Университет ИТМО

Факультет программной инженерии и компьютерной техники

**Лабораторная работа №1**

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3231

Чураков Александр

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2025

# Yandex

#### Задача A «Агроном-любитель»

Пояснение:

Задача в том, что мы должны найти наибольшую подпоследовательность, где нет подряд 3 одинаковых элементов. Проходимся по всем сиволам последовательно и ищем самую длиннкю цепочку где нет 3 подряд одинаковых, если повторение или последний символ, то сравниваем с максимальной найденой.

Сам код работает за O(N+N) = O(2N) = O(N)

Память: O(N) *#include <iostream>*

*#include <vector>*

*using namespace std;*

*int main() {*

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

*cin.tie(nullptr);*

*int n;*

*cin >> n;*

*vector<long long> arr(n);*

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

*cin >> arr[i];*

*}*

*int ansStart = 0, ansEnd = 0;*

*int maxLength = 0;*

*int curStart = 0;*

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

*if (arr[i] == arr[i - 1] && arr[i - 1] == arr[i - 2]) {*

*int curLength = i - curStart;*

*if (curLength > maxLength) {*

*maxLength = curLength;*

*ansStart = curStart;*

*ansEnd = i - 1;*

*}*

*curStart = i - 1;*

*}*

*}*

*int curLength = n - curStart;*

*if (curLength > maxLength) {*

*ansStart = curStart;*

*ansEnd = n - 1;*

*}*

*cout << ansStart + 1 << " " << ansEnd + 1;*

*return 0;*

*}*

#### Задача B «Зоопарк Глеба»

Данная задача также сводится к тому, что мы нумеруем всех животных (строчные буквы) и ловушки (заглавные буквы) особым образом:

* Животные получают индексы в порядке 1,2,3,… по мере чтения строки.
* Для ловушек используется формула i−cnt, где i — позиция символа в исходной строке, а cnt — сколько животных (строчных букв) мы уже встретили.

Таким образом, если мы находимся в позиции i, а до этого встретили cnt животных, то ловушка получает номер (i−cnt). Это позволяет разделить все символы на две группы с независимой нумерацией.

Затем формируем двусвязный список, в который записываем пары (индекс,символ). По сути, в этом списке мы последовательно храним всех участников — сначала в порядке появления в строке, но с локальным индексом, который различает ловушки и животных.

Далее идёт жадная обработка списка с двух сторон:

1. Проверяем, могут ли совпасть (по виду, но разному регистру) **первый** и **последний** элемент списка. Если да — удаляем их и фиксируем, что соответствующая ловушка поймала животное.
2. Если первый и последний не совпадают, смотрим на **первый** и **второй** элемент. Если они подходят, тоже удаляем их и делаем отметку о поимке.
3. Если не совпадает ни одна из этих пар, прерываемся.

Чтобы узнать, совпадают ли символы проверяем разницу в их кодах ASCII: abs(c1​−c2​)=32.

Если к моменту окончания жадного алгоритма список опустел, значит все животные смогли дошли до ловушек.

### Оценка сложности

* **Время работы**: O(N). Каждый символ списка мы рассматриваем в худшем случае один раз, удаляя пару за константное время.
* **Память**: O(N) для хранения исходных данных и служебных структур, где N — длина входной строки (то есть 2n, если n — число животных).

#include <cstddef>

#include <iostream>

#include <list>

#include <optional>

#include <string>

#include <utility>

#include <vector>

constexpr int kShift = 'a' - 'A'; // потому что так хочу

namespace {

bool IsMatchingPair(char x, char y) {

return std::abs(x - y) == kShift;

}

std::optional<std::vector<size\_t>> FindMapping(const std::string& data) {

std::vector<size\_t> output(data.size() / 2);

std::list<std::pair<size\_t, char>> lst;

size\_t cnt = 0;

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

char c = data[i];

if (std::isupper(c)) {

lst.emplace\_back(i - cnt, c);

} else {

lst.emplace\_back(++cnt, c);

}

}

while (!lst.empty()) {

std::pair<size\_t, char> fr = lst.front();

std::pair<size\_t, char> bk = lst.back();

auto sec = std::next(lst.begin());

if (IsMatchingPair(fr.second, bk.second)) {

const std::pair<size\_t, char> trap = std::isupper(fr.second) ? fr : bk;

const std::pair<size\_t, char> animal = std::isupper(fr.second) ? bk : fr;

output[trap.first] = animal.first;

lst.pop\_front();

lst.pop\_back();

} else if (sec != lst.end() && IsMatchingPair(fr.second, sec->second)) {

const std::pair<size\_t, char> trap = std::isupper(fr.second) ? fr : \*sec;

const std::pair<size\_t, char> animal = std::isupper(fr.second) ? \*sec : fr;

output[trap.first] = animal.first;

lst.pop\_front();

lst.pop\_front();

} else {

break;

}

}

if (!lst.empty()) {

return std::nullopt;

}

return output;

}

} // namespace

int main() {

std::string input;

std::cin >> input;

std::optional<std::vector<size\_t>> result = FindMapping(input);

if (result) {

std::cout << "Possible\n";

for (size\_t x : \*result) {

std::cout << x << " ";

}

} else {

std::cout << "Impossible";

}

std::cout << "\n";

return 0;

}