STL map

Henrique Mendes

Escola de Inverno UTFPR 2024

Universidade Tecnológica Federal do Paraná · Curitiba, PR

31 de julho de 2024

O que é o map?

O ‘std::map‘ na biblioteca STL do C++ é uma estrutura de dados que armazena pares de chave-valor, onde cada chave é única. Pense nele como uma lista de contatos, onde você pode rapidamente encontrar o número de telefone (valor) associado a um nome (chave). Ele organiza os elementos de forma ordenada e eficiente, usando uma árvore binária balanceada, o que significa que você pode adicionar, remover e procurar elementos em $\mathcal{O}(\log n)$. O ‘std::map‘ é extremamente útil quando você precisa manter os dados ordenados e realizar buscas rápidas. Ou quando é conveniente trocar o tipo de dados.

Declaração e métodos

Declaração:

```
map<string,string> myMap; / map<string,int> myMap;
```

Métodos:

- begin() – Iterador pro começo do map
- end() – Último elemento do map
- size() – Quantos elementos tem no map
- operator [ ] - consulta valor guardado na chave[1] - $\mathcal{O}(\log n)$
- erase(iterator) – Remove elemento via iterador - $\mathcal{O}(1)$
- erase(key)– Remove elemento pela chave - $\mathcal{O}(\log n)$
- clear() – Remove tudo do map - $\mathcal{O}(n)$

[1] Esse operador insere automaticamente a chave se ela ainda não estiver no map

Exemplo de problema[2]

B. Radio Station

time limit per test: 2 seconds
memory limit per test: 256 megabytes

As the guys fried the radio station facilities, the school principal gave them tasks as a punishment. Dustin's task was to add comments to nginx configuration for school's website. The school has n servers. Each server has a name and an ip (names aren't necessarily unique, but ips are). Dustin knows the ip and name of each server. For simplicity, we'll assume that an nginx command is of form "`command ip;`" where `command` is a string consisting of English lowercase letter only, and `ip` is the ip of one of school servers.

Each ip is of form "`a.b.c.d`" where a, b, c and d are non-negative integers less than or equal to 255 (with no leading zeros). The nginx configuration file Dustin has to add comments to has m commands. Nobody ever memorizes the ips of servers, so to understand the configuration better, Dustin has to comment the name of server that the ip belongs to at the end of each line (after each command). More formally, if a line is "`command ip;`" Dustin has to replace it with "`command ip; #name`" where `name` is the name of the server with ip equal to `ip`.

Dustin doesn't know anything about nginx, so he panicked again and his friends asked you to do his task for him.

[2] `https://codeforces.com/contest/918/problem/B`

Casos de teste

Input

The first line of input contains two integers n and m ($1 \le n, m \le 1000$).

The next n lines contain the names and ips of the servers. Each line contains a string `name`, name of the server and a string `ip`, ip of the server, separated by space ($1 \le |name| \le 10$, *name* only consists of English lowercase letters). It is guaranteed that all ip are distinct.

The next m lines contain the commands in the configuration file. Each line is of form "`command ip;`" ($1 \le |command| \le 10$, `command` only consists of English lowercase letters). It is guaranteed that ip belongs to one of the n school servers.

Output

Print m lines, the commands in the configuration file after Dustin did his task.

Examples

input Copy

```
2 2
main 192.168.0.2
replica 192.168.0.1
block 192.168.0.1;
proxy 192.168.0.2;
```

output Copy

```
block 192.168.0.1; #replica
proxy 192.168.0.2; #main
```

input Copy

```
3 5
google 8.8.8.8
codeforces 212.193.33.27
server 138.197.64.57
redirect 138.197.64.57;
block 8.8.8.8;
cf 212.193.33.27;
unblock 8.8.8.8;
check 138.197.64.57;
```

output Copy

```
redirect 138.197.64.57; #server
block 8.8.8.8; #google
cf 212.193.33.27; #codeforces
unblock 8.8.8.8; #google
check 138.197.64.57; #server
```

Código

```
void solve() {
  int n, m;
  cin >> n >> m;
  map<string,string> ipToName;
  while(n--){
    string name, ip;
    cin >> name >> ip;
    ipToName[ip] = name;
  }
  while(m--){
    string command, ip;
    cin >> command >> ip;
    ip = ip.substr(0, ip.length()-1);
    cout << command << ' ' << ip << "; #" << ipToName[ip] <
  }
}
```

Exemplo de problema[3]

Problema F

Forca Multidimensional

O Jogo da Forca Multidimensional tem regras muito peculiares. De certa forma, é como se você estivesse jogando várias partidas do tradicional Jogo da Forca ao mesmo tempo, com a diferença que as palavras não precisam existir no dicionário. Se você nunca jogou o Jogo da Forca, não se preocupe: toda a informação que você precisa estará abaixo.

