Национальный исследовательский университет “Высшая школа экономики”.

Факультет компьютерных наук. Программная инженерия.

Построение и анализ алгоритмов.

Домашнее задание №2 студента группы БПИ213 Абрамова Александра Сергеевича.

СОДЕРЖАНИЕ

Разработанные материалы

При выполнении задания были разработаны следующие элементы:

1. Папка $algorithms$ содержит программу на языке C++, которая реализует 4 алгоритма поиска всех вхождений шаблона в строку и измерение эффективности их работы.
   1.1. Выбор анализируемого алгоритма происходит при сборке программы путём подключения соответствующего модуля с реализацией функции $search$ соответствующим образом.
   1.2. При запуске программы происходит считывание текста и шаблона из файлов, адреса которых указываются первым и вторым аргументом командной строки соответственно, решение поставленной задачи, а также вывод результата работы и времени выполнения в стандартный поток вывода.
2. Папка $scripts$ содержит вспомогательные программы для генерации тестовых данных, сборки и запуска программ, а также для обработки выходных данных.
   2.1. Файл $config.js$ содержит настройки экспериментов:
      1) $texts$ - список длин анализируемых текстов: 10000 и 100000
      2) $alphabets$ - список алфавитов анализируемых текстов: 01 и abcd
      3) $patterns$ - список длин искомых шаблонов: от 100 до 3000 с шагом 100
      4) $substitutionSymbols$ - список количества символов подстановки в шаблонах: 0 - 4, 30
      5) $algorithms$ - список кодовых имён анализируемых алгоритмов: naive (наивный алгоритм), kmp (алгоритм Кнута-Морриса-Пратта), kmp-optimized (алгоритм Кнута-Морриса-Пратта с использованием уточнённых граней), aho-corasick (алгоритм Ахо-Корасик)
      6) $runs$ - количество запусков программы на каждом наборе тестовых данных при измерении времени выполнения для получения более точного результата.
   2.2. Файл $gen.js$ содержит исходный код программы, которая генерирует наборы тестовых данных и сохраняет их в директории $tests$. Для каждого типа текста (длина и алфавит) случайным образом создаётся соответствующая строка, сохраняемая в файле $text.txt$, а также создаются шаблоны всех заданных в файле конфигурации длин с соответствующим количеством символов подстановки. Полученные данные размещаются в файлах $in.txt$ соответствующих директорий. Также программа производит решение поставленной задачи и сохранение правильного ответа в файлах $out.txt$.
   2.3. Файл $run.js$ содержит исходный код программы, производящей сборку и запуск алгоритмов:
      2.3.1. При указании аргумента командной строки $TEST$ программа производит тестирование всех алгоритмов и выводит результат работы для каждого алгоритма на каждом типе

текста. При этом сборка программы происходит с аргументами командной строки `-fsanitize=address,undefined -fno-sanitize-recover=all -Wall -Wextra` для отлова ошибок во время сборки и выполнения. Исполняемые файлы сохраняются в папке $TEST$ директории bin.

2.3.2. При указании аргумента командной строки $DEBUG$ происходит аналогичное тестирование всех алгоритмов с выводом вердикта для каждого теста. Исполняемые файлы сохраняются в папке $DEBUG$ директории bin.

2.3.3. При указании аргумента командной строки $TIME$ происходит многократный запуск всех алгоритмов поиска подстроки в строке с целью измерения времени их выполнения. Собранные данные записываются в файл $data.json$ папки $report$. При этом для получения более корректных результатов сборка программы происходит с флагом `-O2`, который позволяет компилятору применять необходимые оптимизации. Исполняемые файлы сохраняются в папке $TIME$ директории bin.

3. Файл $package.json$ содержит ряд “скриптов” для запуска соответствующих вспомогательных программ с различными аргументами командной строки.

Эксперименты и анализ

Для получения результатов скрипт $time$ был запущен на системе под управлением $WSL\ Debian$. Результаты сохранены в директории $report$. Файл $SystemInformation.txt$ содержит описание окружения при проведении эксперимента.

