Введение

(\*2 СЛАЙД) Актуальность данной курсовой работы объясняется тем, что временные затраты на анализ сложности различных алгоритмов могут быть легко исключены использованием программы, автоматизирующей данный процесс.

Будем рассматривать временную сложность алгоритмов, которая подсчитывается в количестве арифметических операций, сравнений, пересылок.

Временная сложность алгоритма обычно выражается с использованием нотации «O» большое (Big O).

(\*3-4 СЛАЙД) Существует формальное описание Big O, но обратим внимание на интуитивное. В рамках рассмотрения сложности алгоритмов Big O по своей сути позволяет разделять различные алгоритмы по классам сложности. К таким классам относятся те, что указаны на слайде.

(\*5 СЛАЙД) Задача курсовой работы состоит в разработке программы для автоматизированного оценивания сложности произвольного алгоритма с отображением графика его сложности. Результаты работы программы будут приведены в виде графиков сложности и наименований классов сложности в рамках нотации Big O.

Ввиду существования немалого числа классов сложности на задачу будет наложено следующее ограничение: результирующими классами сложности будут являться те, что указаны на слайде. Если алгоритм принадлежит иному классу сложности, ответ будет дан либо приближенно к одному из вышеуказанных, либо отмечен как **.**

(\*6 СЛАЙД) Научная новизна и уникальность работы заключается в том, что предлагается оригинальный способ программного анализа сложности различных алгоритмов на основе числа элементарных операций, которые они содержат.

(\*7 СЛАЙД) Теоретическая и практическая значимость работы характеризуется пользой, которую может привнести программа, позволяющая разработчикам автоматизированным способом определять сложность рассматриваемых алгоритмов, тем самым исключая излишние временные затраты на ручное определение сложности.

Algolite, трансляция в C++

(\*8 СЛАЙД) Для проведения анализа сложности произвольного алгоритма он должен быть записан в форме, приемлемой для проектируемой программы, в целях чего был разработан язык описания алгоритмов Algolite, представляющий собой небольшой набор команд из языка C++ с соответствующей семантикой и функционалом, достаточным для описания произвольного алгоритма. Пример функционала представлен на следующих слайдах в виде набора правил в форме Бэкуса-Наура (БНФ) (\*ПОКАЗ СЛАЙДОВ С БНФ 9-10).

(\*11-12 СЛАЙД) В качестве ведущего алгоритма синтаксического анализа будем использовать метод рекурсивного спуска, представляющий собой алгоритм нисходящего анализа*, реализуемый путём вызова функций разбора команд, проверяющих эти команды на соответствие установленному набору правил (например, БНФ). Эти функции принимают на вход объекты-токены, возвращаемые* лексическим анализатором, определяющим принадлежность рассматриваемого набора символов допустимому алфавиту

(\*13 СЛАЙД) Для промежуточной формы программы хорошо подходит древовидная структура, состоящая из узлов, которыми являются объекты внутреннего представления, имеющие ссылки друг на друга.

(\*14-15 СЛАЙД) Для проведения анализа синтаксического дерева необходимо организовать его полный проход, для чего хорошо подходит паттерн Посетитель (Visitor). Структура паттерна Visitor в нотации UML указана на слайде 14. *Посетитель представляет собой поведенческий паттерн проектирования, который позволяет добавлять в программу новые операции, не изменяя классы объектов, над которыми эти операции могут выполняться.* Паттерн Посетитель предлагает вынести поведение при анализе каждого такого узла в отдельный класс, примеры которых указаны на слайде 15.

(\*16 СЛАЙД) Так как целью курсовой работы является проведение анализа сложности транслируемого алгоритма, то в процесс генерации кода конечной программы можно внедрить автоматизированную расстановку команд вида counter += N, где N – число элементарных операций в некоторой встреченной конструкции (аналогично ручному методу). Анализ синтаксического дерева позволяет выявить последовательность генерируемых команд для конечной программы.

