

# AtCoder

## AtCoder Beginner Contest 049: *Upsolving*

---

Prof. Edson Alves - UnB/FGA

2020

1. A - UOIAUAI
2. B - Thin
3. C - Daydream
4. D - Connectivity

**A - UOIAUAI**

---

# Problema

Given a lowercase English letter  $c$ , determine whether it is a vowel. Here, there are five vowels in the English alphabet: **a**, **e**, **i**, **o** and **u**.

## Constraints

- $c$  is a lowercase English letter.

## Input

Input is given from Standard Input in the following format:

```
 $c$ 
```

## Output

If  $c$  is a vowel, print **vowel**. Otherwise, print **consonant**.

## Exemplo de entradas e saídas

### Entrada

a

z

s

### Saída

vowel

consonant

consonant

- O problema consiste em determinar se  $c$  é uma das cinco vogais listadas
- Uma maneira de verificar este fato é utilizar e escrever uma expressão que confronte  $c$  com todos os cinco possíveis valores
- Outra forma é utilizar o método `find()` da classe `string` do C++ para procurar por  $c$  em uma string  $s$  formada pelas cinco vogais
- Ambas formas tem complexidade  $O(1)$ , pois serão feitas sempre, no máximo, cinco comparações

# Solução $O(1)$

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int main()
6 {
7     char c;
8     cin >> c;
9
10    string vowels { "aeiou" };
11
12    auto ans = vowels.find(c) == string::npos ?
13        "consonant" : "vowel";
14
15    cout << ans << '\n';
16
17    return 0;
18 }
```



**B - Thin**

---

# Problema

There is an image with a height of  $H$  pixels and a width of  $W$  pixels. Each of the pixels is represented by either `.` or `*`. The character representing the pixel at the  $i$ -th row from the top and the  $j$ -th column from the left, is denoted by  $C_{i,j}$ .

Extend this image vertically so that its height is doubled. That is, print a image with a height of  $2H$  pixels and a width of  $W$  pixels where the pixel at the  $i$ -th row and  $j$ -th column is equal to  $C_{(i+1)/2,j}$  (the result of division is rounded down).

## Constraints

- $1 \leq H, W \leq 100$
- $C_{i,j}$  is either  $.$  or  $*$ .

## Input

Input is given from Standard Input in the following format:

```
H      W
C1,1  ...  C1,W
⋮
CH,1  ...  CH,W
```

## Output

Print the extended image.

# Exemplo de entradas e saídas

## Entrada

2 2

\* .

. \*

1 4

\*\*\* .

## Saída

\* .

\* .

. \*

. \*

\*\*\* .

\*\*\* .

## Exemplo de entradas e saídas

## Entrada

9 20

## Saída

- Há duas formas de se resolver estes problema
- A primeira delas é computar cada uma das  $2H$  linhas utilizando a expressão fornecida no texto do problema
- Neste caso, é preciso atentar ao fato de que a expressão dada numera as linhas de 1 a  $H$ , e não de zero a  $H - 1$ , o que pode levar a uma diferença na saída ou a um erro de invasão de memória
- A segunda é observar que a expressão dada significa que cada linha da imagem original será dobrada
- Assim, basta ler cada linha da entrada e escrevê-la duas vezes na saída
- Ambas soluções tem complexidade  $O(HW)$

# Solução $O(HW)$

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 int main()
6 {
7     ios::sync_with_stdio(false);
8
9     int H, W;
10    cin >> H >> W;
11
12    vector<string> M(H);
13
14    for (int i = 0; i < H; ++i)
15        cin >> M[i];
16
17    for (auto x : M)
18        cout << x << '\n' << x << '\n';
19
20    return 0;
21 }
```

## **C - Daydream**

---



# Problema

You are given a string  $S$  consisting of lowercase English letters. Another string  $T$  is initially empty. Determine whether it is possible to obtain  $S = T$  by performing the following operation an arbitrary number of times:

- Append one of the following at the end of  $T$ : **dream**, **dreamer**, **erase** and **eraser**.

## Constraints

- $1 \leq |S| \leq 10^5$
- $S$  consists of lowercase English letters.

