Kelas : LA95

Kelompok : Caelum Codex Oracle

Anggota Kelompok :

1. 2702242253 – Leony Suhendryck Mao
2. 2702292932 – Kevin Jeremia
3. 2702241244 – Oei, Christopher Bryant Widyanata

Departemen : *Computer Science*

Kampus : Semarang

***AoL (Assurance of Learning) Case Study***

**Mata Kuliah *COMP6049001 – Algorithm Design and Analysis***

1. *Write an analysis of the solutions used in INC 2024!*
2. **Problem A – *Problem C***

**SOAL:**

Problem C

The English language is infamous for having an inconsistent spelling system. One feature in the spelling system that you are interested in is the usage of the letter c. The letter c can be pronounced in multiple different ways, which might be confusing for the speaker. However, there are some simplified patterns to help speakers determine which pronunciation to use.

* + It is pronounced as ⟨k⟩ if it is either: followed by a, o, or u; followed by a consonant letter other than h and y; or at the end of a word. The consonant letters are the English letters excluding the vowel letters: a, e, i, o, and u.
  + It is pronounced as ⟨s⟩ if it is followed by e, i, or y.
  + The pair of adjacent letters ch is pronounced with a unique sound ⟨c⟩.

You are given N words: S1, S2, . . . , SN . For each given word, simplify the spelling system by replacing every occurence of c into k or s (for the sound ⟨k⟩ and ⟨s⟩ respectively) according to the patterns above and every occurence of ch into c. Input The first line consists of an integer N (1 ≤ N ≤ 100). Each of the next N lines consists of a string Si . String Si consists of only lowercase English letters, and its length is between 1 and 100. Output For each word, output a single line containing a string that represents the simplified spelling system of string Si according to the patterns above.

**Input**

The first line consists of an integer N (1 ≤ N ≤ 100). Each of the next N lines consists of a string Si. String Si consists of only lowercase English letters, and its length is between 1 and 100.

**Output**

For each word, output a single line containing a string that represents the simplified spelling system of string Si according to the patterns above.

**Sample Input #1**

10

cyan

circle

ceramic

coccyx

delicacy

chacha

coach

blackjack

church

hiccup

**Sample Output #1**

syan

sirkle

seramik

koksyx

delikasy

caca

koac

blakkjakk

curc

hikkup

**JAWABAN:**

#include <iostream>

#include <string>

using namespace std;

string simplifyWord(string word) {

string result;

int len = word.size();

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

if (word[i] == 'c') {

if (i + 1 < len) {

if (word[i + 1] == 'h') {

result += 'c';

i++;

} else if (word[i + 1] == 'e' || word[i + 1] == 'i' || word[i + 1] == 'y') {

result += 's';

} else {

result += 'k';

}

} else {

result += 'k';

}

} else {

result += word[i];

}

}

return result;

}

int main() {

int N;

cin >> N;

string words[100]; // Maksimal 100 kata sesuai batasan

string results[100]; // Menyimpan hasil transformasi

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

cin >> words[i];

}

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

results[i] = simplifyWord(words[i]);

}

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

cout << results[i] << endl;

}

return 0;

}

**ANALISIS:**

#include <iostream>

#include <string>

**Penjelasan:**

#include <iostream> Menyediakan fungsi untuk membaca input (cin) dan menampilkan output (cout).

#include <string> Memungkinkan penggunaan tipe data string yang diperlukan untuk manipulasi string pada problem ini.

**Alasan**: Dibutuhkan untuk membaca kata-kata (Si) dan memproses setiap huruf dalam string.

using namespace std;

**Penjelasan**: untuk bagian ini hanya untuk menghilangkan penggunaan std:: dalam pemanggilan cin atau cout

string simplifyWord(string word) {

string result;

int len = word.size();

**Penjelasan:**

* string simplifyWord(string word):

Fungsi ini menerima sebuah kata (word) sebagai input dan mengembalikan kata tersebut setelah dilakukan penyederhanaan sesuai aturan.

* string result;

Variabel untuk menyimpan hasil kata yang sudah disederhanakan.

* int len = word.size();

Variabel len menyimpan panjang kata (word). Ini digunakan untuk memeriksa panjang string di dalam loop.

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

**Penjelasan:**

Loop ini digunakan untuk membaca setiap karakter pada kata (word) dari awal hingga akhir.

if (word[i] == 'c') {

**Penjelasan:**

Mengecek apakah karakter saat ini adalah huruf c. Jika iya, maka kita akan menentukan apakah diubah menjadi k, s, atau tetap c sesuai aturan.

if (i + 1 < len) {

**Penjelasan:**

Mengecek apakah ada karakter setelah c.

if (word[i + 1] == 'h') {

result += 'c';

i++;

**Penjelasan:**

Jika huruf setelah c adalah h, maka pasangan ch diubah menjadi c.

i++; -> Mengabaikan h karena sudah diproses bersama c.

} else if (word[i + 1] == 'e' || word[i + 1] == 'i' || word[i + 1] == 'y') {

result += 's';

**Penjelasan:**

Jika huruf setelah c adalah e, i, atau y, maka c diubah menjadi s.

} else {

result += 'k';

}

**Penjelasan:**

Jika tidak ada kondisi di atas, maka c diubah menjadi k.

} else {

result += 'k';

}

**Penjelasan:**

Jika c adalah huruf terakhir dalam kata, maka diubah menjadi k.

} else {

result += word[i];

}

**Penjelasan:**

Jika karakter saat ini bukan c, maka karakter tersebut ditambahkan ke hasil tanpa diubah.

}

return result;

}

**Penjelasan:**

Mengembalikan kata hasil transformasi.

int main() {

int N;

cin >> N;

**Penjelasan:**

**int N**: Variabel untuk menyimpan jumlah kata yang akan diproses.

**cin >> N**: Membaca jumlah kata dari input.

string words[100]; // Maksimal 100 kata sesuai batasan

string results[100]; // Menyimpan hasil transformasi

**Penjelasan:**

* **string words[100]**: Array untuk menyimpan hingga 100 kata.
* **string results[100]**: Array untuk menyimpan hasil transformasi setiap kata.

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

cin >> words[i];

}

**Penjelasan:**

Loop untuk membaca setiap kata dari input dan menyimpannya dalam array words.

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

results[i] = simplifyWord(words[i]);

}

**Penjelasan:**

Loop untuk memproses setiap kata di array words menggunakan fungsi simplifyWord dan menyimpan hasilnya ke array results.

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

cout << results[i] << endl;

}

**Penjelasan:**Loop untuk mencetak hasil transformasi setiap kata dari array results.

return 0;

}

**Penjelasan:**Menandakan bahwa program selesai dijalankan.

**Jawaban code di atas cocok dan bisa untuk menyelesaikan problem karena:**

Kode telah mengikuti semua aturan yang diberikan untuk pengubahan huruf c sesuai soal.

* + - 1. Pasangan huruf ch diubah menjadi c.

Implementasi dalam baris code:

if (word[i + 1] == 'h') {

result += 'c';

i++;

}

* + - 1. Huruf c diubah menjadi s jika diikuti e, i, atau y.

