UNIVERSIDAD DE GRANADA

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

Curso 2023/2024

Guion de prácticas

Kmer3

class Profile

Abril de 2024

Índice

1. Definición del problema

Como ya sabemos, las prácticas tienen como objeto principal trabajar con secuencias de nucleótidos procedentes de genomas de individuos de géneros o especies diferentes. Más concretamente, a la llegada de un nuevo individuo de especie o género desconocido, se quiere hallar el perfil al que más se ajusta (de entre un conjunto de perfiles candidatos) el nuevo genoma para realizar nuestra predicción. En esta práctica vamos a dar un paso más en nuestro proceso de predicción.

De momento, seguimos utilizando los ficheros profile proporcionados en el repositorio, sin preocuparnos de cómo se obtienen estos. De esta forma, el problema de la predicción de un perfil para una secuencia genómica lo vamos a reducir a resolver el siguiente problema: dado un profile, P_x, de género o especie desconocido, y un conjunto de profiles de referencia, $P_1, P_2, \ldots, P_n$, se quiere calcular la distancia de P_x a cada uno de los profiles $P_1, P_2, \ldots, P_n$, y determinar aquel con menor distancia, para asignarle su identificador al profile de valor desconocido.

Para identificar el profile más próximo, es necesario definir una medida de distancia entre dos profiles, esto es, medir el parecido entre dos profiles. Los detalles se muestran en la sección 4. En esta práctica se ampliará la funcionalidad de la clase `Profile` para permitir medir distancias entre objetos de la clase.

Todos los profiles que van a intervenir en nuestro proceso de predicción, se van a almacenar en memoria dinámica; esto nos va a permitir introducirnos en la gestión dinámica de memoria de forma práctica, inicialmente fuera de una clase.

2. Arquitectura de las prácticas

Como ya se indicó en prácticas anteriores, la práctica `Kmer` se ha diseñado por etapas; las primeras contienen estructuras más sencillas, sobre las cuales se asientan otras estructuras más complejas y se van completando con nuevas funcionalidades.

En `Kmer3` se amplía la funcionalidad de la clase `Profile`, bloque **C** de la Figura 1; el resto de clases permanecen iguales.

C Profile.cpp
: Se amplía la clase con la implementación del método `getDistance()`. La componente privada de datos de la clase Profile es la misma que en `Kmer2`: un vector estático de objetos `KmerFreq`, más algún dato adicional. Esto supone cambios mínimos en la clase.

La carga práctica de Kmer3 consiste en la gestión de memoria dinámica para el almacenamiento de los profiles que intervienen en el programa y en la incorporación y tratamiento de nuevos parámetros en la función `main()` para poder leer los argumentos que se pasan al programa desde la línea de comandos.

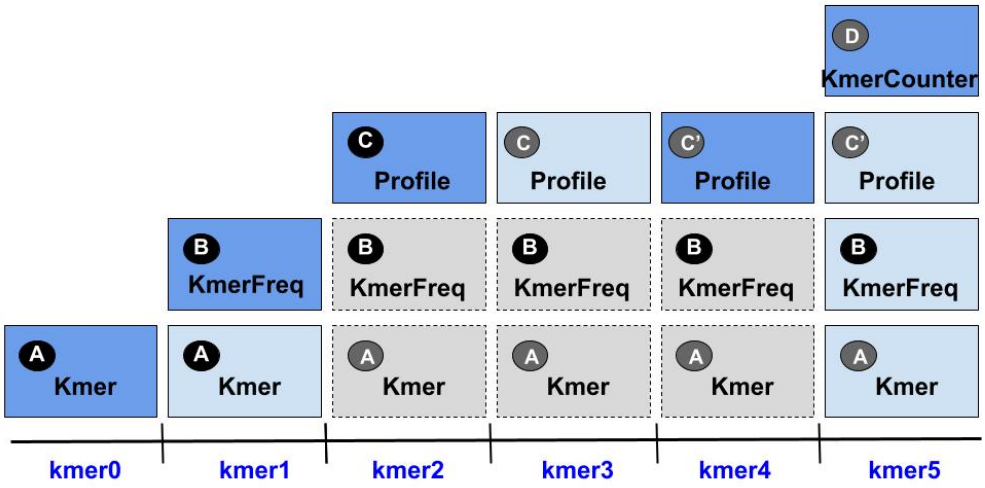

Figura 1: Arquitectura de las prácticas de MP 2024. Los cambios esenciales en las clases (cambio en estructura interna de la clase) se muestran en azul intenso; los que solo incorporan nuevas funcionalidades en azul tenue. En gris se muestran las clases que no sufren cambios en la evolución de las prácticas.

