Федеральное государственное автономное образовательное учреждение высшего образования Национальный исследовательский университет ИТМО

Факультет программной инженерии и компьютерной техники

Алгоритмы и структуры данных

Задачи A, B, C, D (Яндекс.Контест)

Выполнил: студент группы P3208, Васильев Н. А.

Преподаватель: Косяков М. С.

Санкт-Петербург 2025

# Задача A. Агроном-любитель

Этот алгоритм работает за **O(n)**, так как проходит по массиву всего один раз. Каждое изменение curFirst сдвигает левую границу, но не приводит к пересмотру уже обработанных элементов.

Дополнительная память **O(n)** используется только для хранения массива a, что неизбежно, так как входные данные передаются в массив.

При n <= 2 ответ очевиден.

В любом другом случае:

* Если встречаются три одинаковых подряд, начальная граница корректно сдвигается.
* Всегда поддерживается максимальная возможная длина отрезка, обновляя first и last, если найден лучший вариант.
* Так как проверяется только текущая тройка соседних элементов, то всегда гарантируется, что отрезок удовлетворяет условиям.

## Код:

#include <cstdint>

#include <iostream>

#include <vector>

using namespace std;

int main() {

  uint64\_t n;

  cin >> n;

  vector<int64\_t> a(n);

  for (uint64\_t i = 0; i < n; i++) {

    cin >> a[i];

  }

  if (n <= 2) {

    cout << "1 " << n << "\n";

    return 0;

  }

  uint64\_t first = 0;

  uint64\_t last = 1;

  uint64\_t curFirst = 0;

  uint64\_t length = 2;

  if (a[0] == a[1] && a[1] == a[2]) {

    curFirst = 1;

    length = 2;

    first = 0;

    last = 2;

  } else {

    length = 3;

    last = 3;

  }

  for (uint64\_t curLast = 3; curLast < n; ++curLast) {

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

      curFirst = curLast - 1;

    }

    uint64\_t curLength = curLast - curFirst + 1;

    if (curLength > length) {

      length = curLength;

      first = curFirst;

      last = curLast + 1;

    }

  }

  cout << first + 1 << " " << last << "\n";

  return 0;

}

# Задача B. Зоопарк Глеба

Проход по строке **O(n)**, так как каждый символ добавляется или удаляется из стэка один раз.

Дополнительная память **O(n)**, так как используются три стэка и массив индексов.

Каждое животное должно найти свою ловушку, причем порядок должен сохраняться без пересечений. Использование стэков animals и traps позволяет отлавливать ситуации, когда животное и ловушка встречаются в таком порядке, при котором их невозможно отловить без пересечений. Коме того, хранение индексов в animals и traps помогает проще и быстрее определять, какое животное займет конкретную ловушку. Если после обработки строки chars не пуст, значит, некоторые животные или ловушки не смогли быть сопоставлены корректно.

Код:

#include <cctype>

#include <iostream>

#include <stack>

#include <vector>

using namespace std;

int main() {

  string input;

  cin >> input;

  int n = input.size();

  if (n % 2 != 0 || n == 0) {

    cout << "Impossible" << endl;

    return 0;

  }

  stack<pair<char, int>> animals;

  stack<pair<char, int>> traps;

  stack<char> chars;

  int animalIndex = 0;

  int trapIndex = 0;

  vector<int> indexes(n / 2, 0);

  for (int i = 0; i < n; i++) {

    char current = input[i];

    if (!isalpha(current)) {

      cout << "Impossible" << endl;

      return 0;

    }

    if (islower(current)) {

      animalIndex++;

      animals.push({current, animalIndex});

    } else {

      trapIndex++;

      traps.push({current, trapIndex});

    }

    if (chars.empty() || current == chars.top()) {

      chars.push(current);

    } else if (!chars.empty() && tolower(current) == tolower(chars.top()) && !traps.empty() &&

