Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas
Estructuras de Datos
Proyecto del curso, 2015-10
Segunda Parte

1 Descripción del problema

En general, la información genética de un organismo se encuentra contenida en su genoma. Para manipular la información del genoma de un organismo cualquiera, se usan secuencias de información conocidas como códigos genéticos. Estos códigos genéticos están compuestos de letras que representan las cuatro bases nitrogenadas del ácido desoxirribonucleico (ADN) o del ácido ribonucleico (ARN). Mientras en el caso del ADN las bases son adenina (A), timina (T), guanina (G) y citosina (C); para el ARN las bases son adenina (A), uracilo (U), guanina (G) y citosina (C).

Aunque hay muchos avances relacionados con el descubrimiento de los códigos genéticos de los organismos del planeta, mucho trabajo aún queda por hacer. Para guardar la información que se conoce (y señalar la que se desconoce) se usa un formato de códigos genéticos conocido como FASTA.

2 Descripción del proyecto

El objetivo del presente proyecto es construir un sistema que permita algunas manipulaciones sencillas sobre archivos que contienen códigos genéticos

2.1 Formato FASTA

Los archivos en formato FASTA (extensión .fa) son basados en texto. Pueden contener uno o varios códigos genéticos que componen un genoma. Cada código genético empieza con una línea de texto que tiene el formato:

```
>[descripcion de la secuencia]
```

Debe notarse que siempre empieza con el caracter '>' y justo después se presenta una cadena no vacía que describe la secuencia. Las siguientes líneas (hasta la siguiente secuencia o hasta el final del archivo) contienen los datos que describen el código genético. Estas líneas están justificadas y todas tienen un ancho constante, con la posible excepción de la última línea. Un ejemplo de un archivo FASTA puede ser:

```
>Full_SEQUENCE
CTCCGGTGAGAAATTTTGGGATGTATCAAATCACGGTCCTACTAC
TTACCGCCAACACGAGCCGGAACCCCCTAGATCAATTCATGCTTT
TCCCTTCACGCGAAGGAGTCGGAAGTGATCTGTATGAAGCTATTA
CCCTAGGTGGCCACACCTAC
>Incomplete_sequence
URDNSHVNKSCUANADDDVMMDVHRSRGNUNTSBTTMCGNBUUBTUMNAYKSTRYMVBWVNSC
SUUDDYVBCGNDRUURUBDHVGTYAVAVVSAKHUTSWCUBUUMSMDVDKBWRHHBHDBDWUDC
CAUWBRNYSTVTBBCKGDCMSNTKBNTRTNYRNWNWNVCSCNDDWHUHWRRTUMWNKHUBDWA
DUNYUVKHNKSDCYAVARTUWDVSTDVVMMVUHVBCDGVBSKKBHKSVHHGKSAADDVBRKSS
UKURTKTNAWYBKVRRBGKGBMGKUCGSMDTDTKCKVUYNVDWRWNBRCGKDWWUNBYMTGBN
YTAHCUTAGNBKWWGNDRKMNCDVDTKNRGVBRVMKGBKUBGBRHVUSTBCGDV
```

El signifcado de cada letra se explica en la Tabla 1.

2.2 Segunda entrega

A continuación se describen los componentes individuales que conforman la segunda entrega del presente proyecto. El sistema se implementará como una aplicación que recibe comandos textuales.

Código	Significado
A	Adenina
C	Citosina
G	Guanina
T	Timina
U	Uracilo
R	A o G
Y	C, T o U
K	G, T o U
M	A o C
S	C o G
W	A, T o U
B	C, G, T o U
D	A, G, T o U
H	A, C, T o U
V	A, C o G
N	A, C, G, T o U
X	Máscara
-	Espacio de longitud indeterminada

Table 1: Relación de cada código del formato FASTA con su significado biológico (tomada de http://www.ncbi.nlm.nih.gov/blast/fasta.shtml).

2.2.1 Componente 1: Resumen de la información de un genoma

La primera entrega de este proyecto debe estar completamente terminada y funcional (ver documento del enunciado de la primera entrega).

