Bài 1: mã hackerRank:@tiendatbl1910\_điểm:10

Ý tưởng chính:

 **Duyệt chuỗi:** Sử dụng một vòng lặp để đi qua toàn bộ văn bản và trích xuất từng chuỗi con 3 ký tự. Ví dụ, với chuỗi "AAAAAAD", bạn sẽ trích xuất các chuỗi con là "AAA", "AAA", "AAA", "AAD".

 **Đếm tần suất:** Dùng Hash Map để lưu trữ các chuỗi con này. Với mỗi chuỗi con được trích ra sẽ tăng giá trị tương ứng của nó trong Hash Map lên 1.

* 'AAA' ban đầu có giá trị 0, sau đó tăng lên 1, rồi 2, rồi 3.
* 'AAD' ban đầu có giá trị 0, sau đó tăng lên 1.

 **Tìm kết quả:** Sau khi đã duyệt hết chuỗi và có bảng tần suất hoàn chỉnh, chỉ cần duyệt qua các giá trị trong Hash Map để tìm ra số lớn nhất.

#include <iostream>

#include <string>

#include <map>

#include <algorithm>

void solve() {

std::string text;

std::cin >> text;

if (text.length() < 3) {

std::cout << 0 << std::endl;

return;

}

std::map<std::string, int> boba\_counts;

// Duyệt qua chuỗi và đếm tần suất

for (size\_t i = 0; i <= text.length() - 3; ++i) {

std::string boba = text.substr(i, 3);

boba\_counts[boba]++;

}

int max\_freq = 0;

// Tìm tần suất lớn nhất

for (auto const& pair : boba\_counts) {

if (pair.second > max\_freq) {

max\_freq = pair.second;

}

}

std::cout << max\_freq << std::endl;

}

int main() {

std::ios\_base::sync\_with\_stdio(false);

std::cin.tie(NULL);

int T;

std::cin >> T;

while (T--) {

solve();

}

return 0;

}

Bài 2:mã hackerRank:@tiendatbl1910\_điểm:10

# Ý tưởng chính

- Một dãy số là \*\*hoán vị của 1..N\*\* nếu:  
 1. \*\*Tất cả phần tử đều nằm trong đoạn [1..N]\*\*.  
 2. \*\*Không có phần tử nào bị lặp lại\*\* (mỗi số từ 1 đến N xuất hiện đúng 1 lần).  
- Cách kiểm tra:  
 - Tạo mảng đánh dấu `check[1..N]`, ban đầu gán `false`.  
 - Duyệt từng phần tử `Ai`:  
 - Nếu `Ai < 1` hoặc `Ai > N` → chắc chắn không phải hoán vị.  
 - Nếu `check[Ai] == true` → số này đã xuất hiện → không phải hoán vị.  
 - Ngược lại, đánh dấu `check[Ai] = true`.  
 - Sau khi duyệt xong, nếu tất cả số từ 1..N đều xuất hiện đúng 1 lần → in `YES`, ngược lại `NO`.

#include <iostream>

#include <vector>

using namespace std;

int main() {

int N;

cin >> N;

vector<int> a(N);

vector<bool> check(N + 1, false); // check[1..N]

for (int i = 0; i < N; i++) {

cin >> a[i];

}

for (int i = 0; i < N; i++) {

int x = a[i];

if (x < 1 || x > N || check[x]) {

cout << "NO";

return 0;

}

check[x] = true;

}

cout << "YES";

return 0;

}

Bài 3:mã hackerRank:@tiendatbl1910\_điểm:10

# Ý tưởng chính

- Bài toán yêu cầu phân tích N! (giai thừa của N) ra thừa số nguyên tố.  
- Ta cần đếm số mũ của từng số nguyên tố trong phân tích này.  
- Công thức Legendre: số mũ của một số nguyên tố p trong N! được tính bằng:  
  
 e\_p = floor(N/p) + floor(N/p^2) + floor(N/p^3) + ... (cho đến khi p^k > N).  
  
- Các bước thực hiện:  
 1. Sàng Eratosthenes để tìm tất cả số nguyên tố ≤ N.  
 2. Với mỗi số nguyên tố p, tính số mũ e\_p bằng công thức trên.  
 3. Xuất kết quả theo thứ tự các số nguyên tố tăng dần, cách nhau bằng khoảng trắng.

#include <iostream>

#include <vector>

using namespace std;

vector<int> sieve(int n) {

vector<bool> isPrime(n + 1, true);

vector<int> primes;

isPrime[0] = isPrime[1] = false;

for (int i = 2; i <= n; i++) {

if (isPrime[i]) {

primes.push\_back(i);

for (int j = i \* 2; j <= n; j += i)

isPrime[j] = false;

}

}

return primes;

}

int exponentInFactorial(int n, int p) {

int exp = 0;

while (n > 0) {

n /= p;

exp += n;

}

return exp;

}

int main() {

int N;

while (cin >> N) {

vector<int> primes = sieve(N);

for (int i = 0; i < (int)primes.size(); i++) {

cout << exponentInFactorial(N, primes[i]);

if (i < (int)primes.size() - 1) cout << " ";

}

cout << endl;

}

return 0;

}

Bài 4:mã hackerRank:@tiendatbl1910\_điểm:10

# Ý tưởng chính

- Số chữ số 0 tận cùng của N! bằng số lượng thừa số 10 trong N!.  
- Vì 10 = 2 × 5 và trong N! có nhiều thừa số 2 hơn thừa số 5, nên số lượng thừa số 10 bằng số thừa số 5 trong phân tích của N!.  
- Số mũ của 5 trong N! là:  
  
 floor(N/5) + floor(N/5^2) + floor(N/5^3) + ... (cộng cho đến khi 5^k > N).  
  
- Thuật toán: cộng các floor(N/5^k) với k tăng dần (k = 1, 2, 3, ...) — độ phức tạp O(log5 N), rất nhanh ngay cả với N lớn.

#include <iostream>

using namespace std;

int main() {

long long N;

while (cin >> N) {

long long count = 0;

long long p = 5;

while (p <= N) {

count += N / p;

if (p > N / 5) break;

p \*= 5;

}

cout << count << '\n';

}

return 0;

}

Bài 5:mã hackerRank:@tiendatbl1910\_điểm:9

Ý tưởng chính:

· **Chuyển đổi các dãy số thành tập hợp**: Thay vì xử lý các mảng thông thường, bạn nên chuyển từng dãy số x, y, z thành một **tập hợp (set)**. Cấu trúc dữ liệu này giúp tự động loại bỏ các phần tử trùng lặp và hỗ trợ việc tìm kiếm một phần tử có tồn tại hay không với tốc độ rất nhanh (thường là O(1) đối với unordered\_set hoặc O(logn) đối với set).

· **Tìm giao của ba tập hợp**: Sau khi có ba tập hợp, bạn chỉ cần lặp qua các phần tử của một trong ba tập hợp (ví dụ: tập hợp của dãy x). Với mỗi phần tử, kiểm tra xem nó có tồn tại trong cả hai tập hợp còn lại (y và z) hay không. Nếu có, đó chính là một phần tử chung.

· **Lưu trữ và sắp xếp kết quả**: Thu thập tất cả các phần tử chung tìm được vào một danh sách (ví dụ: std::vector). Cuối cùng, **sắp xếp** danh sách này theo thứ tự tăng dần trước khi in ra kết quả. Việc này đảm bảo đáp ứng yêu cầu của bài toán là liệt kê các phần tử chung một cách có thứ tự và không trùng lặp.

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

ios\_base::sync\_with\_stdio(false);

cin.tie(NULL);

int nx, ny, nz;

cin >> nx;

vector<int> x(nx);

for (int i = 0; i < nx; ++i) {

cin >> x[i];

}

cin >> ny;

vector<int> y(ny);

for (int i = 0; i < ny; ++i) {

cin >> y[i];

}

cin >> nz;

vector<int> z(nz);

for (int i = 0; i < nz; ++i) {

cin >> z[i];

}

sort(x.begin(), x.end());

sort(y.begin(), y.end());

sort(z.begin(), z.end());

int i = 0, j = 0, k = 0;

vector<int> common\_elements;

int last\_common = -1;

while (i < nx && j < ny && k < nz) {

if (x[i] == y[j] && y[j] == z[k]) {

if (x[i] != last\_common) {

common\_elements.push\_back(x[i]);

last\_common = x[i];

}

i++;

j++;

k++;

}

else if (x[i] <= y[j] && x[i] <= z[k]) {

i++;

}

else if (y[j] <= x[i] && y[j] <= z[k]) {

j++;

}

else {

k++;

}

}

cout << common\_elements.size() << endl;

for (size\_t l = 0; l < common\_elements.size(); ++l) {

cout << common\_elements[l] << (l == common\_elements.size() - 1 ? "" : " ");

}

cout << endl;

return 0;

}