Университет ИТМО

Факультет программной инженерии и компьютерной техники

**Лабораторная работа №1**

по «Алгоритмам и структурам данных»

Базовые задачи / Timus

Выполнил:

Студент группы P3218

Хромов Даниил Тимофеевич

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2023

Задача №1 «Агроном-любитель»

Пояснение к примененному алгоритму:

В данном решении мы используем указатели и считаем нашу максимальную возможную длину. В общем и целом логика работы программы:  
1. Для каждого end (от 0 до n-1) проверяется, образуют ли текущий элемент и два предыдущих тройку одинаковых.

2. Если тройка обнаружена, начало окна (start) сдвигается на позицию end-1, чтобы исключить тройку из текущего участка.

3.На каждой итерации вычисляется длина текущего участка. Если она больше максимальной найденной, обновляются значения максимальной длины и границ результата.

Код:

#include <sys/types.h>

#include <cstddef>

#include <iostream>

#include <vector>

using namespace std;

pair<u\_int64\_t, u\_int64\_t> solve(const vector<u\_int64\_t>& a) {

u\_int64\_t n = a.size();

u\_int64\_t max\_len = 0;

u\_int64\_t start = 0;

u\_int64\_t result\_start = 0;

u\_int64\_t result\_end = 0;

for (size\_t end = 0; end < n; ++end) {

if (end >= 2 && a[end] == a[end - 1] && a[end] == a[end - 2]) {

start = end - 1;

}

u\_int64\_t current\_len = end - start + 1;

if (current\_len > max\_len) {

max\_len = current\_len;

result\_start = start + 1;

result\_end = end + 1;

}

}

return {result\_start, result\_end};

}

int main() {

u\_int64\_t n;

cin >> n;

vector<u\_int64\_t> a(n);

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

cin >> a[i];

}

pair<u\_int64\_t, u\_int64\_t> result = solve(a);

cout << result.first << " " << result.second << endl;

a.clear();

a.shrink\_to\_fit();

return 0;

}

Задача №2 «Зоопарк Глеба»

Пояснение к примененному алгоритму:

Для решения мы используем структуры stack и map (причем unordered, для повышения оптимизации). Так вот в stack <int> у нас хранятся: animalPosition - индексы животных, trapPosition — индексы ловушек, zooStack — проверяет корректность последовательности. В общем и целом, алгоритм довольно прост, мы если встречаем ловушку то добавляем её индекс в соответствующий стек; а далее проверяем, является ли текущий символ парой к верхнему элементу в стеке zooStack, если не является, то мы добавляем элемент в zooStack, а если да, то связываем текущую ловушку и животное из стеков, добавляя в map, удаляя их из стеков.  
  
В конце концов, мы проверяем, если zooStack пуст — все животные попадут в свои ловушки. Если же нет, то мы выводим «Impossible».

**Временная сложность**: O(n) — каждое животное и ловушка в нашем случае обрабатываются ровно один раз.  
**Пространственная сложность**: O(n) — в худшем случае все символы попадут в стек.

Код:  
#include <bits/stdc++.h>

using namespace std;

bool areMatchingPair(char animal, char trap) {

return toupper(animal) == toupper(trap) && animal != trap;

}

bool processZoo(const string& zoo, vector<int>& result) {

stack<char> zooStack;

stack<int> animalPositions, trapPositions;

unordered\_map<int, int> matches;

int animalIndex = 0, trapIndex = 0;

for (char c : zoo) {

if (isupper(c)) {

trapIndex++;

trapPositions.push(trapIndex);

} else {

animalIndex++;

animalPositions.push(animalIndex);

}

if (zooStack.empty() || !areMatchingPair(zooStack.top(), c)) {

zooStack.push(c);

} else {

matches[trapPositions.top()] = animalPositions.top();

animalPositions.pop();

trapPositions.pop();

zooStack.pop();

}

}

if (!zooStack.empty())

return false;

result.resize(trapIndex);

for (auto& [trap, animal] : matches) {

result[trap - 1] = animal;

}

return true;

}

void printResult(const vector<int>& result) {

cout << "Possible\n";

for (int index : result) {

cout << index << " ";

}

cout << "\n";

}

int main() {

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

cin.tie(nullptr);

string zoo;

cin >> zoo;

vector<int> result;

if (processZoo(zoo, result)) {

printResult(result);

} else {

cout << "Impossible\n";

}

return 0;

}

Задача №3 «Конфигурационный файл»

Пояснение к примененному алгоритму:

Здесь у нас используется уже известная нам структура — unordered\_map<string, int> variables — она собственно отвечает за хранение текущего значения переменных; но также у нас имеется более сложная структура в своем понимании — stack<vector<pair<string, int>>> history — это стек, хранящий вектор (он испоьзуется в качестве массива) пар строка и значение — изменения переменных внутри блоков, которые организует пользователь.  
  
Первым делом программа смотрит, есть ли среди введенных в строке символов «{» - начало блока (если есть, то мы как бы создаем некоторую новую историю с изменениями переменных и записываем ее в наш стек), когда у нас она заканчивается «}» - мы удаляем текущий слой, чтобы не заполнять память и таким образом храним только значения переменных, а не всю их историю изменений. Более подробно про механизм сохранения и отката:  
1. При входе в блок «{» мы создаем новый уровень, как бы новую запись в нашей карточке (историю).  
Сохраняем старое значение в history, чтобы позже откатить его.