## Input

Input is given from Standard Input in the following format:

$S$

## Output

If it is possible to obtain  $S = T$ , print **YES**. Otherwise, print **NO**.

## Exemplo de entradas e saídas

### Entrada

erasedream

dreameraser

dreamerer

### Saída

YES

YES

NO

# Solução

- A principal dificuldade deste problema surge do fato de que há interseção entre o sufixo de algumas palavras serem o prefixo de outras (por exemplo, **dreamer** e **erase**)
- Caso isto não ocorresse, um algoritmo guloso, que testasse cada palavra contra o prefixo de  $s$  seria correto
- Porém, é possível solucionar este problema com este mesmo algoritmo guloso, desde que o processamento se dê em sentido oposto, comparando-se o sufixo de  $S$  com as quatro palavras listadas
- Neste caso, não há mais ambiguidades, e se uma palavra corresponder ao sufixo, ele deve ser descartado e o processo reiniciado
- Se nenhuma palavra corresponder ao sufixo atual, o processo deve ser interrompido
- A resposta será **YES** apenas se o processo terminar com  $S$  vazia
- A complexidade desta solução é  $O(N)$

# Solução $O(N)$

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 vector<pair<string, size_t>> words {
6     { "dream", 5 }, { "dreamer", 7 }, { "erase", 5 }, { "eraser", 6 } };
7
8 bool solve(string& s)
9 {
10     while (not s.empty())
11     {
12         bool ok = false;
13
14         for (auto [w, size] : words)
15         {
16             if (size <= s.size() and s.substr(s.size() - size) == w)
17             {
18                 ok = true;
19
20                 while (size--)
21                     s.pop_back();
22             }
23         }
24     }
25 }
```

## Solução $O(N)$

```
22         }
23     }
24
25     if (not ok)
26         return false;
27 }
28
29 return true;
30 }
31
32 int main()
33 {
34     ios::sync_with_stdio(false);
35
36     string s;
37     cin >> s;
38
39     cout << (solve(s) ? "YES" : "NO") << '\n';
40
41     return 0;
42 }
```

## **D - Connectivity**

---

# Problema

There are  $N$  cities. There are also  $K$  roads and  $L$  railways, extending between the cities. The  $i$ -th road bidirectionally connects the  $p_i$ -th and  $q_i$ -th cities, and the  $i$ -th railway bidirectionally connects the  $r_i$ -th and  $s_i$ -th cities. No two roads connect the same pair of cities. Similarly, no two railways connect the same pair of cities.

We will say city  $A$  and  $B$  are connected by roads if city  $B$  is reachable from city  $A$  by traversing some number of roads. Here, any city is considered to be connected to itself by roads. We will also define connectivity by railways similarly.

For each city, find the number of the cities connected to that city by both roads and railways.



## Constraints

- $2 \leq N \leq 2 \times 10^5$
- $1 \leq K, L \leq 10^5$
- $1 \leq p_i, q_i, r_i, s_i \leq N$
- $p_i < q_i$
- $r_i < s_i$
- When  $i \neq j$ ,  $(p_i, q_i) \neq (p_j, q_j)$
- When  $i \neq j$ ,  $(r_i, s_i) \neq (r_j, s_j)$

## Input

Input is given from Standard Input in the following format:

$N$	$K$	$L$
$p_1$	$q_1$	
$\vdots$		
$p_K$	$q_K$	
$r_1$	$s_1$	
$\vdots$		
$r_L$	$s_L$	

## Output

Print  $N$  integers. The  $i$ -th of them should represent the number of the cities connected to the  $i$ -th city by both roads and railways.

## Exemplo de entradas e saídas

### Entrada

4 3 1

1 2

2 3

3 4

2 3

4 2 2

1 2

2 3

1 4

2 3

### Saída

1 2 2 1

1 2 2 1

## Exemplo de entradas e saídas

### Entrada

7 4 4

1 2

2 3

2 5

6 7

3 5

4 5

3 4

6 7

### Saída

1 1 2 1 2 2 2

## Solução

- Este é um problema interessante por combinar dois problemas que, isoladamente, seriam de fácil solução, mas que combinados formam um problema mais sofisticado
- As conexões via estradas podem ser determinadas em  $O(N + K)$  por meio de uma travessia em profundidade (DFS) ou em largura (BFS)
- De forma análoga, as conexões por linhas de trem podem ser determinadas em  $O(N + L)$
- Um conjunto  $T_i$  de cidades que estão conectadas por estradas corresponde a um componente conectado do grafo
- Da mesma forma, um conjunto  $R_j$  de cidades conectadas por linhas de trem também é um outro componente conectado, segundo este outro critério
- O problema consiste em determinar as interseções entre os conjuntos  $T_i$  e  $R_j$

## Solução

- Comparar diretamente cada  $T_i$  com todos  $R_j$  leva a uma solução quadrática em  $N$  e a um veredito *TLE*
- Uma maneira de se determinar a solução de forma mais eficiente é, na primeira travessia, identificar todos os vértices em um mesmo componente  $T_i$  por meio de um identificador único (um inteiro, por exemplo)
- Na segunda travessia deve se manter um dicionário que associe todos os elementos de um componente  $R_j$  que tenham o mesmo identificador único
- Ao final da detecção de um componente  $R_j$ , a resposta para cada vértice  $u$  guardado no dicionário será o número de vértices que tem identificador igual a  $u$
- Esta solução tem complexidade  $O((N + K + L) \log N)$

## Solução $O((N + K + L) \log N)$

