II Competição Feminina de Programação da UnB 2023
Editorial

Sumário

A Aposta++

Basicamente, a questão pede para responder "YES" se o primeiro número for maior do que o primeiro e "NO" caso ao contrário. Se os dois números forem iguais, a resposta é "NO" porque o enunciado pede que o segundo número seja estritamente maior do que o primeiro.

A.1 Python

```python
a,b = map(int,input().split())

if a < b:
    print('YES')
else:
    print('NO')
```

A.2 C++

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int a, b; cin >> a >> b;

    if (b > a) cout << "YES\n";
    else cout << "NO\n";

    return 0;
}
```

B Sétimo Filho

A questão pede para responder "Sim" se os primeiros 7 caracteres forem iguais a "AAAAAAB", caso ao contrário a resposta é "Nao". Se a string tiver menos do que 7 caracteres, a resposta também é "Nao"

B.1 Python

```python
for _ in range(int(input())):
    if input().startswith('AAAAAAB'):
        print('Sim')
    else:
        print('Nao')
```

B.2 C++

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int n; cin >> n;
    while(n--) {
        string s; cin >> s;
        cout << ((s.substr(0, 7) == "AAAAAAB") ? "Sim\n" : "Nao\n");
    }

    return 0;
}
```

C Lobisomens

Como existe no máximo um lobisomem em uma cidade, existem duas possibilidades.

A primeira é não ter nenhum lobisomem na cidade, nesse caso todos falam a verdade, logo não vai ter nenhuma acusação do tipo 2 x (a pessoa entrevistada acusando alguém de mentir). Assim, se todos as respostas forem do tipo 1 x, a resposta para o problema é -1.

A segunda possibilidade é ter um único lobisomem na cidade. Nesse caso, se o entrevistado for um súdito, ele vai responder de uma das seguintes maneiras - 1 x, sendo x é o índice de um súdito comum - 2 lob, sendo lob é o índice do lobisomem

Já o lobisomem responderá de uma das seguintes maneiras: - 1 lob, sendo lob é o índice do lobisomem (ou seja, ele mente dizendo que ele mesmo sempre fala a verdade) - 2 x, sendo x o índice de um súdito comum (ele mente dizendo que um súdito comum sempre mente)

Como há no máximo um lobisomem, se houver dois ou mais entrevistados acusando a mesma pessoa de mentir, essa pessoa tem que ser o lobisomem. Nesse caso também pode haver um única pessoa acusando outro mentiroso, mas podemos ignorá-la porque essa pessoa é o lobisomem. Se não houver dois ou mais entrevistados acusando a mesma pessoa de mentir, não há nenhuma evidência de lobisomem e portanto a resposta é -1.

Assim o problema se resume a verificar se existe algum índice que duas ou mais pessoas estão acusando de mentir. Caso não houver, a resposta para o problema é -1.

C.1 Python

```python
n = int(input())
acusacoes = [0] * (n + 1)
for _ in range(n):
    q, x = input().split()
    if q == '2':
        i = int(x)
        acusacoes[i] += 1
        if acusacoes[i] > 1:
            print(i)
            break
else:
    print(-1)
```

C.2 C++

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int n; cin >> n;
    // conta quantas pesssoas est o acusando um determinado  ndice
    // de sempre mentir
    map<int, int> mentira;

    for (int i = 0; i < n; i++) {
        int q, x; cin >> q >> x;
        // se a acusa  o for do tipo 2 (o entrevistado est  acusando
        // a pessoa de  ndice  x de sempre mentir)
        if (q == 2) mentira[x]++;
    }

    int ans = -1;
    for (auto [key, value] : mentira) {
        // se houver mais de duas pessoas acusando esse  ndice ,
        // esse    o  ndice  do lobisomem
        if (value >= 2) ans = key;
    }

    cout << ans << '\n';

    return 0;
}
```

D Que Mario?

A questão pede para dizer se a união das das fases que Marco consegue passar e que Polo consegue passar é igual ao conjunto de todas as fases. Para isso podemos usar a estrutura de dados set, que armazena somente elementos distintos. Como o enunciado garante que todas as fases estão entre 1 e n, se no final o tamanho do set for igual a n, significa que eles conseguem passar todas as fases de 1 a n pois o set contém n elementos distintos.

D.1 Python

