PARCIAL BLOQUE 2 RESUELTO

Es el documento de informática que he subido que más me gusta en cuanto a dificultad y rigurosidad (hay dos demostraciones jajajaj). Espero que os sirva 😊

E. T. S. de Ingeniería Informática | Curso: 2018–19

Análisis y Diseño de Algoritmos | Control Bloque 2

Alumno: ______________________________ **Grado**: ______________

Ejercicio 1

Se dice que una cadena s es una supersecuencia de otra cadena w si todos los caracteres de w aparecen en s en el mismo orden, aunque no sea de manera consecutiva. Por ejemplo, "camisa" es una supersecuencia de "casa" (i.e., "**<u>ca</u>mi<u>sa</u>**"). Dadas dos cadenas w_1 y w_2 el problema de la supersecuencia más corta es hallar la cadena s más corta que es a la vez supersecuencia de w_1 y w_2. Nótese que dicha cadena no tiene por qué ser única.

1. Encontrar una expresión recurrente para la longitud de la supersecuencia.
2. Demostrar que el problema exhibe la propiedad de subestructura óptima.
3. Confeccionar un algoritmo de programación dinámica bottom-up para resolver el problema. Sólo se desea conocer el número de caracteres de la supersecuencia más corta.
4. Aplicar a mano este algoritmo para encontrar una supersecuencia de longitud mínima para "gato" y "ratón" (los acentos pueden ignorarse). Es decir, rellenar la tabla.

Ejercicio 2

El CAC de Málaga recibe una colección de cuadros de suma importancia, y el museo, no depositando una gran confianza en sus ciudadanos, decide contratar un servicio de vigilancia. Todos los cuadros se encuentran situados en un pasillo, en la localización $X = \{x_1, x_2, ..., x_n\}$. $x_i \in \mathbb{R}$ representa el punto de colocación de cada cuadro a lo largo del pasillo. Cada vigilante puede proteger todos los cuadros a un metro (a izquierda y derecha) de la posición en la que se encuentre situado.

1. Diseñar un algoritmo mediante la técnica Greedy que coloque a los vigilantes a lo largo del pasillo minimizando el número de vigilantes a contratar.
2. Indicar si el heurístico utilizado encuentra siempre la solución óptima y, en su caso, demostrarlo. En caso contrario, proponer un contraejemplo.

Puntuaciones: [Ejercicio 1: 6 puntos; Ejercicio 2: 4 puntos]

Parcial Bloque 2, 2018/2019

Ejercicio 1 (Programación Dinámica)

1. Sea int[][] S una matriz de tamaño $n \times m$, siendo n el tamaño de w_1 y m el de w_2.

Entonces, $S(i, j)$ representa la cadena más corta que es supersecuencia de los primeros i caracteres de w_1, y los primeros j caracteres de w_2. Notación: $w_1[i]$ representa el carácter i-ésimo de w_1.

$$S(i,j) = \begin{cases} i, \text{ si } j = 0 \\ j, \text{ si } i = 0 \\ \min\left(S(i,j-1)+1, S(i-1,j)+1\right), \\ \quad \text{si } 0<i\leq n, 0<j\leq m, w_1[i] \neq w_2[j] \\ 1 + S(i-1,j-1), \text{ si } 0<i\leq n, \\ \quad 0<j\leq m, \ w_1[i] = w_2[j] \end{cases}$$

2. Supongamos que $\alpha \in \mathbb{N}$ es la solución de $S(i,j)$, con $0 \leq i \leq n$, $0 \leq j \leq m$, y que además es óptima. Como necesitamos ver que cualquier subestructura es óptima, podemos asumir que $i, j \neq 0$, pues está claro que en estos casos la solución es la palabra en sí.

• $w_1[i] \neq w_2[j]$: tenemos que ver que $S(i, j-1)$ y $S(i-1, j)$ son soluciones también óptimas. Haremos la segunda, y la otra es análoga.

Supongamos, por reducción al absurdo, que $S(i-1, j) = \alpha$ no es óptima, esto es, que existe otra cadena de longitud $\beta < \alpha$, y que es supersecuencia de w_1 sin la última letra, y de w_2.

En este caso, podemos añadir la última letra de w_1 a la cadena de longitud β, y nos quedaría una cadena de longitud $\beta + 1 < \underline{\alpha + 1 = S(i, j)}$, y que además es solución a nuestro problema general que era óptimo, lo cual es una contradicción.

Lo subrayado se tiene por cómo hemos definido la ecuación de recurrencia para este caso.

• $w_1[i] = w_2[j]$, es completamente análogo, salvo por que le quitamos la última letra a w_1 y w_2, así que no lo hago :).

Por lo tanto, el problema cumple la propiedad de subestructura óptima.

3. Si trabajamos con las casillas $i, j > 0$, necesitaremos la solución de $i-1, j-1$, luego basta con recorrer de izquierda a derecha, y de arriba abajo.

```
public static int SSM (String w1, String w2){
    int n = w1.length();
    int m = w2.length();
    int[][] S = new int[n+1][m+1];

  for (int i=0; i≤n; i++){
    for (int j=0; j≤m; j++){
        if(j==0) S[i][0] = i+1;
        else if(i==0) S[0][j] = j+1;
        else if (i!=j) S[i][j] = 1+ S[i-1][j-1];
        else{
            S[i][j] = 1+ Math.min (S[i-1][j],
S[i][j-1]);
        }
    }
  } return S[n][m];
}
```

4.

