Reporte

Saavedra Escalona Braulio Rubén
Tinoco Miguel Laura Itzel

1. Introducción
   Se escogió un proyecto hecho con anterioridad para hacerlo nuevamente y notar el desarrollo que se ha adquirido en programación y en el análisis de problemas.

2. Definición del problema

   (a) Antes
   El problema a solucionar era principalmente limpiar una entrada(el archivo de texto) para comenzar a trabajar con ella.
   Se había acordado considerar únicamente las palabras que aparecían en el archivo para graficarlas, sin ninguna otra intervención del usuario. Así, todo estaba determinado por los documentos recibidos.

   (b) Después
   En esta ocasión, además de considerar los archivos de texto, se decidió tomar en cuenta la decisión del usuario para que él contribuyera más en los resultados finales. Se le dejó elegir el número de rebanadas y barras de su gráfica (únicamente en el caso de un solo archivo).

3. Análisis del problema

   (a) General
   Programa que reciba una cantidad de archivos diferentes , esto quiere decir que puede ser extensión .txt, .pdf, .java, etc. indefinida.

   Genera un archivo html con una gráfica de pastel y de barras con las palabras más repetidas en cada archivo, con sus respectivos porcentajes.

   (b) Cambios propuestos

      i. Número de rebanadas
      En el caso de un solo archivo, el usuario podrá elegir el número de rebanadas y barras que quiere para sus gráficas.

      ii. Interfaz gráfica
      Implementación de un interfaz gráfica para que el uso del programa sea más sencillo para el usuario.

      iii. Index
      Un índice, archivo html con hiperreferencias a todos los archivos html generados de cada archivo; para tener un orden.

4. Selección de la mejor alternativa
   Decidimos usar Kotlin y hacer que el programa tuviera interfaz gráfica; esta última característica permite que el usuario tenga una mayor comodidad al trabajar con nuestro sistema. Además, la interfaz gráfica también nos ayuda a reducir a cero los errores que se tenían en la versión pasada que usaba banderas para capturar los datos. La elección de este lenguaje se dio porque nos pareció una buena alternativa para solucionar el problema que modelamos con orientación a objetos. Dado que el proyecto pasado lo habíamos hecho en Java, eso nos permitió un avanzar con mayor rapidez, ya que muchas de las bibliotecas de Java están presentes también en Kotlin.

5. Pseudocódigo

   (a) Clase principal: Interfaz gráfica

      i. Botón Seleccionar → Ruta
      El botón seleccionar abre un explorador de archivos donde el usuario elige todos los archivos que quiera, se imprime la ruta total de cada archivo seleccionado en una área de texto.

      ii. Boton Graficar → html
      Toma las rutas del área de texto y crea un objeto tipo Directorio, de ahí se saca una lista de archivos, que a su vez se itera y por cada archivo se crea una gráfica de barras y una de pastel para después generar el HTML junto con el índice pasando estas gráficas. Las gráficas recibieron la lista de palabras del archivo, un número de rebanadas y una posición en donde se colocará la gráfica según el sistema de SVG.

   (b) Gráficas
   A partir de la lista recibida, se crea otra que tenga la misma cantidad de rebanadas indicadas por el usuario, se crea una colección de colores que tenga cardinalidad igual al número de palabras indicadas por el usuario más uno (si dicha cantidad de palabras es menor al de las palabras diferentes en el documento) o igual al número de palabras diferentes en caso contrario.
   Se insertan las barras o las rebanadas según sea el caso, para las barras simplemente se divide 600 entre el número de barras acordadas con el usuario y ese será el ancho para cada barra. Por otro lado, la altura la calculamos sacando la razón entre el número de ocurrencias de una palabra y la cantidad total. Multiplicamos por 200. Vamos recorriendo el eje x considerando únicamente el ancho de nuestras barras y poniéndolas una al lado de otra. Finalmente, la coordenada y en la que empieza cada barra será calculada de la siguiente manera: escogemos un punto y en el que va a empezar a surgir hacia abajo la barra total (aquella que tiene un tamaño de 200 en altura), sacamos la diferencia entre 200 y la altura de dicha barra, el resultado será el desplazamiento que pondremos a partir de la coordenada y antes mencionada, así con la altura de la barra obtendremos que las barras queden alineadas en una coordenada x.
   Veamos ahora cómo colocar las rebanadas de una gráfica de pastel:
   Es necesario aclarar que como nos va interesar movernos por los puntos que conforman la superficie del círculo, usamos una transformación de coordenadas rectangulares a polares. Para describir el algoritmo, definamos algunas variables:

$c =$ cantidad de veces que aparece la palabra asociada con la rebanada n del pastel, $a = \sum_{i=1}^{n-1} x_i$ donde x_i denota la ocurrencia de la palabra i en el archivo, $t =$ el total de palabras que están en el archivo, $\theta = \frac{c+a}{t} \cdot \pi$, $x = cos(\theta) \cdot 100$, $y = sen(\theta) \cdot 100$, $a_x = 0$, $a_y = 0$, $\theta_c = 0$ Vamos a asumir que se van a graficar todas las rebanadas posibles.
Para cada palabra hacemos $\theta_c = \theta$
$\theta = \theta_x$ donde $\theta_x = \theta$ pero con los nuevos parámetros.
Se crea una curva que empieza desde las coordenadas (a_x, a_y) y gira θ grados en sentido de las manecillas del reloj.
Hacemos $a_x = x$ y $a_y = y$ con los nuevos parámetros
Actualizamos $c += a$
Y así habremos terminado.

6. Mejoras
   Le agregamos al programa una interfaz gráfica que permite al usuario seleccionar varios archivos; la interfaz gráfica también muestra un cuadro de texto con los archivos seleccionados por el usuario, también cuenta con el botón "Graficar" el cual genera archivos html con la gráfica de pastel y de barras de las palabras y cuántas veces parecen por archivo seleccionado.
   Agregamos un Index el cual es un html que tiene hiperreferencias a todos los html generados de los archivos elegidos por el usuario.
   Agregamos la opción de que el usuario pueda decidir el número de barras y particiones de la gráfica, esto se permite si la cantidad de archivos seleccionados es igual a uno y se especifica en el cuadro para insertar texto de la interfaz.

7. Análisis estadístico
   Escribimos un archivo de texto con mil palabras de las cuales había cien diferentes; modificamos el archivo principal de nuestra clase para que escogiera automáticamente cierto archivo en cierto directorio para que no se necesitara apretar ningún botón en la interfaz gráfica, y después de generar los archivos hicimos que la interfaz se cerrara automáticamente; dichas modificaciones nos facilitaron la elaboración de un *script en* bash que corría nuestro programa mil veces y tomaba el tiempo total de ejecución.
   El tiempo tomado fue de 15 minutos con 38 segundos, lo cual nos dice que nuestro programa tarda aproximadamente .938 segundos en tomar un archivo de mil palabras y construir una gráfica de barras y otra de pastel con 20 rebanadas y generar su archivo HTML junto con su index.

8. Situación extraordinaria
   Intentamos darle archivos dañados a la interfaz, así como uno que aunque no estaba dañado, no tenía información para trabajar, pues era un archivo vacío. En ambos casos la interfaz logró avisar del fenómeno.

9. Plan a futuro
   Algunas de las mejoras a futuro para nuestro programa son que el usuario pueda elegir archivos de diferentes directorios y especificar en qué carpeta quiere que se generen sus archivos.

10. Repartición del trabajo

    (a) Braulio
    Programación y documentación de las gráficas, generadores de HTML, la clase Colores, la clase con la excepción personalizada, la clase palabra, la clase contador, manejo de IDE, GitHub y creación del bash.

    (b) Itzel
    Programación y documentación de la clase Archivo, Directorio e Interfaz Gráfica.