Методы построения и анализа алгоритмов

Практические занятия

Введение

Каждое практическое занятие обычно включает в себя следующие пункты:

- реализация некоторых алгоритмов
- проверка корректности реализации алгоритмов
- исследование временной сложности реализованных алгоритмов (замер времени работы на входных данных разного размера)

Реализация алгоритмов

Если специально не указано в задании, для реализации алгоритмов можно использовать любой язык программирования.

Проверка корректности

Проверка корректности обычно выполняется в виде написания и запуска юнит-тестов. В большинстве заданий юнит-тесты даются уже в готовом виде. Для создания юнит-тестов существует много библиотек и фреймворков. Далее рассмотрим создание юнит-тестов для языка C++ с помощью библиотеки Catch (https://github.com/catchorg/Catch2).

Пример теста:

```
#define CATCH_CONFIG_MAIN

#include “catch.hpp”

#include <cstdlib>

TEST_CASE(“Abs function test”) {
     REQUIRE(abs(-2) == 2);
}
```

В примере показан тест для функции abs - получения модуля целого числа - из библиотеки stdlib. В первых двух строках подключается сама библиотека, находящаяся в одном файле catch.hpp (этот файл нужно скачать и поместить в директорию проекта, где его сможет найти компилятор). Определение CATCH_CONFIG_MAIN задействует встроенную функцию main библиотеки, создавать свою функцию main в этом случае **не нужно**. Каждый тест создается с помощью макроса TEST_CASE (таких тестов может быть много, на разные случаи). В параметре макроса указывается название теста, все названия должны быть разными. Каждый тест должен проверять одно или несколько утверждений, истинность которых означает прохождение теста. Для задания утверждений могут использоваться макросы REQUIRE, CHECK, CHECK_TRUE, CHECK_FALSE и т.д. (полный список макросов см. в документации к библиотеке). Все файлы с тестами и исходным кодом нужно собрать в одно приложение и запустить его.

По результатам запуска будет выведен отчет с прошедшими и проваленными тестами. Требуется прохождение всех тестов.

Пример юнит-тестов можно найти в каталоге test_example в репозитории курса.

Замечание: в среде разработки MS Visual Studio может возникать случай, когда окно консольного приложения закрывается сразу после его завершения, без возможности просмотреть вывод (например, результаты тестов). Для предотвращения этого нужно выбрать опцию `/SUBSYSTEM:CONSOLE` в свойствах проекта Компоновщик->Система ->Подсистема (Linker->System->Subsystem), приложение запускать без отладки (`Ctrl+F5`).

Замер времени

Для замера времени работы участка кода обычно используются функции работы со временем. Пример замера времени с использованием функции clock библиотеки time:

```
#include <ctime>
#include <cstdio>

int main() {
      int t1 = clock();
      // тут находится код, время работы которого нужно замерить
      int t2 = clock();
      float seconds = float(t2 - t1)/CLOCKS_PER_SEC;
      printf("Time: %.5f sec\n", seconds);
      return 0;
}
```

Функция clock может иметь низкую точность, а также давать неправильные результаты для многопоточного кода, т.к. замеряет процессорное время. Обычно нас интересует реально прошедшее время, которое заняло исполнение кода. Поэтому в качестве альтернативы рекомендуется использовать средства C++ библиотеки chrono:

```
#include <chrono>
#include <iostream>

int main() {
      auto t1 = std::chrono::high_resolution_clock::now();
      // тут находится код, время работы которого нужно замерить
      auto t2 = std::chrono::high_resolution_clock::now();
      auto seconds = std::chrono::duration<double>(t2-t1).count();
      std::cout << "Time: " << seconds << " sec." << std::endl;
      return 0;
}
```

Важно: для замера времени проект нужно компилировать в режиме с оптимизацией (конфигурация Release в MS Visual Studio; ключи `-O2` или `-O3` для компилятора GCC).

Замечание: для использование библиотеки chrono требуется поддержка компилятором стандарта C++11 минимум. Для включения такой поддержки могут потребоваться дополнительные опции, например `-std=c++11` для компилятора GCC.