Анализ сложности алгоритма

(\*17-19 СЛАЙД) Одним из уникальных принятых решений в рамках данной курсовой работы оказалось введение алгоритма анализа сложности алгоритма на основе используемой в статистике линейной регрессии, представляющей собой регрессионную модель отображения зависимости некоторого числа переменных друг от друга.

В изначальной программе необходимо задать граничные значения, которые будут использоваться для определения числа выполняемых операций на конкретном значении параметра, и шаг, с которым это значение будет изменяться. Для наилучшего результата определения класса сложности необходимо задать большую область между граничными значениями параметра, *чтобы исчерпывающе проанализировать изменение функции в случае принадлежности к классу сложности логарифмического типа.*

*Метод линейной регрессии даёт возможность найти коэффициенты для уравнения прямой, которая наиболее точно соответствует найденным данным.* В общем случае мы имеем следующие типы классов сложности: степенные и логарифмические. Суть предложенного метода анализа алгоритма заключается в предположении, что его функция сложности при приведении к новым переменным принимает вид прямой (на слайдах 17-18 представлены замены для обоих случаев и возможный вид графиков функции сложности в новых переменных). После применения замен и формул для вычисления коэффициентов и (указаны на слайде 19) для набора пар значений параметра и числа выполняемых элементарных операций, необходимо вычислить среднюю ошибку для каждого из случаев. Результирующий выбор упадёт на тот тип класса сложности, чья ошибка показала меньшее значение. Следует отметить, что здесь – это степень x в исходной функции сложности .

Уточнение этапа программной реализации

(\*20 СЛАЙД) Пару слов о процессе программной реализации решения поставленной задачи. В качестве среды разработки была выбрана Microsoft Visual Studio Community 2019. Программа создавалась на языке программирования С++. Для автоматизированного отображения графика сложности в сформированную программу добавляется взаимодействие с библиотекой pbPlots. *Взаимодействие с ней заканчивается на помещении значений параметра и числа выполняемых элементарных операций в два различных вектора и построении графика на основе значений, которые они содержат. В процессе работы с некоторым количеством дробных чисел во время анализа сложности алгоритма значения на графиках, выводимых пользователю, могут не соответствовать действительности, что обусловлено внутренними особенностями работы библиотеки. Библиотека имеет внутренний механизм, подстраивающий график под указанные размеры, тем самым создавая на некоторых осях дробные значения даже в тех случаях, если в наборе точек, используемых для построения графика, таких значений нет.* Для компиляции и дальнейшего запуска исполняемого файла для сгенерированного cpp документа было введено использование отдельного компилятора g++. Инструкция по работе с разработанной программой и пример корректной программы на языке Algolite, описанной в соответствии с инструкцией, представлены на следующих слайдах (\*ПОКАЗ СЛАЙДОВ С ИНСТРУКЦИЕЙ 21-24). В результате работы будет получен график сложности алгоритма с определённым классом сложности.

Результаты исследований

Для анализа эффективности работы разработанной программы достаточно пронаблюдать достоверность вставки счётчиков выполняемых операций в необходимые места (при условии корректно выбранных значений для параметра сложности). Рассмотрим следующий набор алгоритмов: (\*ПОКАЗ СЛАЙДОВ С АЛГОРИТМАМИ 25-31).

Заключение

(\*32 СЛАЙД) В результате выполнения курсовой работы была создана программа, предоставляющая возможность оценивания сложности алгоритма, написанного на разработанном языке Algolite, транслируемого в команды языка C++, с дальнейшим построением графика сложности, описывающего зависимость числа элементарных операций от заданных значений параметра. Благодаря проведённым тестированиям функционала разработанной программы можно прийти к выводу, что данный процесс возможно автоматизировать, тем самым уменьшив временные затраты при внедрении рассматриваемого алгоритма в некоторую систему.

*Анализ сложности алгоритмов занимает у разработчиков большое число времени. Поэтому актуальность автоматизации данного процесса и поиска оптимальных методов его программной реализации неоспорима, а дальнейшее развитие инструментария поиска сложности алгоритмов для улучшения точности определения классов сложности можно охарактеризовать как довольно важное направление.*