3. Objetivos

El desarrollo de esta práctica `Kmer3` persigue los siguientes objetivos:

- Practicar con punteros y memoria dinámica fuera de una clase.
- Profundizar en el paso de parámetros a `main()` desde la línea de comandos (parámetros alternativos, optativos y por defecto).
- Practicar con herramientas como el depurador de NetBeans y `valgrind` para rastrear errores en tiempo de ejecución.

4. Cálculo de distancia entre profiles

Ya sabemos que un objeto `Profile` contiene una lista de $k-meros$ identificados como frecuentes en un determinado género o especie. Los $k-meros$ más frecuentes, se encuentran arriba en el orden y los menos frecuentes al final de la lista.

Vamos a definir una medida de distancia entre dos objetos `Profile` P_1 y P_2. Para calcular la distancia entre los dos objetos, es importante que ambos estén ordenados por frecuencia, de mayor a menor valor. Si no es así, el valor calculado no sería correcto. Indicar que es importante especificar el orden relativo de los profiles a la hora de calcular la distancia, $distance(P_1, P_2)$. Esto es así, porque la medida no es simétrica, como luego mostraremos.

La distancia se calcula de la siguiente forma:

$$distance(P_1, P_2) = \frac{\sum_{kmer_i(P_1)} |rank^{P_1}_{kmer_i(P_1)} - rank^{P_2}_{kmer_i(P_1)}|}{size(P_1) * size(P_2)}$$

donde, $kmer_i(P_j)$ es el i-ésimo k–$mero$ del profile $P_j, j \in \{1, 2\}$, y $rank^{P_k}_{kmer_i(P_j)}$ es la posición en el orden (rank) del i-ésimo $k-mero$ del profile $P_j, j \in \{1, 2\}$ en el profile P_k. El orden de un $k-mero$ es un número entero, de forma que el primer $k-mero$ tiene $orden = 0$, el segundo $orden = 1$ y así sucesivamente.

Como vemos en la anterior fórmula, a la diferencia entre órdenes, se le aplica la función valor absoluto | | para evitar números negativos y que unos términos de la sumatoria se compensen con otros.

También se puede observar que los valores concretos de frecuencia no intervienen en el cálculo de la distancia; tan solo sirven para establecer el $rank$ de un $k-mero$.

Dado el $k-mero$ i-ésimo de P_1, $kmer_i(P_1)$, el valor $rank^{P_1}_{kmer_i(P_1)}$ sería la posición que ocupa en P_1 y $rank^{P_2}_{kmer_i(P_1)}$ sería la posición que ocupa en P_2. Sin embargo, si el $k - mero$ no se encuentra en P_2, se asigna el valor máximo a $rank^{P_2}_{kmer_i(P_1)}$ ($rank^{P_2}_{kmer_i(P_1)} = size(P_2)$). Dicho de otra manera, si un $k-mero$ determinado en P_1 no aparece en P_2, hay una penalización, pero esta no es constante para todos los $k-meros$ no encontrados en P_2. La penalización disminuye conforme el $k-mero$ es menos frecuente en el profile origen P_1. Vea los ejemplos de las tablas 1 y 2 , donde existen varios $k - meros$ no presentes en P_2 o P_3.

Algunas propiedades de la medida de distancia anterior son:

- Es un número real: $distance(P_1, P_2) >= 0$.
- Toma el valor cero para $distance(P_1, P_1)$ ya que coinciden todos los $k - meros$ en las mismas posiciones dentro del orden, luego el numerador siempre es cero.
- La medida de distancia, tal como se ha definido, no es simétrica: $distance(P_1, P_2) \neq distance(P_2, P_1)$.
- La distancia entre pares de objetos, en los que alguno de ellos es un objeto profile vacío, $size(P_1) = 0$ o $size(P_2) = 0$, no está definida (división por cero en el momento de normalizar). Como puede ver en los comentarios del método `getDistance()` en `Profile.h`, en tal caso se debe lanzar una excepción `std::invalid_argument` en el método `getDistance()`.