2.2.2 Componente 2: compresión y decompresión de archivos FASTA

Objetivo: Utilizar el algoritmo de codificación de Huffman para comprimir y descomprimir archivos FASTA.

Un principio muy importante alrededor de la compresión de datos es codificar cada símbolo de un mensaje usando la cantidad mínima posible de bits. Por ejemplo, si un mensaje estuviera escrito en lenguaje ASCII, el cual tiene 256 símbolos diferentes, la cantidad mínima de bits por símbolo requerida sería de 8. Otro principio esencial es que, aquellos símbolos que aparecen más frecuentemente en un mensaje, sería útil codificarlos con menos bits que aquellos menos frecuentes, de tal forma que el mensaje comprimido ocupe el menor espacio posible.

La codificación de Huffman[1] tiene en cuenta los dos principios anteriores. Provee una forma de representar cada símbolo de un mensaje con la menor cantidad posible de bits, y al mismo tiempo permite codificar cada símbolo con una cantidad variable de bits, dependiendo de su frecuencia de ocurrencia en el mensaje. Para realizar el proceso de codificación y decodificación, se utiliza un árbol de Huffman. Éste es un árbol binario que representa una codificación de un conjunto de símbolos óptima: el símbolo que tenga una frecuencia más alta (i.e. el que más se repita) se representa con un número pequeño de bits. Un símbolo poco frecuente se representa con más bits. Un ejemplo de un árbol de Huffman se muestra en la Figura 1.

En este árbol, cada nodo **hoja** almacena un símbolo del lenguaje utilizado en un mensaje y la cantidad de veces que se encuentra dicho símbolo en el mensaje (valor entre paréntesis), al cual llamaremos "frecuencia". Los **nodos intermedios y la raíz** no contienen símbolos, sino sólo un valor correspondiente a la suma de las frecuencias de sus nodos hijos. Note que los símbolos se encuentran exclusivamente en las hojas y cada símbolo aparece una sola vez en todo el árbol (no se repiten en el árbol).

Este árbol se utiliza para codificar y decodificar los símbolos de un mensaje. Cada arista entre un nodo padre y sus hijos se etiqueta con un "0" para el hijo izquierdo y con un "1" para el hijo derecho. Cada camino entre el nodo raíz y las hojas se puede representar como la concatenación de las etiquetas de las aristas que lo conforman. Esta representación corresponde a la codificación para cada uno de los símbolos. En este caso, por ejemplo, el

[1] http://en.wikipedia.org/wiki/Huffman_coding

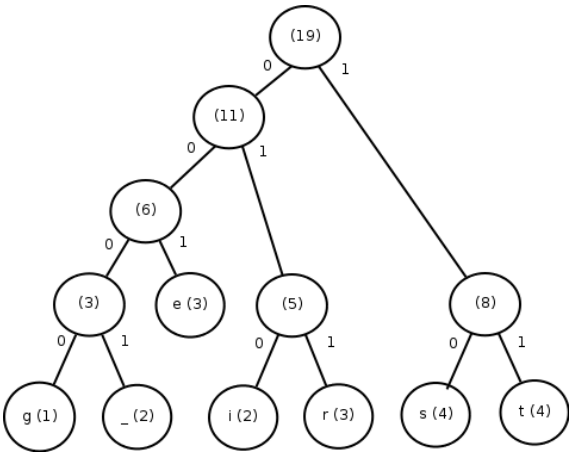


Figure 1: Árbol de Huffman.

símbolo que más se repite es la “t” (4 veces), y su codificación es “11”, el cual corresponde a las etiquetas de las aristas del camino desde la raíz hasta el nodo con el símbolo “t"

Otros ejemplos de codificación son:

- La “s” se codifica como “10”.
- El “ _” se codifica como “0001”.
- La “r” se codifica como “011”.

