

Московский государственный университет имени М.В. Ломоносова

Факультет Вычислительной математики и кибернетики

Кафедра Алгоритмических Языков

КУРСОВАЯ РАБОТА

**«Эффективный по времени алгоритм расчета энтропии двумерных слов методом скользящего окна»**

**Выполнил:**

Студент м114 группы

Седякин Илья

**Научный руководитель:**

д.т.н., профессор

Ульянов Михаил Васильевич

Москва, 2018

**Содержание**

Введение ....................................................................................................................................... 3

Постановка задачи ....................................................................................................................... 4

1 Анализ инструментов для реализации целей работы…………….....………….……………5

1.1 Онтология……..........................................................................................................5

1.2 SQL………................................................................................................................ 6

1.3 AIML………………………….................................................................................. 6

1.4 Анализ базы данных................................................................................................. 7

2 Вопросы о схожести ..………………...................................................................................... 8

2.1 Вопрос об общем сходстве............................................................................... 8

2.2 Вопрос о сходстве по параметру.................................................................... 10

2.3 Конкатенация новых вопросов с механизмом цепочек сообщений………10

3. Результаты работы программы и возникшие проблемы………………………………….10

Заключение……………..…..…………………………………………………………………...12

Список литературы………………………………………………..............................................13

**Введение.**

Вопрос устройства самоорганизующихся систем активно исследуется учеными по всему миру. В этой работе будут рассмотрена одна из таких систем. Хотя рассматриваемая система использует дискретное время и пространство, она моделирует о образование структур в тонких слоях вытянутых гранул под воздействием вибрации. Поражает то, как эта система самоорганизуется и приходит к порядку из, казалось бы, полного хаоса.

Проводить глубокий анализ таких систем и лучше понимать их помогает подсчет энтропии. Иногда графики изменения энтропии в системах наводят на интересные мыли.

Расчёт энтропии может занять множество вычислительных мощностей и времени. В данной работе будут приведены алгоритмы, позволяющие эффективно рассчитать энтропию двумерных слов, будет проведено сравнение их работы и анализ.

**1. Обзор предметной области.**

**1.1 Задача о самоорганизации стержней на торе.**

В своей работе «ОБРАЗОВАНИЕ СТРУКТУР В ДВУМЕРНЫХ СИСТЕМАХ СТЕРЖНЕОБРАЗНЫХ ЧАСТИЦ» Н.И. Лебовка, Ю.Ю. Тарасевич, В.А. Гигиберия, Н.В. Выгорницкий, А.С. Бурмистров, В.В. Лаптев рассматривают следующую задачу.

Пусть дан квадрат, состоящий из квадратных клеток размерности NxN. Данный квадрат является разверткой тора, то есть, он «цикличен» по горизонтали и вертикали. Длина стороны квадрата N может варьироваться от 128 до 2048.

Начальное состояние систем создавалось с использованием алгоритма случайного последовательного осаждения частиц (RSA), который, условно, состоит в следующем:

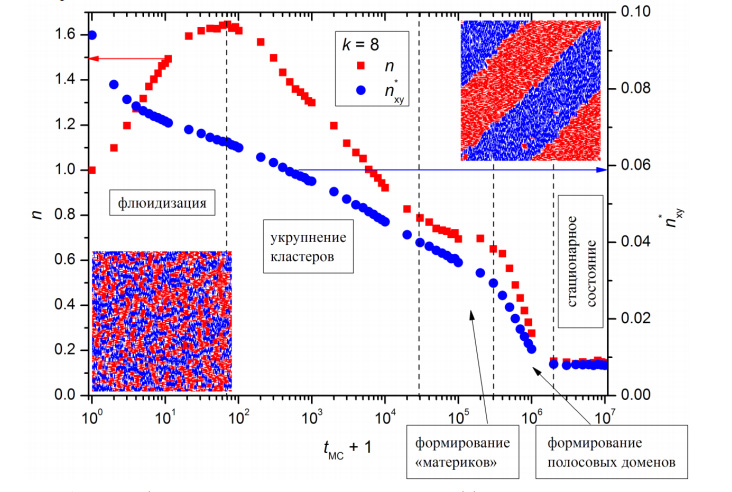
В начале эксперимента на этот тор размещаются стержни, представляющие из себя прямоугольники ширины 1 и длины K, которая варьировалась до 2 до 12. Стержни могли располагаться горизонтально и вертикально, и не могли пересекаться. Стержни размещались поочередно случайным образом до тех пор, пока не возникала ситуация, в которой на развертку тора нельзя положить более ни одного стержня. Эта ситуация называется состоянием джамминга.

