即時新聞 CSIE新聞網

讓專家跌破眼鏡，每個人都忽略這四個詭異的視覺陷阱！我看了3遍都不相信！(4pts)

T: text to be searched from
P: pattern to be searched for
q: the maximum value of the hash value representing a string

Assume that:
(1) All indices of the string (i.e., a character array) start from 0.
(2) T and P consist of numerical characters '0' to '9' only.
(3) You can assume that overflow does not happen

```
N_DIGIT = 8
q = 2**N_DIGIT # A**B: A to the power of B

# ============== Part 1 ==============
# Original Rabin-Karp algorithm taught in the class
function RabinKarpMatcher_1(T, P)
    n = T.length
    m = P.length
    h = 10 ** (m-1) mod q

    t = 0
    p = 0

    for i = 0 to m-1
        t = (10 * t + T[i]) mod q
        p = (10 * p + P[i]) mod q

    for s = 0 to n-m
        if t == p:
            if T[s .. s+m-1] == P[0 .. m-1]
                print "Pattern with shift {s}"
            else
                print "Spurious hit with shift {s}."
        t = (10 * (t - T[s] * h) + T[s+m]) mod q


# ============== Part 2 ==============
# Rabin-Karp with bitwise XOR
# hash value is obtained by XOR'ing all characters of a string
function RabinKarpMatcher_2(T, P)
    n = T.length
    m = P.length

    t = T[0] ^ T[1] ^ ... ^ T[m - 1]    # ^ : bitwise XOR
    p = P[0] ^ P[1] ^ ... ^ P[m - 1]
```

```
    for s = 0 to n-m
        if t == p:
            if T[s .. s+m-1] == P[0 .. m-1]
                print "Pattern with shift {s}"
            else
                print "Spurious hit with shift {s}."
        # TODO: what is the value of next "t"? Calculate it in O(1).


# ============== Part 3 ==============
# Circular Shift on one char
function CS(C, k) # k <= N_DIGIT
    '''
    & : bitwise AND
    | : bitwise OR
    '''
    C1 = (C << k) & (q - 1) # bitwise AND (q-1): C1 keeps track of 8 digits
    C2 = C >> (N_DIGIT - k)
    return (C1 | C2)

# Rabin-Karp with Circular Shift and XOR
function RabinKarpMatcher_3(T, P)
    n = T.length
    m = P.length

    t = CS(T[0], m-1) ^ CS(T[1], m-2) ^ ... ^ CS(T[m-1], 0)
    p = CS(P[0], m-1) ^ CS(P[1], m-2) ^ ... ^ CS(P[m-1], 0)

    for s = 0 to n-m
        if t == p:
            if T[s .. s+m-1] == P[0 .. m-1]
                print "Pattern with shift {s}"
            else
                print "Spurious hit with shift {s}."
        # TODO: what is the value of next "t"? Calculate it in O(1).
```

- Problem 1:
  - (1) Fill in the TODO in `RabinKarpMatcher_2`. What is the value of next "t"?
    - Ans: `t =`
  - (2) Please give an example of T and P which results in at least a spurious hit, when using `RabinKarpMatcher_2`. The example should have both `T.length` and `P.length` less than 6.
    - Ans:
  - (3) Fill in the TODO in `RabinKarpMatcher_3`. What is the value of next "t"?
    - Ans: `t = CS(t, 1) ...`
  - (4) Please give an example of T and P which results in at least a spurious hit, when using `RabinKarpMatcher_3`. The example should have both `T.length` and `P.length` less than 6.
    - Ans:

20歲以前必看！讓全球工程師陷入瘋狂的算法，第4個太誇張了！(8pts)

(In the following problem, we assume the length of the string is less than some constant L)
n: the number of the strings
strings: the strings that need to be sorted

```
void func_A(char ** strings, int n, int digit) {

    char tmp_str[n][L];
    int C[256];

    for(int i = 0; i < n; i++) tmp_str[i] = 0;
    for(int i = 0; i < 256; i++) C[i] = 0;

    for(int i = 0; i < n; i++) C[strings[i][digit]]++;
    for(int i = 1; i < 256; i++) C[i] += C[i - 1];

    for(int i = n - 1; i >= 0; i--) {
        for(int j = 0; j < L; j++) {
            tmp_str[C[strings[i][digit]]][j] = strings[i][j];
        }
        C[strings[i][digit]]--;
    }
    for(int i = 0; i < n; i++) {
        for(int j = 0; j < L; j++) {
            strings[i][j] = tmp_str[i][j];
        }
    }
}

void func_B(char ** strings, int n) {
    for(int i = L - 1; i >= 0; i--) {
        func_A(strings, n, i);
    }
}
```

(1) What does this algorithm do?
(2) What is the time complexity of the algorithm in terms of L and n? Please provide some justification.
(3) In the class, we have talked about the difference between MSB and LSB in radix sort. How about this algorithm? Which of the two is a better choice? Please provide some justification.

