Instituto Tecnológico de Costa Rica

Sede Centro Académico San José

Escuela de Ingeniería en Computación

Ingeniería en Computación

Arquitectura de Computadores

Profesor

Esteban Arias Méndez

Semestre II 2014

Proyecto No. 2

Secuencias de ADN

Implementadas en Ensamblador

Estudiante:

Arburola León, Samantha P.

Abstract

In the following information you would consult about how the local and global algorithms for identity AND sequences were made. These are coding in assembly language**.**

25 de noviembre de 2014, San José.

Enunciado del Proyecto

Para este proyecto usted deberá implementar 3 programas: la implementación de los 2 algoritmos de alineamiento y un programa generador de secuencias de ADN para pruebas.

El programa generador de secuencias, preguntará al usuario el número de bases de ADN a crear, luego, de forma aleatoria deberá generar secuencias de A, C, T y G que conforman el ADN y almacenarlas en un archivo de texto como letras o caracteres.
Por ejemplo el contenido de un archivo de ADN sería como:

ACGGAAATGACGTAGCTAGCTAGCGAGCTGGGTTAACTCTCCATC, etc.

El archivo podría tener 100, 1000, 20000 o un millón de bases (letras A, C, T o G).

El programa deberá crear un archivo de texto con extensión .adn con el nombre que indique el usuario desde el programa. Este archivo contendrá el número de bases de ADN solicitadas por el usuario, como caracteres de texto.

Los otros dos programas serán la implementación del algoritmo de alineamiento local Smith–Waterman y el de alineamiento global llamado Needleman–Wunsch. Para cada algoritmo el programa deberá preguntar al usuario los valores de los parámetros a usar para las operaciones, ya sea los datos default o los que el usuario desee, es decir los valores de match, missmath, gap, etc.

Cada uno de estos programas preguntará al usuario por los 2 archivos de ADN que desea alinear. El programa deberá ejecutar el algoritmo correspondiente y brindar como salida el alineamiento de más bajo coste según cada algoritmo.

Los algoritmos ya están creados, por lo que usted solo deberá implementarlos y brindar al usuario el alineamiento óptimo que se podrá obtener y los valores de comparación correspondientes. No se deberá mostrar la matriz de alineamiento completa al usuario, pero la generación de la misma como salida a un archivo de texto que contenga dicha matriz completa será reconocido con un 10% extra sobre la nota del proyecto.

Generar un archivo de texto con el alineamiento de salida, señalando los gaps, tendrá un valore extra de 5% sobre la nota del proyecto.

Los algoritmos son programas simples que leen los datos de los archivos de texto que previamente generó, los cargan en memoria en una matriz como lo indica cada algoritmo y procede a realizar las operaciones llenando la matriz con valores. Finalmente partiendo de la base de la matriz se busca el alineamiento adecuado para brindar la salida.

Generados de Archivos ADN

El programa recibe el nombre de un archivo y cantidad de bases de ADN (ACTG) desea el usuario y genera el archivo con extensión .adn

Descripción del funcionamiento

Da la bienvenida al programa

Cita instrucciones

Solicita ingresar una cantidad de bases entre 0 y 9999

Muestra un mensaje de corroboración

Solicita ingresar un nombre para el archivo, de máximo 8 caracteres

Muestra un mensaje de corroboración

Repite según la cantidad de bases deseadas un ciclo el cuál contendrá

Genera un número random del cuál se obtine el residuo de dividirlo entre 10

Evalua las condiciones para concatenar a la tira por guardar

Si el residuo es 0, 5, 9: "A"

Si el residuo es 1, 6, 4: "C"

Si el residuo es 2, 7 : "T"

Si el residuo es 3, 8 : "G"

La tira generada es guardada en un archivo creado con el nombre ingresado por el usuario

Si todo el procedimiento concluye con éxito un mensaje lo comunica al usuario, de lo contrario solicita repetir el procedimiento