Implementasi dalam code:

else if (word[i + 1] == 'e' || word[i + 1] == 'i' || word[i + 1] == 'y') {

result += 's';

}

* + - 1. Huruf c diubah menjadi k jika:

Diikuti huruf a, o, u, atau huruf konsonan lain kecuali h dan y dan berada di akhir kata.

Implementasi dalam code:

else {

result += 'k';

} // Ketika tidak ada kondisi lain yang terpenuhi (bukan pasangan ch atau diikuti e, i, y), maka c diubah menjadi k.

* + - 1. c di Akhir Kata

Implementasi dalam code:

else {

result += 'k';

} // Ketika c adalah huruf terakhir dalam kata (tidak ada i + 1), otomatis masuk ke else dan diubah menjadi k.

1. ***Problem C – Double Chunks***

**SOAL:**

Double Chunks

You have a chocolate bar consisting of N chunks (numbered from 1 to N). Chunk i contains Ai peanut bits. You can divide the chocolate bar into several pieces, with each piece consisting of one or more consecutive chunks. Each chunk can only be part of one piece. The total number of peanut bits in a piece is simply the sum of the peanut bits from each of its chunks. A piece is considered a double chunk if and only if it consists of exactly two chunks. You are required to divide the chocolate bar into as many double chunks as possible, all having the same total number of peanut bits. Determine the maximum number of double chunks you can get while satisfying this requirement.

**Input**

The first line consists of an integer N (2 ≤ N ≤ 100 000).

The second line consists of N integers Ai (1 ≤ Ai ≤ 109).

**Output**

Output a single integer representing the maximum number of double chunks you can get while satisfying the requirement.

**Sample Input #1**

10

2 4 1 4 5 2 3 1 1 4

**Sample Output #1**

3

**Explanation for the sample input/output #1**

You can make 3 double chunks, each containing 5 peanut bits. The first piece consists of chunks 2 and 3, which have 4 bits and 1 bit, respectively. The second piece consists of chunks 6 and 7, which have 2 bits and 3 bits, respectively. The third piece consists of chunks 9 and 10, which have 1 bit and 4 bits, respectively.

**Sample Input #2**

7

1 2 1 1 1 2 1

**Sample Output #2**

2

**Sample Input #3**

5

1 2 3 4 5

**Sample Output #3**

1

**JAWABAN**

#include <iostream>

#include <vector>

#include <unordered\_map>

using namespace std;

int main() {

int N\_Caelum;

cin >> N\_Caelum;

vector<int> A\_Oralex(N\_Caelum);

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

cin >> A\_Oralex[i];

}

unordered\_map<int, int> sumCaelumCount;

int maxCodexDoubleChunks = 0;

int i = 0;

while (i < N\_Caelum - 1) {

int doubleChunkSum = A\_Oralex[i] + A\_Oralex[i + 1];

if (i < N\_Caelum - 2 && A\_Oralex[i + 1] + A\_Oralex[i + 2] == doubleChunkSum) {

sumCaelumCount[doubleChunkSum]++;

i += 2;

} else {

sumCaelumCount[doubleChunkSum]++;

i++;

}

}

for (auto &entry : sumCaelumCount) {

maxCodexDoubleChunks = max(maxCodexDoubleChunks, entry.second);

}

cout << maxCodexDoubleChunks << endl;

return 0;

}

**ANALISIS**

#include <iostream>

#include <vector>

#include <unordered\_map>

using namespace std;

* **Penjelasan**: untuk bagian ini hanya untuk menghilangkan penggunaan std:: dalam pemanggilan cin atau cout

int main() {

int N\_Caelum;

cin >> N\_Caelum;

* **Penjelasan**: untuk bagian ini hanya untuk mendeklarasikan integer N\_Caelum dan memasukan inputan user ke dalam variabel N\_Caelum tersebut.

vector<int> A\_Oralex(N\_Caelum);

* **Penjelasan**: ini dibuat untuk membuat suatu array yang dinamis, dimana jika di dalam bahasa Java ini seperti Array List, nah array dinamis ini memiliki nama A\_Oralex untuk dan memiliki tipe menyimpan integer didalamnya. N\_Caelum dilam tanda kurung “( )” adalah berfungsi untuk menentukan besaran/ukuran dari array yang diinginkan berdasarkan inputan user sebelumnya.

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

cin >> A\_Oralex[i];

}

* **Penjelasan**: for disini berfungsi seperti layaknya for biasa dimana memiliki cara kerja menerima input dari user dan memasukkannya ke dalam setiap index di array A\_Oralex.

unordered\_map<int, int> sumCaelumCount;

* **Penjelasan**: untuk fungsi ini berguna untuk menyediakan map, fungsi ini juga sangat mirip seperti hash map yang kita ketahui dimana didalamnya terdapat key (dibagian kiri) dan value nya (dibagian kanan). Untuk kasus ini, key dan value nya kita set dalam tipe integer karena menyesuaikan dengan kasus yang kami dapat. Kemudian, sumCaelumCount hanyalah nama dari hash map ini saja, yang kemudian jika di panggil hanya sebut nama map nya saja. Selain itu, fungsi dari map “sumCaelumCount” ini adalah untuk menyimpan berapa kali nilai penjumlahan antar index dari A\_Oralex itu muncul.

int maxCodexDoubleChunks = 0;

* **Penjelasan**: ini hanya untuk deklarasi nilai yang muncul paling banyak itu sampai berapa kali munculsaja seberapa banyak pasangan angka yang sama itu muncul.

int i = 0;

while (i < N\_Caelum - 1) {

int doubleChunkSum = A\_Oralex[i] + A\_Oralex[i + 1];

* **Penjelasan**: bagian ini untuk menjumlahkan dari nilai dari A\_Oralex index sekarang dengan nilai dari A\_Oralex index selanjutnya, maksudnya adalah misalkan saya ada di nilai dari A\_Oralex index 0 maka jika di tambahkan dengan nilai dari A\_Oralex di index 0+1 = 1 maka hasilnya berapa. Lanjut, jika sekarang saya ada di index 1 dari A\_Oralex maka jika nilai dari index ini ditambahkan dengan nilai dari index berikutnya hasilnya berapa. Nantinya hasil penjumlahan tersebut akan dimasukkan ke dalam doubleChunkSum.

if (i < N\_Caelum - 2 && A\_Oralex[i + 1] + A\_Oralex[i + 2] == doubleChunkSum){

sumCaelumCount[doubleChunkSum]++;

i += 2;

* **Penjelasan**: bagian ini melakukan pengecekan apakah index i kurang dari panjang array dikurangi 2 dan apakah ada hasil yang sama secara berturut-turut, maksudnya adalah misal hasil penjumlahan nilai pada index 0 dan index 1 = 2, maka jika hasil penjumlahan nilai index 1 dan index 2 = 2 juga maka akan masuk ke kondisi IF nya. Pada awalnya kan doubleChunkSum itu di declare = 0, nah di dalam IF ini (sudah memenuhi syarat/kondisi IF nya) doubleChunkSum ini akan menjadi suatu index bagi si map sumCaelumCount, dimana awalnya kan tidak ada nilai di map sumCaelumCount index tersebut (atau bernilai 0), maka akan di ++ atau ditambah 1, artinya nilai penjumlahan itu muncul 1 kali. Begitu juga jika masuk lagi ke sumCaelumCount di index yang sama (misal sudah ada 1), maka akan ditambah lagi 1 yang hasilnya 2, artinya nilai penjumlahannya ada 2 kali muncul. Kemudian index i akan ditambah 2 artinya akan melewati hasil penjumlahan yang sama berturut-turut, maksudnya [0] + [1] = 2, [1] +[2] = 2 itu akan dilewati atau bisa dibilang dihitung 1.