```
1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 const int MAX { 200010 };
6
7 vector<int> adj_c[MAX], adj_r[MAX];
8 bitset<MAX> found;
9
10 void dfs(int u, int color, vector<int>& cs)
11 {
12     if (found[u])
13         return;
14
15     found[u] = true;
16     cs[u] = color;
17
18     for (auto v : adj_c[u])
19         dfs(v, color, cs);
20 }
21
```

## Solução $O((N + K + L) \log N)$

```
22 void dfs2(int u, map<int, vector<int>>& xs, const vector<int>& cs)
23 {
24     if (found[u])
25         return;
26
27     found[u] = true;
28     xs[cs[u]].push_back(u);
29
30     for (auto v : adj_r[u])
31         dfs2(v, xs, cs);
32 }
33
34 vector<int> solve(int N)
35 {
36     vector<int> ans(N + 1), colors(N + 1);
37     int color = 1;
38
39     for (int u = 1; u <= N; ++u)
40         if (not found[u])
41             dfs(u, color++, colors);
42 }
```



## Solução $O((N + K + L) \log N)$

```
43     found.reset();
44
45     for (int u = 1; u <= N; ++u)
46     {
47         if (not found[u])
48         {
49             map<int, vector<int>> xs;
50
51             dfs2(u, xs, colors);
52
53             for (auto& [c, ys] : xs)
54                 for (auto y : ys)
55                     ans[y] = (int) ys.size();
56         }
57     }
58
59     return ans;
60 }
61
```

## Solução $O((N + K + L) \log N)$

```
62 int main()
63 {
64     ios::sync_with_stdio(false);
65
66     int N, K, L;
67     cin >> N >> K >> L;
68
69     while (K--)
70     {
71         int p, q;
72         cin >> p >> q;
73
74         adj_c[p].push_back(q);
75         adj_c[q].push_back(p);
76     }
77
78     while (L--)
79     {
80         int p, q;
81         cin >> p >> q;
82     }
```

## Solução $O((N + K + L) \log N)$

```
83     adj_r[p].push_back(q);
84     adj_r[q].push_back(p);
85 }
86
87 auto ans = solve(N);
88
89 for (int i = 1; i <= N; ++i)
90     cout << ans[i] << (i == N ? '\n' : ' ');
91
92 return 0;
93 }
```

1. [AtCoder Beginner Contest 049 – Problem A: UOIUAUI](#)
2. [AtCoder Beginner Contest 049 – Problem B: Thin](#)
3. [AtCoder Beginner Contest 049 – Problem C: Daydream](#)
4. [AtCoder Beginner Contest 049 – Problem D: Connectivity](#)