Практическое занятие №1: Введение в алгоритмы

Задание:

- реализовать алгоритмы линейного и бинарного поиска на массиве целых чисел
- замерить время работы разработанных алгоритмов поиска на массивах разных размеров и сравнить между собой. Определить тип временной сложности для каждого алгоритма. Т.к. время поиска одного элемента в массиве может быть очень маленьким, предлагается совершать много (от сотен тысяч до миллионов) поисков произвольных чисел в массиве и брать суммарное или среднее время поиска в качестве результата
- реализовать алгоритм проверки отсутствия дубликатов в массиве целых чисел. Замерить время работы алгоритма на массивах разных размеров (заведомо содержащих или не содержащих дубликаты). Определить тип временной сложности разработанного алгоритма. При необходимости предложить вариант более эффективной реализации

Практическое занятие №2: Сортировка за линейное время

Задание:

- реализовать алгоритм сортировки подсчетом массива целых чисел. Исходники проекта находятся в каталоге count_sort репозитория
- запустить прилагающиеся юнит-тесты для проверки корректности реализации
- замерить время работы алгоритма на массивах целых чисел разного размера, сравнить со временем сортировки тех же массивов стандартным (библиотечным) алгоритмом сортировки (std::sort, qsort)

```
Псевдокод алгоритма сортировки подсчетом
count_sort(a, n, start, end):
    1. Создать дополнительный массив counts для значений
       элементов от start до end, изначально заполненный нулями
    2. Подсчитать число вхождений каждого значения путем
       прохода по входному массиву a, результаты сохранить в
       counts
    3. Зная число вхождений каждого значения во входном
       массиве, построить отсортированный массив
```

Практическое занятие №3: Бинарные деревья

Задание:

- реализовать алгоритм декодирования сообщения (функцию `decode`) с использованием кода Хаффмана. Проект находится в папке huffman репозитория. Объявления функций и вспомогательных структур данных находятся в файле `huffman.h`
- В коде Хаффмана каждому символу из некоторого алфавита сопоставляется последовательность из 0 и 1. Функции декодирования на вход подается строка из символов “0” или “1”. Функция должна вернуть расшифрованную строку из оригинальных символов. Для этого нужно использовать дерево Хаффмана, которое создается вызовом функции `buildTree`
- Дерево Хаффмана в задании - бинарное дерево из объектов класса `TreeNode`. Листья дерева содержат код оригинального символа в поле `symbol`. Для декодирования строки нужно, начиная с корня дерева, переходить в левое поддерево (поле `left`), если следующий символ в строке - “0”, или в правое поддерево (`right`), если “1”, пока не будет достигнут лист дерева и получен оригинальный символ. После чего процедура возобновляется с корня дерева, пока вся закодированная строка не будет исчерпана
- Для тестирования реализации использовать данные из файла `data.txt`: каждая строка в нем - отдельное закодированное сообщение

Практическое занятие №4: Графы

Задание:

- реализовать алгоритм проверки наличия пути между парой вершин в ориентированном графе (функция `path_exists` из файла `path.cpp`). Исходный код проекта находится в каталоге graph репозитория
- тип данных “ориентированный граф” уже реализован в виде класса `Graph` (файлы `graph.h` и `graph.cpp`, тесты и примеры использования - `graph_test.cpp`). Вершины графа обозначаются целыми числами. Методы класса, которые могут пригодиться в работе:
    - `std::vector<int> get_vertices()` - возвращает список вершин графа в виде массива
    - `std::vector<int> get_adjacent_vertices(int vertex)` - возвращает список вершин графа, соединенных с вершиной `vertex` исходящими из нее дугами, в виде массива
    - `bool has_vertex(int vertex)` - возвращает `true`, если граф содержит вершину `vertex`, иначе `false`
    - `bool has_arc(int v1, int v2)` - возвращает `true`, если граф содержит дугу `(v1, v2)`, иначе `false`. Допускается существование дуг из вершины в саму себя
    - `void add_vertex(int vertex)` - добавляет вершину `vertex` в граф
    - `void add_arc(int v1, int v2)` - добавляет дугу `(v1, v2)` в граф, а также и сами вершины если они отсутствуют
- функция path_exists должна возвращать `true`, если конечная вершина достижима в графе из стартовой, и `false`, если вершина не достижима или одна или обе не принадлежат графу
- проверить корректность реализации с помощью прилагающихся тестов