} else {

sumCaelumCount[doubleChunkSum]++;

i++;

}

}

* **Penjelasan**: kondisi ini mirip seperti kondisi dalam IF diatas, namun pada index i akan ditambah 1, karena tidak ada hasil penjumlahan nilai yang sama berturut-turut.

for (auto &entry : sumCaelumCount) {

maxCodexDoubleChunks = max(maxCodexDoubleChunks, entry.second);

}

* **Penjelasan**: FOR disini mirip dengan for each yang ada di Java, dalam parameter FOR ini, terdapat auto yang memiliki fungsi sama halnya dengan deklarasi tipe data seperti “int”, “str”, “double”, dll. Kemudian untuk penggunaan simbol “&” disini sangat penting karena kita ingin menuju ke nilai/value asli yang ada di dalam map sumCaelumCount nya. Kemudian di dalam FOR nya itu ada maxDoubleChunks yang sebelumnya sudah di declare = 0. Disini kita ingin membandingkan mana yang lebih besar antara maxDoubleChunks dengan entry.second. Maksud dari entry.second adalah “entry” itu merujuk pada “key” pada map sumCaelumCount dan “second” itu merujuk pada “value” dari “key” yang dituju, jadi mirip sekali konsepnya dengan index dan value. Kemudian, setelah membandingkan kedua nilai tersebut menggunakan fungsi max dan menemukan yang lebih besar diantara keduana, maka maxDoubleChunks yang awalnya 0, nilainya akan diganti dengan nilai yang lebih besar hasil dari fungsi max nya, dilakukan begitu terus sampai selesai.

cout << maxCodexDoubleChunks << endl;

* **Penjelasan**: pada bagian ini sama saja seperti kita akan melakukan print, namun di C++ menggunakan cout untuk print nya. Kemudian, maksud dari endl itu adalah untuk mengeluarkan hasil printnya di baris yang baru atau sama halnya dengan “\n......” di bahasa C, jadi maksud baris ini adalah untuk menampilkan atau print hasil akhir dari maxCodexDoubleChunks di baris yang baru.

return 0;

* **Penjelasan**: ini untuk menandakan bahwa program sudah selesai

}

1. ***Problem M - ICPC Provincial***

**SOAL:**

ICPC Provincial

The University of INC (UOI) is participating in an ICPC Provincial Contest, a qualifier contest for the ICPC Regional Contest. UOI has3N students (numberedfrom1to3N) who are eager to participate in the contest. There will be N teams, each consisting of exactly 3 students. Each student can only be assigned to only one team. As the coach of UOI, you know that student i has a skill rating of Ai. You define the strength of a team as the median of the skill ratings of its members. In order to increase the chance for all UOI teams to qualify for the ICPC Regional Contest, you want to arrange the teams so that the strength of the weakest team is maximized. Determine the maximum strength of the weakest team.

**Input**

The first line consists of an integer N (1 ≤ N ≤ 100000). The second line consists of 3N integers Ai (0 ≤ Ai ≤ 4000)

**Output**

Output a single integer representing the maximum strength of the weakest team

**Sample Input #1**

2

1500 1700 1800 2300 2500 2600

**Sample Output #1**

1800

**Explanation for the sample input/output #1**

Team1consists of students 1, 3, and 5, while team 2 consists of students 2, 4, and 6. The strength of team 1 and 2 are 1800 and 2300, respectively. Other arrangements exist, but none allow the weakest team to have a strength higher than 1800.

**Sample Input #2**

1

2800 2100 3000

**Sample Output #2**

2800

**Explanation for the sample input/output #2**

There is only one team with the strength of 2800.

**Sample Input #3**

3

4000 0 4000 0 4000 0 4000 0 4000

**Sample Output #3**

0

**JAWABAN**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int main() {

int N\_Caelum;

cin >> N\_Caelum;

vector<int> skillOralex(3 \* N\_Caelum);

for (int i = 0; i < 3 \* N\_Caelum; ++i) {

cin >> skillOralex[i];

}

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

int minCodexMedian = 101000;

for (int i = N\_Caelum; i < 3 \* N\_Caelum; i += 2) {

if (minCodexMedian > skillOralex[i]) {

minCodexMedian = skillOralex[i];

}

}

cout << minCodexMedian << endl;

return 0;

}

**ANALISIS**

#include <iostream>

**Penjelasan**: Menyediakan fungsi untuk membaca input (cin) dan menampilkan output (cout). Dibutuhkan untuk menerima input nilai kemampuan mahasiswa dan mencetak hasil akhir.

#include <vector>

**Penjelasan**: Memungkinkan penggunaan struktur data vektor (array dinamis). Digunakan untuk menyimpan kemampuan mahasiswa dalam bentuk array yang fleksibel.

#include <algorithm>

**Penjelasan:** Menyediakan fungsi-fungsi manipulasi data seperti sort(). Dibutuhkan untuk mengurutkan kemampuan mahasiswa agar dapat menentukan median secara optimal.

using namespace std;

**Penjelasan**: Untuk menghindari penggunaan prefiks std:: sebelum fungsi seperti cin, cout, dan struktur data.

cin >> N\_Caelum;

vector<int> skillOralex(3 \* N\_Caelum);

**Penjelasan:**

* cin >> N\_Caelum: Membaca jumlah tim yang harus dibentuk (N\_Caelum).
* vector<int> skillOralex(3 \* N\_Caelum): Membuat vektor untuk menyimpan kemampuan mahasiswa. Ukurannya adalah 3 \* N\_Caelum karena setiap tim membutuhkan 3 anggota.

for (int i = 0; i < 3 \* N\_Caelum; ++i) {

cin >> skillOralex[i];

}

**Penjelasan:**

Loop untuk membaca kemampuan semua mahasiswa. Dibutuhkan untuk menyimpan data kemampuan mahasiswa agar dapat diolah lebih lanjut.