               !animals.empty()) {

      indexes[traps.top().second - 1] = animals.top().second;

      animals.pop();

      traps.pop();

      chars.pop();

    } else {

      chars.push(current);

    }

  }

  if (chars.empty()) {

    cout << "Possible" << endl;

    for (int ind : indexes) {

      cout << ind << " ";

    }

    cout << endl;

  } else {

    cout << "Impossible" << endl;

  }

  return 0;

}

# Задача C. Конфигурационный файл

В худшем случае (O(n) изменений в каждом блоке, вложенных O(n) уровней), сложность удаления данных из blocks может дать O(n²). Однако на практике blocks будет содержать в среднем O(log n) вложенных уровней, что делает удаление значений приближенным к **O(n log n)**.

Основная идея алгоритма – использовать стекоподобную структуру хранения значений переменных для обеспечения их временного изменения в рамках блоков. blocks – вектор, где каждый элемент соответствует уровню вложенности и содержит список переменных, изменённых в данном блоке. Это гарантирует, что при закрытии блока все изменения, сделанные внутри него, откатываются. Если стек значений для переменной пустеет, переменная удаляется из values, что гарантирует уменьшение потребляемой памяти.

Код:

#include <stdint.h>

#include <iostream>

#include <unordered\_map>

#include <vector>

using namespace std;

int main() {

  unordered\_map<string, vector<pair<int, int>>> values;

  int currentBlock = 0;

  vector<vector<string>> blocks;

  string input;

  const string openBlock = "{";

  const string closeBlock = "}";

  while (cin >> input) {

    if (input == openBlock) {

      currentBlock++;

      blocks.emplace\_back();

    } else if (input == closeBlock) {

      if (!blocks.empty()) {

        for (const string& var : blocks.back()) {

          values[var].pop\_back();

          if (values[var].empty()) {

            values.erase(var);

          }

        }

        blocks.pop\_back();

      }

      currentBlock = max(0, currentBlock - 1);

      ;

    } else {

      size\_t pos = input.find('=');

      pair varVal = {input.substr(0, pos), input.substr(pos + 1)};

      if (isdigit(input[pos + 1]) || (input[pos + 1] == '-' && isdigit(input[pos + 2]))) {

        values[varVal.first].emplace\_back(currentBlock, stoi(varVal.second));

        if (!blocks.empty()) {

          blocks.back().push\_back(varVal.first);

        }

      } else {

        int val = 0;

        if (values.count(varVal.second) && !values[varVal.second].empty()) {

          val = values[varVal.second].back().second;

        }

        cout << val << endl;

        values[varVal.first].emplace\_back(currentBlock, val);

        if (!blocks.empty()) {

          blocks.back().push\_back(varVal.first);

        }

      }

    }

  }

  return 0;

}

# Задача D. Профессор Хаос

Используется фиксированное количество переменных (a, b, c, d, k, last), поэтому **O(1)** памяти. Сложность алгоритма в худшем случае составляет **O(k)**, если не срабатывает преждевременный выход.

Использование цикла for гарантирует, что обработка длится максимум k итераций, а преждевременные проверки (a <= 0 и a == last) позволяют оптимизировать выполнение, если число бактерий стабилизируется или обнуляется раньше. Операции min/max позволяют эффективно учесть условия эксперимента (ограничения на удаление c и лимит d), а их выполнение за O(1) делает алгоритм быстрым.

Код:

#include <cstdint>

#include <iostream>

#include <sstream>

#include <string>

using namespace std;

int main() {

  int64\_t a, b, c, d, k = 0;

  cin >> a >> b >> c >> d >> k;

  int64\_t last = a;

  for (int day = 1; day <= k; day++) {

    a = a \* b;

    a = max(a - c, (int64\_t)0);

    a = min(a, d);

    if (a <= 0) {

      a = 0;

      break;

    }

    if (a == last) {

      break;

    }

    last = a;

  }

  cout << a << endl;

  return 0;

}