Veamos con dos ejemplos, cómo se calcula la distancia y cómo interpretar que dos profiles sean más o menos parecidos. Los cálculos que aquí se muestran se han realizado con los ficheros profile: `prep_xx.prf`, `prep_fly.prf` y `prep_hmn.prf`, que se corresponden respectivamente con P_1, P_2 y P_3.

Ejemplo 1 *Cálculo de distancia entre dos profiles*

Sea P_1, el profile que se muestra a continuación, cuyo profileId es desconocido (unknown) y que tiene una longitud de 10 $k-meros$ (con $k = 5$).

```
MP-KMER-T-1.0
unknown
10
TTTTT 271
ATTTT 235
AAAAA 203
AAAAT 187
TATTT 168
AAATA 166
AAATT 163
AATTT 160
TTTTA 153
ATATT 148
```

Disponemos de otro profile P_2, cuyo identificador es conocido, drosophila melanogaster, con el que queremos medir la distancia.

```
MP-KMER-T-1.0
drosophila melanogaster
15
TTTTT 255
AAAAA 243
ATTTT 194
AAAAT 191
AAATA 164
AAATT 164
AATTT 159
TATTT 155
CAAAA 151
TTTTA 151
TAAAA 147
TTTTG 143
TTAAA 142
TTTAA 141
GAAAA 133
```

Vamos a desglosar la distancia, con los valores que aporta cada uno de los 5-meros de P_1. El valor de la distancia sería entonces $distance(P_1, P_2) = 15/150 = 0{,}1$. Los detalles se encuentran en la siguiente tabla.

	$kmer$	$rank(P_1)$	$rank(P_2)$	$\lvert rank(P_1) - rank(P_2) \rvert$	$\in P_2$
	TTTTT	0	0	0	true
	ATTTT	1	2	1	true
	AAAAA	2	1	1	true
	AAAAT	3	3	0	true
	TATTT	4	7	3	true
	AAATA	5	4	1	true
	AAATT	6	5	1	true
	AATTT	7	6	1	true
	TTTTA	8	9	1	true
	ATATT	9	15	6	false
sum				15	

Cuadro 1: Cálculo desglosado de la distancia $distance(P_1, P_2)$

De los 10 5-meros de P_1, 9 se encuentran en P_2 y en las mismas posiciones o muy similares en el orden de los dos profiles. Sin embargo, el último $k - mero$ de P_1 (ATATT) no está en P_2, por lo que se le asigna $rank(P_2) = 15$ (el tamaño de P_2).

Ejemplo 2 *Cálculo de distancia entre dos profiles*

Disponemos de otro profile P_3, cuyo profileId es homo sapiens.

```
MP-KMER-T-1.0
homo sapiens
15
```

```
4  TGTGT 78
5  GTGTG 59
6  CCCAG 47
7  GGGAG 44
8  CTGTG 42
9  GGCTG 41
10 GGAGG 40
11 TGTCT 39
12 CAGGA 38
13 CCTGG 37
14 CCAGC 36
15 CAGCC 35
16 CCAGG 35
17 CCTCC 35
18 CTGGG 35
```

Queremos realizar los mismos cálculos que en el ejemplo anterior para conocer la distancia entre P_1 y P_3.

	$kmer$	$rank(P_1)$	$rank(P_3)$	$\|rank(P_1) - rank(P_3)\|$	$\in P_3$
	TTTTT	0	15	15	false
	ATTTT	1	15	14	false
	AAAAA	2	15	13	false
	AAAAT	3	15	12	false
	TATTT	4	15	11	false
	AAATA	5	15	10	false
	AAATT	6	15	9	false
	AATTT	7	15	8	false
	TTTTA	8	15	7	false
	ATATT	9	15	6	false
sum				105	

Cuadro 2: Cálculo desglosado de $distance(P_1, P_3)$.

Ninguno de los 10 5-meros de P_1 se encuentra en P_3 (valor $false$ en la última columna de la tabla). También puede observarse en la penúltima columna que el valor de penalización decrece conforme bajamos de posición en el orden de P_1. Finalmente, el valor de la distancia sería en este caso $distance(P_1, P_3) = 105/150 = 0{,}7$.