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

**Penjelasan:**

Fungsi sort() mengurutkan elemen-elemen dalam vektor skillOralex secara menaik. Setelah diurutkan, kemampuan mahasiswa dapat dipilih secara optimal untuk menentukan median dari tim.

int minCodexMedian = 101000;

**Penjelasan:**

int minCodexMedian = 101000: Variabel untuk menyimpan median terkecil dari semua tim yang terbentuk. Nilai awal diatur tinggi untuk memastikan dapat diperbarui.

for (int i = N\_Caelum; i < 3 \* N\_Caelum; i += 2) {

if (minCodexMedian > skillOralex[i]) {

minCodexMedian = skillOralex[i];

}

}

**Penjelasan:**

* Loop ini melangkah dari indeks N\_Caelum hingga 3 \* N\_Caelum, memilih nilai kemampuan mahasiswa yang akan menjadi median setiap tim.
* i += 2 memastikan bahwa kita melompati elemen yang tidak berperan sebagai median.
* if (minCodexMedian > skillOralex[i]): Memperbarui nilai minCodexMedian jika nilai median tim saat ini lebih kecil.

cout << minCodexMedian << endl;

**Penjelasan:**

Mencetak nilai minCodexMedian, yaitu median terkecil dari semua tim, yang telah dimaksimalkan. Hasil akhir sesuai dengan tujuan soal untuk memaksimalkan kekuatan tim terlemah.

1. *Upsolve the unsolved problems in INC 2024 after the competition!*

**SOAL:**

Problem B

The Journey of the King

You are very close to becoming the King of Games. The only thing left to do is to win in a card game against the incarnation of the King of Nusantara, Anda, whose soul resides inside you as your split personality. Each player has a deck of cards, each card contains a word. Within each deck, there are no two cards containing the same word. There is also a dictionary consisting of D distinct words: [W1, W2, . . . , WD]. The game consists of N turns. In turn i, Anda will play a card with the word Ai . Then, you can either match his card with one of your remaining cards or skip this turn. Two cards, a and b, match if either the words a + b or b + a exist in the dictionary. The operator + represents the concatenation operation. For instance, the concatenation of words AU and RA is AU + RA = AURA. Once you match a card, you cannot use that card for the rest of the game. Your deck has M cards (numbered from 1 to M); card j contains word Bj. You want to maximize the number of turns in which you successfully match Anda’s card.

**Input**

The first line consists of an integer D (1 ≤ D ≤ 200 000).

Each of the next D lines consists of a string Wk. String Wk consists of only uppercase English letters. The sum of length of Wk does not exceed 200 000. It is guaranteed that Wk ̸= Wk′ for 1 ≤ k < k′ ≤ D.

The following line consists of an integer N (1 ≤ N ≤ 100 000).

Each of the next N lines consists of a string Ai. String Ai consists of only uppercase English letters. The sum of length of Ai does not exceed 100 000. It is guaranteed that Ai ̸= Ai ′ for 1 ≤ i < i′ ≤ N.

The following line consists of an integer M (1 ≤ M ≤ 100 000).

Each of the next M lines consists of a string Bj . String Bj consists of only uppercase English letters. The sum of length of Bj does not exceed 100 000. It is guaranteed that Bj Bj ′ for 1 ≤ j < j′ ≤ M.

**Output**

Output a single integer representing the maximum number of turns you match Anda’s card.

**Sample Input #1**

3

AURA

AURORA

LAURA

3

RA

REO

RORA

2

AU

LAU

**Sample Output #1**

2

**Explanation for the sample input/output #1**

During turn 1, you match RA with LAU to create LAURA.

During turn 2, you skip this turn.

During turn 3, you match RORA with AU to create AURORA.

**Sample Input #2**

3

HARTA

TAHTA

HARU

3

HAR

TAH

HA

3

TA

RU

ARU

**Sample Output #2**

2

**Explanation for the sample input/output #2**

During turn 1, you match HAR with TA to create HARTA.

During turn 2, you skip this turn.

During turn 3, you match HA with RU to create HARU.

**Sample Input #3**

1

AAA

3

A

AA

AAA

2

A

AA

**Sample Output #3**

2

**Sample Input #4**

1

INDONESIA

1

NATIONAL

1

CONTEST

**Sample Output #4**

0

**JAWABAN (CODINGAN SALAH PADA SAAT KOMPETISI):**

#include <iostream>

#include <unordered\_set>

#include <vector>

#include <string>

using namespace std;

int main() {

int D\_Caelum, N\_Oralex, M\_Codex;

cin >> D\_Caelum;

unordered\_set<string> dictionaryCodex;

for (int i = 0; i < D\_Caelum; ++i) {

string wordCaelum;

cin >> wordCaelum;

dictionaryCodex.insert(wordCaelum);

}

cin >> N\_Oralex;

vector<string> andaDeckOralex(N\_Oralex);

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

cin >> andaDeckOralex[i];

}

cin >> M\_Codex;

vector<string> yourDeckCaelum(M\_Codex);

unordered\_set<string> yourDeckSetCodex;

for (int i = 0; i < M\_Codex; ++i) {

cin >> yourDeckCaelum[i];

yourDeckSetCodex.insert(yourDeckCaelum[i]);

}

int totalMatchesOralex = 0;

unordered\_set<string> usedCardsCodex;

for (const string &ai\_Caelum : andaDeckOralex) {

bool matchedOralex = false;

for (const string &bjCodex : yourDeckCaelum) {

if (usedCardsCodex.count(bjCodex)) continue;

string concat1 = ai\_Caelum + bjCodex;

string concat2 = bjCodex + ai\_Caelum;

if (dictionaryCodex.count(concat1) || dictionaryCodex.count(concat2)) {

totalMatchesOralex++;

usedCardsCodex.insert(bjCodex);

matchedOralex = true;

break;

