Exercício 01 - Alef, o Sapo Perdido no Labirinto (com Túneis):

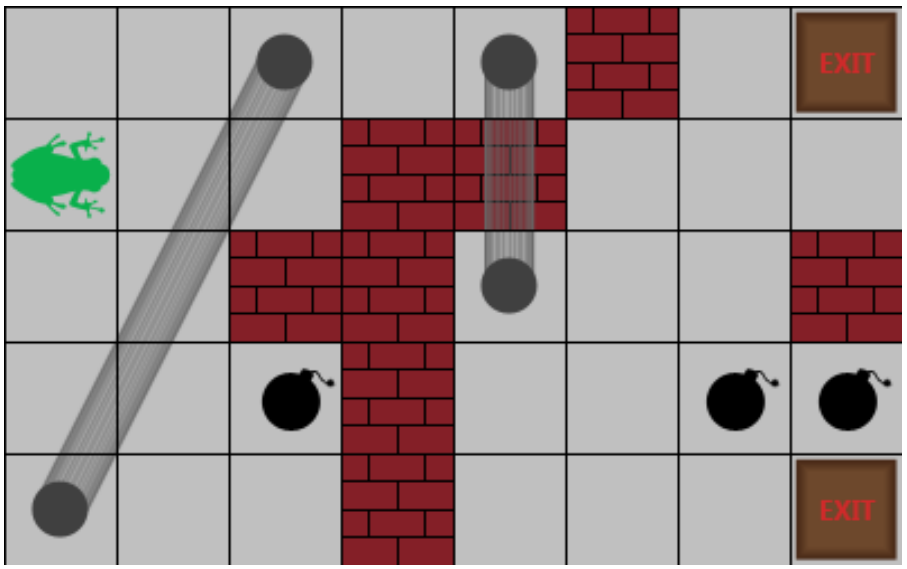
Contexto

Você deve ajudar **Alef, o Sapo** a escapar de um labirinto bidimensional com obstáculos, minas, saídas e túneis mágicos. Alef escolhe aleatoriamente uma das células adjacentes livres para se mover. Se cair em uma mina, ele morre. Se chegar a uma saída, ele escapa. Se entrar em um túnel, será transportado para a outra extremidade do túnel. Após o transporte, ele continua normalmente.

Seu objetivo é escrever um programa que calcule e imprima a **probabilidade** de Alef conseguir escapar do labirinto.

🧠 **Regras do Labirinto**

- Cada célula pode ser:
    - Livre (O)
    - Obstáculo (#)
    - Mina (*)
    - Saída (%)
    - Alef inicialmente (A)
- Alef pode se mover para qualquer célula **adjacente (cima, baixo, esquerda, direita)** desde que não haja obstáculo.
- Túneis conectam **pares de células livres** e funcionam em **duas vias**.
- O labirinto é cercado por paredes (obstáculos).



📥 Formato da Entrada

A primeira linha contém três inteiros: n m k

- n: número de linhas da matriz
- m: número de colunas
- k: número de túneis

As próximas n linhas contêm m caracteres representando o labirinto.

As próximas k linhas contêm quatro inteiros i1 j1 i2 j2

- Representam as posições (linha, coluna) dos dois extremos de um túnel.

📤 Formato da Saída

Um número real representando a **probabilidade de Alef escapar** do labirinto.
Sua resposta será considerada correta se o erro absoluto for menor que 10^{-6}.

📌 Restrições

- **1 ≤ n, m ≤ 20**
- **0 ≤ k ≤ n*m**
- Cada célula tem no máximo uma entrada de túnel.

