Государственное образовательное учреждение высшего профессионального образования
«Московский государственный технический университет имени Н.Э. Баумана»
(МГТУ им. Н.Э. Баумана)

Лабораторная работа №2

по курсу «Анализ алгоритмов»

Умножение матриц

Выполнил: Сорокин А.П., гр. ИУ7-52Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019 г.

Оглавление

Введение

В современном мире почти каждый человек пользуются компьютером и Интернетом в частности. Люди пишут текст в документах, выполняют поиск в поисковых системах, ищут переводы слов и текстов в онлайн-словарях. В таких ситуациях человек часто делает орфографические ошибки или опечатки, и на их исправление он тратит своё время. Чтобы этого избежать, в подобных системах есть опции поиска ошибок и автоисправления. Для такой опции необходим поиск расстояния между строками по алгоритмам Левенштейна и Дамерау-Левенштейна. Также эта задача необходима и в программировании (например, для сравнения текстовых файлов или файлов кода в системах контроля версий) и в биоинформатике (например, для сравнения белков, генов и хромосом).

1. Аналитическая часть

1.1 Задачи

Цель лабораторной работы: исследовать расстояния Левенштейна и Дамерау-Левенштейна. Для достижения этой цели были поставлены следующие задачи:

- изучить алгоритмы вычисления расстояний между строками;
- применить метод динамического программирования для матричных реализаций алгоритмов;
- сравнить матричную и рекурсивную реализации алгоритмов;
- оценить эффективность каждой из реализаций по времени и памяти.

1.2 Описание алгоритмов

1.2.1 Расстояние Левенштейна

Расстояние Левенштейна определяет минимальное количество операций, необходимых для превращения одной строки в другую, среди которых:

- вставка (I - insert);
- удаление (D - delete);
- замена (R - replace).

У каждой операции есть так называемая "цена или "штраф"за её выполнение. Цена каждой операции равна 1, кроме случая совпадения символов (M - match); цена в этом случае равна 0, т. к. при равенстве символов не требуется никаких действий. Соответственно, задача нахождения расстояния Левенштейна заключается в нахождении такой последовательности операции, приводящик одну строку к другой, суммарная цена которых минимальна.

Таким образом, если заданы две строки S_1 и S_2 с длинами m и n соответственно над некоторым алфавитом, то расстояние Левенштейна $D(S_1, S_2)$ между данными строками можно вычислить по следующей рекуррентной формуле [3]:

$$D(S_1[1..m], S_2[1..n]) = \begin{cases} m \ if \ n = 0 \\ n \ if \ m = 0 \\ min \begin{cases} D(S_1[1..m-1], S_2[1..n] + 1) \\ D(S_1[1..m], S_2[1..n-1] + 1) \\ D(S_1[1..m-1], S_2[1..n-1] + (S_1[m] \neq S_2[n])) \end{cases} \end{cases} \tag{1.1}$$

Соотношения в рекурретной формуле отвечают за соотвествующие разрешённые операции:

1. Вставка.
2. Удаление.
3. Замена или совпадение в зависимости от результата ($S_1[m] \neq S_2[n]$).

1.2.2 Расстояние Дамерау-Левенштейна

Расстояние Дамерау-Левенштейна является модификацией расстояние Левенштейна. К исходному набору возможных операций добавляется операция транспозиции (T - transpose), или перестановка двух соседних символов. В своих исследованиях Ф. Дамерау показал, что наиболее частой ошибкой при вводе текста является перестановка двух соседних букв слов [2]. "Цена"данной операции также равняется 1. При вычислении расстояния Левенштейна в такой ситуации потребовалось бы дважды заменить символ. Суммарная цена этих двух операций равнялась бы 2, а транспозиция добавляет в суммарную цену лишь 1. Исходя из этого, можно утверждать, что расстояние Дамерау-Левенштейна даёт лучший результат в сравнении с расстоянием Левенштейна.

