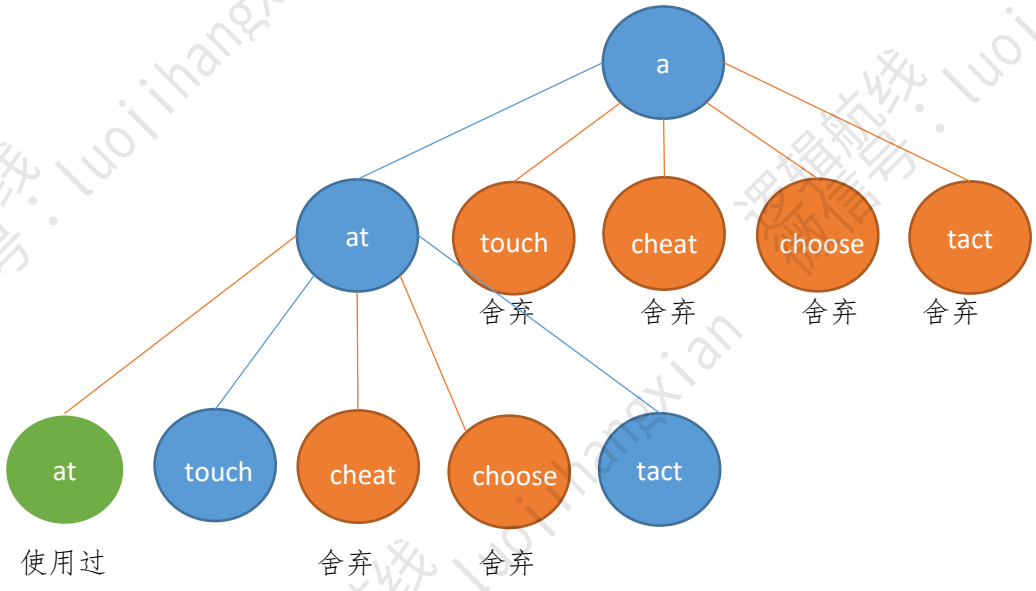
```
5
at
touch
cheat
choose
tact
a
```

输出样例

```
23
```

解析

首先，还是先来构造决策树。这道题的决策树其实非常好理解，就是从根部开始，不停的尝试接入不同的单词，直到找到最长的龙为止，因此决策树图如下：



编码

首先是基本变量定义

```
//待接龙的单词数量和最大长度
int n, length = 0;
//访问列表
int vis[1000] = {0};
//待接龙单词数组
string str[1000];
```

接下来，我们开始读取数据

```
cin >> n;
for (int i = 1; i <= n; i++) {
    //读入n个单词
    cin >> str[i];
}
//读入根节点单词
cin >> str[n + 1];
```

这时，我们开始编写核心的回溯代码

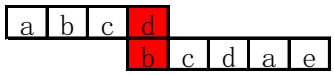
```
// s  当前接龙的根节点单词
// lengthNow 当前接龙长度
void dfs(string s, int lengthNow) {
    //记录当前接龙最大长度
    length = max(length, lengthNow);
    //遍历全部的待接龙单词
    for (int i = 1; i <= n; i++) {
        //如果当前的单词已经被使用过2次，则跳过
        if (vis[i] > 1) continue;
        else {
            //当前字符串与原来字符串的重叠部分的大小
            int nums = checkLength(s, str[i]);
            //即两个字符串有重叠部分
            if (nums != 0) {
                //标记选择
                vis[i]++;
                //继续向下接龙
                //此时新的长度等于原有的长度加上新单词的长度再减去重合长度
                dfs(str[i], lengthNow + str[i].length() - nums);
                //取消选择
                vis[i]--;
            }
        }
    }
}
```

最后，我们来编写求重合部分长度的代码。那么该如何求两个字符串重合部分的长度呢？其实很简单，我们首先记录传入的根字符串的长度len，并将其减1，这样做是防止存在包含的关系。然后开始分别将两个字符串，一个从后往前，一个从前往后的截取1到len，直到截取的部分相同，我们就找到了重合部分的长度。注意全局首个字母需要特判断，如下图所示：

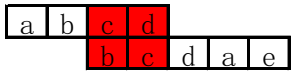
字符串1	a	b	c	d

字符串2	b	c	d	a	e

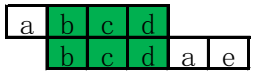
第一次比较



第二次比较



第三次比较




```
int checkLength(string a, string b) {
    //取出根字符串的长度，注意要减1，这样能够避免包含现象的出现
    int len = a.length() - 1;
    //对于全局第一个根字符串需要进行特判
    if (a.compare(str[n + 1]) == 0) {
        len = a.length();
    }
    //尝试截取
    for (int i = 1; i <= len; i++) {
        //从后向前截取
        string s1 = a.substr(a.length() - i, i);
        //从前向后截取
        string s2 = b.substr(0, i);
        //相等，返回重合长度
        if (s1.compare(s2) == 0) {
            return i;
        }
    }
    return 0;
}
```

完整代码

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

using namespace std;
//待接龙的单词数量和最大长度
int n, length = 0;
//访问列表
int vis[1000] = {0};
//待接龙单词数组
string str[1000];

int checkLength(string a, string b) {
    //取出根字符串的长度，注意要减1，这样能够避免包含现象的出现
    int len = a.length() - 1;
    //对于全局第一个根字符串需要进行特判
    if (a.compare(str[n + 1]) == 0) {
        len = a.length();
    }
    //尝试截取
    for (int i = 1; i <= len; i++) {
        //从后向前截取
        string s1 = a.substr(a.length() - i, i);
        //从前向后截取
        string s2 = b.substr(0, i);
        //相等，返回重合长度
        if (s1.compare(s2) == 0) {
            return i;
        }
    }
    return 0;
```

```
}

// s  当前接龙的根节点单词
// lengthNow 当前接龙长度
void dfs(string s, int lengthNow, bool start = false) {
    //记录当前接龙最大长度
    length = max(length, lengthNow);
    //遍历全部的待接龙单词
    for (int i = 1; i <= n; i++) {
        //如果当前的单词已经被使用过2次，则跳过
        if (vis[i] > 1) continue;
        else {
            //当前字符串与原来字符串的重叠部分的大小
            int nums = checkLength(s, str[i]);
            //即两个字符串有重叠部分
            if (nums != 0) {
                //标记选择
                vis[i]++;
                //继续向下接龙
                //此时新的长度等于原有的长度加上新单词的长度再减去重合长度
                dfs(str[i], lengthNow + str[i].length() - nums);
                //取消选择
                vis[i]--;
            }
        }
    }
}

int main() {
    cin >> n;
    for (int i = 1; i <= n; i++) {
        //读入n个单词
        cin >> str[i];
    }
    //读入根节点单词
    cin >> str[n + 1];
    dfs(str[n + 1], 1);
    cout << length;
    return 0;
}
```