Instituto Tecnológico de Estudios Superiores Monterrey

CAMPUS QUERÉTARO

Implementación de métodos computacionales

Pedro Oscar Pérez Murueta

Grupo 602

Actividad Integradora 3.4 Resaltador de sintaxis (evidencia de competencia)

Alumnos:

Alan Fernando Razo Peña - A01703350

Erick Alfredo García Huerta - A01708119

Fecha:

29 de abril de 2022

Solución:

La solución que se planteó para esta actividad fue una que parte desde lo general a lo particular y viceversa. Primero se extrajeron todos los datos del archivo input.cpp en formato de caracteres, seguido se usaron dichos caracteres para construir strings. Para la construcción de estos strings se usó un ciclo recursivo que recorrería el arreglo de caracteres hasta encontrar un signo de puntuación (identificado usando la función char-punctuation?) o un espacio en blanco, sea una tabulación, espacio o salto de línea (usando la función char-whitespace?). Una vez aislados los caracteres que formarían la palabra, se almacenarían en una lista a parte donde se le aplicaría la función list->string que convertiría la lista de caracteres en un string que se almacenaría en una lista general.

Al tener esta lista general, se enviaría a otra función que contendría una serie de regexp-match que filtrarían las palabras por categoría, añadiéndoles su respectiva etiqueta para el archivo html y los uniría a un cuerpo que contenía el encabezado de la página, así como el css de la misma. Por último, usando la función write-file se escribiría el nuevo string general dentro del archivo de salida de datos.

Algoritmos implementados:

```
(define char->string
  (lambda (loc aux result)
    (cond
      [(empty? loc) result]
      [(char-whitespace? (car loc))
       (char->string (cdr loc)
                                    '()
                                    (cons
                                     (list->string aux)
                                     result))]
      [(char-punctuation? (car loc))
       (char->string (cdr loc)
                                    '()
                                    (cons
                                     (list->string
                                      (cons (car loc) '()))
                                     (cons
                                      (list->string aux)
                                      result)))]
      [else
       (char->string (cdr loc)
                                    (append aux (cons (car loc) '()))
                                    result)])))
```

La totalidad del proyecto recurre a ciclos recursivos cuando se necesita de acumular y/o recorrer arreglos. En los momentos donde no se necesita se usan funciones auxiliares con procesos lineales que simplemente facilitan la obtención de datos.

```
(define finalList
  (lambda (inputFile)
    (string-append header (matches (reverse(string-list inputFile)) ""))))
```

Tiempo de ejecución:

Usando la función (time (void)) se obtuvo que la ejecución total del código toma un total de: cpu time: 15 real time: 1 gc time: 0.

Complejidad algorítmica:

Usando la Big O Notation, se puede concluir que la complejidad total del programa es de O(n) debido a que esta es la complejidad más alta presentada a lo largo del programa. Para ver detalles individuales sobre el análisis de complejidad algorítmica revisar comentarios del código.

Conclusión:

La solución planteada se encuentra dentro de las buenas prácticas de la programación, debido a que presenta una complejidad pequeña, así como hace uso de una buena práctica que es la separación de tareas complejas en funciones que llevan a cabo tareas sencillas de apoyo que no contribuyen de forma negativa al número de iteraciones que se llevan a cabo, siendo todas estas funciones auxiliares de complejidad O(1).