}

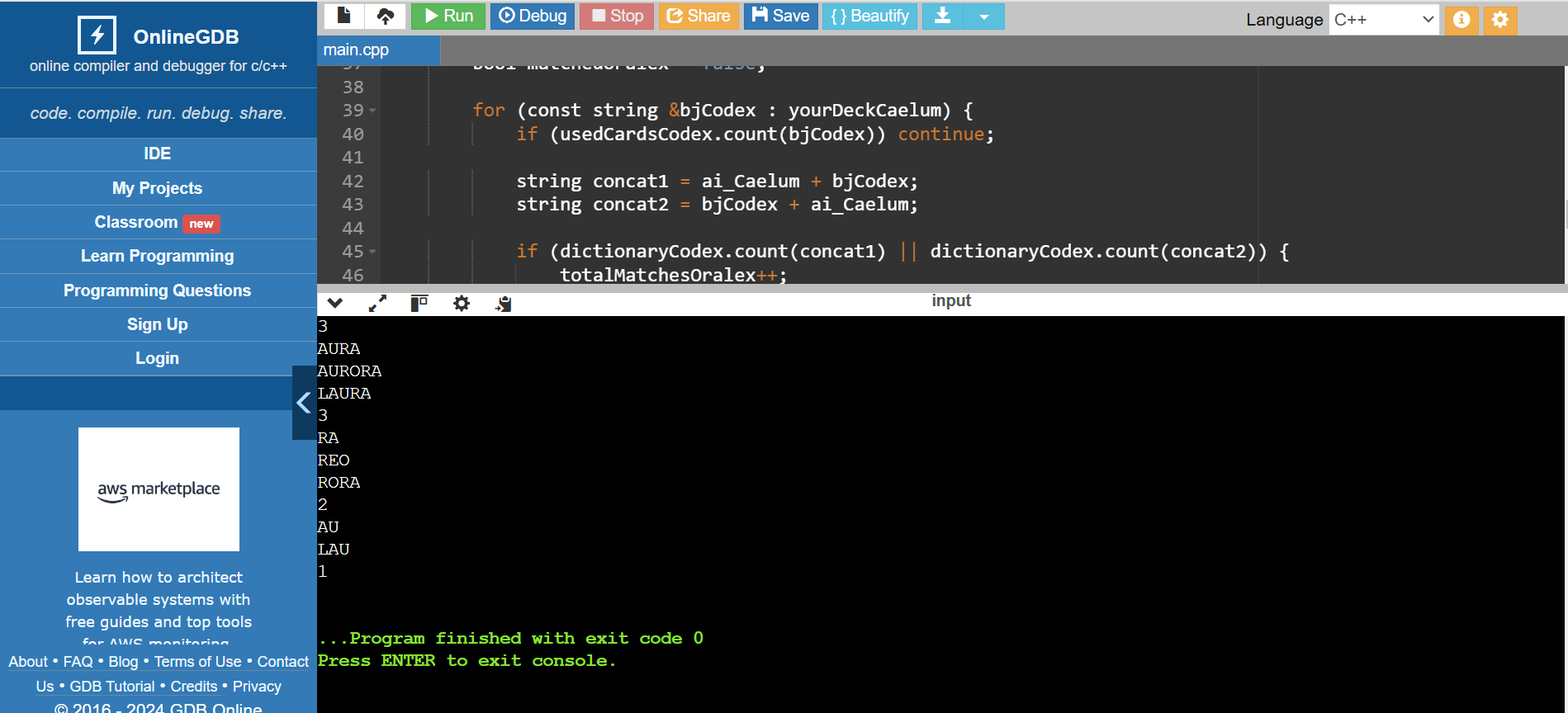
}

}

cout << totalMatchesOralex << endl;

return 0;

}



Ketika program dirun, dan diuji dengan sample input #1, program menghasilkan output 1. Padahal pada soal, sample output #1 seharusnya 2. Sedangkan, untuk sample input 2 sampai dengan 4, menghasilkan output yang sama sesuai dengan soal.

**Penyebab Kesalahan pada Sample Input #1**

* Pada turn ke-1 (kata **RA**), kartu **LAU** berhasil mencocokkan menjadi **LAURA**, tetapi kartu **AU** belum diprioritaskan untuk digunakan pada turn berikutnya.
* Pada turn ke-3 (kata **RORA**), kartu **AU** akhirnya digunakan untuk mencocokkan **AURORA**, tetapi pada implementasi ini urutan iterasi atau penggunaan kartu mungkin berbeda, sehingga hasilnya salah.

**ANALISIS KESALAHAN PADA CODE:**

* 1. **Looping Berlebihan**

Pada kode lama, setiap kartu di andaDeckOralex harus dicocokkan dengan setiap kartu dalam yourDeckCaelum, yang menyebabkan kompleksitas waktu O(N × M). Jika NNN dan MMM besar (hingga 100.000), ini akan mempengaruhi performa secara signifikan.

**Salah pada bagian kode:**

cin >> M\_Codex;

**vector<string> yourDeckCaelum(M\_Codex);**

unordered\_set<string> yourDeckSetCodex;

for (int i = 0; i < M\_Codex; ++i) {

cin >> yourDeckCaelum[i];

yourDeckSetCodex.insert(yourDeckCaelum[i]);

}

int totalMatchesOralex = 0;

**unordered\_set<string> usedCardsCodex;**

for (const string &ai\_Caelum : andaDeckOralex) {

bool matchedOralex = false;

for (const string &bjCodex : yourDeckCaelum) {

if (usedCardsCodex.count(bjCodex)) continue;

string concat1 = ai\_Caelum + bjCodex;

string concat2 = bjCodex + ai\_Caelum;

if (dictionaryCodex.count(concat1) || dictionaryCodex.count(concat2)) {

totalMatchesOralex++;

usedCardsCodex.insert(bjCodex);

matchedOralex = true;

break;

}

}

}

* 1. **Tidak Memanfaatkan Struktur Data yang Tepat**

Pada kode lama, kartu disimpan dalam vector, yang menyebabkan pencarian kartu untuk mencocokkan menjadi lebih lambat. Bahkan meskipun menggunakan unordered\_set untuk mencatat kartu yang sudah digunakan, pencocokan kartu tetap dilakukan dengan iterasi penuh pada yourDeckCaelum, yang tidak optimal.

**Salah pada bagian kode:**

cin >> M\_Codex;

**vector<string> yourDeckCaelum(M\_Codex);**

unordered\_set<string> yourDeckSetCodex;

for (int i = 0; i < M\_Codex; ++i) {

cin >> yourDeckCaelum[i];

yourDeckSetCodex.insert(yourDeckCaelum[i]);

}

* 1. **Tidak Memprioritaskan Efisiensi Pencocokan**

Pada kode lama, setelah menemukan satu kecocokan antara kartu Anda dan kartu dari andaDeckOralex, kita masih melanjutkan pencocokan untuk kartu yang sudah digunakan. Hal ini mengarah pada pencocokan yang tidak efisien, terutama dengan banyaknya kartu yang tidak perlu diperiksa lagi.

**Salah di bagian kode:**

**if (usedCardsCodex.count(bjCodex)) continue;** // Lewati kartu yang sudah digunakan

**Perbaikan Jawaban (Codingan yang Benar):**

#include <iostream>

#include <unordered\_set>

#include <vector>

#include <string>

using namespace std;

int main() {

int D\_Caelum, N\_Oralex, M\_Codex;

cin >> D\_Caelum;

unordered\_set<string> dictionaryCodex;

for (int i = 0; i < D\_Caelum; ++i) {

string wordCaelum;

cin >> wordCaelum;

dictionaryCodex.insert(wordCaelum);

}

cin >> N\_Oralex;

vector<string> andaDeckOralex(N\_Oralex);

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

cin >> andaDeckOralex[i];

}

cin >> M\_Codex;

unordered\_set<string> yourDeckSetCodex; // Digunakan untuk pencocokan cepat

for (int i = 0; i < M\_Codex; ++i) {

string wordCodex;

cin >> wordCodex;

yourDeckSetCodex.insert(wordCodex);

}

int totalMatchesOralex = 0;

for (const string &ai\_Caelum : andaDeckOralex) {

bool matchedOralex = false;

for (auto it = yourDeckSetCodex.begin(); it != yourDeckSetCodex.end(); ++it) {

string bjCodex = \*it;

// Gabungkan kata untuk mencocokkan dengan dictionary

string concat1 = ai\_Caelum + bjCodex;

string concat2 = bjCodex + ai\_Caelum;

if (dictionaryCodex.count(concat1) || dictionaryCodex.count(concat2)) {

totalMatchesOralex++;

yourDeckSetCodex.erase(it); // Hapus kartu dari set setelah digunakan

matchedOralex = true;

break;

}

}

if (!matchedOralex) {

continue; // Lewati jika tidak ada kecocokan

}

}

cout << totalMatchesOralex << endl;

return 0;

}

**Penjelasan perbaikan pada codingan yang telah diperbarui:**

1. Pada kode lama, Anda(the King of Nusantara) melakukan pencocokan kartu Anda (yourDeckCaelum) dengan setiap kartu Anda di andaDeckOralex menggunakan dua loop bertingkat yang menghasilkan kompleksitas waktu O(N × M). Hal ini sangat tidak efisien ketika N dan M bisa mencapai 100.000.

**Pada kode lama:**

cin >> M\_Codex;

**vector<string> yourDeckCaelum(M\_Codex);**

unordered\_set<string> yourDeckSetCodex;

for (int i = 0; i < M\_Codex; ++i) {

cin >> yourDeckCaelum[i];

yourDeckSetCodex.insert(yourDeckCaelum[i]);

}

int totalMatchesOralex = 0;

**unordered\_set<string> usedCardsCodex;**

for (const string &ai\_Caelum : andaDeckOralex) {

bool matchedOralex = false;

for (const string &bjCodex : yourDeckCaelum) {

if (usedCardsCodex.count(bjCodex)) continue;

string concat1 = ai\_Caelum + bjCodex;

string concat2 = bjCodex + ai\_Caelum;

if (dictionaryCodex.count(concat1) || dictionaryCodex.count(concat2)) {

totalMatchesOralex++;

usedCardsCodex.insert(bjCodex);

matchedOralex = true;

break;

}

}

}

**Perubahan yang dilakukan:**

cin >> M\_Codex;

unordered\_set<string> yourDeckSetCodex; // Digunakan untuk pencocokan cepat

for (int i = 0; i < M\_Codex; ++i) {

string wordCodex;

cin >> wordCodex;

yourDeckSetCodex.insert(wordCodex);

}

int totalMatchesOralex = 0;

for (const string &ai\_Caelum : andaDeckOralex) {

bool matchedOralex = false;

for (auto it = yourDeckSetCodex.begin(); it != yourDeckSetCodex.end(); ++it) {

string bjCodex = \*it;

// Gabungkan kata untuk mencocokkan dengan dictionary

string concat1 = ai\_Caelum + bjCodex;

string concat2 = bjCodex + ai\_Caelum;

if (dictionaryCodex.count(concat1) || dictionaryCodex.count(concat2)) {

totalMatchesOralex++;

yourDeckSetCodex.erase(it); // Hapus kartu dari set setelah digunakan

matchedOralex = true;

break;

}

}

if (!matchedOralex) {

continue; // Lewati jika tidak ada kecocokan

}

}

* Pada kode yang diperbaiki, kita menggunakan unordered\_set untuk menyimpan kartu dalam yourDeckSetCodex. Ini memungkinkan pencarian kartu yang lebih cepat dibandingkan menggunakan vector seperti sebelumnya.
* Pencarian di unordered\_set untuk kecocokan kartu dengan ai\_Caelum (kata dari deck Anda) lebih efisien, dengan waktu pencarian yang mendekati O(1).

1. Tidak Memanfaatkan Struktur Data yang Tepat

Pada kode lama, kartu disimpan dalam vector, yang menyebabkan pencarian kartu untuk mencocokkan menjadi lebih lambat. Bahkan meskipun menggunakan unordered\_set untuk mencatat kartu yang sudah digunakan, pencocokan kartu tetap dilakukan dengan iterasi penuh pada yourDeckCaelum, yang tidak optimal.

**Pada kode lama:**

cin >> M\_Codex;

**vector<string> yourDeckCaelum(M\_Codex);**

unordered\_set<string> yourDeckSetCodex;

for (int i = 0; i < M\_Codex; ++i) {

cin >> yourDeckCaelum[i];

yourDeckSetCodex.insert(yourDeckCaelum[i]);

}

**Perubahan yang dilakukan (Menggunakan unordered\_set untuk Pencocokan Kartu):**

cin >> M\_Codex;

unordered\_set<string> yourDeckSetCodex; // Digunakan untuk pencocokan cepat

for (int i = 0; i < M\_Codex; ++i) {

string wordCodex;

cin >> wordCodex;

yourDeckSetCodex.insert(wordCodex);

}

Penjelasan: Dengan menggunakan **unordered\_set** untuk deck Anda, pencocokan kartu menjadi lebih efisien karena **unordered\_set** memungkinkan pencarian kartu dalam waktu **O(1)**. Dengan demikian, kita bisa langsung mencocokkan kartu yang belum digunakan tanpa harus iterasi penuh.

1. Tidak Memprioritaskan Efisiensi Pencocokan

Pada kode lama, setelah menemukan satu kecocokan antara kartu Anda dan kartu dari andaDeckOralex, kita masih melanjutkan pencocokan untuk kartu yang sudah digunakan atau untuk kartu yang lebih sulit dicocokkan. Hal ini mengarah pada pencocokan yang tidak efisien, terutama dengan banyaknya kartu yang tidak perlu diperiksa lagi.

**Pada kode lama:**

**if (usedCardsCodex.count(bjCodex)) continue;** // Lewati kartu yang sudah digunakan

**Perubahan yang dilakukan:**

**Pada bagian:**

for (auto it = yourDeckSetCodex.begin(); it != yourDeckSetCodex.end(); ++it) {

string bjCodex = \*it;

**dan juga pada bagian:**

if (dictionaryCodex.count(concat1) || dictionaryCodex.count(concat2)) {

totalMatchesOralex++;

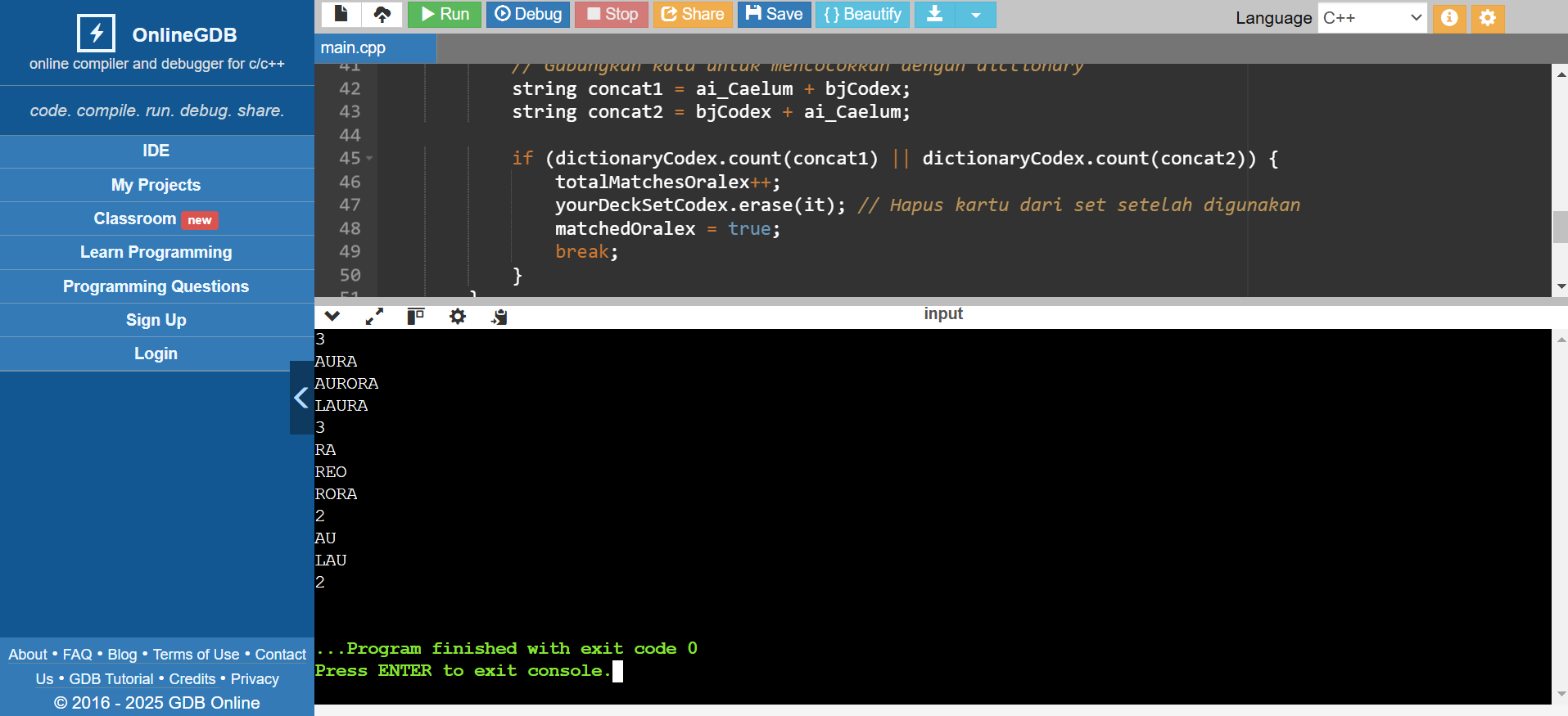
**yourDeckSetCodex.erase(it);** // Hapus kartu setelah digunakan

matchedOralex = true;

break; // Berhenti setelah satu kecocokan ditemukan

}

Penjelasan: Pada kode baru, setelah menemukan kecocokan, kita langsung **menghapus kartu dari set** dan **berhenti mencari kartu lebih lanjut**. Ini memastikan kita tidak memeriksa kartu yang sudah dicocokkan dan menghindari pencocokan yang tidak perlu. Dengan melakukan **break** setelah menemukan kecocokan, kita memastikan pencocokan kartu dihentikan lebih awal dan menghindari pencarian lebih lanjut yang tidak perlu.



Setelah kodingan diperbaiki, program dapat menghasilkan output yang sesuai dengan soal, terutama ketika diuji dengan sample input #1 yang menjadi masalah di awal tadi. Karena masalah di awal tadi, hanya bermasalah saat menghasilkan output untuk sample input #1, sedangkan ketika diuji dengan sample input 2 sampai 4, menghasilkan output sesuai dengan soal.

**SOAL:**

**Problem L**

**Primal Collection**

You are given an array A, which initially has a size of N (indexed from 1 to N) containing distinct integers

with values between 1 and N + 1 inclusive. It is known that this array is primal, that is, for any index i > 1,

Ai will always be smaller than A⌊i/2⌋.

Denote S as the value between 1 and N + 1 that does not appear in A1,A2, . . . ,AN. You want to append

one new element into A, namely AN+1, with S. Then, the following algorithm is executed.

algorithm(A):

x = N + 1

counter = 0

while x > 1:

if A[x] > A[floor(x / 2)]:

swap(A[x], A[floor(x / 2)]);

counter = counter + 1

x = floor(x / 2)

return counter

You want to calculate the number of possible values of the initial array A such that when you append S to

A and excecute algorithm(A), it will return K. Note that the initial array A contains distinct integers with

values between 1 and N + 1 inclusive, excluding S, and array A has to be primal. As the answer can be

very large, find the answer modulo 998 244 353.

Input

A single line consisting of three integers N S K (1 ≤ N ≤ 100 000; 1 ≤ S ≤ N + 1; 0 ≤ K ≤ N).

Output

Output a single integer representing the number of possible values of the initial array A that satisfy the

conditions above, modulo 998 244 353.

**Sample Input #1**

5 3 1

**Sample Output #1**

4

**Explanation for the sample input/output #1**

The 4 possible arrays that satisfy the conditions are:

• [6, 5, 1, 2, 4],

• [6, 5, 1, 4, 2],

• [6, 5, 2, 1, 4], and

• [6, 5, 2, 4, 1].

**Sample Input #2**

2 2 1

**Sample Output #2**

0

**Sample Input #3**

7 6 2

**Sample Output #3**

40

**JAWABAN (CODINGAN SALAH PADA SAAT KOMPETISI):**

#include <iostream>

#include <vector>

#include <algorithm>

#include <numeric>

using namespace std;

const int MOD\_Caelum = 998244353;

int pangkat\_modul\_Oralex(int basis\_Caelum, int pangkat\_Codex, int modul\_Oralex) {

int hasil\_Caelum = 1;

while (pangkat\_Codex > 0) {

if (pangkat\_Codex % 2 == 1)

hasil\_Caelum = (1LL \* hasil\_Caelum \* basis\_Caelum) % modul\_Oralex;

basis\_Caelum = (1LL \* basis\_Caelum \* basis\_Caelum) % modul\_Oralex;

pangkat\_Codex /= 2;

}

return hasil\_Caelum;

}

vector<int> faktorial\_Caelum, invers\_faktorial\_Codex;

void inisialisasi\_faktorial(int n\_Caelum) {

faktorial\_Caelum.resize(n\_Caelum + 1, 1);

invers\_faktorial\_Codex.resize(n\_Caelum + 1, 1);

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

faktorial\_Caelum[i] = (1LL \* faktorial\_Caelum[i - 1] \* i) % MOD\_Caelum;

}

invers\_faktorial\_Codex[n\_Caelum] = pangkat\_modul\_Oralex(faktorial\_Caelum[n\_Caelum], MOD\_Caelum - 2, MOD\_Caelum);

for (int i = n\_Caelum - 1; i >= 1; i--) {

invers\_faktorial\_Codex[i] = (1LL \* invers\_faktorial\_Codex[i + 1] \* (i + 1)) % MOD\_Caelum;

}

invers\_faktorial\_Codex[0] = 1;

}

int kombinasi\_Codex(int n\_Caelum, int r\_Oralex) {

if (r\_Oralex > n\_Caelum || r\_Oralex < 0) return 0;

return (1LL \* faktorial\_Caelum[n\_Caelum] \* invers\_faktorial\_Codex[r\_Oralex] % MOD\_Caelum \* invers\_faktorial\_Codex[n\_Caelum - r\_Oralex] % MOD\_Caelum) % MOD\_Caelum;