A tenor de las dos distancias calculadas, la conclusión es que, P_1 es más próximo a P_2 que a P_3. Dado el profile P_1, el profile de menor distancia sería P_2, y se le asignaría como profileId *drosophila melanogaster*.

5. Práctica a entregar

El programa a desarrollar en esta práctica tiene como objetivo leer varios ficheros profile que se almacenarán (todos menos el primero) en un array dinámico de objetos `Profile`. Entonces se calculará la distancia del primer profile con todos los profiles del array dinámico. Finalmente, el programa debe mostrar el nombre del fichero que contenga el profile más cercano o lejano al primero.

Para la elaboración de la práctica, dispone de una serie de ficheros que se encuentran en Prado en el fichero `Kmer3_nb.zip` o en el repositorio de `github`. Configure su proyecto en Netbeans de forma similar a cómo se hizo en prácticas anteriores.

5.1. Módulos del proyecto

En esta práctica solo necesita hacer modificaciones en la clase `Profile` y en `main.cpp`. Las clases `Kmer` y `KmerFreq` permanecen inalteradas respecto a la práctica Kmer2.

- En la clase `Profile`, solo se necesita implementar el nuevo método `getDistance()`, declarado en el fichero `Profile.h`. Revise, como siempre, si el prototipo proporcionado para este método, es correcto, y corríjalo en su caso. Implemente también tal método en el fichero `Profile.cpp`.
- El módulo `main.cpp` tiene por objetivo leer diferentes ficheros profile `.prf`, con el fin de calcular la distancia del primero de ellos a cada uno de los demás y hallar el profile con mínima/máxima distancia, según se haya especificado en la línea de comandos el parámetro `min` o bien `max`.

  La sintaxis de ejecución del programa desde un terminal es la siguiente:

```
Linux> kmer3 [-t min | max] <file1.prf> <file2.prf> [<file3.prf> ...
↪  <filen.prf>]
```

  Notación: Los `[]` indican que no es obligatorio, es opcional y `|` indica alternativa; en este caso el literal `min` o bien `max`.

  Debemos tener en cuenta los siguientes puntos:

  1. Todos los datos que necesita el programa se le pasan desde la línea de órdenes.
  2. `kmer3` debe calcular la distancia del profile almacenado en <file1.prf> a cada uno de los profiles que aparecen a continuación en la línea de comandos.
  3. El primer parámetro del programa (que es opcional) es el literal `-t` seguido por la cadena `min` o bien por `max`.
     - `-t min` : el programa debe determinar el profile a menor distancia del primero y obtener su identificador.
     - `-t max` : el programa debe determinar el profile de máxima distancia del primero y obtener su identificador.
  4. Si no se introduce el parámetro `-t xxx`, el programa se ejecuta de forma equivalente a si se hubiese introducido `-t min` (opción por defecto).
  5. El programa debe recibir obligatoriamente al menos dos ficheros profile (el número de ficheros adicionales es indeterminado).
  6. El programa debe almacenar la lista de profiles contenidos en los ficheros `.prf` en un vector dinámico de objetos profile. Debe hacer una gestión correcta de la memoria: reservar, usar y liberar.

5.2. Ejemplos de ejecución

En la carpeta `Genomes` del repositorio puede encontrar varios ficheros `.prf` con los que podrá hacer diferentes pruebas. Se detallan aquí algunos ejemplos.

Recuerde también que en cada práctica, se proporcionan una serie de ficheros de tests con extensión `.test`, que contienen tests de integración, y que pueden usarse con la script `runTests.sh` para ejecutar automáticamente tales tests de integración.

Ejemplo 3 *Faltan argumentos para la ejecución del programa. Se necesitan al menos dos ficheros profile.*

```
Linux>kmer3
Linux>kmer3 ../Genomes/5pairsDNA.prf
```

La salida del programa sería la siguiente:

```
ERROR in Kmer3 parameters
Run with the following parameters:
kmer3 [-t min|max] <file1.prf> <file2.prf> [ ... <filen.prf>]

Parameters:
-t min | -t max: search for minimun distances or maximum distances (-t min by
↪  default)
<file1.prf>: source profile file for computing distances
<file2.prf> [ ... <filen.prf>]: target profile files for computing distances

This program computes the distance from profile <file1.prf> to the rest
```