При вычислении расстояния Дамерау-Левенштейна в рекурретную формулу вносится дополнительное соотношение в минимум:

$$D(S_1[1..m-2], S_2[1..n-2]) + 1 \tag{1.2}$$

Соотношение (1.2) вносится в выражение только при выполнении следующих условий:

$$\begin{cases} m > 2, n > 2 \\ S_1[m] = S_2[n-1] \\ S_1[m-1] = S_2[n] \end{cases} \tag{1.3}$$

Таким образом получаем следующую рекурретную формулу:

$$D(S_1[1..m], S_2[1..n]) = \begin{cases} m \ if \ n = 0 \\ n \ if \ m = 0 \\ min \begin{cases} D(S_1[1..m-1], S_2[1..n] + 1) \\ D(S_1[1..m], S_2[1..n-1] + 1) \\ D(S_1[1..m-1], S_2[1..n-1] + (S_1[m] \neq S_2[n])) \\ D(S_1[1..m-2], S_2[1..n-2]) + 1 \end{cases} & \text{if (1.3)} \\ min \begin{cases} D(S_1[1..m-1], S_2[1..n] + 1) \\ D(S_1[1..m], S_2[1..n-1] + 1) \\ D(S_1[1..m-1], S_2[1..n-1] + (S_1[m] \neq S_2[n])) \end{cases} & \text{otherwise} \end{cases} \tag{1.4}$$

2. Конструкторская часть

2.1 Схемы алгоритмов

На рисунках 2.1 - 2.3 представлены схемы алгоритмов трёх реализаций алгоритмов поиск расстояния между строками.

Рис. 2.1: Матричная реализация алгоритма Левенштейна

Рис. 2.2: Матричная реализация алгоритма Дамерау-Левенштейна

Рис. 2.3: Рекурсивная реализация алгоритма Дамерау-Левенштейна

3. Технологическая часть

3.1 Требования к программному обеспечению

На вход подаются две строки максимальной длины в 50 символов, которые входят в таблицу Юникода (UTF-8). На выход программа выдаёт три числовых значения, которые являются результатами вычисления расстояний тремя методам: матричными реализациями алгоритмов Левенштейна и Дамерау-Левенштейна и рекурсивной реализацией алгоритма Дамерау-Левенштейна. В качестве результата для матричных реализаций также выводится матрица расстояний.

3.2 Средства реализации

Для реализации программы был использован язык C++ [4]. Для замера процессорного времени была использована функция rdtsc() из библиотеки stdrin.h.

3.3 Реализации алгоритмов

На листингах 3.1 - 3.3 представлены коды реализации алгоритмов поиска расстояния.

Листинг 3.1: Матричная реализация алгоритма Левенштейна

```
1  unsigned levenshtein(std::string s1, std::string s2, bool to_print)
2  {
3    size_t s1_len = s1.length(), s2_len = s2.length();
4    size_t row_length = s2_len + 1;
5    unsigned row_bytes = row_length * sizeof(unsigned);
6
7    unsigned *prev_row = new unsigned[row_length];
8    unsigned *current_row = new unsigned[row_length];
9    for (size_t i = 0; i < row_length; i++)
10   prev_row[i] = i;
11
12   if (to_print)
13   {
14     for (size_t i = 0; i < row_length; i++)
15       std::cout << prev_row[i] << ' ';
16     std::cout << std::endl;
17   }
18
19   for (size_t i = 1; i <= s1_len; i++)
20   {
21     current_row[0] = i;
22     for (size_t j = 0; j < row_length; j++)
```

```
23      {
24        unsigned match_fault = unsigned(s1[i - 1] != s2[j - 1]);
25        current_row[j] = std::min({current_row[j - 1] + 1,
26                                   prev_row[j] + 1,
27                                   prev_row[j - 1] + match_fault});
28      }
29      if (to_print)
30      {
31        for (size_t k = 0; k < row_length; k++)
32          std::cout << current_row[k] << ' ';
33        std::cout << std::endl;
34      }
35      memcpy(prev_row, current_row, row_bytes);
36    }
37
38    unsigned result = current_row[s2_len];
39
40    delete [] prev_row;
41    delete [] current_row;
42
43    return result;
44  }
```