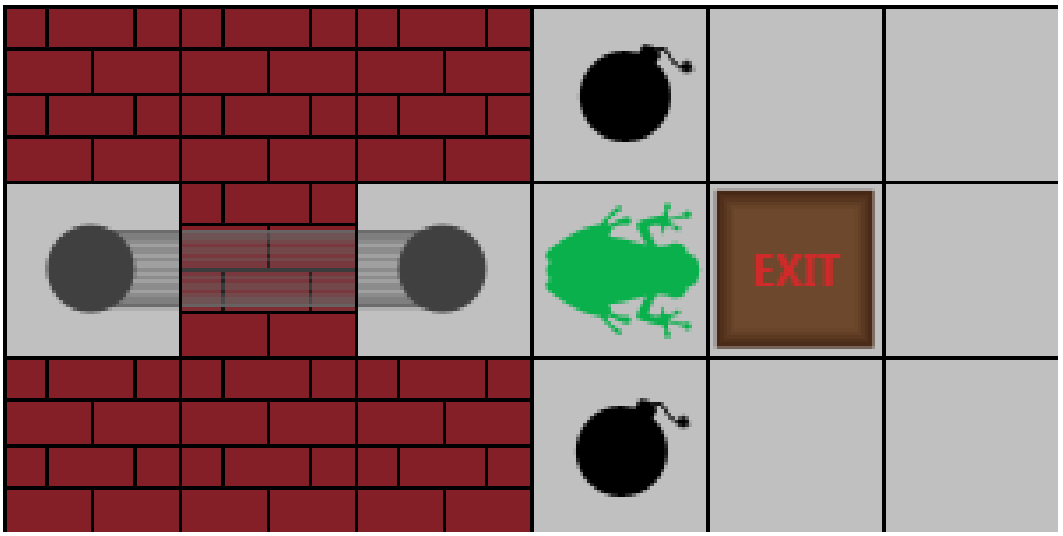
- Não há túneis entre células adjacentes.
- A célula inicial de Alef aparece exatamente uma vez.

🧪 Exemplo de Entrada

```
3 6 1
###*OO
O#OA%O
###*OO
2 3 2 1
```

✅ Saída Esperada

```
0.25
```



🖼️ Explicação

Alef está em uma célula com 4 possibilidades de movimento.

- Cima ou baixo → mina → morte.
- Direita → saída → vitória.
- Esquerda → túnel → célula cercada → preso.

Probabilidade de escapar = 1/4 = 0.25

📌 **Códigos Iniciais (Starter Code)**

Python:

```python
import random

def resolver_labirinto(n, m, k, maze, tunnels):
    # 🧠 Implemente aqui sua lógica
    # Retorne a probabilidade de fuga como float
    return 0.0

def gerar_labirinto(n, m):
    elementos = ['O'] * 5 + ['#', '*', '%']
    labirinto = [''.join(random.choice(elementos) for _ in
range(m)) for _ in range(n)]
    i, j = random.randint(0, n-1), random.randint(0, m-1)
    linha = list(labirinto[i])
    linha[j] = 'A'
    labirinto[i] = ''.join(linha)
    return labirinto

def gerar_tuneis(k, n, m):
    tuneis = set()
    while len(tuneis) < k:
        i1, j1 = random.randint(0, n-1), random.randint(0, m-1)
        i2, j2 = random.randint(0, n-1), random.randint(0, m-1)
        if (i1 != i2 or j1 != j2):
            tuneis.add((i1, j1, i2, j2))
    return list(tuneis)

def main():
    n, m, k = 5, 6, 2
    maze = gerar_labirinto(n, m)
    tunnels = gerar_tuneis(k, n, m)

    print("Maze:")
    for linha in maze:
        print(linha)
    print("Tunnels:", tunnels)

    resultado = resolver_labirinto(n, m, k, maze, tunnels)
    print("Probabilidade de fuga:", resultado)
```

```
if __name__ == '__main__':
    main()
```

JavaScript(Node.js):

```
function resolverLabirinto(n, m, k, maze, tunnels) {
    // 🧠 Implemente aqui sua lógica
    // Retorne um número (probabilidade de fuga)
    return 0.0;
}

function gerarLabirinto(n, m) {
    const elementos = ['O', 'O', 'O', 'O', 'O', '#', '*', '%'];
    const maze = [];

    for (let i = 0; i < n; i++) {
        let linha = '';
        for (let j = 0; j < m; j++) {
            linha += elementos[Math.floor(Math.random() *
elementos.length)];
        }
        maze.push(linha);
    }

    const i = Math.floor(Math.random() * n);
    const j = Math.floor(Math.random() * m);
    maze[i] = maze[i].substring(0, j) + 'A' + maze[i].substring(j
+ 1);

    return maze;
}

function gerarTuneis(k, n, m) {
    const tuneis = new Set();
    while (tuneis.size < k) {
        const i1 = Math.floor(Math.random() * n);
        const j1 = Math.floor(Math.random() * m);
        const i2 = Math.floor(Math.random() * n);
        const j2 = Math.floor(Math.random() * m);
        if (i1 !== i2 || j1 !== j2) {
            tuneis.add(`${i1},${j1},${i2},${j2}`);
```

```
        }
    }
    return Array.from(tuneis).map(e => e.split(',').map(Number));
}

function main() {
    const n = 5, m = 6, k = 2;
    const maze = gerarLabirinto(n, m);
    const tunnels = gerarTuneis(k, n, m);

    console.log("Maze:");
    maze.forEach(row => console.log(row));
    console.log("Tunnels:", tunnels);

    const resultado = resolverLabirinto(n, m, k, maze, tunnels);
    console.log("Probabilidade de fuga:", resultado);
}

main();
```

C:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

float resolverLabirinto(int n, int m, int k, char maze[n][m + 1],
int tunnels[k][4]) {
    // 🧠 Implemente aqui sua lógica
    // Retorne a probabilidade como float
    return 0.0;
}

char gerarElemento() {
    char opcoes[] = {'O', 'O', 'O', 'O', 'O', '#', '*', '%'};
    return opcoes[rand() % 8];
}

int main() {
    srand(time(NULL));
    int n = 5, m = 6, k = 2;
    char maze[n][m + 1];
```

```
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++)
            maze[i][j] = gerarElemento();
        maze[i][m] = '\0';
    }

    int si = rand() % n;
    int sj = rand() % m;
    maze[si][sj] = 'A';

    int tunnels[k][4];
    for (int i = 0; i < k; i++) {
        int i1, j1, i2, j2;
        do {
            i1 = rand() % n;
            j1 = rand() % m;
            i2 = rand() % n;
            j2 = rand() % m;
        } while (i1 == i2 && j1 == j2);
        tunnels[i][0] = i1;
        tunnels[i][1] = j1;
        tunnels[i][2] = i2;
        tunnels[i][3] = j2;
    }

    printf("Maze:\n");
    for (int i = 0; i < n; i++)
        printf("%s\n", maze[i]);

    printf("Tunnels:\n");
    for (int i = 0; i < k; i++)
        printf("%d %d %d %d\n", tunnels[i][0], tunnels[i][1],
tunnels[i][2], tunnels[i][3]);

    float resultado = resolverLabirinto(n, m, k, maze, tunnels);
    printf("Probabilidade de fuga: %.6f\n", resultado);

    return 0;
}
```

Exercício 2: Contando movimento da ordenação

📘 Enunciado

O **Insertion Sort** é um algoritmo de ordenação simples, bastante eficiente para pequenos conjuntos de dados. Cada vez que um número é "empurrado" para a direita para inserir um valor, dizemos que houve um **deslocamento**.

Considere um vetor de inteiros. Você deve implementar uma função que **calcula o número total de deslocamentos** (shifts) necessários para ordená-lo usando Insertion Sort.

🧠 Definição

Se `k[i]` é o número de elementos sobre os quais o elemento na posição `i` teve que passar (deslocar), então o total de deslocamentos é a soma:

k[1]+k[2]+...+k[n]k[1] + k[2] + ... + k[n] k[1]+k[2]+...+k[n]

🧾 Entrada

- Um inteiro `t` representando o número de casos de teste.
- Para cada caso:
    - Um inteiro `n` indicando o tamanho do vetor.
    - Uma linha com `n` inteiros, representando os elementos do vetor.

📤 Saída

Para cada caso de teste, imprima o número de deslocamentos necessários para ordenar o vetor usando Insertion Sort.

📌 Restrições

- $1 \leq t \leq 15$
- $1 \leq n \leq 100000$

- $1 \leq a[i] \leq 1000000$

📥 Exemplo de Entrada

```
2
5
1 1 1 2 2
5
2 1 3 1 2
```

✅ Exemplo de Saída

```
0
4
```

💡 Explicação

No segundo exemplo:

```
Array: 2 1 3 1 2
Passos:
→ 1 2 3 1 2 → deslocamento 1
→ 1 1 2 3 2 → deslocamento 2
→ 1 1 2 2 3 → deslocamento 1
Total: 4 deslocamentos
```

📌 Códigos Iniciais (Starter Code):

JavaScript:

```javascript
'use strict';

function insertionSort(arr) {
    // 🧠 Implemente aqui o algoritmo de Insertion Sort
    // Deve retornar a quantidade de deslocamentos
}

function generateRandomArray(size, min = 1, max = 100) {
    const arr = [];
    for (let i = 0; i < size; i++) {
        arr.push(Math.floor(Math.random() * (max - min + 1)) +
```

