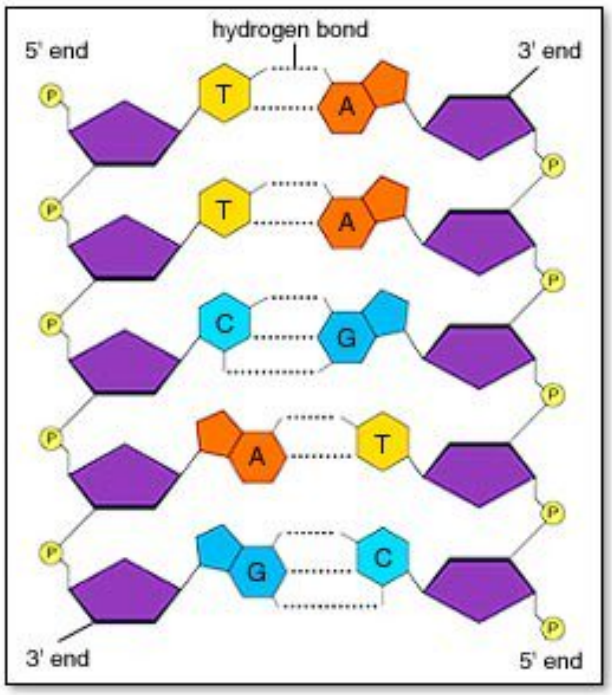

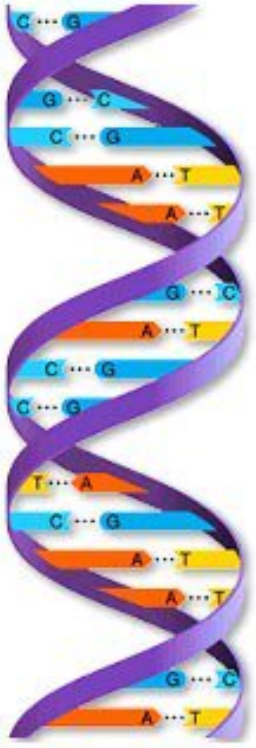

Guion de prácticas 5

Contador Kmer

Mayo 2019

UNIVERSIDAD DE GRANADA

ETSIIT Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación

DECSAI

Metodología de la Programación

DGIM

Curso 2018/2019

Índice

1. Introducción

Esta práctica va a dedicarse a la identificación de secuencias de ADN de origen desconocido, intentando encontrar cuál es la especie conocida más próxima al ADN desconocido. Para ello, el programa deberá analizar, por cada especie conocida, una muestra de ADN proveniente de múltiples individuos de la misma especie, contando, en total, cada kmer aparecido en todos los individuos. Este conteo de kmer de múltiples individuos de una misma especie podría considerarse el perfil genético de esa especie y será representado con un objeto de la clase **SecuenciasKmer** y almacenado en disco. Una vez analizadas todas las secuencias conocidas y almacenados en disco sus perfiles, el programa deberá leer y contar los kmer de una secuencia de ADN desconocida y compararlas con los perfiles de especies conocidas, debiendo indicar el perfil más próximo a la secuencia desconocida. En este caso, las novedades de programación de esta práctica son las siguientes:

1. Los ficheros de datos en la carpeta **data** tienen extensión .dna y son secuencias reales de nucleótidos de diferentes individuos de una misma especie: humano1.dna, humano2.dna. Estas secuencias son de tamaño 10000 cada una. Los ficheros de datos asociados a secuencias de kmers como los estudiados en las prácticas anteriores tendrán extensión .kmer

2. La principal novedad es la clase **ContadorKmer** que se implementa sobre una matriz bidimensional dinámica, con un número de columnas fijo (dado por el tamaño del kmer y el número de caracteres válidos) y un número de filas variable, donde cada fila almacena un perfil conocido. Esta clase aparece descrita en la Figura 1. La matriz bidimensional almacenará un valor entero que será interpretado como la frecuencia de ocurrencia del kmer (biunívocamente asociado al número de columna) y un perfil biológico determinado por la fila de la matriz. Así,

   $$contador[f][c]$$

   representa la frecuencia de ocurrencia del kmer asociado a la columna $c = Indice(Kmer)$ del perfil biológico f.