Por ejemplo, la siguiente matriz de datos (imagen donde cada fila empieza y termina con el símbolo '+'):

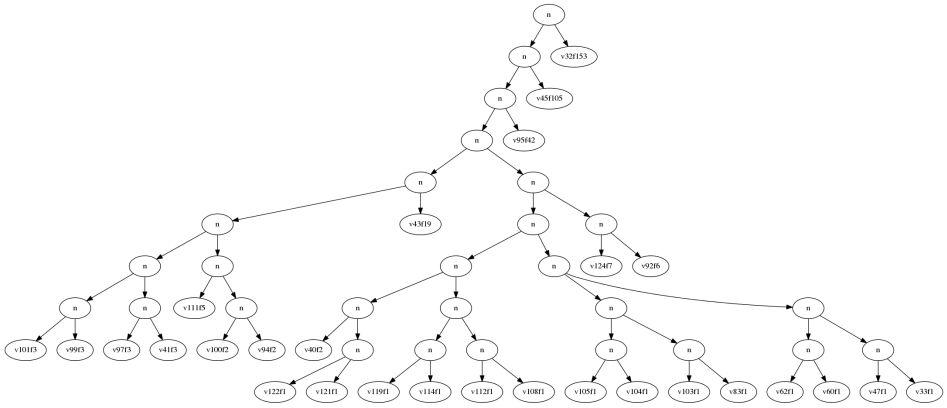
```
+-----------------------------------+
+ ________________________________  +
+ < Soy capaz de hacer el codigo! > +
+ --------------------------------  +
+         \   ^__^                  +
+          \  (oo)\_______          +
+             (__)\       )\/\      +
+                 ||----w |         +
+                 ||     ||         +
+-----------------------------------+
```

Tiene las siguientes características:

- Ancho de la imagen (W) = 36.
- Alto de la imagen (H) = 10.
- Valor (intensidad) máximo posible de la imagen (M) = 255.
- Los caracteres (que hacen las veces de intensidades) y sus frecuencias son:
  - ASCII = 32, caracter = ' ', frecuencia = 153
  - ASCII = 33, caracter = '!', frecuencia = 1
  - ASCII = 40, caracter = '(', frecuencia = 2
  - ASCII = 41, caracter = ')', frecuencia = 3
  - ASCII = 43, caracter = '+', frecuencia = 20
  - ASCII = 45, caracter = '-', frecuencia = 105

- ASCII = 47, caracter = '/', frecuencia = 1
- ASCII = 60, caracter = '<', frecuencia = 1
- ASCII = 62, caracter = '>', frecuencia = 1
- ASCII = 83, caracter = 'S', frecuencia = 1
- ASCII = 92, caracter = '\', frecuencia = 6
- ASCII = 94, caracter = '^', frecuencia = 2
- ASCII = 95, caracter = '_', frecuencia = 42
- ASCII = 97, caracter = 'a', frecuencia = 3
- ASCII = 99, caracter = 'c', frecuencia = 3
- ASCII = 100, caracter = 'd', frecuencia = 2
- ASCII = 101, caracter = 'e', frecuencia = 3
- ASCII = 103, caracter = 'g', frecuencia = 1
- ASCII = 104, caracter = 'h', frecuencia = 1
- ASCII = 105, caracter = 'i', frecuencia = 1
- ASCII = 108, caracter = 'l', frecuencia = 1
- ASCII = 111, caracter = 'o', frecuencia = 5
- ASCII = 112, caracter = 'p', frecuencia = 1
- ASCII = 114, caracter = 'r', frecuencia = 1
- ASCII = 119, caracter = 'w', frecuencia = 1
- ASCII = 121, caracter = 'y', frecuencia = 1
- ASCII = 122, caracter = 'z', frecuencia = 1
- ASCII = 124, caracter = '|', frecuencia = 7

- El árbol asociado es (las hojas tiene asociado un valor "vXfY", donde "X" es el código del símbolo y "Y" es su frecuencia):



El objetivo del componente 2 es implementar un árbol de Huffman para codificar y decodificar archivos FASTA. En este contexto, el "mensaje" sería el conjunto de genomas contenidos en un archivo FASTA y los "símbolos" del mensaje serían los diferentes códigos de las bases del genoma. Dado que la secuencia para codificar un mensaje completo es una secuencia de "0"s y "1"s, este mensaje se puede separar en paquetes de 8 bits y guardar en un archivo binario. El archivo binario (de extension "fabin", para el presente proyecto) tiene la siguiente estructura:

$n\ c_1\ f_1 \cdots c_n\ f_n\ n_s\ l_1 s_{11} \cdots s_{1l_1} \cdots l_{n_s} s_{n_s 1} \cdots s_{n_s l_{n_s}} w_1 x_1 binary\_code_1 \cdots w_{n_s} x_{n_s} binary\_code_{n_s}$

donde:

- n es un número entero de 2 bytes que representa la cantidad de bases diferentes presentes en las secuencia cargadas en ese momento en memoria.

