Университет ИТМО

Факультет программной инженерии и компьютерной техники

«Алгоритмы и структуры данных»

**отчет по блоку задач №1 (Яндекс.Контест)**

**Выполнил:**

Студент группы P3210

Федоров Евгений Константинович

**Преподаватель:**

Тараканов Денис Сергеевич

Санкт-Петербург, 2025г.

**Задача №1 «A. Агроном-любитель»**

**Пояснение к примененному алгоритму:**

Данный алгоритм находит первый максимальный участок без трех повторяющихся цветков (одинаковых чисел).

Есть переменные startIndex – для сохранения начала текущего отрезка, bestIndex – стартовый индекс наибольшего отрезка, maxLength – длина наибольшего отрезка, currentLength – длина текущего отрезка.   
  
На каждой итерации сравнивается текущий и два предыдущих цветка, если они одинаковые, фиксируем стартовый индекс по середине, увеличивая текущих счетчик на 2 (так как последние два цветка из трех повторяющихся удовлетворяют условиям).   
Так же фиксируем длину максимального отрезка при помощи maxLength. Если отрезок становится наибольшим, его стартовый индекс так же фиксируется в bestIndex, это гарантирует, что будет выведен первый максимальный отрезок.

В конце остается прибавить к лучшему стартовому индексу максимальную длину, получив его конечный индекс.

**Алгоритмическая сложность:**

* **По времени**: Алгоритм обрабатывает каждый цветок ровно один раз, что даёт линейную сложность **O(n).**
* **По памяти**: Используется фиксированное (известное) количество переменных, поэтому сложность по памяти **O(1).**

Код:

|  |
| --- |
| #include <iostream>  #include <vector>  using namespace std;  int main() {  int n, startIndex = 0, currentLength = 0, maxLength = 1, bestIndex = 0;  cin >> n;  vector<int> flowers(n);  for (int i = 0; i < n; i++) {  cin >> flowers[i];  }  for (size\_t i = 0; i < flowers.size(); i++) {  if (i >= 2 && flowers[i - 2] == flowers[i] && flowers[i - 1] == flowers[i]) {  startIndex = i - 1;  currentLength = 2;  } else {  currentLength++;  if (currentLength > maxLength) {  maxLength = currentLength;  bestIndex = startIndex;  }  }  }  cout << bestIndex + 1 << " " << bestIndex + maxLength << endl;  return 0;  } |

**Задача №2 «B. Зоопарк Глеба»**

**Пояснение к примененному алгоритму:**

В решении задачи присутствует несколько структур данных. Стек гарантирует, что для каждого из животных найдется ловушка без пересечений, потому что взаимодействовать можно лишь с его верхушкой. Для каждого из животных ведется счетчик – его индекс относительно других животных, так же фиксируется каждый «вид» животного с индексами его экземпляров относительно других животных. Если видим ловушку, добавляем ее в вектор ловушек для дальнейшего сопоставления ловушек с животными. Параллельно смотрим на верхушку стека с ловушками/животными, если видим на верхушке нужную ловушку или животное, снимаем верхушку. После проверяем стек на наличие элементов, если он пустой, то все животные правильно добрались до ловушек без пересечений иначе Impossible. В случае Possible сопоставляем каждую ловушку в векторе с нужным видом животного, забирая с верхушки стека индексов позицию животного.

**Алгоритмическая сложность:**

1. **По времени:** Алгоритм проходит по строке один раз, выполняя константные операции для каждого символа. Таким образом, временная сложность составляет O(n), где n — длина строки
2. **По памяти:** Используются стек и вектор, которые в худшем случае могут содержать до n элементов. В случае с мапой все зависит от количества уникальных букв (26).Таким образом, пространственная сложность также составляет O(n).

Код:

|  |
| --- |
| #include <iostream>  #include <map>  #include <stack>  #include <vector>  using namespace std;  int main() {  stack<char> letterStack;  string input;  cin >> input;  map<char, deque<int>> animalPosition;  vector<char> traps;  int lettersCounter = 0;  for (size\_t i = 0; i < input.length(); i++) {  if (letterStack.empty()) {  letterStack.push(input[i]);  if (islower(input[i])) {  lettersCounter++;  animalPosition[input[i]].push\_back(lettersCounter);  } else {  traps.push\_back(input[i]);  }  } else {  if (islower(input[i])) {  lettersCounter++;  animalPosition[input[i]].push\_back(lettersCounter);  if (isupper(letterStack.top()) && tolower(letterStack.top()) == input[i]) {  letterStack.pop();  } else {  letterStack.push(input[i]);  }  } else {  traps.push\_back(input[i]);  if (islower(letterStack.top()) && toupper(letterStack.top()) == input[i]) {  letterStack.pop();  } else {  letterStack.push(input[i]);  }  }  }  }  if (letterStack.empty()) {  cout << "Possible" << endl;  for (char e : traps) {  int index = animalPosition[tolower(e)].back();  animalPosition[tolower(e)].pop\_back();  cout << index << " ";  }  } else {  cout << "Impossible" << endl;  }  return 0;  } |