$$BASE = |VALIDOS|$$

$$NKMER = BASE^K$$

$$Indice : Kmer \longrightarrow \{0, 1, \ldots NKMER - 1\}$$

$$\forall km \in Kmer, Indice(km) = \sum_{i=0}^{K-1} posicion(km[K-i-1], VALIDOS) * BASE^i$$

$$IndiceInverso : \{0, 1, \ldots NKMER - 1\} \longrightarrow Kmer$$

$$\forall km \in Kmer, IndiceInverso(Indice(km)) = km$$

Por ejemplo, sea $K = 3$, $VALIDOS = \{\_, a, c, t, g\}$, $BASE = 5$, $NKMER = 125$ y $km = \text{“}tga\text{”}$

$$Indice(\text{“}tga\text{”}) = posicion('a', VALIDOS) * 5^0 +$$

$$+posicion('g', VALIDOS) * 5^1 +$$

$$+posicion('t', VALIDOS) * 5^2$$

$$Indice(\text{“}tga\text{”}) = 1 * 1 + 4 * 5 + 3 * 5^2 = 96$$

$$IndiceInverso(96) = \text{“}tga\text{”}$$

3. La clase **SecuenciasKmer**, mostrada en la Figura 2 también sufre algunos cambios, que aparecen descritos con todo detalle en el fichero .h correspondiente. Se abandonan algunos métodos y aparecen otros nuevos. Uno de los métodos nuevos permite comparar dos secuencias de Kmer y calcular una medida de diferenciación, el cual se encuentra implementado en **SecuenciasKmer.cpp**.

4. Ambas clases estarán basadas en el uso de memoria dinámica y deberán, por tanto, tener definidos constructor de copia, destructor y operador de asignación sobrecargado.

```
/**
 * @file ContadorKmer.h
 * @author DECSAI
*/
#ifndef CONTADORKMER_H
#define CONTADORKMER_H

#include "Kmer.h"
#include "SecuenciasKmer.h"

class ContadorKmer {
public:
  static const std::string INI_VALIDOS;
  static const int INI_K;

  ContadorKmer();
  ContadorKmer(const std::string & ok, int k);
  ContadorKmer(const ContadorKmer & orig);
  ~ContadorKmer();
  ContadorKmer& operator=(const ContadorKmer& orig);
  int getNumPerfiles() const;
  long getNumKmer() const;
  void addPerfil();
  void addPerfil(SecuenciasKmer perfil);
  bool addKmer(int perfil, const Kmer &tg);
  SecuenciasKmer getSecuenciasKmer(int perfil, int frecmin) const;
  bool calcularFrecuenciasKmer(int perfil, const char* nfichero);
private:
  int ** _contador,
   _nperfiles;
  std::string _VALIDOS;
  int _K;
  long _NKMER;

  void reservarMemoria(int npersonas);
  void liberarMemoria();
  void copiar(const ContadorKmer & otro);
  Kmer normalizaKmer(const Kmer & ng) const;
  long getIndiceKmer(const Kmer &t) const;
  Kmer getIndiceInversoKmer(long i) const;
};
#endif
```