Alineamiento de Secuencias

Un **alineamiento de secuencias** en bioinformática es una forma de representar y comparar dos o más secuencias o cadenas de ADN, ARN, o estructuras primarias proteicas para resaltar sus zonas de similitud, que podrían indicar relaciones funcionales o evolutivas entre los genes o proteínas consultados. Las secuencias alineadas se escriben con las letras (representando aminoácidos onucleótidos) en filas de una matriz en las que, si es necesario, se insertan espacios para que las zonas con idéntica o similar estructura se alineen

Algoritmo de alineamiento local Smith–Waterman

Usa una matriz de sustitución de aminoácidos o nucleótidos para calificar sus alineamientos. Dicha matriz contiene la puntuación (también llamada *score*) que se le da al alinear un nucleótido o un aminoácido X de la secuencia A con otro aminoácido Y de la secuencia B. Las matrices más usadas para calificar alineamientos de proteínas son la BLOSUM y la PAM (ambas fueron obtenidas midiendo la frecuencia de los aminoácidos en una gran muestra de proteínas). También se permite al usuario definir su propia matriz. El tipo de matriz usada es determinante para los resultados que se obtendrán, el uso de una matriz incorrecta puede llevar a calificar erróneamente los alineamientos y por lo tanto obtener resultados equivocados.

El **algoritmo de Smith-Waterman** es una reconocida estrategia para realizar alineamiento local de secuencias biológicas (ADN, ARN o proteínas); es decir que determina regiones similares entre un par de secuencias.

El algoritmo SW fue propuesto por Temple Smith y Michael Waterman en 1981. Está basado en el uso de algoritmos de programación dinámica, de tal forma que tiene la deseable propiedad de garantizar que el alineamiento local encontrado es óptimo con respecto a un determinado sistema de puntajes que se use (tales como matrices de substitución).

Las alternativas básicas para realizar el alineamiento de un par de secuencias son: el alineamiento local y el alineamiento global. Los alineamientos globales pretenden alinear cada símbolo (o residuo) en cada secuencia. Esta estrategia es especialmente útil cuando las secuencias a alinear son altamente similares y aproximadamente del mismo tamaño. En contraste, los alineamientos locales son más útiles cuando las secuencias a alinear poseen grandes diferencias, pero se sospecha que existen regiones de similitud.

