**Bigram Language Model (BLM)**

User Manual

Пользовательский Мануал

1. Launch Manual
2. Инструкция по запуску программы
3. Notes (не читайте это там скучно)

**Launch Manual**

To launch the program, you need to run an executable .jar file. There are 2 main ways to do that. You can double click on the file and choose “Java Runtime Environment” as the tool to launch it, or you can go to the command prompt or terminal, navigate to the directory that contains the .jar file and run command:

java –jar BLM.jar

IN CASE THE ABOVE METHODS DO NOT WORK:

In case something goes wrong you will need to extract and open the source files and launch the program from a Java IDE (for example IntelliJ IDEA) by building and running the source code itself. Sry for the inconvenience.

----------------------------------------------------------------------------------------------------

When you launch the program, you will see the GUI appear. This is my first attempt in making a GUI in Java, so don’t judge too harshly. Press on the “Choose file” button and choose a **VALID** .txt file from which you want to extract the bigrams. After doing it, you may press any of the remaining 2 buttons: “Generate Name” to generate a random name from all of the bigrams or “Generate Table” to open a window with a graphic representation of the probabilities of each bigram being chosen. To close the program, simply close the GUI using the cross button in the upper-right corner.

**Инструкция по запуску программы**

Для запуска программы, вам необходимо будет запустить исполняемый .jar файл одним из следующих 2 способов. Вы можете просто кликнуть по файлу и выбрать “Java Runtime Environment” для запуска файла, или вы можете открыть командную строку или терминал, зайти в папку с .jar файлом и исполнить команду

java –jar BLM.jar

В СЛУЧАЕ ЕСЛИ ВЫШЕПЕРЕЧИСЛЕННЫЕ МЕТОДЫ НЕ СРАБОТАЮТ:

В случае если что то пойдёт не так и программа не запустится, сорри. Вам нужно будет просто распаковать и открыть исходные файлы проекта в Java IDE (Например в IntelliJ IDEA) и запустить их как любую другую Java программу.

----------------------------------------------------------------------------------------------------

После запуска программы, вы увидите появившийся графический интерфейс. Это моя первая в жизни попытка реализовать графический интерфейс в Java, так что не судите строго. Для начала, нажмите на кнопку “Choose file” и выберите **ПОДХОДЯЩИЙ** текстовый файл из которого вы хотите извлечь и генерировать биграммы. (Я не стал делать строгие ограничения на открываемые файлы, так как я вам доверяю, но лучше не пытайтесь открыть в программе какие либо неподходящие файлы, так как это может привести к непредсказуемой работе программы) После открытия файла, вы можете нажать на любую из оставшихся 2 кнопок: “Generate Name” для генерации рандомного стринга из доступных биграмм, или “Generate Table” для получения графической репрезентации вероятностей всех биграмм. Для закрытия программы, просто закройте окно интерфейса крестиком в правом верхнем углу.

**NOTES:**

This section is not necessary.

Эта секция не обязательна к чтению, тут мои размышления про рантайм, какую структуру данных лучше было выбрать и прочую фигню.

----------------------------------------------------------------------------------------------------

Time complexity analysis:

Here, n stands for the number of words in .txt, k stands for the average length of the word.

* NameGenerator:
  + Constructor: θ(n). Traverses all of the strings in names.txt once, using a sliding window of size 2 to create bigrams. Assume n > k
  + generateBigram: O(n) in the worst case, when the frequency of one bigram is extremely large relatively to the frequencies of others AND random has generated the lowest bound of that bigram.
    - Example: when there are 4 bigrams: ^c cd dc d$ and their frequencies are ^c – 1, cd – 129391239123, dc – 1239834825845 and d$ - 1 and the random has chosen the number 2, it will need to traverse 129391239121 numbers to get to 129391239123 and choose cd. **IMPORTANT: such scenarios are nearly impossible in regular conditions for any arbitrary files, so the average runtime of generateBigram will be O(1)**
  + generateName: Time of generateBigram \* expected word length. So, O(n\*k) in the **worst**, above mentioned case. However, as such cases are nearly impossible, average running time will be **O(k)** where k is the average length of the word in .txt
  + generateTable: Same as generateBigram, as it traverses all entrances in the frequency map exactly once. O(n) when n < k
* BLM:
  + I have no idea how swing elements work, help me.
    - Should actually be O(1) to draw elements, O(n) to choose a file and make a bigram map, O(k) to generate a word, O(n) to create a bigram table.

Space complexity: Uses 3 separate maps of all bigrams, so O(3n) = O(n)

Best Data Structure analysis:

* HashMap (was actually chosen):
  + Due to how HashMaps in Java work, they allow for O(1) search, but because they are not properly ordered, I was forced to make generateBigram to take O(n) time in the worst case.
* Array:
  + Array structure would actually make generateBigram take O(logn) time even in the worst case, but arrays take O(n) time for search, so the overall time complexity of NameGenerator would be O(n^2) because the constructor would need to find the element to increment its frequency every time.
* TreeMap:
  + XDDDDDD, O(nlogn) constructor and constant balancing of the tree would still be worse than using a hashmap.
* Personal Implementation of the ArrayHashMap would help me to avoid the inconvenience in generateBigram, allowing to combine the best things arrays and hashmaps have to offer, decreasing the time of generateBigram to O(logn) and leaving other times untouched, making the time of generateName O(klogn). However, the code would be 100 times worse, incomprehensible, and the program would take much more space. I decided not to bother.

Если вы реально это прочитали, не бойтесь, я в жизни не такой серьёзный аналитик всего и вся, просто это был мой первый опыт в исполнении подобного свободно заданного задания и я решил задокументироваться чтобы потом можно было использовать этот опыт для решения похожих задач :)