Figura 1: Fichero **ContadorKmer.h**

```
/**
* @file SecuenciasKmer.h
* @author MP
*/
#ifndef SECUENCIASKMER_H
#define SECUENCIASKMER_H
#include "Kmer.h"

const std::string MAGIC="MP-KMER";

class SecuenciasKmer{
public:
  SecuenciasKmer();
  SecuenciasKmer(long nkm);
  SecuenciasKmer(const SecuenciasKmer& orig);
  ~SecuenciasKmer();
  SecuenciasKmer& operator=(const SecuenciasKmer& orig);
  Kmer getPosicion(long p) const;
  void setPosicion(long p, const Kmer & km);
  inline long getSize() const { return _nKmer; };
  long findKmer(const std::string& km) const;
  void ordenar();
  bool saveFichero(const char *fichero) const;
  bool loadFichero(const char *fichero);
  double distancia(const SecuenciasKmer & d) const;

  friend std::ostream & operator<<(std::ostream & os, const SecuenciasKmer & s);
  friend std::istream & operator>>(std::istream & os, SecuenciasKmer & s);

private:
  Kmer* _conjunto;
  int _nKmer;

  void reservarMemoria(long n);
  void liberarMemoria();
  void copiar(const SecuenciasKmer& otro);
};

std::ostream & operator<<(std::ostream & os, const SecuenciasKmer & i);
std::istream & operator>>(std::istream & is, SecuenciasKmer & i);
#endif
```

Figura 2: Fichero **SecuenciasKmer.h**

2. Práctica a entregar: aprender a clasificar kmers

2.1. Descripción y material disponible

Descargar el fichero **proyecto.zip** contiene un proyecto de NetBeans con la estructura de la Figura 3.

Se pide implementar los cambios comentados anteriormente (más detalles en **SecuenciasKmer.h** y **ContadorKmer.h**). El mismo programa puede ejecutarse de dos formas diferentes, una para aprender perfiles genéticos y otra para identificar secuencias de ADN desconocidas, dependiendo de la llamada al programa (parámetros de main). En ambos casos el programa utilizará datos contenidos en ficheros y deberá reaccionar adecuadamente a los errores de apertura, lectura o escritura en estos ficheros.

- Ejemplo de llamada en modo “aprender”. Comienza con el parámetro “-l” (learn), a continuación debe aparecer un fichero kmer. Si el fichero existe, se carga en memoria, si no, deberá crearse uno completamente nuevo. A continuación debe aparecer **al menos** un fichero dna. Por cada fichero dna se deben contar sus kmers, añadiendolos a los kmers leídos de disco, si es que se hubiese leído alguno. Al final, la secuencia de kmer resultante, ordenada de mayor a menor frecuencia, y sin contener kmers con frecuencias nulas, se debe grabar en disco con el mismo nombre especificado al comienzo.

```
mp1819practicafinal -f <fichero.kmer> <fichero_1.dna> [<fichero_2.dna> ... <fichero_n.dna>]
```

```
./
├── build
├── data
│   ├── ebola.dna
│   ├── gusano1.dna
│   ├── gusano2.dna
│   ├── humano1.dna
│   ├── humano2.dna
│   ├── mosca1.dna
│   ├── mosca2.dna
│   ├── pez1.dna
│   ├── raton1.dna
│   ├── raton2.dna
│   ├── unknown1.dna
│   ├── unknown2.dna
│   ├── unknown3.dna
│   └── unknown4.dna
├── dist
├── doc
│   └── Kmer.doxy
├── include
│   ├── Kmer.h
│   ├── SecuenciasKmer.h
│   └── ContadorKmer.h
├── Makefile
├── nbproject
├── src
│   ├── Kmer.cpp
│   ├── SecuenciasKmer.cpp
│   ├── ContadorKmer.cpp
│   └── main.cpp
└── zip
```

Figura 3: Estructura del proyecto de la práctica 5

- Ejemplo de llamada en modo “clasificar”. Comienza con el parámetro “-c” (classify), a continuación debe aparecer un fichero dna. Si el fichero existe, se carga en memoria, si no, el programa terminará con un error. A continuación debe aparecer **al menos** un fichero kmer. Se deben contar los kmer del fichero dna, compararlos con los kmer leidos de disco (con la función **distancia** de la clase **SecuenciasKmer**). Finalmente el programa deberá indicar cuál es el fichero de kmer leído previamente que se encuentra a una menor distancia.

```
mp1819practicafinal -c <fichero.dna> <fichero_1.kmer> [<fichero_2.kmer> ... <fichero_n.kmer>]
```