* Если переменная ранее не существовала, фиксируем INT\_MIN, указывая, что её надо удалить после выхода из блока.

При выходе из блока «}»:

* Достаём все сохранённые изменения и восстанавливаем переменные.

**Временная сложность**: O(1) — для каждой операции присваивания или восстановления; O(n) в худшем случае (если весь файл — это вложенные блоки).  
**Пространственная сложность**: O(n) — в худшем случае мы храним все переменные на каждом уровне вложенности.

Код:  
#include <bits/stdc++.h>

using namespace std;

unordered\_map<string, int> variables;

stack<vector<pair<string, int>>> history;

int get\_value(const string& var) {

return variables.count(var) ? variables[var] : 0;

}

int main() {

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

cin.tie(nullptr);

string line;

while (cin >> line) {

if (line == "{") {

history.push({});

} else if (line == "}") {

for (auto it = history.top().rbegin(); it != history.top().rend(); ++it) {

auto& [var, old\_value] = \*it;

if (old\_value == INT\_MIN) {

variables.erase(var);

} else {

variables[var] = old\_value;

}

}

history.pop();

} else {

size\_t eq\_pos = line.find('=');

string var1 = line.substr(0, eq\_pos);

string var2 = line.substr(eq\_pos + 1);

if (!history.empty()) {

if (!variables.count(var1)) {

history.top().emplace\_back(var1, INT\_MIN);

} else {

history.top().emplace\_back(var1, variables[var1]);

}

}

if (isdigit(var2[0]) || var2[0] == '-') {

variables[var1] = stoi(var2);

} else {

int assigned\_value = get\_value(var2);

cout << assigned\_value << endl;

variables[var1] = assigned\_value;

}

}

}

return 0;

}

Задача №4 «Профессор Хаос»

Пояснение к примененному алгоритму:

После чтения нужных данных мы получаем следующий алгоритм решения:

Запускаем симуляцию процесса на k дней:

* Вычисляем количество бактерий после деления: next = current \* b.
* Если после деления количество бактерий next **меньше или равно** c, значит, все бактерии будут уничтожены, и эксперимент завершится (выводим 0).
* Вычитаем c, так как бактерии уничтожаются в опытах.
* Если после этого **бактерий больше** d, то оставляем только d, остальные уничтожаем.
* Если на этом этапе количество бактерий **не изменилось** по сравнению с предыдущим днем, значит, дальнейшие дни ничего не поменяют, и можно вывести результат сразу.

Если цикл завершился без прерывания, выводим текущее количество бактерий.

**Временная сложность:** O(n) – в худшем случае, но такого в реальности не происходит.  
  
Код:  
#include <bits/stdc++.h>

using namespace std;

using ll = long long;

int main() {

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

cin.tie(nullptr);

ll a, b, c, d, k;

cin >> a >> b >> c >> d >> k;

ll current = a;

for (ll day = 0; day < k; day++) {

ll next = current \* b;

if (next <= c) {

cout << "0\n";

return 0;

}

next -= c;

if (next > d)

next = d;

if (next == current) {

cout << current << "\n";

return 0;

}

current = next;

}

cout << current << "\n";

return 0;

}

Задача №1005 «Куча камней»

**Перебор всех разбиений:** Мы перебираем все возможные способы разделить камни на две группы с помощью битовых масок. Маска mask длиной n указывает, какие камни принадлежат первой куче. Если i-й бит маски равен 1, то i-й камень принадлежит первой куче.

**Вычисление разницы сумм:** Для каждого разбиения вычисляем сумму камней в первой куче, а сумма для второй автоматически вычисляется как разница между общей суммой и суммой первой кучи.

**Минимизация разницы:** Мы ищем минимальную разницу между суммами двух куч.

Код:  
#include <climits>

#include <cmath>

#include <iostream>

#include <vector>

using namespace std;

int main() {

int n;

cin >> n;

vector<int> stones(n);

int total\_sum = 0;

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

cin >> stones[i];

total\_sum += stones[i];

}

int min\_diff = INT\_MAX;

for (int mask = 0; mask < (1 << n); ++mask) {

int sum1 = 0;

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

if (mask & (1 << i)) {

sum1 += stones[i];

}

}

int sum2 = total\_sum - sum1;

min\_diff = min(min\_diff, abs(sum1 - sum2));

}

cout << min\_diff << endl;

return 0;

}  
  
Задача №2025 «Стенка на стенку»

Равномерное распределение: Разделить n бойцов на k команд так, чтобы размеры команд отличались не более чем на 1.

Формула для схваток: Использовать формулу суммы попарных произведений через сумму квадратов:

Код:  
#include <iostream>

using namespace std;

int main() {

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

cin.tie(nullptr);

int T;

cin >> T;

while (T--) {

long long n, k;

cin >> n >> k;

long long m = n / k;

long long r = n % k;

long long sum\_sq = r \* (m + 1) \* (m + 1) + (k - r) \* m \* m;

long long ans = (n \* n - sum\_sq) / 2;

cout << ans << '\n';

}

return 0;

}