TC2025 / Programación avanzada

Assignment 8

Tecnológico de Monterrey

Programación Avanzada: Assignment 9

ITESM - Campus Estado de México

Profesor: Miguel Angel Medina Pérez

Fecha límite para entrega de solución: 12 de noviembre del 2019

Clave y grupo: TC2025-201913.1

Nombre: _____ Matrícula: _____

Compromiso de integridad académica

Apegándome al Reglamento de Integridad Académica del Instituto Tecnológico y de Estudios Superiores de Monterrey, me comprometo a que mi actuación en este assignment esté regida por la Integridad Académica. En congruencia con el compromiso adquirido con dicho reglamento, realizaré este assignment de forma honesta y personal, para reflejar, a través de él, mi conocimiento y aceptar, posteriormente, la evaluación obtenida.

Marque el recuadro para aceptar: ☐

Instrucciones generales

Usted debe anexar este documento en la publicación del examen en Google Classroom titulada Assignment 8 con los datos que se le solicitaron hasta este punto. Tenga en cuenta que existen muchas herramientas gratis y de paga en la web que les permitirá completar los datos que se les pide en este formulario en formato `pdf` (ejemplo, https://www.sejda.com/pdf-editor).
También deberá anexar a dicha publicación los archivos con sus códigos fuente. Si usted no entrega este documento con los datos solicitados, no entrega los códigos fuente, entrega o modifica alguno de estos documentos una vez transcurrida la hora final del assginment, su solución será invalidada.

Ejercicios

Su empresa desarrollará una plataforma en C con algoritmos de Reconocimiento de Patrones (RP). A usted se le ha encargado implementar los siguientes requerimientos que serán la base para desarrollar los algoritmos de la plataforma:

1. Los especialistas en RP suelen representar como tuplas las instancias del problema que desean resolver. Para modelar las instancias usted deberá implementar una estructura con alias `Instance` compuesta de tres campos:
   a. El campo `features` es un arreglo unidimensional representado por un puntero a `double`.
   b. El campo `featureTypes` es un arreglo unidimensional representado por un puntero a `FeatureType` el cual es un alias de un enumerado con dos posibles valores: `numerical` y `nominal`.

c. El campo `length` indica la longitud de los arreglos `features` y en `featureTypes`.

2. Programe la función `computeEuclideanDissimilarity` que recibe como parámetros dos punteros constantes a instancias constantes (`Instance`) y retorna la disimilitud (`double`) entre ellas. Para calcular la disimilitud, la función itera por los elementos de `features` válidos (desde 0 hasta `length` – `1`), suma el cuadrado de las diferencias de los elementos y retorna la raíz cuadrada de la suma. La diferencia de dos elementos de `features` consiste en la resta aritmética si dichos elementos están etiquetados como `numerical`; en caso contrario, o sea cuando están etiquetados `nominal`, la diferencia es `0` si los elementos de `features` son iguales, mientras que la diferencia es `1` si los elementos son diferentes. Programe las cuatro funciones de prueba descritas en la lista de cotejo para `computeEuclideanDissimilarity`.
3. Programe la función `findNearestNeighbor` que recibe cuatro parámetros que no se modificarán dentro de la función: un arreglo de instancias representado por un puntero constante a una instancia constante (`Instance)`, una variable constante `length` indicando la longitud del arreglo y una variable `query` que contiene una instancia representada por un puntero constante a instancia constante (`Instance`), un puntero a una función `computeDissimilarity` (El prototipo de esta función es el mismo que el de la función `computeEuclideanDissimilarity`) que calcula la disimilitud entre instancias. La función retorna un puntero a la instancia en el arreglo que es menos disimilar a `query`. Programe las cuatro funciones de prueba descritas en la lista de cotejo para `findNearestNeighbor`.
4. **Inciso de bonificación (5 puntos)**. Programe la función `averageDataset` que recibe un arreglo de instancias representado por un puntero constante a instancia constante (`Instance`), una variable constante `length` indicando la longitud del arreglo. La función retorna un puntero a una nueva instancia (`Instance`) que contiene el promedio de las instancias contenidas en el arreglo pasado por parámetro. Tenga en cuenta que el promedio de varias instancias (`Instance`) se calcula promediando los valores de cada elemento de `Features` entre pares de instancias. En el caso de los elementos etiquetados `nominal`, use la moda en lugar de promediar los valores. Como ejemplo, suponga que tiene cuatro instancias con los siguientes valores válidos (desde 0 hasta `length` – `1`) de `features`:

   [0, 0, 0]
   [1, 1, 2]
   [0, 1, 1]
   [1.2, 1, 3.4]

   de las cuales sabemos que la componente 0 y la 2 están etiquetadas `numerical` mientras que la componente 1 está etiquetada `nominal`; la instancia promedio (`refInstance`) deberá contener los siguientes valores en el campo `features`:

   [0.55, 1, 1.6]

   mientras que en el campo `featureTypes` deberá contener:

   [`numerical`, `nominal`, `numerical`]

   y el campo `length` deberá tener valor 3.