Código en JAVA

```
public class SmithWaterman {
	char[] mSeqA;
	char[] mSeqB;
	int[][] mD;
	int mScore;
	String mAlignmentSeqA = "";
	String mAlignmentSeqB = "";

	void init(char[] seqA, char[] seqB) {
		mSeqA = seqA;
		mSeqB = seqB;
		mD = new int[mSeqA.length + 1][mSeqB.length + 1];
		for (int i = 0; i <= mSeqA.length; i++) {
			mD[i][0] = 0;
		}
		for (int j = 0; j <= mSeqB.length; j++) {
			mD[0][j] = 0;
		}
	}

	void process() {
		for (int i = 1; i <= mSeqA.length; i++) {
			for (int j = 1; j <= mSeqB.length; j++) {
				int scoreDiag = mD[i-1][j-1] + weight(i, j);
				int scoreLeft = mD[i][j-1] - 1;
				int scoreUp = mD[i-1][j] - 1;
				mD[i][j] = Math.max(Math.max(Math.max(scoreDiag,
scoreLeft), scoreUp), 0);
			}
		}
	}

	void backtrack() {
		int i = 1;
		int j = 1;
		int max = mD[i][j];

		for (int k = 1; k <= mSeqA.length; k++) {
			for (int l = 1; l <= mSeqB.length; l++) {
				if (mD[k][l] > max) {
					i = k;
					j = l;
					max = mD[k][l];
				}
			}
		}

		mScore = mD[i][j];

		int k = mSeqA.length;
		int l = mSeqB.length;

		while (k > i) {
			mAlignmentSeqB += "-";
			mAlignmentSeqA += mSeqA[k - 1];
			k--;
		}
		while (l > j) {
			mAlignmentSeqA += "-";
			mAlignmentSeqB += mSeqB[l - 1];
			l--;
		}

		while (mD[i][j] != 0) {
			if (mD[i][j] == mD[i-1][j-1] + weight(i, j))
{
				mAlignmentSeqA += mSeqA[i-1];
				mAlignmentSeqB += mSeqB[j-1];
```

```
                        i--;
                        j--;
                        continue;
                } else if (mD[i][j] == mD[i][j-1] - 1) {
                        mAlignmentSeqA += "-";
                        mAlignmentSeqB += mSeqB[j-1];
                        j--;
                        continue;
                } else {
                        mAlignmentSeqA += mSeqA[i-1];
                        mAlignmentSeqB += "-";
                        i--;
                        continue;
                }
        }

        while (i > 0) {
                mAlignmentSeqB += "-";
                mAlignmentSeqA += mSeqA[i - 1];
                i--;
        }
        while (j > 0) {
                mAlignmentSeqA += "-";
                mAlignmentSeqB += mSeqB[j - 1];
                j--;
        }

        mAlignmentSeqA = new StringBuffer(mAlignmentSeqA).reverse().toString();
        mAlignmentSeqB = new StringBuffer(mAlignmentSeqB).reverse().toString();
}

private int weight(int i, int j) {
        if (mSeqA[i - 1] == mSeqB[j - 1]) {
                return 2;
        } else {
                return -1;
        }
}

void printMatrix() {
        System.out.print("D =       ");
        for (int i = 0; i < mSeqB.length; i++) {
                System.out.print(String.format("%4c ", mSeqB[i]));
        }
        System.out.println();
        for (int i = 0; i < mSeqA.length + 1; i++) {
                if (i > 0) {
                        System.out.print(String.format("%4c ", mSeqA[i-1]));
                } else {
                        System.out.print("     ");
                }
                for (int j = 0; j < mSeqB.length + 1; j++) {
                        System.out.print(String.format("%4d ", mD[i][j]));
                }
                System.out.println();
        }
        System.out.println();
}

void printScoreAndAlignments() {
        System.out.println("Score: " + mScore);
        System.out.println("Sequence A: " + mAlignmentSeqA);
        System.out.println("Sequence B: " + mAlignmentSeqB);
        System.out.println();
}

public static void main(String [] args) {
        char[] seqB = { 'A', 'C', 'G', 'A' };
        char[] seqA = { 'T', 'C', 'C', 'G' };

        SmithWaterman sw = new SmithWaterman();
```

```
                sw.init(seqA, seqB);
                sw.process();
                sw.backtrack();

                sw.printMatrix();
                sw.printScoreAndAlignments();
        }
}
```

Algoritmo de alineamiento global Needleman–Wunsch

El algoritmo Needleman-Wunsch realiza un alineamiento global de dos secuencias (aquí llamadas A y B).

Es comúnmente usado en bioinformática para alinear secuencias de nucleótidos o proteínas. Fue propuesto en 1970 por Saul Needleman y Christian Wunsch en su paper *A general method applicable to the search for similarities in the amino acid sequence of two proteins*, J Mol Biol. 48(3):443-53.

El algoritmo Needleman–Wunsch es un ejemplo de programación dinámica, y está garantizado que encuentre el alineamiento con el puntaje máximo. Needleman–Wunsch fue la primera aplicación de programación dinámica para la comparación de secuencias biológicas.

Los puntajes para caracteres alineados son especificados por una matriz de similitud. Aquí, S(i,j) es la similitud de los caracteres i y j. Esta usa una penalidad por hueco (gap) lineal, aquí llamada d.

