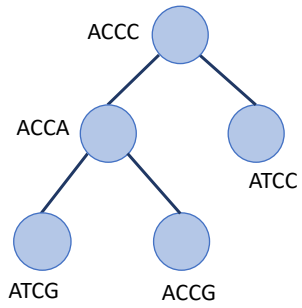
Práctica 2 - Estructuras de Datos: Árboles binarios filogenéticos y mínima parsimonia

Definición del problema

En bilogía, la filogenia[1] hace referencia al origen y desarrollo evolutivo de las especies. La filogenética estudia la evolución de las diferentes especies basándose en la información genética. Esta información normalmente está incompleta y por tanto es posible llegar a diferentes *hipótesis*, por lo que es necesario establecer algún criterio que nos permita seleccionar la hipótesis más probable.

El *árbol filogenético* es una forma de representar las relaciones evolutivas de diferentes especies. En estos árboles los nodos están etiquetados por secuencias genéticas, de manera que el nodo raíz representa a un ancestro común al resto de nodos descendientes. En la Figura 1 se muestra una representación gráfica (izquierda) y otra textual (derecha) de un árbol filogenético binario (los nodos internos tienen a lo sumo dos hijos). La representación de la derecha utiliza la notación textual de Newick[2].



((ATCG, ACCG) ACCA, ATCC) ACCC

Figura 1. Ejemplo de árbol filognético. Izq. representación gráfica. Dch. notación de Newick

Supongamos que en el árbol de la Figura 1 los 3 nodos hoja representan al gorila (ATCG), al chimpancé (ACCG) y al ser humano (ATCC) respectivamente. El nodo interno etiquetado con ACCA representa un ancestro común a gorila y al chimpancé. Como vemos, la etiqueta de esta especie difiere en algunas posiciones con las etiquetas de sus descendientes el gorila y el chimpancé, las que se mantienen representan características que se han heredado. El nodo raíz (ACCC) es un ancestro común a las tres especies.

En esta práctica vamos a realizar el cálculo de la mínima *parsimonia* que nos permitirá *estimar una hipótesis con el menor número cambios evolutivos* posibles.

Descripción del método

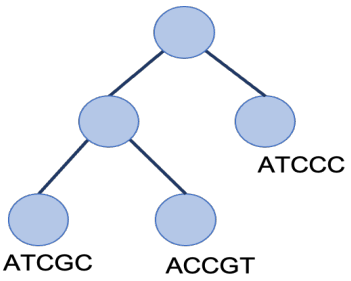


Figura 2. Árbol filogenético de entrada

Existen varios algoritmos para el cálculo de la mínima parsimonia. Nosotros vamos a utilizar una versión simplificada del algoritmo de Sankoff.

La entrada del algoritmo es un árbol binario filogenético en el que solo están etiquetados los nodos hoja, como aparece en la Figura 2. Todas las etiquetas tienen la misma longitud m (en la figura $m = 5$) y la longitud del

[1] https://dle.rae.es/filogenia
[2] https://en.wikipedia.org/wiki/Newick_format

alfabeto es *n* (en nuestro caso *n* = 4 porque el alfabeto está compuesto por los nucleótidos A, C, G y T). Además, el algoritmo utiliza una matriz *delta* de tamaño *n* x *n* (en nuestro caso 4x4) que almacena el coste de cambiar un nucleótido por otro. Vamos a suponer que el coste es 1 si cambiamos el nucleótido por otro diferente, y 0 si no se cambia.

El algoritmo de Sankoff calcula la puntuación de todas las posibles etiquetas de los nodos del árbol (tanto de los nodos hoja como de los internos). Para cada nodo las puntuaciones de las posibles etiquetas se almacenan en una matriz **s** de tamaño *m* x *n*. La posición **s**[i][j] almacena el coste de tener el nucleótido j en la posición i de la etiqueta de ese nodo. El algoritmo comienza calculando la matriz de puntuación **s** de los nodos hojas y termina calculando la matriz **s** de la raíz.