```python
n = int(input())

conj = set()

for idx,x in enumerate(input().split()):
    if idx > 0:
        conj.add(x)

for idx,x in enumerate(input().split()):
    if idx > 0:
        conj.add(x)

if len(conj) == n:
    print('Sou eu, Mario!')
else:
    print('Que Mario?')
```

D.2 C++

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

int main(){

    set<int> fases;

    int n; cin >> n;

    int p; cin >> p;
    for(int i=0; i<p; i++){
        int x; cin >> x;
        fases.insert(x);
    }

    int q; cin >> q;
    for(int i=0; i<q; i++){
        int x; cin >> x;
        fases.insert(x);
    }

    cout << (fases.size() == n ? "Sou eu, Mario!" : "Que Mario?") << endl;

    return 0;
}
```

E Olha a cobra!

Uma maneira de resolver essa questão é olhando a paridade do índice de cada linha. Vamos dizer que a primeira linha tem índice 0, a segunda índice 1 e assim por diante. As linhas pares $(0, 2, 4, 6, 8, 10, 12 \dots)$, ou seja, as linhas cujo índice quando dividido por 2 deixa resto 0, vão ser sempre completamente preenchidas com #. Já para as linhas de número ímpar, há dois casos: ou a linha é completamente preenchida com . exceto pelo primeiro caractere ou a linha é completamente preenchida por . exceto pelo último caractere. Assim, as linhas $1, 5, 9, \dots$, ou seja, linhas cujo índice quando dividido por 4 deixa resto 1 tem # no último caractere e as linhas $3, 7, 11, \dots$, ou seja, as linhas cujo índice quando dividido por 4 deixa resto 3 tem # no primeiro caractere.

Outro jeito de resolver esse problema é inicialmente preencher todos o tabuleiro com # e depois, começando pela linha de índice 1, ir percorrendo as linhas ímpares e preenchendo com . e usar uma variável booleana para alterar a posição da # entre a primeira e a última posição.

E.1 Python

```python
n, m = map(int, input().split())

for i in range(n):
    for j in range(m):
        if (i % 2 == 0):
            print("#", end="")
        elif (i % 4 == 1):
            if j == m - 1:
                print("#", end="")
            else:
                print(".", end="")
        elif (i % 4 == 3):
            if j == 0:
                print("#", end="")
            else:
                print(".", end="")
    print(end="\n")
```

E.2 C++

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int n, m; cin >> n >> m;
    char grid[n][m];

    for (int i = 0; i < n; i+=2) {
        for (int j = 0; j < m; j++) {
            grid[i][j] = '#';
        }
    }

    bool flag = true;
    for (int i = 1; i < n; i+=2) {
        for (int j = 0; j < m; j++) {
            grid[i][j] = '.';
        }
        if (flag) grid[i][m - 1] = '#';
        else grid[i][0] = '#';
        flag = !flag;
    }

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            cout << grid[i][j];
        }
        cout << '\n';
    }

    return 0;
}
```

F Eu acho que ouvi um gatinho

Uma maneira de resolver essa questão é converter a string toda para lowercase, e então construir uma nova string adicionando somente os caracteres que são diferente do caractere que foi adiciona mais recentemente. Se a string construída for igual a "meow" no final, então a resposta é "Yes", caso ao contrário a resposta é "No".

Outra maneira de resolver é percorrer a string marcando qual foi o último caractere adicionado à string e qual é o próximo caractere de "meow" que estamos procurando. Se em algum momento o caractere atual for diferente do último caractere adiciona e do caractere que estamos procurando, então a resposta é "No". Quando encontramos todos os caracteres, podemos usar algum outro caractere para marcar que encontramos todos os caracteres que procurávamos.

F.1 Python

```python
t = int(input())
for _ in range(t):
    n = int(input())
    s = input().lower()

    nova = s[0]

    for i in range(len(s)):
        if i != 0 and s[i] != s[i - 1]:
            nova += s[i]

    if nova == "meow":
        print("YES")
    else:
        print("NO")
```

F.2 C++

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int t; cin >> t;
    while (t--) {
        int n; cin >> n;
        string s; cin >> s;

        char flag = 'm', prev = 'm';
        bool ans = true;

        for (int i = 0; i < n; i++) {
            if (tolower(s[i]) == flag) {
                if (flag == 'm') {
                    flag = 'e';
                    prev = 'm';
                } else if (flag == 'e') {
                    flag = 'o';
                    prev = 'e';
                } else if (flag == 'o') {
                    flag = 'w';
                    prev = 'o';
                } else if (flag == 'w') {
                    flag = '*';
                    prev = 'w';
                }
            }

            else if (tolower(s[i]) != prev) ans = false;
        }

        if (flag != '*') ans = false;

        cout << (ans ? "YES\n" : "NO\n");
    }

    return 0;
}
```

G As Meninas Super Programadoras!

O que o exercício pede é que dados dois números a e b começando com zero, você os transforme nos dois números dados como entrada realizando uma das seguintes operações:

a Adicionar um número k (escolhido por você) a ambos a e b

b Adicionar um número k (escolhido por você) ao número a e subtrair esse mesmo k do número b

c Adicionar um número k (escolhido por você) ao número b e subtrair esse mesmo k do número a

Você pode realizar essa operação várias vezes, podendo escolher um k diferente a cada operação realizada. A resposta da questão é o mínimo de operações necessárias para transformar 0 e 0 nos números fornecidos na entrada ou imprimir -1 caso seja impossível fazer isso.

A ideia subtrair do menor número a metade da diferença entre os números e somar a metade da diferença ao outro número. Então é só usar a operação do tipo a para obter os números desejados. Note que temos 4 casos:

- A resposta será -1 se a diferença entre os números for ímpar (não dá pra pegar a metade da diferença nesse caso)
- A resposta é 0 se ambos os números forem 0 (não é necessário realizar nenhuma operação)
- A reposta é 1 se a e b foram iguais, pois a única operação necessário é somar k = a (e logo k = b) a ambos os números
- A reposta é 2 para todos os demais casos, pois podemos realizar as duas operações descritas acima

G.1 Python

```python
t = int(input())
for _ in range(t):
    a, b = map(int, input().split())
    if abs(a - b) % 2 == 1:
        print(-1)
    elif a == 0 and b == 0:
        print(0)
    elif a == b:
        print(1)
    else:
        print(2)
```

G.2 C++

```cpp
#include <bits/stdc++.h>

using namespace std;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int t; cin >> t;
    while (t--) {
        int c, d; cin >> c >> d;
        if (abs(c - d) % 2 == 1) {
            cout << -1 << '\n';
        } else if (c == 0 && d == 0) {
            cout << 0 << '\n';
        } else if (c == d) {
            cout << 1 << '\n';
        } else {
            cout << 2 << '\n';
        }
    }

    return 0;
}
```

H Pera com Gorgonzola

O elevador 1 demora a - 1 para chegar ao primeiro andar, já o elevador 2 demora a diferença entre os andares b e c mais c - 1, ou seja, o tempo de ir do andar b para o andar c e depois para o andar 1. Como o andar b pode estar antes ou depois do andar c, pegamos o valor absoluto para garantir que a diferença não seja negativa.

H.1 Python

```python
t = int(input())

for _ in range(t):
    a, b, c = map(int, input().split())

    if abs(b - c) + c - 1 > a - 1:
        print(1)
    elif abs(b - c) + c - 1 < a - 1:
        print(2)
    else:
        print(3)
```

H.2 C++

```cpp
#include<bits/stdc++.h>

using namespace std;

const int MAX = 2e5+17;

int main() {
    ios::sync_with_stdio(false);
    cin.tie(NULL);

    int t; cin >> t;

    while (t--) {
        int a, b, c; cin >> a >> b >> c;

        if (abs(b - c) + c - 1 > a - 1) cout << "1\n";
        else if (abs(b - c) + c - 1 < a - 1) cout << "2\n";
        else cout << "3\n";
    }

    return 0;
}
```