Por ejemplo, si la matriz de similitud era:

-	A	G	C	T
A	10	-1	-3	-4
G	-1	7	-5	-3
C	-3	-5	9	0
T	-4	-3	0	8

Entonces el alineamiento es:

AGACTAGTTAC

CGA—GACGT

Con una penalidad por hueco de -5, tendríamos el siguiente puntaje…

$$S(A,C)+S(G,G)+S(A,A)+3\times d+S(G,G)+S(T,A)+S(T,C)+S(A,G)+S(C,T)$$
$$= -3+7+10-3\times 5+7+-4+0+-1+0=1$$

Para encontrar el alineamiento con el puntaje más alto, una matriz bidimensional es asignada. Esta matriz a menudo es llamada matriz F y su (i,j)ésima entrada frecuentemente es denotada *Fij* . Allí hay una columna para cada carácter de la **secuencia A**, y una fila para cada carácter de la **secuencia B**. Así si estamos alineando secuencias de tamaños n y m, el tiempo de ejecución del algoritmo es O(nm) y la cantidad de memoria utilizada es O(nm).

Sin embargo hay una versión modificada del algoritmo que usa solo **O(n+m)** espacio, al costo de un tiempo de ejecución más grande. Esta modificación es de hecho una técnica general que aplicamos a muchos algoritmos de programación dinámica; este método fue introducido en el algoritmo de Hirschberg para resolver el problema de la subcadena más larga en común.

Cuando el algoritmo progresa, *Fij* puede ser asignada para ser el puntaje óptimo para el alineamiento de los primeros i caracteres en A y los primeros j caracteres en B. El principio de optimización es entonces aplicado como sigue:
Base:

$F0j = d * j$
$Fi0 = d * i$
Recursión, basada en el principio de optimización:

$Fij = \max(Fi-1,j-1 + S(Ai-1,Bj-1),Fi,j-1 + d,Fi-1,j + d)$

El pseudo-código para el algoritmo que computa la matriz A por lo tanto luce algo así (los índices de arreglos y secuencias empiezan en 0):

```
for i=0 to length(A)-1
     F(i,0) <- d*i
   for j=0 to length(B)-1
     F(0,j) <- d*j
   for i=1 to length(A)
     for j = 1 to length(B)
     {
       Choice1 <- F(i-1,j-1) + S(A(i), B(j))
       Choice2 <- F(i-1, j) + d
       Choice3 <- F(i, j-1) + d
       F(i,j) <- max(Choice1, Choice2, Choice3)
     }
```

Una vez que la matriz F es computada, note que la esquina inferior derecha de la matriz es el máximo puntaje para cualquier alineamiento. Para computar que alineamiento da actualmente ese puntaje, puedes empezar desde la celda que se encuentra al fondo a la derecha, y comparar el valor con las tres posibles fuentes (Elección1, Elección2, Elección3) para ver de donde proviene.

Si era Elección1, entonces A(i) y B(i) están alineadas, si era Elección2 entonces A(i) está alineado con un gap, y si era Elección3, entonces B(i) está alineada con un hueco (gap).

```
AlignmentA <- ""
   AlignmentB <- ""
   i <- length(A)
   j <- length(B)
   while (i > 0 AND j > 0)
   {
     Score <- F(i,j)
     ScoreDiag <- F(i - 1, j - 1)
     ScoreUp <- F(i, j - 1)
     ScoreLeft <- F(i - 1, j)
     if (Score == ScoreDiag + S(A(i), B(j)))
     {
       AlignmentA <- A(i-1) + AlignmentA
       AlignmentB <- B(j-1) + AlignmentB
       i <- i - 1
       j <- j - 1
     }
     else if (Score == ScoreLeft + d)
     {
       AlignmentA <- A(i-1) + AlignmentA
       AlignmentB <- "-" + AlignmentB
       i <- i - 1
     }
     otherwise (Score == ScoreUp + d)
     {
       AlignmentA <- "-" + AlignmentA
       AlignmentB <- B(j-1) + AlignmentB
       j <- j - 1
     }
   }
   while (i > 0)
   {
```

```
     AlignmentA <- A(i-1) + AlignmentA
     AlignmentB <- "-" + AlignmentB
     i <- i - 1
   }
   while (j > 0)
   {
     AlignmentA <- "-" + AlignmentA
     AlignmentB <- B(j-1) + AlignmentB
     j <- j - 1
   }
```