**Задача №3 «C. Конфигурационный файл»**

**Пояснение к примененному алгоритму:**

Используем map<string, stack<string>> inputs для хранения значения переменных на каждом уровне вложенности и stack<stack<string>> для хранения самих переменных на каждом уровне. Добавляем переменные в соответствующие структуры, в inputs добавляем значения для каждой переменной, в стек currentChars добавляем сами переменные. Если введенная переменная (если она не число) после оператора присваивания не найдена в inputs, переменной присваивается 0, иначе переменной присваивается значение и добавляется в соответствующие структуры. Если после оператора присваивания стоит число, оно так же присваивается переменной. Как только встречается “{“ , мы уходим на уровень «глубже» (добавляя на стек еще один стек, который и является хранилищем для нового уровня вложенности), если видим “{“, удаляем последнее значение переменной в мапе по ключу из стека текущего уровня (currentChars).

**Алгоритмическая сложность:**

1. **По времени:** В худшем случае O(n), где n – количество строк. Операции открытия нового блока по времени составляет O(1), а закрытие O(n) в худшем случае, опрерации присваивания значения переменной так же проходит за O(1).
2. **По памяти:** В худшем случае так же получаем O(n), так как хранение в мапе – O(n), в стеке с большой вложенностью O(n).

Код:

|  |
| --- |
| #include <iostream>  #include <map>  #include <stack>  using namespace std;  bool isNumber(string input) {  if (input.empty())  return false;  if (input == "-0" || input == "0")  return true;  size\_t start = (input[0] == '-') ? 1 : 0;  return start < input.size() && input.find\_first\_not\_of("0123456789", start) == string::npos;  }  int main() {  string input;  map<string, stack<string>> inputs;  stack<stack<string>> currentChars;  currentChars.push(stack<string>());  while (getline(cin, input)) {  if (input == "{") {  currentChars.push(stack<string>());  } else if (input == "}") {  if (!currentChars.empty()) {  while (!currentChars.top().empty()) {  inputs[currentChars.top().top()].pop();  currentChars.top().pop();  }  currentChars.pop();  }  } else {  size\_t equalPos = input.find("=");  string varName = input.substr(0, equalPos);  string value = input.substr(equalPos + 1);  string result;  if (isNumber(value)) {  result = value;  } else {  if (!inputs.count(value) || inputs[value].empty()) {  result = "0";  } else {  result = inputs[value].top();  }  cout << result << endl;  }  currentChars.top().push(varName);  inputs[varName].push(result);  }  }  return 0;  } |

**Задача №4 «D. Профессор Хаос»**

**Пояснение к примененному алгоритму:**

Основная идея – рассмотрение нескольких возможных исходов в задаче. Если мы хотя бы раз превысили значение d, то меньше оно не станет, соответственно не имеет смысла смотреть на результат следующих итераций. Если число бактерий меньше c, следующие итерации так же не имеют смысла (по условию задачи используются все бактерии). Так же проверяем, что количество бактерий меняется от итерации к итерации (a!=buffer), иначе выходим.

**Алгоритмическая сложность:**

1. **По времени:** В худшем случае выполнится за O(k), где k – количество дней, если не будет выполнено ни одно из терминальных условий (превысило d, количество не будет меняться от итерации к итерации).
2. **По памяти:** Программа использует фиксированное количество переменных (a, b, c, d, k), которые не зависят от размера входных данных. Таким образом, сложность по памяти составляет O(1).

Код:

|  |
| --- |
| #include <iostream>  #include <vector>  using namespace std;  int main() {  vector<int> params(5);  for (size\_t i = 0; i < params.size(); ++i) {  cin >> params[i];  }  int a = params[0], b = params[1], c = params[2], d = params[3], k = params[4];  int buffer = 0;  for (int i = 0; i < k; ++i) {  buffer = a \* b - c;  if (buffer < 0) {  a = 0;  break;  }  if (buffer == a) {  break;  }  if (buffer < d) {  a = buffer;  } else {  a = d;  break;  }  }  cout << a;  return 0;  } |