2.2. Validación de la práctica

```
(SHELL)$ dist/Debug/GNU-Linux/mp1819practicafinal -l data/gusano.kmer data/gusano1.dna data/gusano2.dna
Abriendo fichero data/gusano.kmer
ERROR abriendo fichero data/gusano.kmer
Abriendo fichero data/gusano1.dna
Procesados 9996 Kmers
Abriendo fichero data/gusano2.dna
Procesados 9996 Kmers
Guardando fichero data/gusano.kmer
Escribiendo 1023 Kmers

(SHELL)$ head data/gusano.kmer
MP-KMER
1023
ttttt 270
aaaaa 263
atttt 171
aaaat 162
aaatt 148
gaaaa 147
aattt 138
tttta 124

(SHELL)$ dist/Debug/GNU-Linux/mp1819practicafinal -l data/mosca.kmer data/mosca1.dna
Abriendo fichero data/mosca.kmer
ERROR abriendo fichero data/mosca.kmer
Abriendo fichero data/mosca1.dna
Procesados 9996 Kmers
Guardando fichero data/mosca.kmer
Escribiendo 1021 Kmers

(SHELL)$ dist/Debug/GNU-Linux/mp1819practicafinal -l data/mosca.kmer data/mosca2.dna
Abriendo fichero data/mosca.kmer
Leyendo 1021 Kmers
Abriendo fichero data/mosca2.dna
Procesados 9996 Kmers
Guardando fichero data/mosca.kmer
Escribiendo 1024 Kmers

(SHELL)$ head data/mosca.kmer
MP-KMER
1024
tgctg 63
ctgct 53
gctgc 53
aaaca 48
gctgg 48
gagga 47
aacaa 46
aaata 44
```

Figura 4: Secuencia de ejecuciones de aprendizaje de los perfiles genéticos de dos especies. El perfil **gusano.kmer** se obtiene en una única ejecución, mientras que el perfil **mosca.kmer** se obtiene en dos ejecuciones sucesivas.

2.3. Entrega de la práctica

Se debe de entregar através de **PRADO** un fichero zip, **proyecto.zip** con la estructura de directorios ya expuesta en la Figura 3. Observese que

```
(SHELL)$ dist/Debug/GNU-Linux/mp1819practicafinal -c data/unknown1.dna data/gusano.kmer data/humano.kmer
data/raton.kmer data/mosca.kmer data/pez.kmer data/ebola.kmer
Abriendo fichero data/unknown1.dna
Procesados 9997 Kmers
Abriendo fichero data/gusano.kmer
Leyendo 1023 Kmers
Abriendo fichero data/humano.kmer
Leyendo 1015 Kmers
Abriendo fichero data/raton.kmer
Leyendo 1017 Kmers
Abriendo fichero data/mosca.kmer
Leyendo 1024 Kmers
Abriendo fichero data/pez.kmer
Leyendo 977 Kmers
Abriendo fichero data/ebola.kmer
Leyendo 1014 Kmers
[1, data/gusano.kmer]= 0.295075
[2, data/humano.kmer]= 0.155519
[3, data/raton.kmer]= 0.179153
[4, data/mosca.kmer]= 0.253456
[5, data/pez.kmer]= 0.244955
[6, data/ebola.kmer]= 0.223391

Sugerencia = data/humano.kmer

(SHELL)$ dist/Debug/GNU-Linux/mp1819practicafinal -c data/unknown2.dna data/gusano.kmer
data/humano.kmer data/raton.kmer data/mosca.kmer data/pez.kmer data/ebola.kmer
Abriendo fichero data/unknown2.dna
Procesados 9997 Kmers
Abriendo fichero data/gusano.kmer
Leyendo 1023 Kmers
Abriendo fichero data/humano.kmer
Leyendo 1015 Kmers
Abriendo fichero data/raton.kmer
Leyendo 1017 Kmers
Abriendo fichero data/mosca.kmer
Leyendo 1024 Kmers
Abriendo fichero data/pez.kmer
Leyendo 977 Kmers
Abriendo fichero data/ebola.kmer
Leyendo 1014 Kmers
[1, data/gusano.kmer]= 0.232718
[2, data/humano.kmer]= 0.231177
[3, data/raton.kmer]= 0.190038
[4, data/mosca.kmer]= 0.282785
[5, data/pez.kmer]= 0.179083
[6, data/ebola.kmer]= 0.262668

Sugerencia = data/pez.kmer
```