- c_i y f_i son dos números entero de 1 y 8 bytes, respectivamente, que representan un código de la base de genoma y su frecuencia asociada (i.e. cuántas veces aparece en todas las secuencias.
- n_s es un número entero de 4 bytes que representa la cantidad de secuencias que hay en el archivo.
- l_i es un número entero de 2 bytes que representa el tamaño del nombre de la i-ésima secuencia.
- s_{ij} es el caracter que se encuentra en la j-ésima posición del nombre de la i-ésima secuencia.
- w_i un número entero de 8 bytes que representa la longitud de la i-ésima secuencia.
- x_i un número entero de 2 bytes que representa la indentación de la i-ésima secuencia.
- $binary\_code_i$ es la secuencia binaria que representa la i-ésima secuencia. Note que si la secuencia no es múltiplo de 8, se debe completar (*pad*) con "0".

Los comandos que usted debe desarrollar son:

- **comando:** `encode <filename.fabin>`
  salida en pantalla:
  (mensaje de error) Could not save loaded sequences into `<filename.fabin>`
  descripción: El comando debe generar el archivo binario con la correspondiente codificación de Huffman en el formato descrito más arriba, almacenándolo en disco bajo el nombre `<filename.fabin>`.
- **comando:** `decode <filename.fabin>`
  salida en pantalla:
  (mensaje de error) Could not load sequences from `<filename.fabin>`
  descripción: El comando debe cargar en memoria las secuencias contenidas en el archivo binario `<filename.fabin>`, que contiene una codificación Huffman de un conjunto de secuencias en el formato descrito más arriba.

2.3 Interacción con el sistema

La interfaz de la aplicación a construir debe ser una consola interactiva donde los comandos correspondientes a los componentes serán tecleados por el usuario. El indicador de línea de comando debe ser el caracter '$'. Para cada comando, se debe incluir una ayuda de uso que indique la forma correcta de hacer el llamado (i.e. el comando '`help <command>`' debe existir, además de los descritos anteriormente). El comando debe presentar en pantalla los mensajes de resultado especificados antes. Los comandos de los componentes deben ensamblarse en un único sistema (es decir, funcionan todos dentro de un mismo programa, no como programas independientes por componentes).

3 Evaluación

Las entregas se harán en la correspondiente actividad de Uvirtual, hasta la media noche del día indicado. Se debe entregar un archivo comprimido (únicos formatos aceptados: .zip, .tar, .tar.gz, .tar.bz2, .tgz) que contenga documentos (único formato aceptado: .pdf) y código fuente (.h, .hxx, .hpp, .c, .cxx, .cpp). Si la entrega contiene archivos en cualquier otro formato, será descartada y no será evaluada, es decir, la nota definitiva de la entrega será de 0 (cero) sobre 5 (cinco).

3.1 Entrega (viernes 1 de mayo de 2015)

Componente 2 completo y funcional. Esta entrega se compone de:

- (20%) Completar la funcionalidad que aún no haya sido desarrollada de la primera entrega.
- (30%) Documento de diseño. El documento de diseño debe seguir las pautas de ingeniería que usted ya conoce: Diagrama de clases (si ya vio ADOO o POO) o diseño precondiciones, poscondiciones e invariantes (si no ha visto ADOO). Además, se exigirá un(os) esquemático(s) que resuma(n) el(los) problema(s) y su(s) solución(ones) gráficamente.
- (50%) Código fuente compilable en el compilador gnu-g++ (versión 4.0.0, como mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando.