RESOLUÇÃO DE PROBLEMAS

Grupo: BIT BY BIT

Estruturas Intermediárias (set, map, priority queue, union-find)

A) ANTONS AND LETTERS (https://codeforces.com/problemset/problem/443/A)

Explicação:

O problema é contar quantas letras minúsculas distintas aparecem na entrada (entre chaves e vírgulas).

Basta filtrar somente caracteres minúsculos e armazená-los em um conjunto.

O tamanho desse conjunto será o número de letras distintas.

Código em PYTHON:

```python

entrada = input()
entrada = entrada.strip('{}').strip()

# Se o conjunto estiver vazio após tirar chaves e espaços, não há letras
if not entrada:
    print(0)
else:
    # Cria set para remover duplicatas; cada item é separado por ", "
    letras = set(entrada.split(', '))
    print(len(letras))
```

B) TWO GRAM (https://codeforces.com/problemset/problem/977/B)

Explicação:

Dada uma string, extraímos cada par (two-gram) de caracteres consecutivos.

Contamos quantas vezes cada par aparece e escolhemos o mais frequente.

Código em C++:

```cpp
#include <iostream>
#include <unordered_map>
#include <string>
using namespace std;

int main() {
    int n;
    string s;
    cin >> n >> s;
    unordered_map<string, int> freq;
    string maxTwoGram;
    int maxCount = 0;

    // Percorre a string de i até i+2
    for (int i = 0; i < n - 1; i++) {
        string twoGram = s.substr(i, 2);
        freq[twoGram]++;

        // Atualiza o par mais frequente
        if (freq[twoGram] > maxCount) {
            maxCount = freq[twoGram];
            maxTwoGram = twoGram;
        }
    }

    cout << maxTwoGram << endl;
    return 0;
}
```

Código em PYTHON:

```python
n = int(input())
s = input()
contagem = {}

# Percorre cada par consecutivo
for i in range(n - 1):
    par = s[i:i+2]
    contagem[par] = contagem.get(par, 0) + 1

# Encontra o que teve maior frequência
mais_frequente = max(contagem, key=contagem.get)
print(mais_frequente)
```

C) DAWIG AND BAG OF CANDIES (https://codeforces.com/problemset/problem/1230/A)
Explicação:

Há 4 valores que precisam ser divididos em dois grupos de soma igual.

Testam-se todas as formas possíveis de separar esses números, inclusive casos de 3 contra 1.

Código em C++:

```cpp
#include <iostream>
using namespace std;

int main() {
    int a[4];
    for (int i = 0; i < 4; i++) {
        cin >> a[i];
    }

    // Verifica cada modo de dividir em grupos de soma igual
    if (a[0] + a[1] == a[2] + a[3]) { cout << "YES\n"; return 0; }
    if (a[0] + a[2] == a[1] + a[3]) { cout << "YES\n"; return 0; }
    if (a[0] + a[3] == a[1] + a[2]) { cout << "YES\n"; return 0; }
    if (a[0] == a[1] + a[2] + a[3]) { cout << "YES\n"; return 0; }
    if (a[1] == a[0] + a[2] + a[3]) { cout << "YES\n"; return 0; }
    if (a[2] == a[0] + a[1] + a[3]) { cout << "YES\n"; return 0; }
    if (a[3] == a[0] + a[1] + a[2]) { cout << "YES\n"; return 0; }

    cout << "NO\n";
    return 0;
}
```

Código em PYTHON:

```python
from itertools import combinations

a = list(map(int, input().split()))
possible = False

# Testa combinações de 1 elemento (contra 3)
for group in combinations(a, 1):
    if sum(group) == sum(a) - sum(group):
        possible = True
        break

# Testa combinações de 2 elementos (contra 2)
if not possible:
    for group in combinations(a, 2):
        if sum(group) == sum(a) - sum(group):
            possible = True
            break

print("YES" if possible else "NO")
```

D) THOR (https://codeforces.com/problemset/problem/705/C)
Explicação:

Simula um sistema de notificações com três tipos de evento:

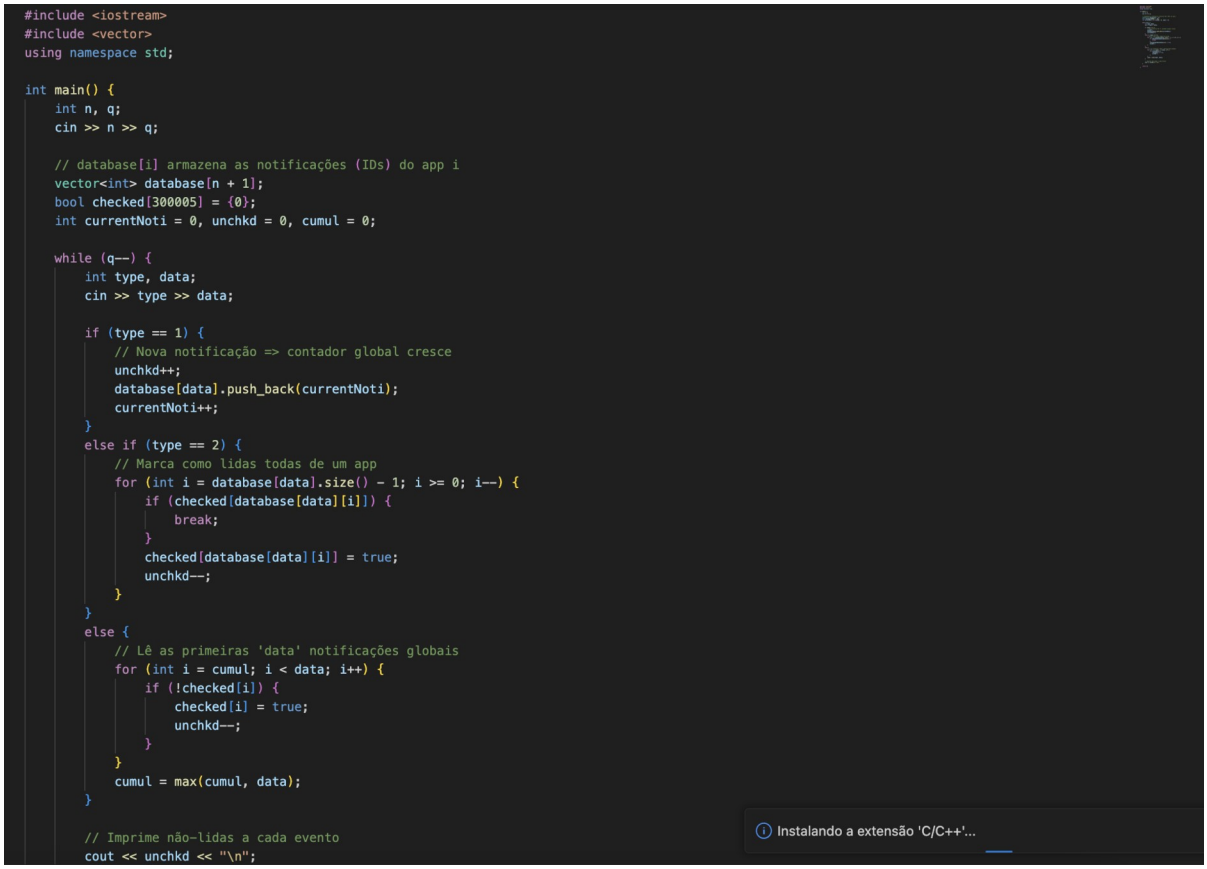
1) Adicionar notificação a um app,

2) Ler todas notificações de um app,

3) Ler as primeiras x notificações globais.

Mantém-se um contador de notificações não-lidas e um array que marca se cada notificação foi lida.

Código em C++:



```cpp
#include <iostream>
#include <vector>
using namespace std;

int main() {
    int n, q;
    cin >> n >> q;

    // database[i] armazena as notificações (IDs) do app i
    vector<int> database[n + 1];
    bool checked[300005] = {0};
    int currentNoti = 0, unchkd = 0, cumul = 0;

    while (q--) {
        int type, data;
        cin >> type >> data;

        if (type == 1) {
            // Nova notificação => contador global cresce
            unchkd++;
            database[data].push_back(currentNoti);
            currentNoti++;
        }
        else if (type == 2) {
            // Marca como lidas todas de um app
            for (int i = database[data].size() - 1; i >= 0; i--) {
                if (checked[database[data][i]]) {
                    break;
                }
                checked[database[data][i]] = true;
                unchkd--;
            }
        }
        else {
            // Lê as primeiras 'data' notificações globais
            for (int i = cumul; i < data; i++) {
                if (!checked[i]) {
                    checked[i] = true;
                    unchkd--;
                }
            }
            cumul = max(cumul, data);
        }

        // Imprime não-lidas a cada evento
        cout << unchkd << "\n";
```

Instalando a extensão 'C/C++'...

Código em PYTHON:

```python
import sys

def main():
    data = sys.stdin.read().split()
    if not data:
        return

    it = iter(data)
    n = int(next(it))
    q = int(next(it))

    # database[x] é a lista de IDs das notificações do app x
    database = [[] for _ in range(n + 1)]
    checked = [False] * (q + 5)
    currentNoti = 0
    unchkd = 0
    cumul = 0
    res = []

    for _ in range(q):
        typ = int(next(it))
        if typ == 1:
            x = int(next(it))
            unchkd += 1
            database[x].append(currentNoti)
            currentNoti += 1
            res.append(str(unchkd))
        elif typ == 2:
            x = int(next(it))
            # Lê todas notificações do app x
            for notif in reversed(database[x]):
                if checked[notif]:
                    break
                checked[notif] = 
                unchkd -= 1
            res.append(str(unchkd))
        else:
            t = int(next(it))
            # Lê as primeiras t notificações globais
            while cumul < t:
                if not checked[cumul]:
                    checked[cumul] = True
                    unchkd -= 1
                cumul += 1
            res.append(str(unchkd))

    sys.stdout.write("\n".join(res))

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

E) EQUALIZE THE ARRAY (https://vjudge.net/contest/705045#problem/E **)**

Explicação:

O objetivo é deletar o mínimo de elementos para que reste apenas o valor que aparece com maior frequência.

Basta encontrar essa frequência máxima e subtrair do tamanho total.

Código em C++:

```cpp
#include <iostream>
#include <vector>
#include <unordered_map>
#include <algorithm>
#include <climits>

using namespace std;

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

    int t;
    cin >> t;
    while (t--) {
        int n;
        cin >> n;
        vector<int> arr(n);
        for (int i = 0; i < n; i++) {
            cin >> arr[i];
        }

        // Frequência de cada número
        unordered_map<int, int> freqMap;
        for (int x : arr) {
            freqMap[x]++;
        }

        // Frequências em um vetor separado
        vector<int> freq_list;
        freq_list.reserve(freqMap.size());
        for (auto &p : freqMap) {
            freq_list.push_back(p.second);
        }

        // Ordena a lista de frequências
        sort(freq_list.begin(), freq_list.end());

        long long total = 0;
        for (int f : freq_list) {
            total += f;
        }

        // Caso não existam frequências (n==0), resposta é 0 (mas em geral n>=1)
        if (freq_list.empty()) {
            cout << 0 << "\n";
            continue;
        }

        long long best = LLONG_MAX;
        // Maior frequência
        int maxF = freq_list.back();

        // Testa todos os c de 1 até maxF
        for (int c = 1; c <= maxF; c++) {
            // Acha quantos valores são >= c
            auto it = lower_bound(freq_list.begin(), freq_list.end(), c);
            int count = freq_list.end() - it;
            if (count == 0) {
                break;
            }
            long long removals = total - (long long)count * c;
            if (removals < best) {
                best = removals;
            }
        }

        cout << best << "\n";
    }

    return 0;
}
```

Código em PYTHON:

```python
def main():
    t = int(input().strip())
    results = []
    from bisect import bisect_left

    for _ in range(t):
        n = int(input().strip())
        arr = list(map(int, input().split()))

        # Conta as ocorrências de cada número
        freq = {}
        for x in arr:
            freq[x] = freq.get(x, 0) + 1

        # Lista de frequências ordenada
        freq_list = sorted(freq.values())
        total = sum(freq_list)
        m = len(freq_list)
        best = total
        max_f = freq_list[-1] if freq_list else 0

        # Testa todos os candidatos para C
        for C in range(1, max_f + 1):
            cnt = m - bisect_left(freq_list, C)
            if cnt == 0:
                break
            removals = total - cnt * C
            if removals < best:
                best = removals
        results.append(str(best))

    print("\n".join(results))

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

F) REGISTRATION SYSTEM (https://vjudge.net/contest/705045#problem/F **)**
Explicação:

Faz o cadastro de novos nomes de usuário. Se já existir, gera-se uma variação com número e incrementa-se o contador no mapa.

Código em C++:

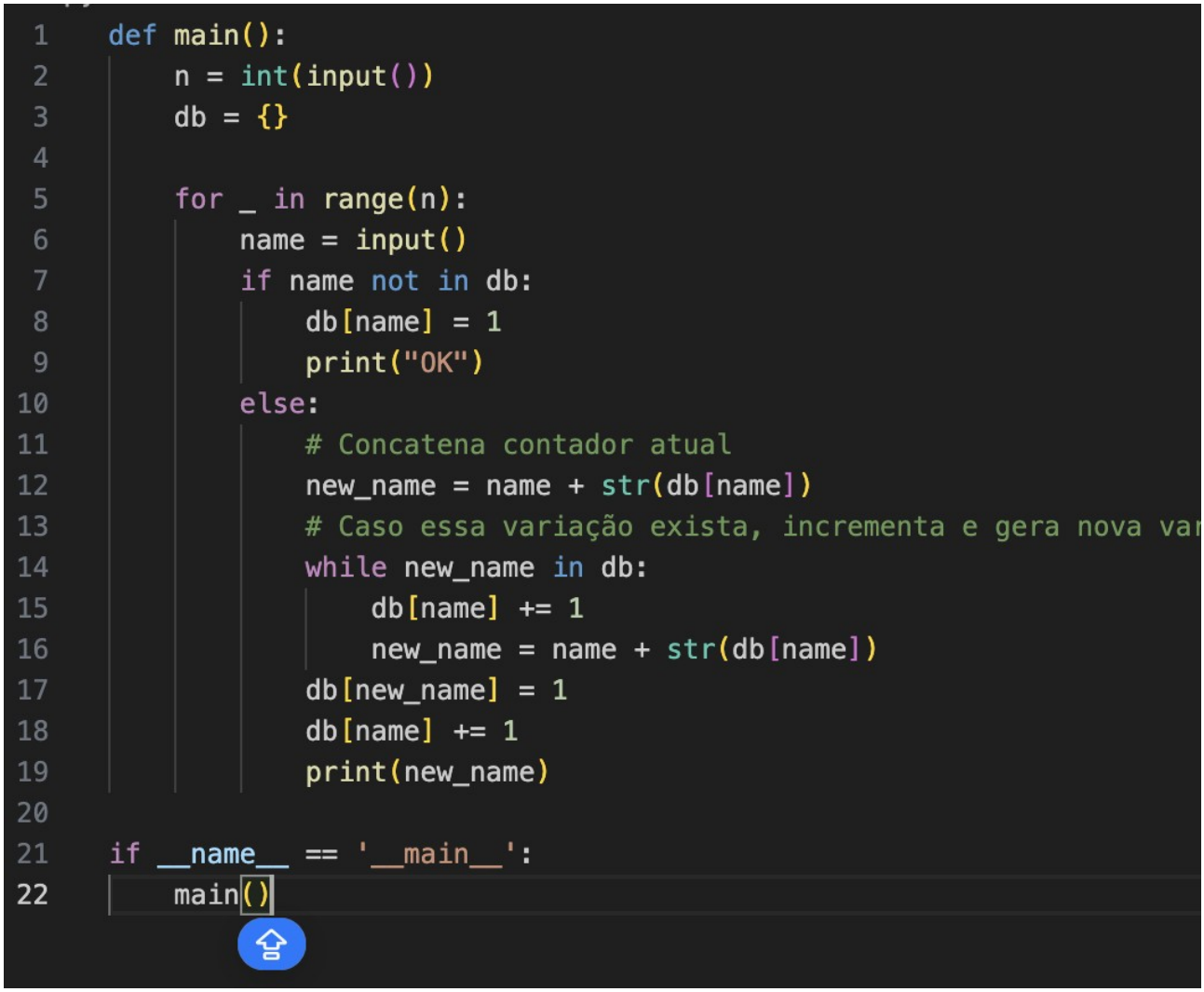
```cpp
#include <iostream>
#include <unordered_map>
#include <string>
using namespace std;

int main() {
    int n;
    cin >> n;
    unordered_map<string,int> nameCount;
    string name;

    while(n--) {
        cin >> name;

        // Se nome é novo, imprime "OK"
        if (nameCount[name] == 0) {
            cout << "OK\n";
            nameCount[name] = 1;
        } else {
            // Se já existe, concatena contador
            cout << name << nameCount[name] << "\n";
            nameCount[name]++;
        }
    }
    return 0;
}
```

Código em PYTHON:



```python
def main():
    n = int(input())
    db = {}

    for _ in range(n):
        name = input()
        if name not in db:
            db[name] = 1
            print("OK")
        else:
            # Concatena contador atual
            new_name = name + str(db[name])
            # Caso essa variação exista, incrementa e gera nova var
            while new_name in db:
                db[name] += 1
                new_name = name + str(db[name])
            db[new_name] = 1
            db[name] += 1
            print(new_name)

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

G) KEYBOARD (https://codeforces.com/problemset/problem/474/A **)**

Explicação:

O teclado está deslocado uma tecla para a esquerda ou para a direita.

Para corrigir, encontramos o índice de cada caractere digitado na string do layout e ajustamos esse índice conforme a direção.

Código em C++:

```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    char direction;
    string typed;
    cin >> direction >> typed;

    string keyboard = "qwertyuiopasdfghjkl;zxcvbnm,./";

    // Ajuste de índice: se deslocado para a direita, pegar pos - 1; se para a esquerda, pos + 1
    for (char &c : typed) {
        size_t pos = keyboard.find(c);
        if (direction == 'L') {
            cout << keyboard[pos + 1];
        } else {
            cout << keyboard[pos - 1];
        }
    }
    cout << endl;
    return 0;
}
```

Código em PYTHON:

```python
direction = input().strip()
typed = input().strip()
keyboard = "qwertyuiopasdfghjkl;zxcvbnm,./"

# Se deslocado para 'R', índice diminui; se 'L', índice aumenta
shift = -1 if direction == 'R' else 1
original_message = ""

for char in typed:
    index = keyboard.index(char)
    original_message += keyboard[index + shift]

print(original_message)
```

H) CHAT ORDER (https://vjudge.net/contest/705045#problem/H **)**

Explicação:

Mantém uma lista de conversas na ordem. Quando Polycarp envia mensagem a alguém,

remove-se esse contato (se já existir) e insere-se no topo,

preservando a sequência dos demais.

Código em C++:

```cpp
#include <iostream>
#include <unordered_map>
#include <list>
#include <string>
using namespace std;

int main() {
    int n;
    cin >> n;

    list<string> chatList;
    unordered_map<string, list<string>::iterator> position;
    string name;

    for (int i = 0; i < n; ++i) {
        cin >> name;
        // Se já existir na lista, remove da posição antiga
        if (position.count(name)) {
            chatList.erase(position[name]);
        }
        // Insere no topo
        chatList.push_front(name);
        position[name] = chatList.begin();
    }

    // Imprime do topo para baixo
    for (const string& s : chatList) {
        cout << s << '\n';
    }

    return 0;
}
```

Código em PYTHON:

```python
from collections import OrderedDict
import sys

input = sys.stdin.readline
n = int(input())
chats = OrderedDict()

for _ in range(n):
    name = input().strip()
    # Se já existir, apaga para reinserir no final (último = topo)
    if name in chats:
        del chats[name]
    chats[name] = True

# O último inserido fica no final; queremos ordem do mais recente para o menos recente
for name in reversed(chats):
    print(name)

```

I) CHAT ORDER (https://codeforces.com/problemset/problem/547/B**)**

Explicação:

Para cada tamanho x (1 até n), encontra-se o maior valor entre os mínimos de todas as janelas de tamanho x.

Usamos pilhas para descobrir onde cada elemento atua como mínimo (buscando o próximo menor à esquerda e à direita) e assim saber o maior “raio de influência” de cada valor.

Código em C++:

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

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

    int n;
    cin >> n;
    vector<long long> v(n);
    for (int i = 0; i < n; ++i) {
        cin >> v[i];
    }

    // left[i] = índice do elemento menor à esquerda de i
    // right[i] = índice do elemento menor à direita de i
    vector<int> left(n, -1), right(n, n);
    stack<int> st;

    // Calcula menor à esquerda
    for (int i = 0; i < n; i++) {
        while (!st.empty() && v[st.top()] >= v[i]) {
            st.pop();
        }
        if (!st.empty()) {
            left[i] = st.top();
        }
        st.push(i);
    }

    while (!st.empty()) {
        st.pop();
    }

    // Calcula menor à direita
    for (int i = n - 1; i >= 0; i--) {
        while (!st.empty() && v[st.top()] >= v[i]) {
            st.pop();
        }
        if (!st.empty()) {
            right[i] = st.top();
        }
        st.push(i);
    }

    vector<long long> ans(n + 1, 0);

    // length = quantos elementos pode formar esse valor sendo o mínimo
    for (int i = 0; i < n; i++) {
        int length = right[i] - left[i] - 1;
        ans[length] = max(ans[length], v[i]);
    }

    // Ajusta preenchendo do maior segmento ao menor
    for (int i = n - 1; i >= 1; i--) {
        ans[i] = max(ans[i], ans[i + 1]);
    }

    // Imprime as respostas para cada x
    for (int i = 1; i <= n; i++) {
        cout << ans[i] << " ";
    }
    cout << "\n";

    return 0;
}

```

Código em PYTHON:

```python
def max_strength_for_each_group(n, a):
    left = [-1] * n
    right = [n] * n
    stack = []

    # Próximo menor à esquerda
    for i in range(n):
        while stack and a[stack[-1]] >= a[i]:
            stack.pop()
        if stack:
            left[i] = stack[-1]
        stack.append(i)

    stack.clear()

    # Próximo menor à direita
    for i in range(n - 1, -1, -1):
        while stack and a[stack[-1]] >= a[i]:
            stack.pop()
        if stack:
            right[i] = stack[-1]
        stack.append(i)

    res = [0] * (n + 1)

    # right[i] - left[i] - 1 = quantos elementos o a[i] pode cobrir como mínimo
    for i in range(n):
        length = right[i] - left[i] - 1
        res[length] = max(res[length], a[i])

    # Preenche do fim para o início
    for i in range(n - 1, 0, -1):
        res[i] = max(res[i], res[i + 1])

    return res[1:]

n = int(input())
a = list(map(int, input().split()))
result = max_strength_for_each_group(n, a)
print(' '.join(map(str, result)))
```