Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по «Алгоритмам и структурам данных»

Базовые задачи

Выполнила:

Студентка группы P3230

Вавилина Е. А.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2024

Задача №1 «A. Агроном-любитель»

Пояснение к примененному алгоритму:

Будем последовательно проходить по каждому цветку и отслеживать три последних встреченных нами цветка.

В начале выполнения алгоритма начало последовательности – первый цветок, а максимум – последовательность длины 0. Далее к последовательности добавляется по 1 цветок.

Если встречаются три одинаковых подряд, текущая последовательность завершается, и ее длина сравнивается с наибольшей найденной ранее. Если текущая последовательность оказался длиннее, его границы сохраняются как новый максимум.

После обнаружения трёх одинаковых цветков, начало новой последовательности ставится на 2 из 3 одинаковых цветков.

Алгоритм продолжает обработку до конца последовательности. В конце выполнения проверяем длину последней последовательности (если она больше, чем максимальная, то обновим границы).

После этого наибольшая допустимая последовательность гарантированно найдена, т.к.

1. Или мы не встретили 3 идущих подряд одинаковых цветов и все наша последовательность максимальная
2. Или мы встретили 3 одинаковых цветка подряд, тогда хотя бы раз обновлялся максимум.

Оценки:

1. Основной цикл for проходит по каждому элементу ровно один раз, выполняя константное количество операций (считывание, проверки, обновления). Сложность O(n)

   Следовательно, общий порядок сложности **O(n)**, **линейная**.

2. Код использует фиксированное количество переменных:

   Целочисленные индексы – 5 переменных. Массив arr[3] для хранения последних трех элементов. Дополнительной памяти для хранения всех n элементов не требуется, так как массив arr всегда остаётся размером 3.

   Следовательно, сложность по памяти **константная**, **O(1)**

Код:

```
#include <array>
#include <cstddef>
#include <iostream>

int main() {
  size_t numb = 0;
```

```
  std::cin >> numb;

  size_t beg = 0;
  size_t end = 0;

  size_t max_beg = 0;
  size_t max_end = 0;

  std::array<int, 3> arr = {0, 0, 0};

  std::cin >> arr[0];

  if (numb >= 2) {
    std::cin >> arr[1];
    end = 1;
    max_end = 1;
  }

  for (size_t i = 2; i < numb; i++) {
    std::cin >> arr[2];
    if (arr[0] == arr[1] and arr[1] == arr[2]) {
      if (end - beg > max_end - max_beg) {
        max_end = end;
        max_beg = beg;
      }
      beg = end;
    }

    end += 1;
    arr[0] = arr[1];
    arr[1] = arr[2];
  }

  if (end - beg > max_end - max_beg) {
    max_end = end;
    max_beg = beg;
  }

  std::cout << max_beg + 1 << " " << max_end + 1 << '\n';
}
```

Задача №2 «B. Зоопарк Глеба»

Пояснение к примененному алгоритму:

Реализуем алгоритм, похожий на применяемый для проверки правильности скобочных последовательностей.

Будем проходить по последовательности животных и ловушек и записывать каждый встреченный элемент в стек по следующим правилам:

- Если текущая верхушка стека – это животное, а новый элемент – соответствующая ему ловушка, тогда убираем животное из стека. Аналогично, если верхушка – ловушка, а новый элемент – соответствующее животное, то убираем ловушку из стека.
- Если не образована пара – записываем элемент в стек.

Если после прохода по всем элементам стек оказался пуст – можно распределить всех животных по ловушкам.

Для вывода пути параллельно будем вести 2 счетчика (номер животного и номер ловушки) и стек, хранящий номера животных/ловушек. На каждом этапе выполним 1 из следующих действий:

- Если текущая верхушка стека – это животное, а новый элемент – соответствующая ему ловушка, тогда убираем животное из стека. и увеличиваем счетчик лоушек на 1. В путь записываем, что в ловушку с номером равным счетчику животных попало животное с номером, снятым со стека.
- Если верхушка – ловушка, а новый элемент – соответствующее животное, то убираем номер ловушки из стека и увеличиваем счетчик животных на 1. В путь записываем, что в ловушку с номером, снятым со стека, попало животное с номером равным счетчику животных
- Если встретили животное и пары нет – увеличиваем счетчик животных и кладем его в стек номеров.
- Если встретили ловушку и пары нет – увеличиваем счетчик ловушек и кладем его в стек номеров.

