Таким образом, из рассмотренных сортировок наибольшим потенциалом обладают *быстрая* и *интроспективная*. Это не случайно, поскольку интроспективная является быстрой сортировкой с контролем глубины рекурсии. Если глубина превышает установленный лимит, то быстрая сортировка останавливается и вызывается другой алгоритм, выполняющий досортировку. Выполним оптимизацию её работы по следующим направлениям: выбор лимита глубины рекурсии; уменьшение количества математических операций; выбор запасного алгоритма досортировки.

Лимит глубины рекурсии H можно установить по-разному. Автор алгоритма Дэвид Мюссер предложил ориентироваться на величину H=log(n) [1]. Почему предложен именно десятичный логарифм не совсем понятно, т.к. в процессе выполнения рекурсивно вызываются 2 функции для левого и правого подмножеств, поэтому получается бинарное дерево, глубина которого определяется как $H=\log_2(n)$. В этом случае, например, для массива n=1млн. H=20. Здесь следует осознать тот факт, что между n и H есть прямая связь. Чем больше n, тем больше H, следовательно, вычислять текущую глубину с помощью $\log_2(n)$ на каждом шаге не будем, так как эта операция не очень быстрая. Вместо этого заведём счётчик числа вызовов функции сортировки подмножества. Когда значение счётчика превысит объём массива, то будем считать, что произошло переполнение и вызовем второй алгоритм. Для увеличения значения счётчика используем операцию +, вместо инкремента inc(). Измерения проводились при n = 500 тыс.

Таблица 2 – Замер времени при оптимизации

	Начальная реализация	Переход на + вместо inc()	Переход от $\log_2$ к объёму	Динамическое измен. границы
Время, мс	64,449	64,377	63,402	52,218

Девид Мюссер в своей работе указал, что запасным алгоритмом следует использовать *пирамидальную* сортировку, т.к. она всегда сортирует за n*log(n). Однако в тесте выше было показано, что гребешковая и сортировка слиянием в среднем работают быстрее, поэтому проверим все 3 вида сортировок. Худший результат оказался именно у пирамидальной.

Таблица 3 – Результаты теста работы с запасным алгоритмом

	Быстрая без ограничения	Быстрая + пирамидальная	Быстрая + гребешковая	Быстрая + слиянием
Время, мс	10,452	23,471	14,223	18,099

Заключение. Проведённые измерения показывают, что наиболее быстродействующими алгоритмами без использования распараллеливания являются быстрая и интроспективная сортировки. В качестве запасного алгоритма для интроспективной выбираем гребешковую сортировку.

1. Описание алгоритмов сортировок [Электронный ресурс]: Коллекция сортировок / Валерий Макаров. – РФ, 2015. – Режим доступа: http://sorting.valemak.com. – Дата доступа: 15.12.2017.
2. Седжвик, Р. Фундаментальные алгоритмы на С. Ч. 1–4. Анализ. Структуры данных. Сортировка. Поиск [Текст] : Пер. с англ. / Роберт Седжвик. – СПб. : ООО «ДиаСофтЮП», 2003. – 672 с.
3. Точное время: измеряем, применяем [Электронный ресурс]: Социальное Интернет СМИ об IT / begin_end. – РФ, 2009. – Режим доступа: http://habrahabr.ru/post/75234/. – Дата доступа: 15.12.2017.
4. Musser, D. Introspective Sorting and Selection Algorithms / D. Musser // Software: Practice and Experience. – 1997. – August, volume 27, issue 8. – Pages 983–999.

О ИЗМЕРЕНИЯХ ВРЕМЕННЫХ ЗАТРАТ АЛГОРИТМОВ, РЕАЛИЗОВАННЫХ НА ЯЗЫКАХ ВЫСОКОГО УРОВНЯ

М.Г. Семёнов
Витебск, ВГУ имени П.М. Машерова

Сравнение производительности различных алгоритмов является важной задачей при изучении таких дисциплин, как «Методы алгоритмизации и программирования», «Алгоритмы и структуры данных», «Вычислительные методы», а также при разработке программного обеспечения. Одной из наиболее важных характеристик производительности являются

временная сложность, которая характеризует скорость работы. Классически для оценки временной сложности применяется асимптотический подход. Суть данного подхода заключается в асимптотической оценке скорости работы, как функции от размера входных данных. Однако, поведение алгоритмов, принадлежащих одному классу по сложности в асимптотической терминологии, могут иметь значительные различия в скорости для недостаточно больших объемов входных данных.

Альтернативами асимптотическому методу является подсчет числа элементарных операций (для итерационных алгоритмов возможен подсчет числа итераций). Стоит отметить, что в таком подходе определенную сложность вызывает выбор элементарной операции. Например, операции сложения и деления значительно отличаются необходимым для своего выполнения временем. В то же время, если при оценке временных затрат полностью пренебречь операциями, требующими относительно небольшое время, то, при большом количестве таких операций, возможна значительная погрешность. Кроме того, даже при одинаковом количестве элементарных операций языка высокого уровня, временные затраты могут отличаться в несколько раз за счет особенностей работы аппаратного обеспечения.

Наиболее общим методом является непосредственное измерение временных затрат, необходимых на работу конкретной реализации некоторого алгоритма. Стоит отметить, что, при реализации алгоритма на языке высокого уровня, во время преобразования исходного кода в машинный может произойти ряд не всегда очевидных оптимизаций, что может существенно повлиять на результаты измерений, по сравнению с рассчитанными теоретически. Особенностям измерения временных затрат в таких ситуациях и посвящена настоящая работа.