Ejemplo 4 *Algún parámetro no es válido. En el primer caso el párametro* `-f` *no es válido. En el segundo caso no es válido* `-t MIN`*, ya que debe ser* `-t min`*.*

```
Linux>kmer3 -f ../Genomes/5pairsDNA.prf
↪  ../Genomes/5pairsDNA.prf
Linux>kmer3 -t MIN ../Genomes/5pairsDNA.prf
↪  ../Genomes/5pairsDNA.prf
```

La salida del programa sería de nuevo la siguiente:

```
ERROR in Kmer3 parameters
Run with the following parameters:
kmer3 [-t min|max] <file1.prf> <file2.prf> [ ... <filen.prf>]

Parameters:
-t min | -t max: search for minimun distances or maximum distances (-t min by
↪  default)
<file1.prf>: source profile file for computing distances
<file2.prf> [ ... <filen.prf>]: target profile files for computing distances

This program computes the distance from profile <file1.prf> to the rest
```

Ejemplo 5 *Calcular la distancia mínima entre* `human1.prf` *y otros 3 profiles suministrados.*

```
Linux>kmer3 ../Genomes/human1.prf ../Genomes/human2.prf
↪  ../Genomes/fly1.prf ../Genomes/worm1.prf
```

La salida del programa sería la siguiente:

```
Distance to ../Genomes/human2.prf: 0.264996
Distance to ../Genomes/fly1.prf: 0.29156
Distance to ../Genomes/worm1.prf: 0.330618
Nearest profile file: ../Genomes/human2.prf
Identifier of the nearest profile: homo sapiens
```

Ejemplo 6 *Calcular la distancia máxima del fichero profile* `prep_xx.prf` *con varios profiles cuyo género o especie es conocida.*

```
Linux>kmer3 -t max ../Genomes/prep_xx.prf ../Genomes/prep_chpz.prf
↪  ../Genomes/prep_fly.prf ../Genomes/prep_hmn.prf ../Genomes/prep_covid.prf
```

La salida es como sigue:

```
Distance to ../Genomes/prep_chpz.prf: 0.215385
Distance to ../Genomes/prep_fly.prf: 0.0866667
Distance to ../Genomes/prep_hmn.prf: 0.7
Distance to ../Genomes/prep_covid.prf: 0.625
Farthest profile file: ../Genomes/prep_hmn.prf
Identifier of the farthest profile: homo sapiens
```

6. Código para la práctica

6.1. Profile.h

```
#ifndef PROFILE_H
#define PROFILE_H


#include <iostream>
#include "KmerFreq.h"

/**
 * @class Profile
 * @brief It defines a model (profile) for a given biological species. It
 * contains a vector of pairs Kmer-frequency (objects of the class KmerFreq)
 * and an identifier (string) of the profile.
 */
class Profile {
public:

    /**
     * @brief Base constructor. It builds a Profile object with "unknown" as
     * identifier, and an empty vector of pairs Kmer-frequency.
     */
    Profile();

 ...


    /**
     * @brief Gets the distance between this Profile object (\f$P_1\f$) and
     * the given argument object @p otherProfile (\f$P_2\f$).
     * The distance between two Profiles \f$P_1\f$ and \f$P_2\f$ is
     * calculated in the following way:
     *
     * \f$d = \frac{ \sum_{kmer_i(P_1)} | rank_{kmer_i(P_1)}^{P_1} -
     * rank_{kmer_i(P_1)}^{P_2} | }{size(P_1) * size(P_2) }\f$,
     *
     * where \f$kmer_i(p_j)\f$ is the kmer \f$i\f$ of the Profile \f$p_j,
     * j \in \{1, 2\}\f$ and \f$rank_{kmer_i(p_j)}^{p_k}\f$ is the ranking
     * of the kmer \f$i\f$ of the Profile \f$p_j, j \in \{1, 2\}\f$ in the
     * Profile \f$p_k\f$.
     *
     * The rank of a kmer is the position in which it
     * appears in the list of KmerFreq. We consider 0 as the
     * first position (rank equals to 0). When calculating
     * \f$rank_{kmer_i(P_1)}^{P_2}\f$, if the kmer \f$kmer_i(P_1)\f$
     * does not appears in the Profile \f$P_2\f$ we consider that the rank
     * is equals to the size of Profile \f$P_2\f$.
     * Query method
     * @param otherProfile A Profile object. Input parameter
     * @pre The list of kmers of this and otherProfile should be ordered in
     * decreasing order of frequency. This is not checked in this method.
     * @throw Throws a std::invalid_argument exception if the implicit object
     * (*this) or the argument Profile object are empty, that is, they do not
     * have any kmer.
     * @return The distance between this Profile object and the given
     * argument @p otherProfile.
     */
    double getDistance(Profile otherProfile);

 ...

 private:
    static const int DIM_VECTOR_KMER_FREQ = 2000; ///< The capacity of the array _vectorKmerFreq
    static const std::string MAGIC_STRING_T; ///< A const string with the magic string for text files

    std::string _profileId; ///< profile identifier
    KmerFreq _vectorKmerFreq[DIM_VECTOR_KMER_FREQ]; ///< array of KmerFreq
    int _size; ///< Number of used elements in _vectorKmerFreq
};

#endif /* PROFILE_H */
```