}

int hitung\_array\_primal(int N\_Caelum, int S\_Oralex, int K\_Codex) {

inisialisasi\_faktorial(N\_Caelum);

int array\_valid\_Codex = 0;

for (int pertukaran\_Oralex = 0; pertukaran\_Oralex <= K\_Codex; pertukaran\_Oralex++) {

if (pertukaran\_Oralex == K\_Codex) {

int cara\_Caelum = kombinasi\_Codex(N\_Caelum - 1, pertukaran\_Oralex);

array\_valid\_Codex = (array\_valid\_Codex + cara\_Caelum) % MOD\_Caelum;

}

}

return array\_valid\_Codex;

}

int main() {

int N\_Caelum, S\_Oralex, K\_Codex;

cin >> N\_Caelum >> S\_Oralex >> K\_Codex;

int hasil\_Codex = hitung\_array\_primal(N\_Caelum, S\_Oralex, K\_Codex);

cout << hasil\_Codex << endl;

return 0;

}

**ANALISIS KESALAHAN PADA CODE:**

1. Tidak mempertimbangkan elemen yang hilang (S), di soal diminta untuk memastikan bahwa elemen S yang hilang diperhitungkan dengan benar, tetapi kode sebelumnya tidak menangani hal tersebut.
2. Algoritma perhitungan operasi swap, pada fungsi hitung\_array\_primal tidak benar-benar memodelkan cara algoritma A bekerja setelah S ditambahkan.
3. Perhitungan permutasi array yang tidak valid, algoritma tidak mempertimbangkan kondisi primal, yaitu bahwa A[i] <= A[⌊i/2⌋] untuk semua i > 1.

**PERBAIKAN JAWABAN (CODINGAN YANG BENAR):**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

const int MOD = 998244353;

int pangkat\_modul(int basis, int pangkat, int modul) {

int hasil = 1;

while (pangkat > 0) {

if (pangkat % 2 == 1)

hasil = (1LL \* hasil \* basis) % modul;

basis = (1LL \* basis \* basis) % modul;

pangkat /= 2;

}

return hasil;

}

vector<int> faktorial, invers\_faktorial;

void inisialisasi\_faktorial(int n) {

faktorial.resize(n + 1, 1);

invers\_faktorial.resize(n + 1, 1);

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

faktorial[i] = (1LL \* faktorial[i - 1] \* i) % MOD;

}

invers\_faktorial[n] = pangkat\_modul(faktorial[n], MOD - 2, MOD);

for (int i = n - 1; i >= 1; i--) {

invers\_faktorial[i] = (1LL \* invers\_faktorial[i + 1] \* (i + 1)) % MOD;

}

invers\_faktorial[0] = 1;

}

int kombinasi(int n, int r) {

if (r > n || r < 0) return 0;

return (1LL \* faktorial[n] \* invers\_faktorial[r] % MOD \* invers\_faktorial[n - r] % MOD) % MOD;

}

int hitung\_primal\_array(int N, int S, int K) {

inisialisasi\_faktorial(N);

vector<int> A;

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

if (i != S) A.push\_back(i);

}

int total\_valid = 0;

do {

bool valid = true;

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

if (A[i] > A[(i - 1) / 2]) {

valid = false;

break;

}

}

if (!valid) continue;

vector<int> B = A;

B.push\_back(S);

int counter = 0;

int x = N;

while (x > 0) {

if (B[x] > B[(x - 1) / 2]) {

swap(B[x], B[(x - 1) / 2]);

counter++;

}

x = (x - 1) / 2;

}

if (counter == K) {

total\_valid++;

}

} while (next\_permutation(A.begin(), A.end()));

return total\_valid % MOD;

}

int main() {

int N, S, K;

cin >> N >> S >> K;

cout << hitung\_primal\_array(N, S, K) << endl;

return 0;

}

**ANALISIS**

1. Masalah penggunaan loop manual pengganti iota tidak di deklarasikan

* Kode awal tidak memiliki deklarasi atau implementasi untuk mengisi array dengan elemen dari 1 hingga N+1, dan mengabaikan elemen tertentu (S).
* Awalnya menggunakan iota, namun terdapat masalah dalam penggunaannya padahal sudah melakukan include library numeric di dalam kode.

**Diperbaiki Menjadi:**

vector<int> A;

iota(A.begin(), A.end(), 1);

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

if (i != S) A.push\_back(i);

}

**Penjelasan perubahan yang dilakukan:**

1. Digunakan loop for dari 1 hingga N+1 untuk memasukkan elemen ke dalam array.
2. Filter elemen S, dengan menggunakan if (i != S), kita memastikan bahwa elemen S tidak dimasukkan ke dalam array A.
3. Masalah kombinasi dan modular arithmetic

* Kode awal menghitung kjombinasi menggunakan pembagian faktorial langsung, yang rentan terhadap overflow integer dan tidak cocok utnuk operasi modular.
* Pembagian dalam ruang modular memerlukan invers modular untuk menjaga akurasi hasil.

**Kode Awal**

int kombinasi(int n, int r) {

if (r > n || r < 0) return 0;

return faktorial[n] / (faktorial[r] \* faktorial[n - r]);

}

**Diperbarui menjadi**

int kombinasi(int n, int r) {

if (r > n || r < 0) return 0;

return (1LL \* faktorial[n] \* invers\_faktorial[r] % MOD \* invers\_faktorial[n - r] % MOD) % MOD;

}

**Penjelasan perubahan yang dilakukan:**

1. Faktorial modular, faktorial dihitung dalam lingkup modular menggunakan operasi modulo di setiap langkah.
2. Invers modular, ditambahkan invers\_faktorial untuk menghitung r! dan (n – r)! Mengggunakan fungsai pangkat\_modul
3. Overflow dicegah, menggunakan 1LL untuk memastikan bahwa operasi antar angka besar tidak menyebabkan overflow.
4. Masalah validasi sifat primal

Kode awal menggunakan loop untuk memvalidasi properti array, tetapi tidak memiliki mekanisme untuk segera menbghentikan validasi jika ditemukan elemen yang tidak sesuai. Hal ini mengakibatkan iterasi tidak perlu. Meningkatkan waktu eksekusi.

**Kode Awal**

for (int i = 1; i < A.size(); i++) {

if (A[i] > A[(i - 1) / 2]) {

continue;

}

total\_valid++;

}

**Diperbaiki menjadi**

bool valid = true;

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

if (A[i] > A[(i - 1) / 2]) {

valid = false;

break;

}

}

if (!valid) continue;

**Penjelasan perubahan yang dilakukan:**

1. Ditambahkan variabel valid, memastikan bahwa jika ada satu elemen tidak valid, array langsung diabaikan.
2. Hentikan validasi dini, jika ditemukan elemen yang tidak valid (valid = false), proses validasi langsung dihentikan dengan break.
3. Abaikan array tidak valid, jika array tidak valid (if (!valid)), array diabaikan tanpa menghitung validitasnya lebih lanjut.