驚！！整整流傳四十年的「這個算法」居然有這些性質，你一定要看看！(8pts)

Problem Description

During the spring break of NTU, the class leader of CSIE, NeVeR_LosEs (abbreviated NL), along with his friends, went hiking and camping in a remote campsite in Hualien. So mischievous is NL that he invited his friends to go on an adventure in an attempt to discover some unknown places. They strolled all the way out of the camp zone and found a mysterious cave with some illegible handwriting bearing some resemblance to an archaic cuneiform. Discovering deeper into the cave, they missed the way back, but fortunately, a dictionary was hidden under the subterranean lake in the midst of the cave. Scrutinizing the dictionary, NL found out that it is not only a thesaurus but also a record of ancient aboriginals who discovered this secret safe place to avoid their archenemy. To prevent further pursuit from their enemies or their descendants, they carved some sentences on the path in the cave. To distinguish the correct one to the outer world, it is indicated on the

dictionary that the correct paths are the ones with synonyms at both sides of the path, while the path to the dead end is not. Please help NL and his friends to find the way out! NL, however, is too lazy to check the inscriptions on the wall and search in the book simultaneously, so he would determine the synonym based on his knowledge, what he memorized when there are no writings on the path, only.

Input Format

The first line contains an integer N, denoting the number of the operations.
The following N lines are the operations given in the following format:

- `union a b` indicates that `a` and `b` are synonyms.
- `compare A B`: `A` and `B` are sentences comprising multiple words like `a` and `b`.

Output Format

For each `compare A B`, based on the knowledge of NL, output "True" if `A` and `B` have the same meaning, or "False" otherwise. Here, we define `A` and `B` have same meaning iff $\text{len}(A) = \text{len}(B)$ and let $A = A_1 A_2 \dots A_i \dots A_{\text{len}(A)}$, $B = B_1 B_2 \dots B_i \dots B_{\text{len}(B)}$, (A_i, B_i) are synonym $\forall i \in [1, \text{len}(A)]$, while A_i and B_i are words.

Input Constraint

- Each word (i.e. `a`, `b`) consists of 4 alpha-numeric characters (\^[0-9a-zA-Z]{4}$)
- $0 \leq \text{len}(A), \text{len}(B) \leq 100$ **words**, it is assured that A and B can be uniquely recognized (i.e. words do not overlap)
- Subtask 1 (50%)
  $N \leq 10^3$
- Subtask 2 (50%)
  $N \leq 10^5$

Sample Input

```
5
compare nlnl7414 7414nlnl
union nlnl 7414
compare nlnl7414 7414nlnl
union mkmk nlnl
compare nlnl7414 nlnl8787
```

Sample Output

```
False
True
False
```

Sample Code

Constructing a disjoint set forest could help you solve the problem.
While we encourage you to write your code from scratch, you can refer to the following code and finish the **TODO**s to solve the problem.

```
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>

typedef struct DisjointSet{
    // TODO: Determine fields to use by your method
} disjoint_set;

disjoint_set ds[1 << 24];
bool set[1 << 24] = {};

int c2i(char c) {
    if ('0' <= c && c <= '9') return c - '0';
    else if ('a' <= c && c <= 'z') return c - 'a' + 10;
    else if ('A' <= c && c <= 'Z') return c - 'A' + 36;
    return -1;
}

int hash(const char* s) {
    int ret = 0;
    for (int i = 0; i < 4; ++i)
        ret = (ret << 6) | c2i(s[i]);
    return ret;
}

void makeset(const char* s){
    int i = hash(s);
    // TODO: Initialize a set
}

inline void static init(const char* s) {
    int i = hash(s);
    if (!set[i]) {
        makeset(s);
        set[i] = 1;
    }
}

int find_set(const char* s) {
    init(s);
    int i = hash(s);
    // TODO: Implement your find algorithm here
    return /* something */;
}

void link(const char *ra, const char *rb) {
    int a = find_set(ra), b = find_set(rb);
    // TODO: Implement your union algorithm here
}

bool same_set(const char *a, const char *b) {
    return (find_set(a) == find_set(b));
}
```

```
bool stringcompare(const char *a, const char *b) {
    // implement your string compare
}

int main() {
    int n;
    scanf("%d", &n);
    char cmd[16], a[512], b[512];
    for (int i = 0; i < n; ++i) {
        scanf("%s %s %s", cmd, a, b);
        if (!strcmp(cmd, "union")) {
            init(a);
            init(b);
            link(a, b);
        } else {
            bool same = stringcompare(a, b);
            if(same) puts("True");
            else puts("False");
        }
    }
}
```

Code submission

Please submit your code here.