```
min);
    }
    return arr;
}

function main() {
    const arr = generateRandomArray(10);
    console.log("Array de entrada:", arr);

    const result = insertionSort(arr);
    console.log("Deslocamentos realizados:", result);
}

main();
```

C:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int insertionSort(int arr_count, int* arr) {
    // 🧠 Implemente aqui o algoritmo de Insertion Sort
    // Deve retornar a quantidade de deslocamentos
    return 0;
}

void generateRandomArray(int* arr, int size, int min, int max) {
    for (int i = 0; i < size; i++) {
        arr[i] = min + rand() % (max - min + 1);
    }
}

void printArray(int* arr, int size) {
    printf("Array de entrada: [");
```

```c
    for (int i = 0; i < size; i++) {
        printf("%d", arr[i]);
        if (i < size - 1) printf(", ");
    }
    printf("]\n");
}

int main() {
    srand(time(NULL));
    int n = 10;
    int arr[n];

    generateRandomArray(arr, n, 1, 100);
    printArray(arr, n);

    int result = insertionSort(n, arr);
    printf("Deslocamentos realizados: %d\n", result);

    return 0;
}
```

Python:

```python
import random

def insertionSort(arr):
    # 🧠 Implemente aqui o algoritmo de Insertion Sort
    # Deve retornar a quantidade de deslocamentos
    return 0

def generate_random_array(size, min_val=1, max_val=100):
    return [random.randint(min_val, max_val) for _ in range(size)]

def main():
    arr = generate_random_array(10)
    print("Array de entrada:", arr)

    result = insertionSort(arr)
    print("Deslocamentos realizados:", result)
```

```
if __name__ == '__main__':
    main()
```

Exercício 03: Notificação de Gastos Suspeitos

O banco HackerLand National possui uma política para alertar clientes sobre possíveis atividades suspeitas em suas contas. Se o valor gasto por um cliente em um determinado dia for maior ou igual ao dobro da **mediana** dos gastos dos `d` **dias anteriores**, o cliente recebe uma **notificação de possível fraude**.

O banco **só começa a monitorar e enviar notificações após ter dados de pelo menos `d` dias anteriores**. Seu objetivo é calcular quantas notificações serão enviadas ao longo de `n` dias.

📌 Exemplo

Considere o vetor de gastos:

```
expenditure = [10, 20, 30, 40, 50]
d = 3
```

Nos três primeiros dias, os dados são apenas coletados. No 4º dia, os últimos 3 gastos são `[10, 20, 30]`, e a mediana é `20`. O gasto no 4º dia foi `40`, que é igual a `2 × 20`, portanto uma **notificação será enviada**.

No 5º dia, os últimos 3 gastos são `[20, 30, 40]`, mediana = `30`. O gasto do dia é `50`, que é menor que `2 × 30`. **Nenhuma notificação será enviada**.

Total: 1 notificação enviada.

📌 Observação

A mediana de um conjunto de números é:

- O valor central se o número de elementos for ímpar.
- A média dos dois valores centrais, se o número for par.

✅ Sua Tarefa

Implemente uma função que receba:

- `expenditure[]`: lista de gastos diários
- `d`: número de dias anteriores usados para calcular a mediana

E retorne:

- A quantidade total de notificações enviadas.

📥 Entrada

- Um inteiro `n` (número total de dias) e um inteiro `d` (dias anteriores usados para mediana)
- Um vetor de `n` inteiros representando os gastos diários

📤 Saída

- Um único número inteiro representando a quantidade de notificações enviadas

---

🎯 Restrições

- `1 ≤ n ≤ 200000`
- `1 ≤ d ≤ n`
- `0 ≤ expenditure[i] ≤ 200`

🧪 Exemplo de Entrada

```
9 5
2 3 4 2 3 6 8 4 5
```

✅ Exemplo de Saída

```
2
```

🧠 Explicação

- No 6º dia, os gastos anteriores são `[2, 3, 4, 2, 3]`, mediana = `3`. Gasto = `6 → 6 ≥ 2×3` → notificação.
- No 7º dia, `[3, 4, 2, 3, 6]`, mediana = `3`. Gasto = `8 → 8 ≥ 2×3` → notificação.
- Dias seguintes não geram alertas.

Total = **2 notificações**.