   Este es un ejercicio de todo o nada por lo que no se incluye ningún instrumento de evaluación. El ejercicio deberá estar resuelta sin ningún error y, además, deberán incluirse las siguientes funciones de pruebas:

a. Una función de prueba para el caso que el arreglo esté vacío.
b. Una función de prueba para el caso que el arreglo contenga instancias con cantidades diferentes de elementos válidos en `Features`.
c. Una función de prueba para el caso que el arreglo contenga al menos dos instancias con etiquetas diferentes para una misma posición en el arreglo `featureTypes`.
d. Una función de prueba que muestre el correcto funcionamiento calculando el promedio de al menos 10 instancias que tengan 4 valores de `features` cada una donde dos elementos estén etiquetados `nominal` y dos etiquetados `numerical`.

El profesor ejecutará sus propias pruebas personalizadas. La solución del estudiante deberá pasar estas pruebas para considerarse válida.

Lista de cotejo

(100/31 cada elemento)

i. ☐ El código fuente compila sin errores con `GCC`.
ii. ☐ El programa se ejecuta como aplicación de consola.
iii. ☐ El estudiante crea un archivo de cabecera en C con las definiciones de los tipos de datos (estructuras y enumerados) y con el prototipo de las funciones. También crea un archivo de código en C con la implementación de la función. La firma de las funciones en ambos archivos coincide y no tiene errores de sintaxis.
iv. ☐ La estructura que representa las instancias tiene como alias `Instance`.
v. ☐ El campo `features` es un arreglo unidimensional representado por un puntero a `double` en la estructura `Instance`.
vi. ☐ El campo `featureTypes` es un arreglo unidimensional representado por un puntero a `FeatureType` en la estructura `Instance`.
vii. ☐ El código incluye un enumerado con dos posibles valores: `numerical` y `nominal`.
viii. ☐ El código incluye un enumerado con alias `FeatureType` que tiene dos posibles valores: `numerical` y `nominal`.
ix. ☐ El programa incluye el prototipo y, además, la implementación de la función `computeEuclideanDissimilarity`.
x. ☐ La firma de la función `computeEuclideanDissimilarity` respeta las restricciones del problema (recibe como parámetros dos punteros constantes a instancias constantes `Instance` y retorna la disimilitud `double` entre ellas) y no tiene errores de implementación.
xi. ☐ La función `computeEuclideanDissimilarity` hace la cantidad de iteraciones correctas.
xii. ☐ La función `computeEuclideanDissimilarity` itera por cada uno de los elementos del campo `features` de al menos una de las instancias pasadas por parámetro.
xiii. ☐ La función `computeEuclideanDissimilarity` itera por cada uno de los elementos del campo `features` de las dos instancias pasadas por parámetro.
xiv. ☐ La función `computeEuclideanDissimilarity` calcula correctamente la diferencia de elementos `features` etiquetados `numerical`.
xv. ☐ La función `computeEuclideanDissimilarity` calcula correctamente la diferencia de elementos `features` etiquetados `nominal`.

xvi. ☐ La función `computeEuclideanDissimilarity` calcula correctamente la disimilitud total entre las instancias comparadas.

xvii. ☐ El programa incluye una función de prueba que verifica que la función `computeEuclideanDissimilarity` retorna `-1` cuando al menos una de las instancias comparadas tiene `length` con valor `0`. Esta función de prueba no usa variables globales, crea objetos de las estructuras reservando espacio dinámicamente (no es válido en esta prueba crear objetos de la siguiente manera: `Instance someInstance;`) y elimina toda la memoria que reservó dinámicamente. En el archivo de cabecera del programa se incluye el prototipo de esta función. Esta función no imprime nada en consola. La función de prueba retorna `1` si la prueba pasa y `0` en caso contrario.

xviii. ☐ El programa incluye una función de prueba que verifica que la función `computeEuclideanDissimilarity` retorna `-2` cuando las instancias comparadas tienen valores diferentes del campo `length`. Esta función de prueba no usa variables globales, crea objetos de las estructuras reservando espacio dinámicamente (no es válido en esta prueba crear objetos de la siguiente manera: `Instance someInstance;`) y elimina toda la memoria que reservó dinámicamente. En el archivo de cabecera del programa se incluye el prototipo de esta función. Esta función no imprime nada en consola. La función de prueba retorna `1` si la prueba pasa y `0` en caso contrario.

xix. ☐ El programa incluye una función de prueba que verifica que la función `computeEuclideanDissimilarity` retorna `-3` cuando las instancias comparadas tienen valores iguales del campo `length` pero al menos un elemento de los respectivos arreglos `featureTypes` contienen etiquetas diferentes (una etiqueta es `numerical` y la otra es `nominal`). Esta función de prueba no usa variables globales, crea objetos de las estructuras reservando espacio dinámicamente (no es válido en esta prueba crear objetos de la siguiente manera: `Instance someInstance;`) y elimina toda la memoria que reservó dinámicamente. En el archivo de cabecera del programa se incluye el prototipo de esta función. Esta función no imprime nada en consola. La función de prueba retorna `1` si la prueba pasa y `0` en caso contrario.

