Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнил:

Студент группы P3216

Брагин Р.А.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2025

Задача №A «Агроном-любитель»

```cpp
#include <iostream>
#include <map>
#include <vector>
using namespace std;

pair<int, int> findMaxUniqueSequence(const vector<int>& flowers, int n) {
 int left = 0;
 pair<int, int> result = {0, 0};
 int mx_len = 0;

 if (n < 3) {
  return {0, n - 1};
 }

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

  if (right >= 2 && flowers[right] == flowers[right - 1] &&
      flowers[right] == flowers[right - 2]) {
    left = right - 1;
  }

  if (right - left + 1 > mx_len) {
    mx_len = right - left + 1;
    result = {left, right};
  }
 }

 return result;
}

int main() {
 int n;
 cin >> n;

 vector<int> flowers(n);
```

```
 for (int i = 0; i < n; i++) {
   cin >> flowers[i];
 }
 pair<int, int> result = findMaxUniqueSequence(flowers, n);

 cout << result.first + 1 << " " << result.second + 1 << endl;

 return 0;
}
```

Пояснение к примененному алгоритму:

Алгоритм использует два указателя: левый отмечает начало подпоследовательности, а правый проходит по массиву. Если встречаются три одинаковых элемента подряд, левый указатель сдвигается, чтобы исключить их. На каждом шаге длина текущей подпоследовательности сравнивается с максимальной, и если она больше, обновляются границы результата. В итоге возвращаются индексы самой длинной подпоследовательности без трёх повторений. Работает за O(n).

Задача №B «Зоопарк Глеб»

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

using namespace std;

bool matchTrapsAndAnimals(const string& str, vector<int>& matches) {
 stack<char> chars;
 stack<int> animals;
 stack<int> traps;
 int animalCount = 0;

 for (size_t i = 0; i < str.size(); ++i) {
   if (islower(str[i])) {
     animalCount++;
```

```
    animals.push(animalCount);
  } else {
    traps.push(i - animalCount);
  }

  if (chars.empty() || str[i] == chars.top()) {
    chars.push(str[i]);
  } else if (tolower(str[i]) == tolower(chars.top())) {
    matches[traps.top()] = animals.top();
    traps.pop();
    animals.pop();
    chars.pop();
  } else {
    chars.push(str[i]);
  }
 }
 return chars.empty();
}

int main() {
 string input;
 cin >> input;

 vector<int> matches(input.size() / 2);
 bool isPossible = matchTrapsAndAnimals(input, matches);

 if (isPossible) {
  cout << "Possible" << endl;
  for (size_t i = 0; i < matches.size(); ++i) {
    cout << matches[i] << " ";
  }
  cout << endl;
 } else {
  cout << "Impossible" << endl;
 }
 return 0;
```

Пояснение к примененному алгоритму:

Алгоритм использует три стека: char для символов, animal для номеров животных и traps для индексов ловушек. Проходим по строке, для каждого символа проверяем, является ли он животным или ловушкой, и добавляем в соответствующий стек. Если текущий символ и верхний элемент стека chars — одна и та же буква в разных регистрах, сопоставляем ловушку с животным, записываем результат в вектор matches и удаляем элементы из стеков. Если символы не совпадают, просто добавляем текущий символ в стек. В конце проверяем, пуст ли стек char`: если да, выводим "Possible" и порядок сопоставления, иначе — "Impossible". Алгоритм работает за O(n).

Задача №C «Конфигурационный файл»

```
#include <iostream>
#include <stack>
#include <string>
#include <unordered_map>

using namespace std;

void assignVariable(
  const string& line,
  unordered_map<string, int>& variables,
  stack<pair<string, int>>& changes_stack
) {
 size_t splitter = line.find('=');
 string variable = line.substr(0, splitter);
 string value = line.substr(splitter + 1);

 if (isdigit(value[0]) || (value[0] == '-' && isdigit(value[1]))) {
  if (variables.find(variable) != variables.end()) {
   changes_stack.push({variable, variables[variable]});
  } else {
   changes_stack.push({variable, -1});
  }
  variables[variable] = stoi(value);
 } else {
  if (variables.find(variable) != variables.end()) {
```

```
    changes_stack.push({variable, variables[variable]});
  } else {
    changes_stack.push({variable, -1});
  }
  variables[variable] = variables[value];
  cout << variables[value] << endl;
 }
}

int main() {
 unordered_map<string, int> variables;
 stack<pair<string, int>> changes_stack;

 string line;
 while (getline(cin, line)) {
  if (line == "{") {
    changes_stack.push({"", 0});
  } else if (line == "}") {
    while (!changes_stack.empty() && changes_stack.top().first != "") {
      auto [var, value] = changes_stack.top();
      if (value == -1) {
        variables.erase(var);
      } else {
        variables[var] = value;
      }
      changes_stack.pop();
    }
    if (!changes_stack.empty()) {
      changes_stack.pop();
    }
  } else {
    assignVariable(line, variables, changes_stack);
  }
 }

 return 0;
```

```
}
```

Пояснение к примененному алгоритму:

Алгоритмы читает строки и обрабатывает их в зависимости от содержимого. Если строка — это присваивание (например, `x=10` или `y=x`), он смотрит, является ли значение числом или другой переменной. Если это число, он сохраняет текущее значение переменной в стеке (если оно было) и обновляет её в словаре. Если это другая переменная, он берёт её значение из словаря, обновляет текущую переменную и выводит это значение на экран. Если встречается `{`, это начало блока, и в стек добавляется специальная метка. Если встречается `}`, это конец блока, и код откатывает все изменения, сделанные внутри этого блока, восстанавливая значения переменных до состояния до блока. Это позволяет работать с вложенными блоками и корректно управлять изменениями переменных. Алгоритм работает за O(N * n), где N — количество строк, а n — средняя длина строки.

Задача №D «Профессор Хаос»

```
#include <iostream>
using namespace std;
int getBactOnKDay(int a, int b, int c, int d, int k) {
 int curr_day = 0;
 int previous = a;
 while (curr_day < k) {
  if (curr_day > 1 && previous == a) {
    return a;
  }
  previous = a;
  a *= b;
  if (a - c > 0) {
    a -= c;
```

```
  } else {
    return 0;
  }
  if (a >= d) {
    a = d;
  }
  curr_day++;
 }
 return a;
}

int main() {
 int a, b, c, d, k;
 cin >> a >> b >> c >> d >> k;
 int res = getBactOnKDay(a, b, c, d, k);
 cout << res;
}
```

Пояснение к примененному алгоритму:

Код моделирует рост бактерий по дням. На входе дается начальное число бактерий (a), коэффициент размножения (b), число погибших бактерий каждый день (c), максимальная вместимость среды (d) и день, на который нужно узнать количество бактерий (k). В цикле, пока не достигнут день (k), количество бактерий умножается на b, затем вычитается (c). Если бактерий становится меньше нуля, возвращается 0 (все погибли). Если бактерий больше, чем (d), их число ограничивается до (d). Если количество бактерий не меняется два дня подряд, возвращается текущее значение (популяция стабилизировалась). В конце выводится количество бактерий на день k. Алгоритм работает за O(k).