📌 **Códigos Iniciais (Starter Code):**

JavaScript:

```javascript
function activityNotifications(expenditure, d) {
    // 🧠 Implemente aqui sua lógica
    // Deve retornar o número de notificações
    return 0;
}

function gerarGastos(n, max = 200) {
    const gastos = [];
    for (let i = 0; i < n; i++) {
        gastos.push(Math.floor(Math.random() * (max + 1)));
    }
    return gastos;
}

function main() {
    const n = 10, d = 5;
    const gastos = gerarGastos(n);

    console.log(`n = ${n}, d = ${d}`);
    console.log("Gastos:", gastos);

    const resultado = activityNotifications(gastos, d);
    console.log("Notificações:", resultado);
}

main();
```

C:

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int activityNotifications(int* expenditure, int n, int d) {
    // 🧠 Implemente aqui sua lógica
    // Retorne o número de notificações
    return 0;
}

void gerarGastos(int* arr, int n, int max) {
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % (max + 1);
    }
}

void printArray(int* arr, int n) {
    printf("Gastos: [");
    for (int i = 0; i < n; i++) {
        printf("%d", arr[i]);
        if (i < n - 1) printf(", ");
    }
    printf("]\n");
}

int main() {
    srand(time(NULL));
    int n = 10, d = 5;
    int gastos[n];

    gerarGastos(gastos, n, 200);

    printf("n = %d, d = %d\n", n, d);
    printArray(gastos, n);

    int resultado = activityNotifications(gastos, n, d);
    printf("Notificações: %d\n", resultado);

    return 0;
}
```

Python:

```python
import random

def activityNotifications(expenditure, d):
    # 🧠 Implemente aqui sua lógica
    # Deve retornar o número de notificações enviadas
    return 0

def gerar_dados(n, max_val=200):
    return [random.randint(0, max_val) for _ in range(n)]

def main():
    n, d = 10, 5
    gastos = gerar_dados(n)

    print(f"n = {n}, d = {d}")
    print("Gastos:", gastos)

    resultado = activityNotifications(gastos, d)
    print("Notificações:", resultado)

if __name__ == '__main__':
    main()
```

Exercício 04: Contagem de Cadeias Reconhecidas Por Expressões Regulares

📘 Enunciado

Uma **expressão regular** é usada para descrever um conjunto de cadeias de caracteres (strings). Neste exercício, o alfabeto é restrito apenas às letras `'a'` e `'b'`.

Uma expressão regular `R` é válida se:

1. `R` é `"a"` ou `"b"`.
2. `R` é da forma `(R₁ R₂)`, onde `R₁` e `R₂` são expressões regulares — representa a concatenação.
3. `R` é da forma `(R₁|R₂)`, onde `R₁` e `R₂` são expressões regulares — representa a união (alternância).
4. `R` é da forma `(R₁*)`, onde `R₁` é uma expressão regular — representa zero ou mais repetições de `R₁`.

Todas as expressões são bem formadas, com os parênteses sempre corretamente balanceados.

🧠 Reconhecimento de Strings

Dado `R`, o conjunto de strings reconhecido por ela é definido por:

- `"a"` reconhece a string `"a"`.
- `"b"` reconhece a string `"b"`.
- `(R₁ R₂)` reconhece todas as concatenações `s₁ + s₂`, onde `s₁ ∈ R₁` e `s₂ ∈ R₂`.
- `(R₁|R₂)` reconhece strings que pertencem a `R₁` ou a `R₂`.

- (R_1*) reconhece a concatenação de zero ou mais cópias de strings em R_1.

🧩 Tarefa

Dada uma expressão regular R e um inteiro L, determine **quantas strings de tamanho L** são reconhecidas por R.

Como a resposta pode ser grande, imprima o valor **modulo $10^9 + 7$**.

📥 Entrada

- A primeira linha contém um inteiro T representando o número de casos de teste.
- Cada uma das próximas T linhas contém:
    - Uma expressão regular válida R.
    - Um inteiro L.

📤 Saída

Para cada caso de teste, imprima uma linha com o número de strings de tamanho L reconhecidas pela expressão R.

🔢 Restrições

- $1 \le T \le 50$
- $1 \le |R| \le 100$ (tamanho da string da expressão)
- $1 \le L \le 10^9$
- Garante-se que a expressão R é válida.

🧪 Exemplo de Entrada