Figura 5: Secuencia de ejecuciones de clasificación de las secuencias **unknown1.dna** y **unknown2.dna**. La primera es correctamente clasificada como humano. La segunda es de ratón, pero es clasificada incorrectamente como pez.

las carpetas **dist** y **build** deben estar vacías. Los comandos para llevar a cabo el empaquetamiento son los siguientes (se recomienda tenerlos integrados en Netbeans en la herramienta “Send To...”):

```
cd <carpeta-del-proyecto>
rm -rf zip/* dist/* build/* doc/html doc/latex
zip -r zip/proyecto.zip * -x nbproject/private/*
```

```
(SHELL)$ dist/Debug/GNU-Linux/mp1819practicafinal -l data/mosca.kmer data/noexiste.dna data/mosca2.dna
Abriendo fichero data/mosca.kmer
ERROR abriendo fichero data/mosca.kmer
Abriendo fichero data/noexiste.dna
Error al abrir el fichero data/noexiste.dna
Abriendo fichero data/mosca2.dna
Procesados 9996 Kmers
Guardando fichero data/mosca.kmer
Escribiendo 1021 Kmers

(SHELL)$ dist/Debug/GNU-Linux/mp1819practicafinal -c data/unknown2.dna data/gusano.kmer data/humano.kmer
data/noexiste.kmer data/raton.kmer data/mosca.kmer data/noexiste.kmer data/pez.kmer data/ebola.kmer
Abriendo fichero data/unknown2.dna
Procesados 9997 Kmers
Abriendo fichero data/gusano.kmer
Leyendo 1023 Kmers
Abriendo fichero data/humano.kmer
Leyendo 1015 Kmers
Abriendo fichero data/noexiste.kmer
ERROR abriendo fichero data/noexiste.kmer
Abriendo fichero data/raton.kmer
Leyendo 1017 Kmers
Abriendo fichero data/mosca.kmer
Leyendo 1021 Kmers
Abriendo fichero data/noexiste.kmer
ERROR abriendo fichero data/noexiste.kmer
Abriendo fichero data/pez.kmer
Leyendo 977 Kmers
Abriendo fichero data/ebola.kmer
Leyendo 1014 Kmers
[1, data/gusano.kmer]= 0.232718
[2, data/humano.kmer]= 0.231177
[3, data/raton.kmer]= 0.190038
[4, data/mosca.kmer]= 0.303017
[5, data/pez.kmer]= 0.179083
[6, data/ebola.kmer]= 0.262668

Sugerencia = data/pez.kmer

(SHELL)$ dist/Debug/GNU-Linux/mp1819practicafinal data/unknown2.dna data/gusano.kmer data/humano.kmer
data/noexiste.kmer data/raton.kmer data/mosca.kmer data/noexiste.kmer data/pez.kmer data/ebola.kmer
ERROR en la llamada
   kmer [-l f.kmer fa.dna [fn.dna]] [-c f.dna fa.kmer [fn.kmer]]

(SHELL)$ dist/Debug/GNU-Linux/mp1819practicafinal
ERROR en la llamada
   kmer [-l f.kmer fa.dna [fn.dna]] [-c f.dna fa.kmer [fn.kmer]]

(SHELL)$ dist/Debug/GNU-Linux/mp1819practicafinal -c data/noexiste.dna data/gusano.kmer data/humano.kmer
data/noexiste.kmer data/raton.kmer data/mosca.kmer data/noexiste.kmer data/pez.kmer data/ebola.kmer
Abriendo fichero data/noexiste.dna
Error al abrir el fichero data/noexiste.dna
```

Figura 6: Secuencia de llamadas que contienen algunos errores y las respuestas que debe dar el programa.