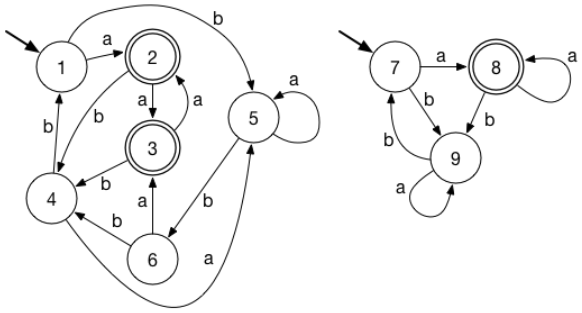
Problem E

O algoritmo de Hopcroft-Karp

para determinar se dois autómatos finitos deterministas são equivalentes



(exemplo do *sample input*)

O objectivo deste problema é a implementação dum algoritmo clássico da literatura em linguagens formais e teoria da computação, o algoritmo Hopcroft-Karp. Espera-se que a solução seja concisa, eficaz e elegante.

Problem

O algoritmo de Hopcroft-Karp para a equivalência de autómatos permite determinar a equivalência sem passar pela construção dos autómatos minimais.

A publicação original deste algoritmo em 1971, de J. E. Hopcroft e de R. M. Karp, encontra-se neste link (aqui)

Considere dois autómatos finitos deterministas $M_1 \triangleq (Q_1, \Sigma, \delta_1, s_1, F_1)$ e $M_2 \triangleq (Q_2, \Sigma, \delta_2, s_2, F_2)$ sobre, para simplificar, o alfabeto $\Sigma = \{a, b, c \ldots, z\}$. Consideramos igualmente que $Q_1 \cap Q_2 = \emptyset$.

Considere o autómato $M \triangleq (Q_1 \cup Q_2, \Sigma, \delta, s_1, F_1 \cup F_2)$, onde

$$\delta(q,a) \triangleq \begin{cases} \delta_1(q,a) & se \quad q \in Q_1 \\ \delta_2(q,a) & se \quad q \in Q_2 \end{cases}$$

Então podemos afirmar que M_1 e M_2 são equivalentes (i.e. $L(M_1) = L(M_2)$ se e só se $s_1 \approx s_2$ em M.

Relembramos que $\approx$ é uma relação de equivalência (reflexiva, simétrica e transitiva) sobre o conjunto de estados e que , para dois estados p e q,

- $p \approx q \ \triangleq \ \forall w \in \Sigma^*, \bar{\delta}(p,w) \in F \Leftrightarrow \bar{\delta}(q,w) \in F$
- $\forall a \in \Sigma, \ p \approx q \Rightarrow \delta(p,a) \approx \delta(q,a)$
- $\forall w \in \Sigma^*, \ p \approx q \Rightarrow \bar{\delta}(p,w) \approx \bar{\delta}(q,w)$

O algoritmo faz uso estratégico de uma pilha S e de uma estrutura de dados, **Union-Find**, especializada para lidar com conjuntos equipados de uma relação de equivalência (*disjoint-sets*).

Esta estrutura de dados disponibiliza três operações:

- MakeSet q: cria um conjunto com um só elemento, $\{q\}$

- Find q: devolve o elemento representativo do conjunto que contém q
- Union p q: junta num só conjunto os conjuntos que contêm p e q

Assim sendo, o algoritmo é o seguinte:

1. para cada estado q de $Q_1 \cup Q_2$, $MakeSet\ q$
2. $Union\ s_1\ s_2$
3. $Push\ S\ (s_1, s_2)$
4. Enquanto S não estiver vazia
   (a) $(q_1, q_2) \leftarrow\ Pop\ S$
   (b) Para cada caracter a de Σ
      i. $r_1 \leftarrow\ Find\ \delta(q_1, a)$
      ii. $r_2 \leftarrow\ Find\ \delta(q_2, a)$
      iii. se $r_1 \neq r_2$ então
         A. $Union\ r_1\ r_2$
         B. $Push\ S\ (r_1, r_2)$
5. no final, $L(M_1) = L(M_2)$ se e só se nenhum dos sub-conjuntos da estrutura Union-Find contém simultaneamente um estado final e não final.

Input

A entreada deste problema consiste na descrição dos dois autómatos $M_1 \triangleq (Q_1, \Sigma, \delta_1, s_1, F_1)$ e $M_2 \triangleq (Q_2, \Sigma, \delta_2, s_2, F_2)$.

Para simplificar o formatos dos dados em entrada admitiremos aqui que o conjunto Q_1 é sempre da forma $\{1 \ldots n\}$ (n inteiro) e que Q_2 é sempre da forma $\{n+1 \ldots n+m\}$ (m inteiro). Assim, é garantido que $Q_1 \cap Q_2 = \emptyset$.

Admitiremos igualmente que Σ é o alfabeto português $\{a - z\}$.

Assim $M_1 \triangleq (\{1, \ldots, n\}, \Sigma, \delta_1, s_1, F_1)$ pode ser introduzido por:

- uma linha com o inteiro n, especificando o conjunto $Q_1 = \{1, \ldots, n\}$;
- uma linha com o inteiro representa o estado inicial s_1 ($s_1 \in Q_1$);
- uma linha com o inteiro f_1 (cardinalidade do conjunto F_1 dos estados finais);
- uma linha com f_1 inteiros distintos que formam o conjunto dos estados finais;
- uma linha com o numero t_1 de transições (a cardinalidade de δ_1));
- t_1 linhas em que cada uma delas introduz uma transição sob a forma de `i c j`, sendo `i` o inteiro representando o estado de partida da transição, `c` o caracter no rótulo da transição e `j` o inteiro que representa o estado de chegada.

As restants linhas introduzem M_2 da mesma forma. Assim $M_2 \triangleq (\{n+1, \ldots, n+m\}, \Sigma, \delta_2, s_2, F_2)$ pode ser introduzido por:

- uma linha com o inteiro m, especificando o conjunto $Q_2 = \{n+1, \ldots, n+m\}$;
- uma linha com o inteiro representa o estado inicial s_2 ($s_2 \in Q_2$);
- uma linha com o inteiro f_2 (cardinalidade do conjunto F_2 dos estados finais);

- uma linha com f_2 inteiros distintos que formam o conjunto dos estados finais;
- uma linha com o numero t_2 de transições (a cardinalidade de δ_2));
- t_2 linhas em que cada uma delas introduz uma transição sob a forma de `i c j`, sendo `i` o inteiro representando o estado de partida da transição, `c` o caracter no rótulo da transição e `j` o inteiro que representa o estado de chegada.

Output

Uma linha com `YES` se $L(M_1) = L(M_2)$, `NO` senão.

Constraints

$1 \leq n, m \leq 50$
$1 \leq t_1, t_2 \leq 1000$

Sample Input

```
6
1
2
2 3
12
1 a 2
1 b 5
2 a 3
2 b 4
3 a 2
3 b 4
4 a 5
4 b 1
5 a 5
5 b 6
6 a 3
6 b 4
3
7
1
8
6
7 a 8
7 b 9
8 a 8
8 b 9
9 a 9
9 b 7
```

Sample Output

```
YES
```