```
3
((ab)|(ba)) 2
((a|b)*) 5
((a*)(b(a*))) 100
```

✅ Exemplo de Saída

```
2
32
100
```

🧠 Explicação

- No primeiro caso: apenas `"ab"` e `"ba"` têm tamanho 2 e são reconhecidas.
- No segundo caso: a expressão reconhece todas as strings com apenas `'a'`s e `'b'`s. Há $2^5 = 32$ combinações de tamanho 5.
- No terceiro caso: strings com **um `b` cercado por qualquer número de `a`**. Existem exatamente 100 strings de tamanho 100 que têm exatamente um `b`.

📌 **Códigos Iniciais (Starter Code):**

JavaScript:

```javascript
function countRecognizedStrings(R, L) {
    // 🧠 Implemente aqui sua lógica
    // Retorne a quantidade de strings reconhecidas de tamanho L
    return 0;
}

function gerarExpressao() {
    const bases = ['a', 'b'];
    let exp = bases[Math.floor(Math.random() * 2)];
    for (let i = 0; i < Math.floor(Math.random() * 3) + 1; i++) {
        const op = ['|', '', '*'][Math.floor(Math.random() * 3)];
        if (op === '') {
            exp = `(${exp}${bases[Math.floor(Math.random() *
2)]})`;
        } else if (op === '|') {
            exp = `(${exp}|${bases[Math.floor(Math.random() *
2)]})`;
        } else if (op === '*') {
            exp = `(${exp}*)`;
        }
    }
    return exp;
}

function main() {
    const T = 3;
    const casos = [];

    for (let i = 0; i < T; i++) {
        const R = gerarExpressao();
        const L = Math.floor(Math.random() * 6) + 1;
        casos.push([R, L]);
    }

    casos.forEach(([R, L]) => {
```

```
        console.log(`Expressão: ${R}, Tamanho: ${L}`);
        const resultado = countRecognizedStrings(R, L);
        console.log("Reconhecidas:", resultado);
    });
}

main();
```

C:

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

int countRecognizedStrings(char* R, int L) {
    // 🧠 Implemente aqui sua lógica
    // Deve retornar a quantidade de strings reconhecidas de
tamanho L
    return 0;
}

void gerarExpressao(char* buffer, int max_len) {
    const char* bases[] = {"a", "b"};
    strcpy(buffer, bases[rand() % 2]);

    for (int i = 0; i < rand() % 3 + 1 && strlen(buffer) < max_len
- 5; i++) {
        int op = rand() % 3;
        if (op == 0) { // concatenação
            char c[2] = { "ab"[rand() % 2], '\0' };
            char temp[100];
            sprintf(temp, "(%s%s)", buffer, c);
            strcpy(buffer, temp);
        } else if (op == 1) { // alternância
            char c[2] = { "ab"[rand() % 2], '\0' };
            char temp[100];
            sprintf(temp, "(%s|%s)", buffer, c);
            strcpy(buffer, temp);
        } else if (op == 2) { // repetição
            char temp[100];
```

```
            sprintf(temp, "(%s*)", buffer);
            strcpy(buffer, temp);
        }
    }
}

int main() {
    srand(time(NULL));
    int T = 3;

    for (int i = 0; i < T; i++) {
        char R[100];
        gerarExpressao(R, sizeof(R));
        int L = rand() % 6 + 1;

        printf("Expressão: %s, Tamanho: %d\n", R, L);
        int resultado = countRecognizedStrings(R, L);
        printf("Reconhecidas: %d\n", resultado);
    }

    return 0;
}
```

Python:

```python
import random

def countRecognizedStrings(R, L):
    # 🧠 Implemente aqui sua lógica
    # Deve retornar um inteiro com a quantidade de strings
reconhecidas de tamanho L
    return 0

def gerar_expressao():
    # Simples gerador de expressões balanceadas
    bases = ["a", "b"]
    exp = random.choice(bases)
    for _ in range(random.randint(1, 3)):
        op = random.choice(["|", "", "*"])
        if op == "":
            exp = f"({exp}{random.choice(bases)})"
        elif op == "|":
            exp = f"({exp}|{random.choice(bases)})"
        elif op == "*":
            exp = f"({exp}*)"
    return exp

def main():
    T = 3
    casos = []
    for _ in range(T):
        R = gerar_expressao()
        L = random.randint(1, 6)
        casos.append((R, L))

    for R, L in casos:
        print(f"Expressão: {R}, Tamanho: {L}")
        resultado = countRecognizedStrings(R, L)
        print("Reconhecidas:", resultado)

if __name__ == '__main__':
```

```
    main()
```

Exercício 05: Contagem de Pares Similares em Árvore

Enunciado

Dado um grafo em forma de **árvore**, com `n` nós numerados de 1 a `n`, defina um **par similar** como um par de nós `(a, b)` que satisfaça:

1. O nó `a` é **ancestral** de `b`.
2. `abs(a - b) ≤ k`, onde `k` é um valor inteiro dado.

Você deverá implementar uma função que, dado `n`, `k` e a estrutura da árvore, conte **quantos pares similares** existem.