В настоящей работе в качестве технологий разработки применяется программная платформа .NET и язык программирования C#, однако несложно перенести основные результаты на другие языки высокого уровня.

Цель работы – проанализировать методические особенности измерения временных затрат алгоритмов, реализованных на языках высокого уровня, и спроектировать архитектуру приложения, которое будет производить такого рода измерения. Такое приложение позволит производить эффективное сравнение алгоритмов, как в исследовательских целях, так и в выявлении слабых сторон программных продуктов, разрабатываемых в рамках курсовых, дипломных и магистерских проектов с целью улучшения их производительности.

Материал и методы. В исследовании в качестве рабочего материала использовались различные источники: публикации IT-специалистов, видео-материалы, официальная документация и интернет-ресурсы. Применялись такие методы, как изучение и обобщение опыта работы специалистов, статистические методы, объектно-ориентированные методы разработки и аспектно-ориентированная парадигма.

Результаты и их обсуждение. При измерении временных затрат можно выделить следующие этапы: постановка задачи, выбор метрики и инструментов, проведение эксперимента, анализ результатов и выводы.

Одной из ключевых характеристик при выборе инструмента измерения является его разрешающая способность, т.е. минимальный интервал времени различимый инструментом. Зачастую при проведении измерений на программной платформе .NET применяются классы System.DateTime (метод UtcNow()) и System.Diagnostics.Stopwatch (метод GetTimeStamp()). Абсолютные значения разрешающих способностей данных инструментов зависят от аппаратного и программного обеспечения, однако на большинстве систем Stopwatch.GetTimeStamp() значительно опережает [2] своего конкурента.

Для этапа проведения эксперимента отметим следующие особенности:

измерения для различных алгоритмов и исходных данных, результаты которых предполагается сравнить, должны проходить при одинаковых условиях: версии операционной системы, компилятора, CLR, JIT, GC, а также аппаратного обеспечения и режима сборки (рекомендуется release, а не debug);

измерения необходимо проводить при максимальном уровне производительности системы, закрыв все посторонние приложения и отключив службы проверки и установки обновлений, антивирусные службы;

при измерении слишком быстрых алгоритмов (время работы которых имеет порядок разрешающей способности) рекомендуется замерять не каждый вызов, а множество вызовов и вычитать время холостого прогона цикла;

измерения должны повторяться множество раз для одних и тех же условий (результатом эксперимента в таком случае является выборка, а не одно значение измерения);

при использовании JIT-компиляции результаты измерения первого вызова не учитываются, поскольку они будут завышены.

На этапе анализа результатов применяются различные статистические методы и вычисляются числовые характеристики выборки полученной по итогам проведения эксперимента. При анализе результатов, особое внимание следует уделять оптимизациям, которые могут существенно влиять на результаты измерений и зависят от версии JIT и аппаратного обеспечения.

Как можно заметить, на этапе проведения эксперимента существует много однотипных действий, которые необходимо выполнить. В рамках данной работы разработана архитектура приложения, направленного на автоматизацию значительной части процесса эксперимента.

Заключение. Временные затраты – важная характеристика алгоритмов. В современных условиях, при наличии дополнительных слоев абстракций между исходным кодом на языке высокого уровня и машинным кодом, реальные временные затраты могут значительно отличаться от расчетных. В работе выявлены основные этапы процесса измерения временных затрат и их особенности, а также способ автоматизации данного процесса.

1. Cormen, T. Introduction to Algorithms // T.H. Cormen,ʃCh.E. Leiserson,ʃR.L. Rivest,ʃC. Stein – 3rd Edition. – MIT Press, 2009. – 1292 p.
2. Acquiring high-resolution time stamps [Электронный ресурс]: Microsoft development network – Режим доступа: https://msdn.microsoft.com/ library/ windows/ desktop/ dn553408.aspx – Дата доступа: 19.12.2017.

КОДИРОВАНИЕ ТЕКУЩЕГО СОСТОЯНИЯ ДЕТЕРМИНИРОВАННОГО КОНЕЧНОГО АВТОМАТА ЗНАЧЕНИЕМ СЧЕТЧИКА КОМАНД

С.В. Сергеенко
Витебск, ВГУ имени П.М. Машерова

При анализе текста широкое распространение получили регулярные языки и описывающие их регулярные выражения [1]. Актуальным способом эффективного сопоставления текста с заданным регулярным выражением является использование алгоритма, построенный на основе детерминированного конечного автомата, который представляет собой математическую модель, обладающую относительно простой операционной семантикой. [2]

Цель исследования – выяснить возможность реализации детерминированного конечного автомата, в которой текущее состояние кодируется значением счетчика команд. То есть предоставить компилятору как можно больше информации о поведении конечного автомата.

Материал и методы. Материалом исследования служит детерминированный конечный автомат, его программная реализация на языке C++, в которой текущее состояние автомата задается значением счетчика команд. Поставленная цель достигается средствами обобщенного программирования посредством шаблонов в языке программирования C++. Кроме того, были использованы методы математического моделирования и общенаучные методы.

Результаты и их обсуждение. Алфавит и набор состояний конечного автомата, а также типы входного потока и получаемого результата задаются как псевдонимы типов, объявленные в рамках класса, указываемого как параметр шаблона класса `Common`, инкапсулирующего реализацию детерминированного конечного автомата. Для удобства дальнейшего использования, общий для различных конечных автоматов код вынесен в пространство имен `DFA`. Кроме того, введен вспомогательный шаблон класса `DfaTraits`, отвечающий за определение базовых типов, на которых основано определение класса `Common`.

```
namespace DFA {
 template<class States, class CharT, class Result = bool,
 class CharTraits=std::char_traits<CharT>>
```