Código en Java

```
/*
 * NeedlemanWunsch.java
 *
 * Created on 9 december 2006, 18:24
 */

package bioinfo;

import java.util.ArrayList;

/**
 *
 * @author DS
 */
public class NeedlemanWunsch {
    public static final int A=0, G=1, C=2, T=3;
    public static final int d = -5;
                                    // A    G   C   T
    public static int[] simularity = {10, -1, -3, -4, //A
                                      -1,  7, -5, -3, //G
                                      -3, -5,  9,  0, //C
                                      -4, -3,  0,  8};//T
    public static void main(String[] arg)
    {
//        int[] ar = NeedlemanWunsch.convertStringToArr("TCGAAGCT");
//        System.out.println(ar);
          System.out.println(NeedlemanWunsch.simular(0,0));
          System.out.println(NeedlemanWunsch.simular(1,0));
          int[][] ar = NeedlemanWunsch.calculateMatrix(convertStringToArr("AGACTAGTTAC"),
convertStringToArr("AGACTAGTTAC"));
          for (int y = 0; y < ar.length; y++)
          {
              System.out.println("");
              for (int x = 0; x < ar[y].length; x++)
                  System.out.print(ar[y][x] +"\t ");
          }
          NeedlemanWunsch.getAlignments(ar, convertStringToArr("AGACTAGTTAC"),
convertStringToArr("AGACTAGTTAC"), "AGACTAGTTAC","AGACTAGTTAC");

    }
    public static void getAlignments(int[][] ar, int[] A, int[] B, String sA, String sB)
    {
        String alA = "";
        String alB = "";
        int i = sA.length();
        int j = sB.length();
        while (i > 0 && j > 0)
        {
            int score = ar[i][j];
            int scorediag = ar[i-1][j-1];
```

```
            int scoreup = ar[i][j-1];
            int scoreleft = ar[i-1][j];
            if (score == scorediag + simular(A[i-1], B[j-1]))
              {
                alA = sA.charAt(i-1) + alA;
                alB = sB.charAt(j-1) + alB;
                i--;j--;
              }
            else if (score == scoreleft + d)
            {
                alA = sA.charAt(i-1) + alA;
                alB = "-" + alB;
                i--;
            }
            else if(score == scoreup + d)
            {
                alA = "-" + alA;
                alB = sB.charAt(j-1) + alB;
                j--;
            }
        }
        while(i > 0)
        {
            alA = sA.charAt(i - 1) + alA;
            alB = "-" + alB;
            i--;
        }
        while(j > 0)
        {
            alA = "-" + alA;
            alB = sB.charAt(j - 1) + alB;
            j--;
        }
        System.out.println(alA+"\n");
        System.out.println(alB+"\n");
    }

    public static int[][] calculateMatrix(int[] source, int[] dest)
    {
        int[][] res = new int[source.length+1][dest.length+1];
        for (int y = 0; y < source.length; y++)
            res[y][0] = d * y;


        for (int x = 0; x < dest.length; x++)
            res[0][x] = d * x;

        for (int y = 1; y < source.length + 1; y++)
            for (int x = 1; x < dest.length +1; x++)
                {
                    int k = res[y-1][x-1] + simular(source[y-1] , dest[x-1]);
                    int l = res[y-1][x] + d;
                    int m = res[y][x-1] + d;
                    k = Math.max(k,l);
                    res[y][x] = Math.max(k,m);
                }
        return res;
    }
    public static int[] convertStringToArr(String str)
    {
        ArrayList l = new ArrayList();
        for (int i = 0; i < str.length(); i++)
        {
            int n = -1;
            if(str.charAt(i) == 'A')
                n= 0;
            if(str.charAt(i) == 'G')
                n= 1;
            if(str.charAt(i) == 'C')
                n= 2;
            if(str.charAt(i) == 'T')
```

```
                n= 3;

            l.add(new Integer(n));
        }
        int[] arr = new int[l.size()];
        for (int i = 0; i < l.size();i++)
            arr[i] = ((Integer)l.get(i)).intValue();
        return arr;
    }
    public static int simular(int first, int second)
    {
        return simularity[first * 4 + second];
    }

}
```