Formato da Entrada

- A primeira linha contém dois inteiros `n` (quantidade de nós) e `k` (limite de similaridade).
- As próximas `n-1` linhas contêm dois inteiros `pai filho`, indicando uma aresta da árvore (ou seja, `pai` é ancestral direto de `filho`).

Exemplo

Entrada:

```
5 2
3 2
3 1
1 4
1 5
```

Representação da Árvore:

Saída Esperada:

```
4
```

Python:

```python
import random

def similar_pair(n, k, edges):
    # IMPLEMENTAR AQUI
    return 0

def generateTestCases():
    return [
        (5, 2, [(3, 2), (3, 1), (1, 4), (1, 5)]),
        (6, 3, [(1, 2), (1, 3), (2, 4), (3, 5), (3, 6)]),
        (4, 1, [(1, 2), (2, 3), (3, 4)]),
        (7, 4, [(1, 2), (1, 3), (2, 4), (2, 5), (3, 6), (6, 7)]),
        (3, 0, [(1, 2), (2, 3)])
    ]

def main():
    testCases = generateTestCases()
    for idx, (n, k, edges) in enumerate(testCases, 1):
        print(f"🧪 Teste {idx}")
        print(f"n = {n}, k = {k}, edges = {edges}")
        result = similar_pair(n, k, edges)
        print(f"➡️ Resultado: {result}\n")

if __name__ == "__main__":
    main()
```

JavaScript:

```
function similarPair(n, k, edges) {
    // IMPLEMENTAR AQUI
    return 0;
}

function generateTestCases() {
    return [
        [5, 2, [[3, 2], [3, 1], [1, 4], [1, 5]]],
        [6, 3, [[1, 2], [1, 3], [2, 4], [3, 5], [3, 6]]],
        [4, 1, [[1, 2], [2, 3], [3, 4]]],
        [7, 4, [[1, 2], [1, 3], [2, 4], [2, 5], [3, 6], [6, 7]]],
        [3, 0, [[1, 2], [2, 3]]],
    ];
}

function main() {
    const tests = generateTestCases();
    tests.forEach(([n, k, edges], idx) => {
        console.log(`🧪 Teste ${idx + 1}`);
        console.log(`n = ${n}, k = ${k}, edges =
${JSON.stringify(edges)}`);
        const result = similarPair(n, k, edges);
        console.log(`➡️ Resultado: ${result}\n`);
    });
}

main();
```

C:

```c
#include <stdio.h>

int similarPair(int n, int k, int edges[][2], int m) {
    // IMPLEMENTAR AQUI
    return 0;
}

void runTest(int n, int k, int edges[][2], int m, int testNum) {
    printf("🧪 Teste %d\n", testNum);
    printf("n = %d, k = %d\nedges = ", n, k);
    for (int i = 0; i < m; i++) {
        printf("(%d,%d) ", edges[i][0], edges[i][1]);
    }
    printf("\n➡️ Resultado: %d\n\n", similarPair(n, k, edges, m));
}

int main() {
    int test1[][2] = {{3, 2}, {3, 1}, {1, 4}, {1, 5}};
    int test2[][2] = {{1, 2}, {1, 3}, {2, 4}, {3, 5}, {3, 6}};
    int test3[][2] = {{1, 2}, {2, 3}, {3, 4}};
    int test4[][2] = {{1, 2}, {1, 3}, {2, 4}, {2, 5}, {3, 6}, {6,
7}};
    int test5[][2] = {{1, 2}, {2, 3}};

    runTest(5, 2, test1, 4, 1);
    runTest(6, 3, test2, 5, 2);
    runTest(4, 1, test3, 3, 3);
    runTest(7, 4, test4, 6, 4);
    runTest(3, 0, test5, 2, 5);

    return 0;
}
```