All in all

Johanna Beltran y Diego Triviño

2012

Índice

1. Introducción

'**All in all**' es un problema de programación el cual encontramos en el juez virtual UVA con el número **10340**.
Este documento busca mostrar una de las tantas soluciones desde el enfoque matemático teniendo en cuenta que el objetivo es realizar el código con la solución del problema en cualquier lenguaje de programación con la ayuda de este documento.
Este problema puede ser resuelto utilizando la metodología de algoritmos voraces.
Los algoritmos voraces tienden a ser bastante eficientes y pueden implementarse de forma relativamente sencilla. Su eficiencia se deriva de la forma en que trata los datos, llegando a alcanzar muchas veces una complejidad de orden lineal.

Se deben definir los siguientes elementos según el problema que abordan:

1. **El conjunto C:** de candidatos es decir las entradas del problema.
2. **Función solución S:** Comprueba, en cada paso, si el subconjunto actual de candidatos elegidos forma una solución (no importa si es óptima o no lo es).
3. **Función de selección:** Informa de cuál es el elemento más prometedor para completar la solución. Éste no puede haber sido escogido con anterioridad. Cada elemento es considerado una sola vez.
4. **Función de factibilidad:** determina si un conjunto es completable, es decir, si añadiendo a este conjunto nuevos candidatos es posible alcanzar una solución al problema, suponiendo que esta exista.
5. **Función objetivo:** Es aquella que queremos maximizar o minimizar, el núcleo del problema.

2. Definición del problema

Dadas dos cadenas s y t , usted tiene que decidir si s es una subsecuencia de t , es decir, si se puede quitar caracteres de t tales que la concatenación de los caracteres restantes es s .
Se deben tener en cuenta las siguientes restricciones para la solución del problema:

1. Por cada caso de prueba solo se deben ingresar dos cadenas.
2. Se tiene en cuenta las mayúsculas y las minúsculas de cada carácter. Para esto es importante el código ASCII.

2.1. Entrada

La entrada contiene varios casos de prueba. Cada uno está especificado por dos cadenas s, t separados por espacios en blanco. Las entradas finalizan con EOF.

EJEMPLO

secuencia subsecuencia

persona compresión

VERDI vivaVittorioEmanueleReDiItalia

caseDoesMatter CaseDoesMatter

2.2. Salida

Para cada caso de prueba de salida, se debe imprimir 'YES' si S es una subsecuencia de T y 'NO' si no lo es.

EJEMPLO ANTERIOR

Yes

No

Yes

No

3. Modelamiento matemático

S y T son listas de caracteres donde S es la cadena que se debe buscar en la cadena T.

1. **Conjunto de candidatos C:** Todos los caracteres que conforman la lista T.
2. **Factibilidad (Factibilidad):** La lista de caracteres $C \subseteq S$.
3. **Función de selección (SeleccioneCaracter):** El primer carácter $x \in T$ tal que $x \in S$ y $x \neg \in C$.
4. **Función solución (Objetivo):** Lista C = lista S.

4. Planteamiento de la solución

Para cada caso de prueba se definen los elementos de los algoritmos voraces mencionados anteriormente. Para empezar, con la función de selección se elige un elemento x; luego se verifica si el elemento seleccionado $x \bigcup C$ cumple con la función de factibilidad, si cumple con la función de factibilidad el elemento x se adiciona al conjunto de candidatos. Por último, cuando no existan mas elementos por seleccionar se verifica la función solución devolviendo así la respuesta correcta para cada caso de prueba.

4.1. Pseudocódigo propuesto

Funcion Voraz (lista S, lista T)
Define lista C
Define entero j

mientras S != C **ó** $j <$ **Tamaño** (T)

$x :=$ **SeleccioneCaracter** (T,j)
si (**Factibilidad**(S,x))
entonces $C := C + x$
$j := j+1$

si (**Objetivo**(S,C))
entonces devuelve 'YES'
si (!**Objetivo**(S,C))
entonces devuelve 'NO'

5. Conclusiones

1. Por las características de este problema se recomienda utilizar el enfoque de algoritmos voraces puesto que una vez incorporado un candidato a la solución, permanece ahí hasta el final; y cada vez que un candidato es rechazado, lo es para siempre.

2. Debemos diseñar una función de selección adecuada ya que algunas pueden garantizar la solución óptima y otras pueden ser más heurísticas.