Оценки:

1. Основной цикл for проходит по каждому элементу ровно один раз, выполняя константное количество операций (проверки, pop, push). Сложность O(n).

   Следовательно, общий порядок сложности **O(n)**, **линейная**.

2. Код использует:
   - вектор длиной n/2, следовательно сложность по памяти O(n), линейная
   - стек длиной не более n для хранения списка животных и ловушек, следовательно сложность по памяти O(n), линейная
   - вектор длиной не более n для хранения порядковых номеров, следовательно сложность по памяти O(n), линейная

   Следовательно, сложность по памяти **линейная**, **O(n)**

Код:

```
#include <cctype>
#include <cstddef>
#include <iostream>
```

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

int main() {
  std::string str;
  std::cin >> str;

  int numb_trp = 0;
  int numb_ani = 0;

  std::vector<int> path(str.length() / 2);

  std::stack<char> stk_animals;
  std::stack<int> stk_numbers;

  for (size_t i = 0; i < str.length(); i++) {
    const char curr_letter = str[i];

    if (std::isupper(curr_letter) != 0) {
      numb_trp += 1;
    } else {
      numb_ani += 1;
    }

    if (!stk_animals.empty() && (std::tolower(stk_animals.top()) ==
std::tolower(curr_letter)) &&
        (stk_animals.top() != curr_letter)) {
      if (std::isupper(stk_animals.top()) != 0) {
        path[stk_numbers.top() - 1] = numb_ani;
      } else {
        path[numb_trp - 1] = stk_numbers.top();
      }

      stk_animals.pop();
      stk_numbers.pop();
      continue;
    }

    stk_animals.push(curr_letter);

    if (std::isupper(curr_letter) != 0) {
      stk_numbers.push(numb_trp);
    } else {
      stk_numbers.push(numb_ani);
    }
  }

  if (stk_animals.empty()) {
```

```
    std::cout << "Possible" << '\n';
    for (const int num : path) {
      std::cout << num << " ";
    }
  } else {
    std::cout << "Impossible" << '\n';
  }
}
```

Задача №3 «C. Конфигурационный файл»

Пояснение к примененному алгоритму:

Реализуем рекурсивный алгоритм.

На каждом шаге рекурсии будем идти по последовательности и действовать в зависимости от встреченного символа. Что бы обеспечить сохранение изменений внутри { } будем хранить все измененные на данном уровне { } переменные.

Возможные действия:

- Встретили { - переходим на новый уровень рекурсии, начинаем записывать дальнейшие изменения переменных
- Встретили строку: если это присвоение переменой числа – закидываем старое значение переменной в бекап, если еще не закидывали на данном уровне вложенности и присваиваем число. Если это присвоение переменой другой еременной – закидываем старое значение переменной в бекап, если еще не закидывали на данном уровне вложенности и присваиваем число из второй переменной. Затем добавляем записанное число в конец списка результатов.
- Встречаем } – возвращаем все значения переменных, сохраненные в бекап и поднимаемся на уровень рекурсии выше.

В конце получим последовательность чисел, которые присваивались через переменные.

Оценки:

1. Основной цикл while проходит по каждому элементу ровно один раз. Хотя рекурсия и присутствует, все данные передаются ссылками, а значит каждый while продолжается с того места, где был рекурсивный вызов/возврат из рекурсии.

   Поиск и вставка в unsorted map работают за линию, O(n).

   Бекап старых значений после цикла максимум восстанавливает n переменных, что делается с линейной сложностью, O(n).

   Следовательно, общий порядок сложности **O(n)**, **линейная**.

2. Код использует:
   - Массив строк размером n, следовательно сложность по памяти O(n), линейная

- Значения присвоенные из переменных – не более n штук, сложность по памяти O(n)
- На каждом этапе рекурсии создается мапа размером не более чем n символов. Глубина рекурсии не более чем n/2. Сложность $O(n^2)$

Следовательно, сложность по памяти **квадратичная**, **$O(n^2)$**

Код:

```
#include <cstddef>
#include <iostream>
#include <set>
#include <stdexcept>
#include <string>
#include <unordered_map>
#include <vector>

namespace {
bool TryToConvert(const std::string& str) {
  try {
    std::stoi(str);
    return true;
  } catch (const std::invalid_argument& e) {
    return false;
  }
}

std::vector<int> Funk(
    std::vector<std::string>& arr, std::unordered_map<std::string, int>& dict,
size_t& start
) {
  std::unordered_map<std::string, int> backup;
  std::vector<int> res;

  while (start < arr.size()) {
    if (arr[start] == "{") {
      start++;
      std::vector<int> res_inside = Funk(arr, dict, start);
      res.insert(res.end(), res_inside.begin(), res_inside.end());
      continue;
    }

    if (arr[start] == "}") {
      start++;
      break;
    }

    const size_t pos = arr[start].find('=');
    const std::string var = arr[start].substr(0, pos);
    const std::string value = arr[start].substr(pos + 1);
```

```
    if (backup.find(var) == backup.end()) {
      backup[var] = dict[var];
    }

    if (TryToConvert(value)) {
      dict[var] = std::stoi(value);
    } else {
      dict[var] = dict[value];
      res.push_back(dict[value]);
    }

    start++;
  }

  for (const auto& var : backup) {
    dict[var.first] = var.second;
  }
  return res;
}
}  // namespace

int main() {
  std::vector<std::string> arr;
  std::string current_srt;
  std::unordered_map<std::string, int> dict;
  size_t start = 0;

  while (std::cin >> current_srt) {
    arr.push_back(current_srt);
  }

  for (const int num : Funk(arr, dict, start)) {
    std::cout << num << '\n';
  }
}
```

Задача №4 «D. Профессор Хаос»

Пояснение к примененному алгоритму:
Будем проводить симуляцию по дням. Для каждого дня выполняются следующие шаги:

- Количество бактерий умножается на коэффициент деления.
- Из полученного результата вычитается число бактерий, уходящих на эксперименты. Если бактерий меньше, чем требуется для эксперимента, устанавливается нулевое количество.
- Далее применяется ограничение контейнера: если оставшихся бактерий больше, чем может вместить контейнер, выбирается максимум, равный вместимости.

- Если наступил последний (k-й) день и количество бактерий отлично от 0 – это искомый результат.
- Если число бактерий 0 – прерываем эксперимент. Искомый результат – 0.
- Если в начале и в конце дня одинаково число бактерий, то это наш искомый результат (т.к. мы попали в цикл и дальнейшие шаги будут давать нам тот же исход).
- Если еще не нашли нужное число – переходим к следующему дню

Оценки:

1. Основной цикл расчета в худшем случае выполняется k раз, по количеству дней эксперимента. Сложность **O(k), линейная**.
2. Программа использует:
    - Небольшой набор целочисленных переменных для хранения текущего количества бактерий, счётчика дней и результата.
    - Вектор фиксированного размера для хранения параметров эксперимента.

    Сложность по памяти **O(1)**

Код:

```
#include <algorithm>
#include <iostream>
#include <vector>

int main() {
  int a = 0;

  std::vector<int> val(4, 0);

  int now = 1;
  int result = 0;

  std::cin >> a;
  for (int i = 0; i < 4; i++) {
    std::cin >> val[i];
  }

  while (now <= val[3]) {
    const int a_start = a;

    a = a * val[0];

    a = (a >= val[1]) ? a - val[1] : 0;

    a = std::min(a, val[2]);

    if (now == val[3] and a != 0) {
      result = a;
```

```
      break;
    }

    if (a == 0) {
      result = 0;
      break;
    }

    if (a == a_start) {
      result = a;
      break;
    }

    now += 1;
  }

  std::cout << result;
}
```