Листинг 3.2: Матричная реализация алгоритма Дамерау-Левенштейна

```
1  unsigned damerau(std::string s1, std::string s2, bool to_print)
2  {
3    size_t s1_len = s1.length(), s2_len = s2.length();
4    size_t row_length = s2_len + 1;
5    unsigned row_bytes = row_length * sizeof(unsigned);
6
7    unsigned *prev2_row = new unsigned[row_length];
8    unsigned *prev_row = new unsigned[row_length];
9    unsigned *current_row = new unsigned[row_length];
10   for (size_t i = 0; i < row_length; i++)
11   {
12     prev2_row[i] = 0;
13     prev_row[i] = i;
14   }
15
16   if (to_print)
17   {
18     for (size_t i = 0; i < row_length; i++)
19       std::cout << prev_row[i] << ' ';
20     std::cout << std::endl;
21   }
22
23   for (size_t i = 1; i <= s1_len; i++)
24   {
25     current_row[0] = i;
26     for (size_t j = 0; j < row_length; j++)
27     {
28       unsigned match_fault = unsigned(s1[i - 1] != s2[j - 1]);
29       current_row[j] = std::min({current_row[j - 1] + 1,
```

```
                    prev_row[j] + 1,
                    prev_row[j - 1] + match_fault});
      if (i >= 2 && j >= 1)
        if (s1[i - 1] == s2[j - 2] && s1[i - 2] == s2[j - 1])
          current_row[j] = std::min(current_row[j],
                    prev2_row[j - 2] + 1);
    }

    if (to_print)
    {
      for (size_t k = 0; k < row_length; k++)
        std::cout << current_row[k] << ' ';
      std::cout << std::endl;
    }

    memcpy(prev2_row, prev_row, row_bytes);
    memcpy(prev_row, current_row, row_bytes);
  }

  unsigned result = current_row[s2_len];

  delete [] prev2_row;
  delete [] prev_row;
  delete [] current_row;

  return result;
}
```

Листинг 3.3: Рекурсивная реализация алгоритма Дамерау-Левенштейна

```
unsigned damerau_r(std::string s1, std::string s2, bool to_print)
{
  size_t s1_len = s1.length(), s2_len = s2.length();
  if (s1_len == 0)
    return s2_len;
  if (s2_len == 0)
    return s1_len;

  unsigned match_fault = unsigned(s1[s1_len - 1] != s2[s2_len - 1]);

  unsigned result = std::min({damerau_r(s1.substr(0, s1_len - 1),
                s2.substr(0, s2_len)) + 1,
                damerau_r(s1.substr(0, s1_len),
                s2.substr(0, s2_len - 1)) + 1,
                damerau_r(s1.substr(0, s1_len - 1),
                s2.substr(0, s2_len - 1)) + match_fault});

  if (s1_len > 1 && s2_len > 1)
    if (s1[s1_len - 1] == s2[s2_len - 2] &&
      s1[s1_len - 2] == s2[s2_len - 1])
      return std::min(result, damerau_r(s1.substr(0, s1_len - 2),
                    s2.substr(0, s2_len - 2)) + 1);

  return result;
```

```
25 }
```

3.4 Тесты

Для проверки корректности работы были подготовлены функциональные тесты, представленные в таблице 3.1. В данной таблице λ означает пустую строку, а числа в столбцах "Ожидание"и "Результат"соответствуют результатам работы алгоритмов в следующем порядке:

1. Матричная реализация алгоритма Левенштейна.
2. Матричная реализация алгоритма Дамерау-Левенштейна.
3. Рекурсивная реализация алгоритма Дамерау-Левенштейна.

Таблица 3.1: Функциональные тесты

Строка 1	**Строка 2**	**Ожидание**	**Результат**
λ	λ	0 0 0	0 0 0
λ	а	1 1 1	1 1 1
а	λ	1 1 1	1 1 1
а	а	0 0 0	0 0 0
а	б	1 1 1	1 1 1
азы	базы	1 1 1	1 1 1
компютер	компьютер	1 1 1	1 1 1
данны	данные	1 1 1	1 1 1
email.ru	mail.ru	1 1 1	1 1 1
programmmer	programmer	1 1 1	1 1 1
mail.rus	mail.ru	1 1 1	1 1 1
ашибка	ошибка	1 1 1	1 1 1
алгоритм	алгорифм	1 1 1	1 1 1
копия	копии	1 1 1	1 1 1
укрсовой	курсовой	2 1 1	2 1 1
аглоритм	алгоритм	2 1 1	2 1 1
универе	универ	2 1 1	2 1 1
курс	курсовой	4 4 4	4 4 4
курсовой	курс	4 4 4	4 4 4
курсовой	курсовик	2 2 2	2 2 2
код	закодировать	9 9 9	9 9 9
закодировать	код	9 9 9	9 9 9
ccoders	recoding	5 5 5	5 5 5
header	subheader	3 3 3	3 3 3
subheader	header	3 3 3	3 3 3
subheader	overheader	4 4 4	4 4 4

В результате проверки все реализации алгоритмов прошли все поставленные функциональные тесты.