Conclusiones y Observaciones

La primera recomendación para implementar en lenguaje ensamblador este proyecto más adelante es crear previo al proyecto programas pequeños con funciones para manipular strings; entre los cuales puedo mencionar concatenación, recorrido y búsqueda y generar números aleatorios. Esto con el fin de conocer el comportamiento de estas funciones, la localización del resultado y la capacidad necesaria.

La segunda recomendación, informarse sobre el lenguaje, si se van a hacer consultas, buscar desde el principio quién domine NASM o bien TASM ya que hay muy pocos recursos humanos cuales son escasos.

La tercera recomendación, a los estudiantes de biología les alegra que le pregunten sobre el tema y les apoyaran en cuánto puedan, su conocimiento y apoyo son invaluables, no hay que dudar en consultarles y solicitarles una explicación del tema.

La última recomendación, armarse de paciencia, les dejo a ustedes aplicarlo en los axiomas que crean necesarios.

La falta de conocimiento sobre el lenguaje ensamblador fue el mayor obstáculo para desarrollar el proyecto. Las desmotivación que se presentó al pedir ayuda a otros compañeros o bien en la búsqueda de un tutor extracurricular dieron espacio a comentarios que reservaré como aprendizaje y lecciones de vida.

Referencias

Bassi, S. (2010). *Python for bioinformatics*. Boca Raton: CRC Press.

Bioinformatica.cecalc.ula.ve,. (2014). *Portal de Bioinformática | Taller de Herramientas para Análisis de Secuencias (THAS) marzo 2014 (Actualizado)*. Recuperado 26 November 2014, a partir de http://bioinformatica.cecalc.ula.ve/?p=230015

Bioinformaticos,. (2014). *Algoritmo Needleman-Wunsch*. Recuperado 26 November 2014, a partir de http://www.bioinformaticos.com.ar/algoritmo-needleman-wunsch/

Bioinfouab.uab.cat,. (2014). *Curso de Perl - 10 Ejercicios aplicados a la Bioinform*  *tica*. Recuperado 26 November 2014, a partir de http://bioinfouab.uab.cat/bioinfouab_new/Resources/CursoPerl/CursoPerl.asp

Blast.ncbi.nlm.nih.gov,. (2014). *BLAST: Basic Local Alignment Search Tool*. Recuperado 26 November 2014, a partir de http://blast.ncbi.nlm.nih.gov/Blast.cgi

Claverie, J., & Notredame, C. (2003). *Bioinformatics for dummies*. New York, NY: Wiley Pub.

Code.google.com,. (2014). *SmithWaterman.java - himmele - Daniel's Blog Repository - Google Project Hosting*. Recuperado 26 November 2014, a partir de https://code.google.com/p/himmele/source/browse/trunk/Bioinformatics/SmithWaterman/src/SmithWaterman.java

Model, M. (2010). *Bioinformatics programming using Python*. Sebastopol, CA: O'Reilly Media.

Virtual.unal.edu.co,. (2014). *SEDE BOGOTA*. Recuperado 26 November 2014, a partir de http://www.virtual.unal.edu.co/cursos/ingenieria/2001832/lecciones/needleman.html