Авторы пишут, что в состоянии джамминга плотность частиц составляла примерно  
 p = 0,72. Таким образом поверхность тора получалась хаотично заполненной стрежнями.

После этого случайным образом выбиралась одна частица и предпринималась попытка сместить её на одну клетку в произвольно выбранном направлении. Число попыток, равное полному числу частиц в рассматриваемой системе, принималось за один шаг моделирования.

Моделирование типично продолжалось до710 шагов.

В результате система приходила к самоорганизации: Стержни одинаковой направленности выстраивались в цикличные диагональные полосы на торе. Это происходило только в случае, если длины стержней K >= 6.



**1.2 Энтропия двухмерных слов.**

В рамках данной задачи хочется определить понятие энтропии и измерить ее для различных шагов моделирования. Это позволит понять, как состояние системы постепенно приходит к самоорганизации.

Введем понятие энтропии, вычисляемой методом скользящего окна.

Обозначим каждую клетку развертки тора одной из цифр:

* 0 – клетка пуста
* 1 – через клетку проходит вертикальный стержень
* 2 – через клетку проходит горизонтальный стержень

Выберем на матрице окно – квадрат размера MxM клеток. M2 цифр, расположенных в нем, создают собой слово длины М2, составленное из алфавита размерности 3.

Теперь, пройдемся окном по всей матрице (NxN позиций), и запишем, какое из слов сколько раз встретилось – ji.

Теперь мы можем рассчитать энтропию по формуле

E = - ,

где q – некая константа.

**2. Постановка задачи исследования.**

Подсчет энтропии методом скользящего окна – трудоемкая задача. Трудоемкость растет с увеличением размеров матрицы, окна и количества оцениваемых шагов моделирования.

Главную сложность составляет подсчет количества встреченных слов. Рассчитать же формулу по найденному количеству слов – не представляет труда.

Таким образом, целью работы является:

1. Исследовать алгоритмы, применимые для подсчета слов, определить их особенности реализации в рамках задачи.
2. Реализовать данные алгоритмы и сравнить их эффективность для различных начальных условий.
3. Выбрать лучший алгоритм для определенных начальных условий.

**3. Алгоритмы подсчета энтропии и их особенности реализации.**

**3.1 Линейный поиск.**

Линейный поиск – самый простой в понимании и реализации алгоритм. Однако он заведомо имеет одну из худших производительностей.

Программа поочередно считывает окна из матрицы и пытается уложить их в неупорядоченный массив пар «окно – количество». Для этого она пробегает этот массив от начала до конца. Если подобное окно будет найдено – увеличивает количество оных, если нет – добавляет новое окно в конец.

Как можно заметить, асимптотическая сложность данного алгоритма составляет О(N4\*M2), где N – размер матрицы, M размер окна. Это происходит потому, что N2 окон добавляются в массив, который так же может достичь размера N2, и при каждом сравнении производятся М2 операций.

Единственным плюсом данного алгоритма можно отметить возможность его реализации без динамического выделения памяти. По предположениям, линейный поиск должен был показать худшие результаты на тестировании, и он был приведен в работе лишь для сравнения.

**3.2 Преобразование в индекс.**

Первое, что приходит на ум при решении поставленной задачи и кажется самым быстрым – неким образом преобразовать окно в индекс массива. Тогда поиск окон будет происходить за О(1), и сложность алгоритма будет составлять лишь преобразование окна в индекс.

Здесь стоит разделить задачу на две области.

* Если размер окна М <= 4, мы можем объявить в памяти массив размера 3M\*M, и преобразовывать наше окно в индекс по следующему правилу: Возьмем поочередно каждую цифру {0,1,2} из окна как очередную цифру в троичной записи числа, обозначающего индекс. Переведем это число в десятичную систему счисления и получим наш индекс. Для того, что бы перевести окно в десятичную систему был использован следующий алгоритм:  
  Берется результат, изначально равный нулю. Затем для каждой цифры окна результат умножался на 3, после чего к нему прибавлялась эта цифра.
* Если же размер окна больше 4, мы не сможем объявить столь большой массив. Тогда нам необходимо придумать хеш-функцию для соответствующего преобразования. Вычисление хеш-функции для квадратного массива – дело нетривиальное. Производительность такого алгоритма будет целиком зависеть от выбора хеш-функции. Поэтому этот вариант алгоритма реализован не был.