xx. ☐ El programa incluye una función de prueba que verifica que la función `computeEuclideanDissimilarity` calcula correctamente la disimilitud de dos instancias que tienen `length >= 4`, tiene dos o más elementos etiquetados `numerical` y tiene dos o más elementos etiquetados `nominal`. Esta función de prueba no usa variables globales, crea objetos de las estructuras reservando espacio dinámicamente (no es válido en esta prueba crear objetos de la siguiente manera: `Instance someInstance;`) y elimina toda la memoria que reservó dinámicamente. En el archivo de cabecera del programa se incluye el prototipo de esta función. Esta función no imprime nada en consola. La función de prueba retorna `1` si la prueba pasa y `0` en caso contrario.

xxi. ☐ El programa incluye el prototipo y, además, la implementación de la función `findNearestNeighbor`. El prototipo de la función se encuentra en el archivo de cabecera del programa.

xxii. ☐ El encabezado y prototipo de la función `findNearestNeighbor` no tiene errores de implementación y respeta las restricciones del problema.

xxiii. ☐ La función `findNearestNeighbor` itera correctamente por todas las instancias.

xxiv. ☐ La función `findNearestNeighbor` calcula la disimilitud entre la instancia pasada por parámetro y cada instancia por la que se itera, evaluando correctamente la función `computeDissimilarity`.

xxv. ☐ La función `findNearestNeighbor` calcula la menor disimilitud de manera correcta.

xxvi. ☐ La función `findNearestNeighbor` retorna el puntero a la instancia correcta: la menos disimilar.

xxvii. ☐ El programa incluye una función que prueba que `findNearestNeighbor` retorna `-1` cuando la instancia `query` o alguna de las instancias del arreglo tiene valor `0` en el campo `length`. Esta función de prueba no usa variables globales, crea objetos de las estructuras reservando espacio dinámicamente (no es válido en esta prueba crear objetos de la siguiente manera: `Instance someInstance;`) y elimina toda la memoria que reservó dinámicamente. En el archivo de cabecera del programa se incluye el prototipo de esta función. Esta función no imprime nada en consola. La función de prueba retorna `1` si la prueba pasa y `0` en caso contrario.

xxviii. ☐ El programa incluye una función que prueba que `findNearestNeighbor` retorna `-2` cuando la instancia `query` y alguna de las instancias del arreglo tienen valores diferentes del campo `length`. Esta función de prueba no usa variables globales, crea objetos de las estructuras reservando espacio dinámicamente (no es válido en esta prueba crear objetos de la siguiente manera: `Instance someInstance;`) y elimina toda la memoria que reservó dinámicamente. En el archivo de cabecera del programa se incluye el prototipo de esta función. Esta función no imprime nada en consola. La función de prueba retorna `1` si la prueba pasa y `0` en caso contrario.

xxix. ☐ El programa incluye una función que prueba que `findNearestNeighbor` retorna `-3` cuando la instancia `query` y alguna de las instancias del arreglo tienen valores iguales del campo `length` pero al menos un elemento de los respectivos arreglos `featureTypes` contienen etiquetas diferentes (una etiqueta es `numerical` y la otra es `nominal`). Esta función de prueba no usa variables globales, crea objetos de las estructuras reservando espacio dinámicamente (no es válido en esta prueba crear objetos de la siguiente manera: `Instance someInstance;`) y elimina toda la memoria que reservó dinámicamente. En el archivo de cabecera del programa se incluye el prototipo de esta función. Esta función no imprime nada en consola. La función de prueba retorna `1` si la prueba pasa y `0` en caso contrario.

xxx. ☐ El programa incluye una función de prueba que verifica que la función `findNearestNeighbor` retorna el correcto índice de la instancia en el arreglo menos disimilar a `query`. Esta función trabaja con un arreglo de al menos 4 instancias que tienen `length >= 4`, tiene dos o más elementos etiquetados `numerical` y tiene dos o más elementos etiquetados `nominal`. Esta función de prueba no usa variables globales, crea objetos de las estructuras reservando espacio dinámicamente (no es válido en esta prueba crear objetos de la siguiente manera: `Instance someInstance;`) y elimina toda la memoria que reservó dinámicamente. En el archivo de cabecera del programa se incluye el prototipo de esta función. Esta función no imprime nada en consola. La función de prueba retorna `1` si la prueba pasa y `0` en caso contrario.

xxxi. ☐ Los nombres de variables y parámetros en todo el programa tienen sentido, excepto las variables de iteración de ciclos donde por convenio se aceptan nombres como `i`, `j` y `k`.

Tecnológico de Monterrey