Algoritmo de Hirschberg

- Luis Ernesto Ibarra Vázquez C411
- Luis Enrique Dalmau Coopat C411

Resumen

En el trabajo se detalla el algoritmo de Hirschberg y cómo este puede ser usado para resolver problemas ligados a la comparación de cadenas de texto. También se presentan los códigos de dicho algoritmo así como la implementación de los problemas que puede resolverse mediante el uso de este. Además muestra una pequeña comparación entre las variantes de algoritmos con Hirschberg y sin este.

Desarrollo

Algoritmo de Hirschberg (AH)

Este algoritmo resuelve el problema de hallar la alineación óptima global entre dos cadenas de texto. Este problema es ya resuelto con el algoritmo de Needleman-Wunsch (NW) pero con un cambio en la complejidad espacial. El algoritmo de Needleman-Wunsch mantiene en todo momento una matriz en la que se realiza la programación dinámica, esto conlleva a un gasto de O(nm) de memoria, con n y m siendo el tamaño de la primera y segunda cadena respectivamente, AH solo necesita de dos filas de esta matriz, utilizando la primera como base para calcular la segunda y luego intercambiando de lugares las filas, este procedimiento requiere de solo O(max(n,m)) de memoria. Con respecto a la complejidad temporal ambos algoritmos trabajan en O(nm), pero en la práctica AH se desempeña mejor.

El AH es un algoritmo divide y vencerás, en el divide se reduce el tamaño de las cadenas hasta que tenga alguna longitud 0 o 1 y luego se procede a realizar Needleman-Wunsch sobre estas. La estrategia de división se basa en la idea de que dada la alineación optima:

$$(X, Y)$$

Existe una partición de X y de Y

$$X = X^l + X^r \qquad Y = Y^l + Y^r$$

Tal que al concatenar las subsecuencas óptimas de estas esta se óptima:

$$NW(X, Y) = NW(X^l, Y^l) + NW(X^r, Y^r)$$

Para encontrar las particiones se elige como punto de corte en la primera cadena al índice y en la segunda cadena al índice que maximice la alineación de la parte izquierda de X con Y y el reverso de la parte derecha de X con el reverso de Y. Una vez se tiene estos dos índices se devuelve el la concatenación de las alineaciones óptimas de las mitades correspondientes.

Al igual que NW la entrada de AH, además de las cadenas en que se desea hallarles la alineación óptima, consiste en una función de costo que indica el costo que tiene sustituir un caracter por otro y un coste por dejar un hueco en la alineación.

Usos

AH se puede usar para resolver varios problemas relacionados con la similitud entre cadenas, cambiando la función de coste se pueden crear variantes del algoritmo cuyos resultados son equivalentes. Entre los problemas que se pueden resolver usando este algoritmo están la distancia de Levenshtein y la subsecuencia común más larga (LCS). Las variantes clásicas de estos algoritmos trabajan con una complejidad espacial de O(nm), pero usando AH se puede reducir a O(max(n,m)).

Para modelar la distancia de Levenshtein se tiene como costo de hueco a -1 y como función de costo a la siguiente:

$$f(a,b) = \begin{cases} -1 & a \neq b \\ 0 & a = b \end{cases}$$

De esta manera el costo de una alineación óptima es equivalente a la distancia de Levenshtein, ya que la función de sustitución cuenta las sustituciones en la cadena con coste -1 y los huecos representarían el insertado y borrado de caracteres de una cadena a otra, haciendo el costo equivalente a la métrica que usa la distancia de Levenshtein. Como se está maximizando no habrá otra cadena con mayor costo, lo que se traduce a que no habrá una transcripción con menor cantidad de cambios, lo cual representa la distancia de Levenshtein.

Otro problema que se puede resolver mediante AH es el problema de la subsecuencia común más larga. Para el modelado de LCS se tiene como costo de hueco a 0 y como función de costo:

$$f(a,b) = \begin{cases} 1 & a = b \\ 0 & a \neq b \end{cases}$$

De esta manera se logra asociar el costo de la transcripción a la cantidad de elementos que son iguales en las subcadenas de forma ascendente, al AH maximizar esta métrica se obtiene como resultado una de longitud máxima.

Además de estos usos este algoritmo es usado en la biología computacional para comparar secuencias de ADN y otras proteínas. Aunque se usan computadoras potentes para el cálculo de estas.

Implementación

En la carpeta **codigos** se enuentran las implementaciones de los diferentes algoritmos con sus respectivas versiones.

- Needleman-Wunsch
- Hirschberg
- Levenshtein
- LCS

Para probar los algoritmos correr el script *main.py* dentro de **codigos**. Este script presenta un método interactivo para probar los diferentes algoritmos pudiendo introducir cadenas y que estos devuelvan los respectivos resultados.

Comparaciones

Para ver las comparaciones en *main.py* descomente la función **run_comp()** dentro de **main()**.

Se realizaron diferentes comparaciones entre los diferenes algoritmos para ver el desempeño de estos. De las comparaciones se sacaron las siguientes tiempos en segundos:

Cantidad de caracteres	Lev Hirs	Lev	LCS Hirs	LCS	Hirschberg	Needl.-Wunsch
Cad 1 y Cad 2: 311	1.11	0.2	1.2	0.15	1.22	1.44
Cad 1 y Cad 2: 422	2.22	0.38	2.18	0.29	2.08	2.44
Cad 1 y Cad 2: 481	2.81	0.49	2.74	0.32	2.71	3.22
Cad 1 y Cad 2: 605	4.13	0.77	4.14	0.57	4.15	5.26
Cad 1 y Cad 2: 726	6.28	1.18	6.17	0.73	6.05	7.54
Cad 1 y Cad 2: 881	9.35	1.74	9.34	1.21	9.25	11.92

Como se puede observar las variantes de Hirschberg de los algoritmos de Levenshtein y LCS son mucho más lentas que sus contrapartes dinámicas, esto se puede deber al overhead que produce la transformación de este algoritmo a estas variantes a ir directamente a la solución. En la comparación entre Hirschberg y Needleman-Wunsch se oberva que el primero es mejor.

Referencias

[1] *Hirschberg's algorithm* Wikipedia https://en.wikipedia.org/wiki/Hirschberg%27s_algorithm Consultado el 30/12/2021

[2] *Algoritmo de Hirschberg algoritmo de optimización de la distancia de edición mínima* https://programmerclick.com/article/35691234352/ Consultado el 30/12/2021