En el caso de un nodo hoja, **s**[i][j] es 0 si en la posición i de su etiqueta está el nucleótido j. En otro caso **s**[i][j] vale infinito. Para el árbol de la Figura 2, la matriz **s** del nodo hoja con etiqueta ATCGC sería $\mathbf{s_1}$ y la del nodo con etiqueta ACCGT sería la $\mathbf{s_2}$:

$$s_1 = \begin{array}{c} \text{Nucleótido} \\ \text{A C G T} \\ \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & \infty & \infty & 0 \\ \infty & 0 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & 0 & \infty & \infty \end{pmatrix} \end{array} \begin{array}{c} \\ \\ 0 \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \qquad s_2 = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix}$$

En el caso de un nodo interno, **s**[i][j] depende de la matriz de puntuación de sus hijos y de la matriz delta. Vamos a llamar ***sl*** y ***sr*** a las matrices de puntuación del hijo izquierdo y derecho respectivamente. De este modo, para un nodo interno el valor de **s**[i][j] se calcula como:

$$s[i][j] = min_k\ (\boldsymbol{sl}[\text{i}][\text{k}] + \ \text{delta}[\text{j}][\text{k}]) + min_r(\boldsymbol{sr}[i][r] + delta[j][r])$$

Por ejemplo, la matriz **s** del nodo padre de ATCGC y ACCGT se calcularía utilizando $\mathbf{s_1}$ y $\mathbf{s_2}$ y la matriz delta, y el resultado sería la matriz $\mathbf{s_3}$:

$$delta = \begin{array}{c} \text{A C G T} \\ \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix} \end{array} \begin{array}{c} \text{A} \\ \text{C} \\ \text{G} \\ \text{T} \end{array} \qquad s_3 = \begin{pmatrix} 0 & 2 & 2 & 2 \\ 2 & 1 & 2 & 1 \\ 2 & 0 & 2 & 2 \\ 2 & 2 & 0 & 2 \\ 2 & 1 & 2 & 1 \end{pmatrix}$$

Cuando ya se ha calculado la matriz s del nodo raíz, la parsimonia del árbol se calcula sumando la mínima puntuación posible para cada posición de la etiqueta. Es decir, si la matriz $\mathbf{s_5}$ es la matriz de puntuación del nodo raíz del árbol del ejemplo, la puntuación de parsimonia del árbol es igual a 3 (sumando el valor mínimo de cada fila 0 + 1 + 0 + 1+1).

$$s_5 = \begin{pmatrix} 0 & 2 & 2 & 2 \\ 3 & 2 & 3 & 1 \\ 2 & 0 & 2 & 2 \\ 2 & 1 & 1 & 2 \\ 2 & 1 & 3 & 1 \end{pmatrix}$$

Cuando ya se ha calculado la matriz s del nodo raíz, el algoritmo de Sankoff reconstruye las etiquetas de los nodos internos. Para nuestro ejemplo, la puntuación de parsimonia igual a 3 indica que existe al menos una etiqueta (del nodo raíz) que puede transformarse en cualquier etiqueta de los nodos hoja haciendo a lo sumo 3 cambios. Utilizando la matriz $\mathbf{s_5}$ podemos inferir que estas etiquetas tendrán en la posición 0 el nucleótido A, en la posición 1 el T, en la

posición 2 tendrán el C, en la posición 3 tendrán el C o el G y en la posición 4 el C o el T. Como ya hemos dicho antes, **en la práctica no vamos a reconstruir las etiquetas de los nodos internos**.

Para la implementación del algoritmo os proponemos implementar dos funciones. La función computeParsimonyScore recibe un árbol filogenético y devuelve la puntuación de parsimonia. Para calcular dicho valor, utiliza la función auxiliar computeScoreRec que dado un árbol filogenético calcula de forma recursiva la matriz de puntuación s del nodo raíz. En las Figuras Figura 3 y Figura 4 se muestra el pseudo-código de ambas funciones.

```
fun computeParsimonyScore(↓t: PhylogeneticTree, ↑parsimony:int)
vars
  s[0..m-1][0..n-1]:int
min, i, j: int
begin
  computeScoreRec(t, s)
  for i ← 0 to m-1 do
    min = s[i][0]
    for j← 1 to n-1 do
      if(s[i][j] < min)
        min = s[i][j]
      endif
    endfor
    parsimony+= min
  endfor
end
```