- замерить время работы алгоритма на произвольных графах разного размера (для получения значительного времени проводить много поисков между парами разных вершин и брать суммарное или среднее время поиска). Определить тип временное сложности алгоритма

Практическое занятие №5: Метод полного перебора

Задание:

- реализовать алгоритм генерации всех возможных паролей для заданного алфавита и максимальной длины пароля (функцию `all_words`). Проект находится в папке bruteforce репозитория. Объявления функций находятся в файле `bruteforce.h`, реализация - в `brutefroce.cpp`
- входными параметрами функции `all_words` являются алфавит - строка символов - и максимальная длина пароля. Функция должна вернуть все возможные пароли (слова), которые можно составить из символов алфавита, длиной от 1 до максимальной включительно. Результат возвращается как массив (вектор) строк. Каждый уникальный пароль должен встречаться в массиве только один раз, порядок паролей в массиве не важен
- проверить корректность реализации с помощью прилагающихся тестов
- провести исследование времени работы функции подбора пароля `bruteforce` для разных размеров алфавита и максимальной длины пароля

Практическое занятие №6: Динамическое программирование

Задание:

- задача: даны достоинства монет и некоторая сумма. Требуется разменять указанную сумму монетами указанных достоинств, использовав при этом минимальное число монет. Монету любого достоинства можно использовать неограниченное число раз
- реализовать алгоритм решения задачи (функцию `get_change`), используя метод динамического программирования. Проект находится в папке change репозитория. Объявления функций находятся в файле `change.h`, реализация - в `change.cpp`.
- входными параметрами функции `get_change` являются массив достоинств монет `denominations` (целые числа) и сумма на размен `amount` (тоже целое число). Достоинства монет в массиве `denominations` могут располагаться в произвольном порядке. Функция должна вернуть результат в виде массива использованных монет (каждая монета обозначается ее достоинством; порядок монет не важен).
- проверить корректность реализации с помощью прилагающихся тестов
- провести исследование времени работы алгоритма для разных значений суммы

```
Решение задачи методом динамического программирования
Пусть d₁, d₂, … , dₙ - достоинства монет
```

```
Пусть F(n) - минимальное число монет для суммы n
Тогда:
F(0) = 0
F(n) = 1 + min_{i, di >= n}{F(n - d_i)}
Чтобы найти сами достоинства монет для решения, нужно взять те d_i,
на которых достигается минимум F(n - d_i) для соответствующих n.
Чтобы не вычислять заново одинаковые F(k), результаты нужно
кэшировать в таблице. Альтернативный способ - начать заполнять
таблицу последовательно: F(0), F(1), F(2), и так до F(amount).
```

Практическое занятие №7: Жадные алгоритмы

Задание:

- задача: дано множество занятий (активностей) a_i, для каждого занятия известно время его начала и окончания: $[s_i, f_i)$, $0 <= s_i < f_i$. Занятия a_i и a_j считаются совместимыми, если не пересекаются по времени: $s_j >= f_i$ или $s_i >= f_j$. Требуется найти максимальное по размеру множество взаимно совместимых занятий
- реализовать жадный алгоритм решения задачи (функцию `get_max_activities`). Проект находится в папке activities репозитория. Объявления функций находятся в файле `activities.h`, реализация - в `activities.cpp`.
- входным параметром функции `get_max_activities` является массив занятий (элементов типа данных `Activity`). Занятия в массиве могут располагаться в произвольном порядке. Функция должна вернуть результат - максимальное по размеру множество совместимых занятий - также в виде массива, порядок занятий в массиве не важен
- проверить корректность реализации с помощью прилагающихся тестов
- провести исследование времени работы алгоритма для входных массивов разного размера

```
Жадный алгоритм решения задачи
Пока множество занатий-кандидатов не исчерпано:
  - включить в результат занятие с минимальным временем окончания
    из рассматриваемых (a_k)
  - оставить для рассмотрения только те занятия, которые
    начинаются после выбранного (s_i >= f_k)
  - повторить
```

Практическое занятие №8

Задание будет выложено позже