Для построения графиков следует открыть файл $graphs.html$ директории $report$ в любом браузере и загрузить файл $data.json$ эксперимента. По графикам можно отметить следующее:

1. Алгоритм Ахо-Корасик достаточно неэффективен при отсутствии символов подстановки из-за того, что алгоритм не предназначен для поиска одной подстроки в строке. Но с увеличением количества символов подстановки задача трансформируется в поиск нескольких подстрок в строке одновременно, что улучшает относительную эффективность алгоритма Ахо-Корасик.
2. Использование уточнённых граней действительно позволяет повысить эффективность алгоритма Кнута-Морриса-Пратта: улучшенная “версия” показывает себя лучше на всех тестовых данных.
3. Добавление символов подстановки, ожидаемо, значительно снижает эффективность алгоритма Кнута-Морриса-Пратта независимо от типа используемых граней. При большом количестве символов подстановки время работы алгоритма КМП оказывается заметно больше времени выполнения алгоритма Ахо-Корасик, несмотря на противоположный результат при отсутствии символов подстановки.
4. Наивный алгоритм оказывается наиболее эффективным среди рассмотренных. Несмотря на то, что алгоритм КМП с уточнёнными гранями незначительно превосходит его по эффективности на текстах длины 100000 с шаблонами без символов подстановки, наивный алгоритм показывает себя лучше остальных на других тестах, в том числе с большим количеством символов подстановки. Хотя этот результат и не коррелирует с асимптотической сложностью алгоритма, он ожидаем. В виду высокой случайности данных наивному алгоритму не требуется проверять весь шаблон для каждой подстроки текста - вывод об их сходстве может быть сделан уже после небольшого количества сравнений. Более того, для “поддержки” символов подстановки требуются лишь незначительные модификации, практически не влияющие на сложность алгоритма. Таким образом, в совокупности с общей простотой алгоритма и низкой “константой”, наивный алгоритм оказывается самым эффективным из проанализированных.
5. Изменение алфавита оказывает неоднозначное влияние на эффективность рассматриваемых алгоритмов. Теоретически можно ожидать, что размер алфавита не оказывает влияния на наивный алгоритм и алгоритм Кнута-Морриса-Пратта, но снижает эффективность алгоритма Ахо-Корасик. Тем не менее результаты эксперимента показывают, что увеличение алфавита повышает

эффективность наивного алгоритма и алгоритма КМП, а снижение эффективности алгоритма Ахо-Корасик наблюдается лишь для текста длины 10000. Это также связано с высокой случайностью данных и, как следствие, маленьким количеством совпадающих префиксов и суффиксов подстрок, на анализе которых построены алгоритмы Кнута-Морриса-Пратта и Ахо-Корасик.

6. Изменение длины текста влияет на анализируемые алгоритмы неодинаково: при увеличении длины текста в 10 раз, время работы увеличивается в 2.5 - 3 раза для алгоритма Ахо-Корасик и в 8 - 9 раз для остальных алгоритмов. Действительно, доля времени, затрачиваемого на анализ шаблона, в общем времени выполнения алгоритма значительно больше для алгоритма Ахо-Корасик, чем для остальных алгоритмов, что и приводит к уменьшению влияния размера текста на общее время выполнения.
7. Как и ожидается теоретически, время выполнения всех анализируемых алгоритмов линейно зависит от размера шаблона, что подтверждается как видом кривых графиков, так и коэффициентом детерминации линий тренда, который всегда близок к единице для алгоритма Ахо-Корасика, за исключением отдельных тестов близок к единице для наивного алгоритма, близок к единице на всех тестах с текстом длины 10000 и некоторых тестах с текстом длины 100000 для алгоритма КМП. Низкие коэффициенты детерминации во всех случаях объясняются высокой случайностью входных данных, так как отклонения от линии тренда наблюдаются в обе стороны, а общий вид кривой всё равно линейный.

Вывод

Если ожидаются данные, сгенерированные случайным образом без явных закономерностей, стоит использовать наивный алгоритм поиска шаблона в строке. Как показал эксперимент, этот алгоритм работает наиболее эффективно на таких данных.

Если в данных ожидаются определённые закономерности (в частности, совпадения префиксов подстрок с суффиксами), стоит рассмотреть необходимость использование алгоритма, эффективность которого меньше зависит от случайности данных:

1. Если требуется найти вхождения строки в текст, следует использовать алгоритм Кнута-Морриса-Пратта с уточнёнными гранями.
2. Тем не менее этот алгоритм неэффективно обрабатывает символы подстановки. В таком случае следует использовать алгоритм Ахо-Корасик, который позволяет находить вхождения нескольких строк в текст за один проход, что можно использовать для эффективной обработки символов подстановки путём поиска всех вхождений частей шаблона в текст и последующего отбора индексов.
3. Если доля символов подстановки в шаблоне велика, операция отбора индексов становится достаточно сложной, из-за чего алгоритм Ахо-Корасик теряет эффективность. В таком случае имеет смысл вновь использовать наивный алгоритм поиска шаблона в строке, который способен обрабатывать символы подстановки без значительных модификаций алгоритма.