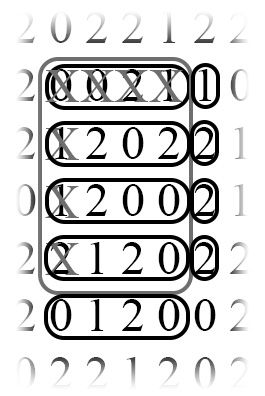
**3.3 Дерево поиска, состоящее из списков.**

Одна из эффективных структур поиска элементов – балансированное дерево поиска. Эта структура представляет собой бинарное дерево из элементов, которые обладают свойством сравнения, т.е. для любых двух элементов можно определить, какой из них больше. Каждый узел дерева хранит один из элементов и два поддерева (левое и правое), которые могут быть пустыми. При этом соблюдается правило: в левом поддереве все элементы меньше элемента в узле, а в правом – больше.

Стандартная библиотека шаблонов STL уже имеет хорошую реализацию такого дерева. Это шаблон std::map, который представляет собой балансированное (чёрно-красное) дерево поиска пар (элемент, значение). Эта реализация и была использована для экспериментов.

Заметим следующую особенность нашей задачи. Когда окно сдвигается, не все его элементы изменяются. К нему лишь добавляется новый ряд справа или снизу, и убирается соответствующий ряд слева или сверху. Все остальные же элементы остаются в окне, и лишь сдвигаются на одну позицию. Таким образом, лучший вариант реализации окна для нас – список списков.

Предположим такую реализацию: Внешний список хранит M списков строк, а каждый из M списков строк хранит M элементов, соответствующих столбцам. Тогда для того, чтобы сдвинуть окно влево или вниз нам будет достаточно О(М) операций (а не О(М2)), так как список позволяет добавлять и удалять элементы за О(1).

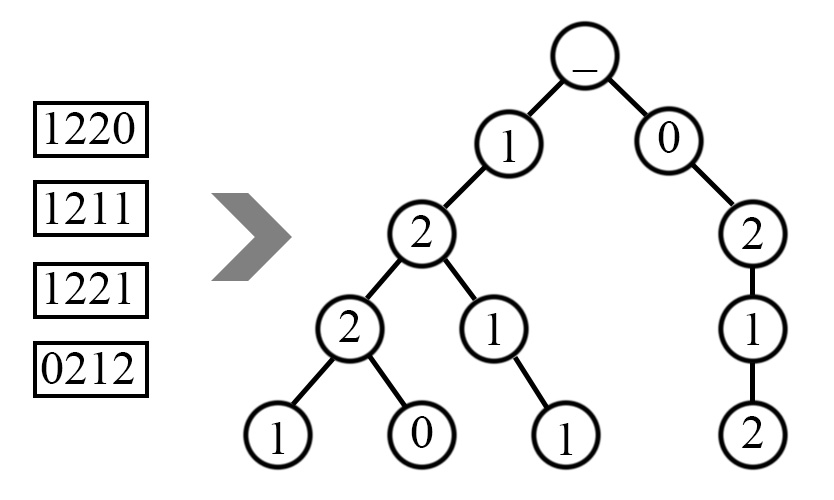


На сложность сравнения элементов в дереве это никак не повлияет – и массивы и списки сравниваются поэлементно. Поиск же элементов в дереве можно оценить как О(log(N2)\*M2).

**3.4 Дерево префиксов.**

Один из главных кандидатов на лучшее время работы – дерево префиксов.

Дерево префиксов представляет собой корневое дерево, каждое ребро которого помечено каким-то символом так, что для любого узла все рёбра, соединяющие этот узел с сыновьями, помечены разными символами.



Таким образом, наши двумерные слова (которые можно так же представить, как одномерные) будут разложены по частям, и для того, чтобы прочитать любое из слов, необходимо будет пройти от корня дерева к одному из его листьев.

Один из главных вопросов реализации префиксного дерева – каким образом хранить сыновей узла. Зачастую для этого используют упорядоченный массив (чтобы искать необходимого сына бинарным поиском).

Было решено поступить следующим образом. Наверное, лучший вариант реализации дерева – хранить в каждом узле по одной цифре окна. Это дает два преимущества:

* Не нужно делать лишних преобразований элементов
* Можно хранить сыновей в виде простого массива размером 3, и получать доступ к ним за О(1) времени.

В итоге, мы получаем следующий алгоритм: Для каждого из окон мы проходим по его элементам, одновременно с этим спускаясь по префиксному дереву. Дойдя до конца, увеличиваем тот счетчик того листа, в который пришли.