6.2. main.cpp

```
/*
 * Metodología de la Programación: Kmer3
 * Curso 2023/2024
 */

/**
 * @file main.cpp
 * @author Silvia Acid Carrillo <acid@decsai.ugr.es>
 * @author Andrés Cano Utrera <acu@decsai.ugr.es>
 * @author Luis Castillo Vidal <L.Castillo@decsai.ugr.es>
 * @author Javier Martínez Baena <jbaena@ugr.es>
 *
 * Created on 16 November 2023, 14:15
 */
```

```
#include <iostream>

#include "Profile.h"

using namespace std;


/**
 * Shows help about the use of this program in the given output stream
 * @param outputStream The output stream where the help will be shown (for example,
 * cout, cerr, etc)
 */
void showEnglishHelp(ostream& outputStream) {
    outputStream << "ERROR in Kmer3 parameters" << endl;
    outputStream << "Run with the following parameters:" << endl;
    outputStream << "kmer3 [-t min|max] <file1.prf><file2.prf> [ ... <filen.prf>]" << endl;
    outputStream << endl;
    outputStream << "Parameters:" << endl;
    outputStream << "-t min | -t max: search for minimun distances or maximum distances (-t min by default)"
      << endl;
    outputStream << "<file1.prf>: source profile file for computing distances" << endl;
    outputStream << "<file2.prf> [ ... <filen.prf>]: target profile files for computing distances" << endl;
    outputStream << endl;
    outputStream << "This program computes the distance from profile <file1.prf> to the rest" << endl;
    outputStream << endl;
}

/**
 * This program reads an undefined number of Profile objects from the set of
 * files passed as parameters to main(). All the Profiles object, except the
 * first one, must be stored in a dynamic array of Profile objects. Then,
 * for each Profile in the dynamic array, this program prints to the
 * standard output the name of the file of that Profile and the distance from
 * the first Profile to the current Profile.
 * Finally, the program should print in the standard output, the name of
 * the file with the Profile with the minimum|maximum  distance to the Profile
 * of the first file and its profile identifier.
 *
 * At least, two Profile files are required to run this program.
 *
 * This program assumes that the profile files are already normalized and
 * sorted by frequency. This is not checked in this program. Unexpected results
 * will be obtained if those conditions are not met.
 *
 * Running sintax:
 * > kmer3 [-t min|max] <file1.prf><file2.prf> [  ... <filen.prf>]
 *
 * Running example:
 * > kmer3 ../Genomes/human1.prf ../Genomes/worm1.prf ../Genomes/mouse1.prf
Distance to ../Genomes/worm1.prf: 0.330618
Distance to ../Genomes/mouse1.prf: 0.224901
Nearest profile file: ../Genomes/mouse1.prf
Identifier of the nearest profile: mus musculus
 *
 * Running example:
 * > kmer3 -t max ../Genomes/human1.prf ../Genomes/worm1.prf ../Genomes/mouse1.prf
Distance to ../Genomes/worm1.prf: 0.330618
Distance to ../Genomes/mouse1.prf: 0.224901
Farthest profile file: ../Genomes/worm1.prf
Identifier of the farthest profile: worm
 */
int main(int argc, char* argv[]) {

    // Process the main() arguments

    // Allocate a dynamic array of Profiles

    // Load the input Profiles

    // Calculate and print the distance from the first Profile to the rest

    // Print name of the file and identifier that takes min|max distance to the first one

    // Deallocate the dynamic array of Profile
    return 0;
}
```