Figura 3. Pseudo-código de la función computeParsimony

```
fun computeScoreRec(↓t: PhylogeneticTree, ↑s[1..m][1..n]:int)
vars
  label[m]: char
  i, j, k, sl [1..m][1..n], sr[1..m][1..n]: int
const
  delta = {0 1 1 1, 1 0 1 1,1 1 0 1, 1 1 1 0}
  alphabet = {'A', 'C', 'G', 'T'}
begin
  if (t es nodo hoja)
    label = t.root()
    for i ←0 to m-1 do
      for j← 0 to n-1 do
         if(label[i] == alphabet[j])
           s[i][j] = 0; // no se cambia el nucleótido
         else
           s[i][j] = MAX_INTEGER; // sí se cambia el nucleótido
        endif
       endfor
    endfor
  else if (t es nodo interno)
    computeScoreRec(t.left(), sl)
    computeScoreRec(t.right(),sr)
    for i←0 to m-1 do
       for j←0 to n-1 do
         s[i][j] = min_k{s_left[i][k] + delta[j][k]} +
                   min_k{s_right[i][k] + delta[j][k]}
      endfor
    endfor
  endif
  return s;
end
```

Figura 4. Pseudo-código de la función computeScoreR

Nota 1: $\min_k$ no es una función Java predefinida. Hay que calcular el valor mínimo de s[i][k] + delta[j][k], donde k toma valores entre 0 y (n-1).
Nota 2: si para alguna combinación de i,j s[i][j] es ∞ (MAX_INTEGER), no hay que sumarle delta[j][k] ya que $\infty + 1 = \infty$, pero MAX_INTEGER+1 desborda y toma un valor negativo.

Realización de la práctica

1. Completar la implementación de la clase PhylogeneticTree que permitirá almacenar árboles filogenéticos. La clase implementa la interfaz BinaryTree<String>, es decir que las etiquetas de cada nodo del árbol son de tipo String.
2. Completar la implementación de la clase estática PhylogeneticUtils que consta de los métodos:
    a. computeParsimonyScore: dado un árbol filogenético en el que solo están etiquetados los nodos hoja, devuelve la puntuación de parsimonia mínima.
    b. computeScoreRec: dado un árbol filogenético, devuelve la matriz de puntuación **s** del nodo raíz.

    El estudiante puede implementar los métodos auxiliares que crea necesarios.

Además de estas dos clases, se proporcionan:

- Phylogenetic1.6.jar: esta librería contiene algunas de las interfaces de TADs vistos en el tema 2, entre otras la interfaz BinaryTree. Para importar la librería en el proyecto eclipse:
    - Copiar el fichero jar en el proyecto. Por ejemplo, puedes crear una carpeta lib dentro del proyecto para mantener los ficheros organizados.
    - Sobre el nombre del proyecto, pulsa el botón secundario y selecciona Configure Build Path > Pestaña Libraries.
    - Si en el área central te aparecen Module Path y Class Path, selecciona Class Path. Si no te aparece, puedes seguir con el siguiente paso.
    - Pulsa el botón Add JARs y selecciona el fichero Phylogenetic.6.jar
- TestPhylogeneticUtils: en esta clase se encuentra en método main (no hay que modificarla). Para probar la implementación debéis pasar como argumentos de entrada -f <nombrefichero.txt>, donde <nombrefichero.txt> es un fichero que contiene un árbol filogenético en notación de Newick.
- Ficheros de prueba (treeX.txt) y sus correspondientes salidas (treeX.out).

Entrega y evaluación

La evaluación de la práctica consiste en la entrega y evaluación automática del código del alumno y una memoria (de no más de 3 páginas) en la que el alumno realice una defensa de su solución. A continuación, se detalla la nota que se puede obtener:

Nota	Descripción
0	No pasa ninguna prueba en laboratorio ni entrega memoria o el sistema detecta plagio en el código o la memoria
[1,4]	No pasa ninguna prueba en laboratorio. Entrega la memoria y pasa las pruebas fuera de plazo
[4,6]	Pasa algunas pruebas en el laboratorio. Entrega memoria y pasa todas las pruebas fuera de plazo
[6,10]	Pasa todas las pruebas en laboratorio y entrega memoria

Para la evaluación automática del código, el estudiante debe subir los ficheros PhylogeneticTree.java y PhylogeneticUtils.java a una actividad del campus que realizará una serie de pruebas sobre la implementación. El código hay que subirlo siempre al final de la sesión de laboratorio correspondiente. Si no se pasan todas las pruebas tendrá una segunda oportunidad fuera de plazo. La fecha se comunicará al final de la sesión de laboratorio.