4. Экспериментальная часть

4.1 Примеры работы

На рисунке 4.1 представлен пример работы программы, демонстрирующий различие в работе алгоритмов наглядно: различаются матрицы расстояний и результаты.

Рис. 4.1: Пример работы программы

4.2 Сравнение работы алгоритмов Левенштейна и Дамерау-Левенштейна

Для сравнения времени работы алгоритмов Левенштейна и Дамерау-Левенштейна были использованы строки длиной от 10 до 70 с шагом 10. Эксперимент для более точного результата повторялся 100 раз. Итоговый результат рассчитывался как средний из полученных результатов. Результаты измерений показаны в таблице 4.1 и на рисунке 4.2.

Таблица 4.1: Время работы алгоритмов при чётных размерах матриц в тактах процессора

Длина слова	Стандартный	Алг-м Виногада	Оптимиз. алг-м Винограда
100	11378576	8801206	1183784
200	94712580	77139807	6132265
300	342637546	273553296	13960164
400	863174672	684590971	24498851
500	1792773181	1404195438	38730481
600	3521402245	2810464339	65344034
700	5996792976	4687597625	90821373
800	10493249242	8357682976	125045114
900	14519644100	11520503989	150466748
1000	23147499368	18975832272	182725943

Алгоритм Левенштейна выигрывает по времени в среднем не более, чем на 10%. Алгоритм Дамерау-Левенштейна выполняется дольше за счёт добавления небольшого количества операций.

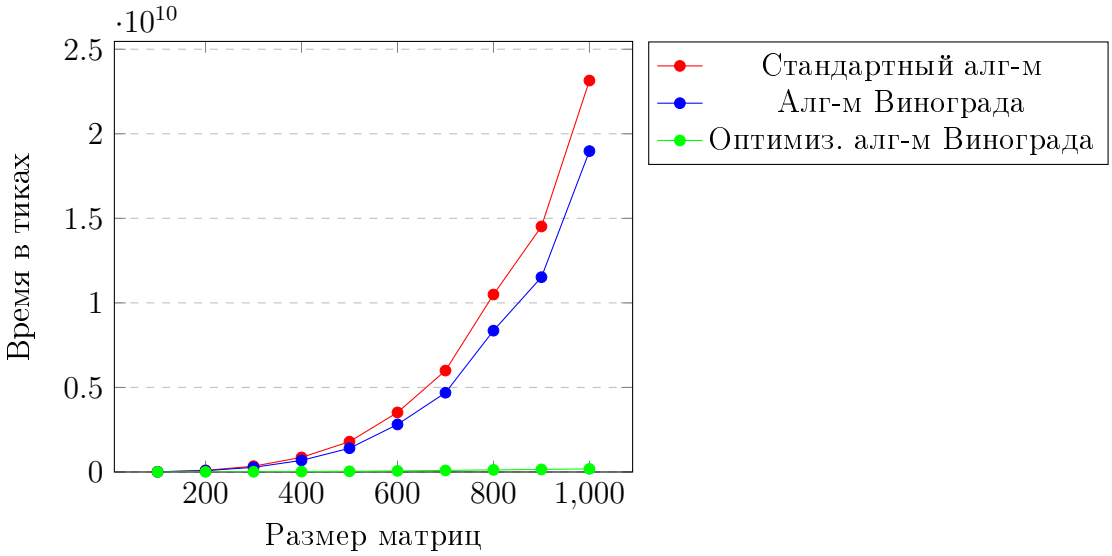

Рис. 4.2: График времени работы алгоритмов при чётных размерах матриц

4.3 Сравнение работы реализаций алгоритма Дамерау-Левенштейна

Для сравнения времени работы матричной и рекурсивной реализаций алгоритма Дамерау-Левенштейна были использованы строки длиной от 1 до 10 с шагом 1. Эксперимент для более точного результата повторялся 100 раз. Итоговый результат рассчитывался как средний из полученных результатов. Результаты измерений показаны в таблице 4.2 и на рисунках 4.3 и 4.4.