Na versão multidimensional do jogo, existem várias palavras no tabuleiro, inicialmente desconhecidas, todas de mesmo comprimento. Em cada momento do jogo, você conhece alguns caracteres de certas posições das palavras (a maneira como estes caracteres foram descobertos não é importante para este problema). Em determinado momento, quando resta apenas um caractere desconhecido em cada palavra do tabuleiro, o jogo parte para a fase de *tudo ou nada*. Neste momento, você deve escolher uma palavra que maximize o número de compatibilidades com as palavras do tabuleiro. Para uma escolha de palavra P, dizemos que ela é compatível com uma palavra T do tabuleiro se todas as letras conhecidas de T ocorrem exatamente nas mesmas posições em P.

Dadas as informações conhecidas sobre as palavras do tabuleiro, você deve determinar qual palavra escolher para o *tudo ou nada*, que maximiza o número de compatibilidades. Se houver mais de uma solução, imprima a menor lexicograficamente. Dizemos que uma palavra P é lexicograficamente menor que uma palavra Q se $P_i < Q_i$ onde P_i é o i-ésimo caractere de P, Q_i é o i-ésimo caractere de Q e i é o menor índice tal que $P_i \neq Q_i$.

Entrada

A primeira linha da entrada contém dois inteiros N e C satisfazendo $1 \leq N \leq 10^4$ e $1 \leq C \leq 12$, indicando número de palavras no tabuleiro e comprimento das palavras que ele contém. Em seguida, N linhas descrevem, cada uma, uma palavra de comprimento C composta apenas de caracteres de "a" a "z" exceto por uma das posições, que conterá um caracter "*", indicando que o caractere daquela posição ainda é desconhecido.

Saída

Você deve imprimir uma única linha, contendo, respectivamente, uma palavra T, de comprimento C, e um inteiro M, onde M é o maior número de compatibilidades que uma palavra pode ter com as palavras da entrada e T é a palavra lexicograficamente menor dentre aquelas com compatibilidade M.

[3] https://codeforces.com/gym/103960/problem/F

Entrada

Exemplo de entrada 1	Exemplo de saída 1
5 4 rat* ru*d rot* r*ta r*ta	rata 3

Exemplo de entrada 2	Exemplo de saída 2
5 4 bon* fon* n*no *eto *ano	nano 2

Algumas ideias

- Contar frequência dos caracteres?
- Alguma estratégia gulosa diferente?

Algumas ideias

- Contar frequência dos caracteres?
- Alguma estratégia gulosa diferente?
- Contar todas as palavras possíveis!!! Brute Force mesmo

Mas por que funcionaria?

- Quantas palavras temos?
- Quantos caracteres desconhecemos?
- Qual o tamanho do alfabeto?
    - Sim é importante lembrar o tamanho do alfabeto

Qual a complexidade (P) disso?

Quantidade de palavras (NP): $10^4 \cdot 26 = 2.6 \cdot 10^5$
Complexidade map: $\mathcal{O}(n \log n)$
Custo da comparação entre 2 palavras: 12

$$P = NP \cdot \log(NP)$$

$$2.6 \cdot 10^5 \cdot \log(2.6 \cdot 10^5) = 4.4 \cdot 10^6$$

$$12 \cdot 4.4 \cdot 10^6 = 5.3 \cdot 10^7$$

Código

Código

```cpp
#include <bits/stdc++.h>
using namespace std;

#define endl '\n'

void solve(){
    int n, c;
    cin >> n >> c;
    map<string,int> freq;
    while(n--){
        string s;
        cin >> s;
        for (int i = 0; i < c; i++)
            if (s[i] == '*'){
                string s2 = s;
                for (char a = 'a'; a <= 'z'; a++){
                    s2[i] = a;
                    freq[s2]++;
                }
            }
    }
    string ans;
    int freqAns = -1;
    for (auto [s, c]: freq){
        if (c == freqAns)
            ans = min(ans, s);
        else if (c > freqAns)
            ans = s, freqAns = c;
    }
    cout << ans << ' ' << freqAns << endl;
}

signed main(){
    cin.tie(0); cout.tie(0);
    ios_base::sync_with_stdio(0);
    solve();
    return 0;
}
```