Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнила:

Студентка группы P3215

Павличенко Софья Алексеевна

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2025

Задача A «Агроном-любитель»

Код:

```cpp
#include <iostream>
#include <vector>

using namespace std;

int main() {
  int n;
  cin >> n;
  vector<int> a(n);
  for (int i = 0; i < n; ++i)
    cin >> a[i];

  int i = 0, j = 1, t = 1, l = 0, r = 0;
  while (i < n) {
    if (j == n || (t == 2 && a[j - 1] == a[j])) {
      if (j - i > r - l) {
        l = i;
        r = j;
      }
      if (j - i < 3)
        t--;
      i++;
    } else {
      if (a[j - 1] == a[j])
        t++;
      else
        t = 1;
      j++;
    }
  }
  cout << l + 1 << " " << r;
  return 0;
}

```

Пояснение к примененному алгоритму:

Используем два указателя: i – левая граница, j – правая. j двигается вправо, пока это возможно, но, если подряд встречаются три одинаковых элемента, сдвигаем i. Отрезок рассматривается в полуинтервале [i, j).

Таким образом, каждый элемент обрабатывается не более двух раз: j расширяет отрезок, а i корректирует его при необходимости. Это обеспечивает эффективность **O(n)**. По памяти алгоритм использует **O(1)**, так как хранит только несколько целочисленных переменных для границ отрезка и счетчика подряд идущих элементов. В итоге находим самый длинный корректный подотрезок и выводим его границы.

Задача B «Зоопарк Глеба»

Код:

```cpp
#include <iostream>
#include <stack>
#include <vector>

using namespace std;

int main() {
  string s;
  cin >> s;
  int n = s.size(), a = 0, t = 0;
  stack<pair<char, int>> st;
  vector<int> res(n / 2);
  for (int i = 0; i < n; ++i) {
    if (!st.empty() && isupper(s[i]) && st.top().first == tolower(s[i])) {
      res[t++] = st.top().second;
      st.pop();
    } else if (!st.empty() && islower(s[i]) && st.top().first == toupper(s[i])) {
      res[st.top().second - 1] = ++a;
      st.pop();
    } else {
      if (isupper(s[i]))
        st.push({s[i], ++t});
      else
        st.push({s[i], ++a});
    }
  }
  if (!st.empty()) {
    cout << "Impossible";
    return 0;
  }
  cout << "Possible\n";
  for (int c : res)
    cout << c << " ";
  return 0;
}
```

Пояснение к примененному алгоритму:

Используем стек для отслеживания соответствий животных и их ловушек. Проходим по строке:

- Если текущий символ — ловушка и на вершине стека есть соответствующее животное, фиксируем совпадение и удаляем животное из стека.
- Если текущий символ — животное и на вершине стека находится его ловушка, также фиксируем совпадение и удаляем ловушку.

В остальных случаях добавляем символ с его позицией в стек.

В конце проверяем стек:

- Если стек пуст, все животные могут попасть в свои ловушки, и мы выводим "Possible" с порядком захвата.
- Если стек не пуст, значит, есть несовместимые пары — выводим "Impossible".

Каждый символ обрабатывается не более двух раз: добавляется в стек и, при нахождении пары, удаляется. Это обеспечивает эффективность **O(n)**.

По памяти алгоритм использует **O(n)** для хранения стека и массива с результатами.

Задача C «Конфигурационный файл»

Код:

```cpp
#include <iostream>
#include <stack>
#include <unordered_map>
#include <vector>

using namespace std;

int main() {
  string s;
  unordered_map<string, stack<int>> vars;
  stack<vector<string>> changed;
  changed.push({});
  while (getline(cin, s)) {
    if (s[0] == '{') {
      changed.push({});
    } else if (s[0] == '}') {
      for (const string& var : changed.top())
        vars[var].pop();
      changed.pop();
    } else {
      string var1;
      string var2;
      size_t i = 0;
      while (s[i] != '=') {
        var1 += s[i];
        i++;
      }
      i++;
      for (; i < s.size(); ++i)
        var2 += s[i];
      bool is_number = true;
      int n_var2 = 0;
      i = (var2[0] == '-');
      for (; i < var2.size(); ++i) {
        if (!isdigit(var2[i])) {
          is_number = false;
          break;
        }
        n_var2 = n_var2 * 10 + (var2[i] - '0');
      }
      if (var2[0] == '-')
        n_var2 = -n_var2;
      if (is_number) {
        vars[var1].push(n_var2);
      } else {
        vars[var1].push(vars[var2].empty() ? 0 : vars[var2].top());
        cout << vars[var1].top() << endl;
      }
      changed.top().push_back(var1);
    }
  }
  return 0;
}
```

Пояснение к примененному алгоритму:

Используем два стека:

- vars — хранит для каждой переменной стек её значений.
- changed — хранит стек с векторами, где каждый вектор содержит переменные, изменённые в текущем блоке.

При встрече **{** добавляем новый пустой вектор в changed.
При **}** восстанавливаем все переменные, изменённые в текущем блоке, удаляя вершины их стеков.
Для строки вида **var1=number** добавляем значение в стек переменной var1 и фиксируем её изменение.
Для строки **var1=var2** копируем текущее значение var2 (или 0, если var2 не определена) в var1, выводим это значение и фиксируем изменение.

Каждая строка обрабатывается один раз, операции со стеком занимают O(1), что обеспечивает **O(n)** по времени.
Память **O(n)** используется для хранения значений переменных и истории изменений.

Задача D «Профессор Хаос»

Код:

```cpp
#include <iostream>

using namespace std;

int main() {
  int a, b, c, d, k;
  cin >> a >> b >> c >> d >> k;
  for (int i = 0; i < k && a > 0; ++i) {
    int n = min(max(a * b - c, 0), d);
    if (a == n)
      break;
    a = n;
  }
  cout << a;
  return 0;
}

```

Пояснение к примененному алгоритму:

Итеративно моделируем процесс деления и уничтожения бактерий на протяжении k дней.

Каждый день:

1. Увеличиваем количество бактерий в b раз.
2. Уничтожаем c бактерий (или все, если их меньше c).
3. Если бактерий осталось больше d, сохраняем только d из них.

Если после любого дня количество бактерий не изменилось, эксперимент прекращается досрочно.

Временная сложность **O(k)** — мы симулируем каждый день.
Память **O(1)** — используем только несколько переменных для хранения текущего состояния.