Оглавление

[Тема 1](#_Toc62592023)

[Преамбула 1](#_Toc62592024)

[Задачи 1](#_Toc62592025)

[Отчет 4](#_Toc62592026)

# Тема

Алгоритм Вагнера-Фишера, редакционное предписание для алгоритма Вагнера-Фишера. Распараллеливание.

# Преамбула

В этом задании вы реализуете алгоритм Вагнера-Фишера, разработаете функцию, которая строит редакционное предписание на основании матрицы, заполненной алгоритмом Вагнера-Фишера, как рассмотрено в лекции 4.

Конечным шагом задания будет разработка алгоритма, который позволит вычислять расстояние между последовательностями на 2-х процессорах параллельно. Это важно, если обрабатываются большие последовательности, как, например, в биоинформатике.

Прежде чем приступать к выполнению задания необходимо прочесть часть лекции относительно близости строк, алгоритма Вагнера-Фишера и построения редакционного предписания.

# Задачи

1. Начните новый консольный проект LWF. Подключите возможность консольного ввода-вывода.
2. Добавьте в проект класс WF. Его удобно сделать inline, т.е. полностью определить в заголовочном файле.
3. В алгоритме используется целочисленная матрица …. Поэтому, в класс необходимо включить указатель на массив целых (int \* pm) и целочисленные переменные для хранения размерностей – M и N.

Как при этом должны измениться конструктор и деструктор?

При заполнении матрицы нам понадобятся и значения строк. Строки можно сохранить в полях типа string. Например, string s, t;

1. Создайте объект класса WF в main и убедитесь, что проект собирается.
2. Матрица и строки будут заполняться в методе void Init(const char \*sс, const char \*tс). Пусть метод пока просто:
   1. Запоминает строки (можно s=string(sc), t=string(tc))
   2. выделяет память и заполняет M и N.

Подумайте, к чему приведет повторная инициализация? Как решить эту проблему?

1. Чтобы удобно было работать с объектами класса WF добавьте в него 4 метода:
   1. int Rows() const – возвращает количество строк матрицы;
   2. int Columns() const – возвращает количество столбцов матрицы;
   3. int Get(int i, int j) const – возвращает значение i-го,j-го элемента матрицы;
   4. void Set(int i, int j, int val) – задает значение i-го,j-го элемента матрицы;

Обычно, матрицы располагаются в памяти по строкам, сначала все элементы первой строки, затем второй и т.д. Как при этом правильно вычислить номер элемента одномерного массива?

Позаботьтесь и о контроле значений i, j. В случае недопустимых значений применяйте throw out\_of\_range(“<Имя метода>: Index out of range”).

Проверьте, что проект по-прежнему собирается.

Теперь осталась самая важна часть. Реализовать алгоритм Вагнера-Фишера. Псевдокод алгоритма описан в лекции 4.

1. Завершите метод Init, разместив в нем реализацию алгоритма заполнения матрицы.
2. Проинициализируйте объект в main строками из лекции (“no” и “ono”).   
   Выведите матрицу на консоль. Воспользуйтесь для этого методами WF: Rows, Columns и Get.  
     
   Проверьте, что результат соответствует значению, приведенному в лекции:

|  |  |  |  |
| --- | --- | --- | --- |
| 0 | 1 | 2 | 3 |
| 1 | 1 | 1 | 2 |
| 2 | 1 | 2 | 1 |

Если это не так, то найдите и исправьте ошибки реализации.

1. Выведите на консоль расстояние между строками.
2. Мы не можем доказать, что полученная величина является метрикой в пространстве строк. Но можем проверить все свойства метрики на практике.   
     
   Проделайте необходимые проверки.
3. Подумайте, что будет происходить при копировании и присваивании объектов WF друг другу? Нужно ли нам копировать эти объекты?   
   Запретите копирование и присваивание объектов класса WF. Запрет копирования делается такой конструкцией:

WF(const WF&) = delete;

Аналогично запрещается и присваивание.

На этом с классом WF закончено. Используя этот класс можно получить и редакционное предписание. Однако, этим заниматься не будем!

Двинемся к решению другой проблемы. Вы можете убедиться, что разработанный алгоритм не позволяет распараллеливания. Т.е., нельзя вычислить расстояния между отдельными частями строк параллельно и найти общее расстояние, как сумму отдельных расстояний (сработает правило треугольника ☺). Получается, что вычисления могут происходить только последовательно!

Но имеется идея, которая лежит в основе алгоритма Хиршберга [[Hirschberg's Algorithm (monash.edu)](http://users.monash.edu/~lloyd/tildeAlgDS/Dynamic/Hirsch/)].

Разобьем строку s на 2 примерно равные части s1 и s2. Заполним матрицу Вагнер-Фишера для s1, t как и раньше. А для s2, t заполним ее “*в обратном порядке*”, т.е. с конца.

Рассмотрим “*в обратном порядке*” более подробно. Теперь нужно инициализировать порядковыми целыми (0, 1,…) не первую строку и столбец матрицы, а последние. Причем в обратном порядке - … 1, 0! Начинаем с нижнего правого угла и движемся справа налево снизу- вверх.

В общем, алгоритм довольно просто переписывается из прямого!

Когда заполнены обе матрицы, то остается найти наименьшее из суммы последнего столбца первой матрицы и первого столбца второй! Это и будет расстояние по Левенштейну. Но теперь матрицы могут заполняться независимо, а значит, параллельно на двух процессорах.

Приступим к реализации!

1. Добавьте новый класс RWF аналогичный классу WF, но метод void Init(const char \*sс, const char \*tс) в нем должен заполняться в соответствие с изложенным выше алгоритм “*обратной инициализации*”. Больше ничего не меняется!

Кроме, разумеется индекса элементов матрицы где находится расстояние между строками.

1. Разработайте функцию int Distance (const WF &wf, const RWF &rwf), которая получает ссылки на 2 объекта и вычисляет расстояние между строкой s и суммой строк t в объектах wf и rwf.
   1. Пусть функция проверяет, что количество строк одного и другого объектов совпадает. Если это не так, то вбрасывает исключение logic\_error.
   2. Затем ищет построчно (начиная с 0 строки), минимальную сумму значений последнего столбца wf и первого столбца rwf и возвращает результат!
2. В основной программе проверьте, что результат вычисленный новым способом совпадает с результатом полученным алгоритмом Вагнера-Фишера.
   1. Приготовьте 3 строки s, v и u. v и u представляют собой разбиение некоторой строки t = v+u.
   2. Создайте объект WF и проинициализируйте его строками s и v.
   3. Создайте объект RWF и проинициализируйте его строками s и u.
   4. С помощью функции Distance найдите расстояние между s и v+u.
   5. Вычислите расстояние между s и v+u обычным способом. Создайте дополнительный объект WF, проинициализируйте его строками s и v+u и получите расстояние.
   6. Сравните расстояния, если они различаются, ищите ошибку.
   7. Повторите проверку для нескольких разных строк.
3. Очистите проект от бинарных и вспомогательных файлов и поместите его в zip-архив. Это будет часть отчета о выполнении задания.

# Отчет

Подготовьте небольшой отчет в Word. Отчет должен содержать:

* Описание “на пальцах”, т.е. без кода, алгоритма Вагнера-Фишера;
* Описание “на пальцах”, т.е. без кода, алгоритм построения предписания;
* Опишите своими словами идею распараллеливания;
* Опишите тесты, которые вы проводили для проверки правильности параллельного алгоритма.

Отчет и архив проекта поместите как ответ на задание.