1. BÀI 1. BỘ BA (BOBA.\*)

Ý tưởng:

Sử dụng một map để lưu trữ tần suất xuất hiện của các "bộ ba".

Sử dụng một biến max để lưu trữ tần suất lớn nhất, khởi tạo bằng 0.

Sử dụng một vòng lặp for để duyệt chuỗi S từ đầu đến cuối. Vòng lặp sẽ chạy từ chỉ số i = 0 đến S.length() - 3 để đảm bảo luôn lấy được một chuỗi con có đủ 3 ký tự. Trong mỗi lần lặp, trích xuất chuỗi con gồm 3 ký tự từ vị trí i.

Nếu bộ ba lặp lại thì tang biến đếm lên.

Cập nhật max nếu tần suất của bộ ba lớn hơn giá trị hiện tại.

Sử dụng một vòng lặp for để duyệt qua tất cả các cặp (key, value) trong map và tìm giá trị value lớn nhất.

Code:

#include <iostream>

#include <string>

#include <map>

using namespace std;

int main() {

int T;

cin >> T;

while (T--) {

string s;

cin >> s;

map<string, int> tanso;

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

string bo\_ba = s.substr(i, 3);

tanso[bo\_ba]++;

}

int max\_tanso = 0;

if (!tanso.empty()) {

for (map<string, int>::iterator it = tanso.begin(); it != tanso.end(); it++) {

if (it->second > max\_tanso) {

max\_tanso = it->second;

}

}

}

cout << max\_tanso << endl;

}

return 0;

}

Điểm :10

1. BÀI 2. DÃY SỐ HOÁN VỊ (DAYHV.\*)

Ý tưởng:

Duyệt qua dãy N số nguyên để kiểm tra xem có nằm trong khoảng từ 1->N không . Nếu có số nằm ngoài thì in ra “NO”. Nếu có số xuất hiện lại thì in ra “NO”. Nếu số hợp lệ và chưa xuất hiện, đánh dấu nó là đã xuất hiện bằng cách gán check[A\_i] = true.

Nếu vòng lặp hoàn thành mà không có lỗi nào được tìm thấy (tức là không in ra "NO" và kết thúc), điều đó có nghĩa là tất cả các số từ 1 đến N đều đã xuất hiện đúng một lần. Khi đó, in ra "YES".

Code:

#include <iostream>

#include <vector>

using namespace std;

int main() {

int N;

cin >> N;

vector<bool> check(N + 1, false);

bool DHV = true;

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

int a;

cin >> a;

if (a < 1 || a > N) {

DHV = false;

}

else if (check[a]) {

DHV = false;

}

else {

check[a] = true;

}

}

if (DHV) {

cout << "YES" << endl;

}

else {

cout << "NO" << endl;

}

return 0;

}

Điểm: 10

1. BIỂU DIỄN N! (ANALYSE.\*)

Ý tưởng:

Tìm tất cả các số nguyên tố nhỏ hơn hoặc bằng N.

Đối với mỗi số nguyên tố p tìm được ở bước 1, áp dụng công thức Legendre để tính số mũ của nó trong phân tích của N.

Lưu các số mũ đã tính được vào một mảng hoặc vector.

In ra các số mũ theo thứ tự đã lưu, cách nhau bằng dấu cách.

Code:

#include <iostream>

#include <vector>

using namespace std;

vector<int> sieve(int n) {

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

LSNT[0] = LSNT[1] = false;

for (int p = 2; p \* p <= n; ++p) {

if (LSNT[p]) {

for (int i = p \* p; i <= n; i += p) {

LSNT[i] = false;

}

}

}

vector<int> SNT;

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

if (LSNT[p]) {

SNT.push\_back(p);

}

}

return SNT;

}

long long legendre(int n, int p) {

long long M = 0;

while (n > 0) {

n /= p;

M += n;

}

return M;

}

int main() {

int N;

while (cin >> N) {

if (N < 2) {

cout << "" << endl;

continue;

}

vector<int> SNT = sieve(N);

for (size\_t i = 0; i < SNT.size(); ++i) {

long long M = legendre(N, SNT[i]);

cout << M;

if (i < SNT.size() - 1) {

cout << " ";

}

}

cout << endl;

}

return 0;

}

Điểm: 10

1. SỐ CHỮ SỐ KHÔNG TẬN CÙNG N! (SOKHONG.\*)

Ý tưởng:

Tính trước số chữ số 0 tận cùng của N cho một loạt các giá trị N (ví dụ từ 1 đến 100).

Lưu các kết quả này vào một map, với khóa (key) là N và giá trị (value) là số chữ số 0 tận cùng.

Khi người dùng nhập một số N, chỉ cần tra cứu khóa N trong bảng băm.

Nếu tìm thấy, in ra giá trị tương ứng.

Code:

#include <iostream>

#include <map>

using namespace std;

long long DSK(long long n) {

if (n < 0) return 0;

long long count = 0;

while (n >= 5) {

n /= 5;

count += n;

}

return count;

}

int main() {

map<int, long long> BL;

for (int i = 1; i <= 1000; ++i) {

BL[i] = DSK(i);

}

int N;

while (cin >> N) {

if (N <= 1000) {

cout << BL[N] << endl;

}

else {

cout << DSK(N) << endl;

}

}

return 0;

}

Điểm: 10

1. A

Ý tưởng:

Tạo ba hash set (set\_x, set\_y, set\_z) để lưu các phần tử duy nhất từ mỗi dãy số. Điều này giúp loại bỏ các phần tử trùng lặp trong từng dãy ban đầu.

Sử dụng một hash map (counts) để lưu tần suất xuất hiện của các phần tử.

Duyệt qua set\_x và tăng giá trị đếm cho mỗi phần tử trong counts.

Làm tương tự với set\_y và set\_z.

Duyệt qua counts và thu thập tất cả các phần tử có giá trị đếm bằng 3 (tức là chúng xuất hiện trong cả 3 tập hợp).

Sắp xếp các phần tử này theo thứ tự tăng dần và in ra.

Code:

#include <iostream>

#include <vector>

#include <unordered\_set>

#include <unordered\_map>

#include <algorithm>

using namespace std;

void NhapMang(int n, unordered\_set<int>& s) {

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

int val;

cin >> val;

s.insert(val);

}

}

int main() {

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

cin.tie(NULL);

int nx, ny, nz;

cin >> nx;

unordered\_set<int> set\_x;

NhapMang(nx, set\_x);

cin >> ny;

unordered\_set<int> set\_y;

NhapMang(ny, set\_y);

cin >> nz;

unordered\_set<int> set\_z;

NhapMang(nz, set\_z);

unordered\_map<int, int> counts;

for (int num : set\_x) {

counts[num]++;

}

for (int num : set\_y) {

counts[num]++;

}

for (int num : set\_z) {

counts[num]++;

}

vector<int> PTC;

for (const auto& pair : counts) {

if (pair.second == 3) {

PTC.push\_back(pair.first);

}

}

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

cout << PTC.size() << endl;

for (size\_t i = 0; i < PTC.size(); ++i) {

cout << PTC[i];

if (i < PTC.size() - 1) {

cout << " ";

}

}

cout << endl;

return 0;

}

Điểm: 10