Таблица 4.2: Время работы алгоритмов при нечётных размерах матриц в тактах процессора

Длина слова	Стандартный	Алг-м Виногада	Оптимиз. алг-м Винограда
101	11889847	9424292	1314911
201	97600814	77692429	5905464
301	356044760	286028727	15421621
401	939094754	750760519	27668312
501	2032880488	1606779606	42609865
601	4139224911	3146415361	71468977
701	6253434863	4930691171	95735554
801	9771180988	7753844038	122309346
901	14620168149	11646686358	156299495
1001	23445735293	19176489090	193180038

Время выполнения рекурсивной реализации алгоритма резко возрастает с увеличением длины слов: так при длине слова 5 рекурсивная выполняется в 15 раз дольше, чем матричная, а при длине слова 10 - приблизительно в 40000 раз. Рекурсивная реализация выигрывает по времени только при длине слов, равной 1 (в 2 раза), но это тривиальный случай. Можно сделать вывод о том, что матричная реализация алгоритма значительно эффективнее рекурсивной при любой длине слова.

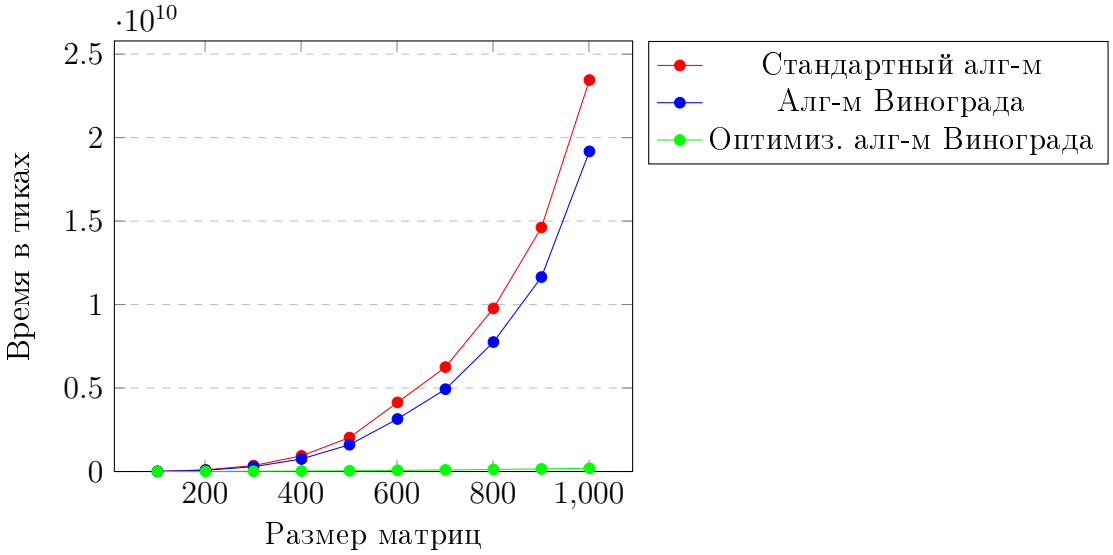

Рис. 4.3: График времени работы алгоритмов при нечётных размерах матриц

Заключение

В ходе лабораторной работы были изучены и реализованы алгоритмы нахождения расстояния Левенштейна и Дамерау-Левенштейна. Для этого были реализованы три различные реализации алгоритмов с применением навыка динамического программирования для матричных.
Экспериментально потверждена эффективность матричных реализаций над рекурсивной: при длине слов выше 10 символов применение рекурсивной реализации алгоритма Дамерау-Левенштейна является нецелесообразной, т. к. проигрывает по памяти и по времени матричных в несколько порядков. Также было экспериментально установлено, что применение матричной реализации Дамерау-Левенштейна допустимо, так как данный алгоритм уступает по времени алгоритму Левенштейна лишь на 10%, но при это при решении определённых задач может давать меньший результат.

Литература

[1] Двоичные коды с исправлением выпадений, вставок и замещений символов. Доклады Академий Наук СССР, 1965. В. И. Левенштейн.

[2] A technique for computer detection and correction of spelling errors. Damerau Fred J.

[3] Indexing methods for approximate dictionary searching. Journal of Experimental Algorithmics, 2011. L. M. Boytsov

[4] https://cppreference.com/ [Электронный ресурс]