**4. Проведение экспериментов**

Тестирование алгоритмов производилось следующим образом:

Программы, реализующие алгоритмы на языке С++ запускались на ноутбуке ASUS n76 с процессором Intel CORE i7 и оперативной памятью 8Gb. (Часть из этих ресурсов была затрачена на работу операционной системы Windows 10)

Для чистоты эксперимента при проведении замеров из программ удалялся участок кода, который, непосредственно, считал энтропию по массиву встретившихся слов. Как уже говорилось ранее, эта процедура производится очень быстро на любой структуре.

Для того, чтобы максимально приблизить результаты к тем, которые были бы получены при измерении энтропии в задачи предметной области, замеры производились на результатах, полученных про моделировании этой задачи. Это 65 файлов с матрицами 256х256, соответствующие шагам самоорганизации под номерами 1, 2, .., 10, 20, .., 100, 200, …, …, 1000000. (В ходе исследований предметной области было выявлено, что значительные изменения в матрице происходят с «логарифмическим разрывом» между шагами.)

Так же, для всех программ и тестовых файлов замеры производились для разных размеров окон M=2,4,6,8. Однако, для метода преобразования в индекс замеров для M>4 не было, так как алгоритм просто не может поддерживать такие окна. Так же, если любой алгоритм показывал очень плохие результаты на определенном размере окна, для больших окон замеров не производилось.

Время замерялось с помощью стандартной функции clock(). Для каждой программы было замерено минимальное, максимальное и среднее время выполнения на всех тестах.

**5. Результаты и рекомендации.**

Результаты описанных экспериментов представлены в следующей таблице.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Метод | Тип замера | М = 2 | М = 4 | М = 6 | М = 8 |
| Преобразование в индекс | Мин. время, мс | 1256 | 1279 |  |  |
| Ср. время, мс | 1315,89 | 1380,31 | невозможно | невозможно |
| Макс. время, мс | 1469 | 1581 |  |  |
| Линейный поиск | Мин. время, мс | 58 | 12538 |  |  |
| Ср. время, мс | 82,7 | 22416,7 | нет данных | нет данных |
| Макс. время, мс | 132 | 27782 |  |  |
| map <list <list <int> > > | Мин. время, мс | 370 | 1339 |  |  |
| Ср. время, мс | 417,622 | 1456,78 | нет данных | нет данных |
| Макс. время, мс | 486 | 1586 |  |  |
| Дерево префиксов | Мин. время, мс | 1 | 14 | 120 | 305 |
| Ср. время, мс | 1,61 | 19,93 | 201,292 | 538,75 |
| Макс. время, мс | 5 | 28 | 299 | 589 |

Анализируя результаты можно сделать следующие выводы:

* От преобразования в индекс ожидалась лучшая производительность на малых значениях. Однако эксперименты показали обратное. Видимо, множественные операции умножения на 3 сильно замедлили работу системы.
* Линейный поиск – результаты вполне ожидаемы: хорошая производительность для M = 2 и степенной рост времени при увеличении размера окна.
* Дерево поиска – ожидаемые результаты с умеренным ростом времени
* Дерево префиксов – наилучшие результаты для любых размеров окон.

Исходя из проведенных экспериментов, можно сказать, что лучшим решением для поиска энтропии для данной задачи является префиксное дерево, в котором сыновья хранятся в виде массива размерностью 3. Независимо от размеров окон, этот алгоритм показывает наилучшие результаты по времени работы.

**Заключение.**

В данной работе была рассмотрена задача вычисления энтропии двумерных слов методом скользящего окна на матрице 256х256. При этом алфавит слов состоит из трех символов.

Были приведены возможные алгоритмы решения, и произведены замеры времени работы этих алгоритмов. В результате был определен самый быстродействующий метод – Префиксное дерево.

**Список литературы.**

1. Онтология в информатике, http://dic.academic.ru/dic.nsf/ruwiki/16269, 2016.

2. Лапшин В. А. Онтологии в компьютерных системах. — М.: Научный мир, 2010.

3. SQLite official site, https://www.sqlite.org/copyright.htm, 2016

4. Wikipedia – AIML, https://ru.wikipedia.org/wiki/AIML, 2016

5. Alicebot official site, http://www.alicebot.org, 2016

6. AIML, http://www.alicebot.org/aiml.html, 2016

7. Брайан Керниган, Деннис Ритчи. Язык программирования C. — Москва: Вильямс, 2015. — 304 с. — ISBN 978-5-8459-1975-5