			R	A	T	Ó	N
		0	1	2	3	4	5
	0	0	1	2	3	4	5
G	1	1	2	3	4	5	6
A	2	2	3	3	4	5	6
T	3	3	4	5	4	5	6
O	4	4	5	6	5	5	6

Solución:
GRATÓN. (6)

Ejercicio 2 (Algoritmos Voraces)

1. Consideraremos como estructura de datos para almacenar la solución, una lista L de enteros, donde $L = \{l_i\}$, $l_i = j$ significa que el i-ésimo vigilante se encuentra en la posición j.

- esFinal $(L) = \forall x_i \in X, \exists \alpha, 0 \leq \alpha < L.size() /$

$x_i \in [l_\alpha - 1, l_\alpha + 1]$.

- esVálida (L, k) = True. (Siempre le puedo meter más vigilantes, para ser válida no tiene por qué ser eficiente ni final).

- Sea $X' = \{x_1', \ldots, x_n'\}$ tal que

X' es una permutación de X que verifica que $x_i' \leq x_j'$ $\forall i \leq j$. Entonces:

$$\text{Sele}(L) = x_i' + 1, \text{ donde } \exists_j \in \{0, \ldots, L.size()-1\}$$

con $x_{i-1}' \in [l_j - 1, l_j + 1]$, pero $x_i' \notin [l_k - 1, l_{k+1}]$ $\forall k \in \{0, \ldots, L.size()-1\}$, esto es, x_i' es el menor elemento todavía no cubierto por ningún vigilante.

- ini $= \{\}$.

No nos piden implementar el código, así que pasaré directamente al apartado 2.

Me refiero a nuestro profesor.

2. Nuestro heurístico siempre encuentra la solución óptima. Vamos a demostrarlo por inducción sobre el número de vigilantes, de la solución aportada por nuestro heurístico, que pertenecen a una solución óptima.

- Caso base: Supongamos que queremos vigilar a x_1', esto es, el cuadro más a la izquierda, y sea $L = \{l_0, \dots, l_m\}$ la solución dada por nuestro heurístico, y $OPT = \{o_0, \dots, o_m\}$ una solución óptima, y sin pérdida de gen., ordenada.

  - Si $l_0 = o_0$, entonces $l_0 \in OPT$, luego tenemos un vigilante en una solución óptima, y hemos acabado.

  - Si $l_0 \neq o_0$, entonces $l_0 > o_0$, pues de lo contrario, $x_1' \notin [o_0 - 1, o_0 + 1]$, y como OPT está ordenada, OPT no sería solución.

  Pero entonces, como x_1' es el primer cuadro, $[o_0, x_1') \cap X = \emptyset$, y como $[x_1', o_0 + 1] \subset$ $\subset [x_1' = l_0 - 1, l_0 + 1]$, entonces $OPT' = \{l_0, o_1, \dots, o_n\}$ es una solución óptima (tiene el mismo número de vigilantes que OPT).

- El caso inductivo se razona de la misma forma, suponiendo que coinciden los k primeros vigilantes y viendo que el $k+1$ vigilante puede ser sustituido, pues nuestro heurístico selecciona el menor de los cuadros no vigilados.

Por lo tanto, si una solución óptima tiene tamaño m, nuestra función de selección devuelve una solución de tamaño m, luego también es óptima.

PARCIAL BLOQUE 2 RESUELTO

Es el documento de informática que he subido que más me gusta en cuanto a dificultad y rigurosidad (hay dos demostraciones jajajaj). Espero que os sirva 😊

E. T. S. de Ingeniería Informática — Curso: 2018–19

Análisis y Diseño de Algoritmos — Control Bloque 2

Alumno: ______________________ **Grado**: ____________

Ejercicio 1

Se dice que una cadena s es una supersecuencia de otra cadena w si todos los caracteres de w aparecen en s en el mismo orden, aunque no sea de manera consecutiva. Por ejemplo, "camisa" es una supersecuencia de "casa" (i.e., "**ca**mi**sa**"). Dadas dos cadenas w_1 y w_2 el problema de la supersecuencia más corta es hallar la cadena s más corta que es a la vez supersecuencia de w_1 y w_2. Nótese que dicha cadena no tiene por qué ser única.

1. Encontrar una expresión recurrente para la longitud de la supersecuencia.
2. Demostrar que el problema exhibe la propiedad de subestructura óptima.
3. Confeccionar un algoritmo de programación dinámica bottom-up para resolver el problema. Sólo se desea conocer el número de caracteres de la supersecuencia más corta.
4. Aplicar a mano este algoritmo para encontrar una supersecuencia de longitud mínima para "gato" y "ratón" (los acentos pueden ignorarse). Es decir, rellenar la tabla.

Ejercicio 2

El CAC de Málaga recibe una colección de cuadros de suma importancia, y el museo, no depositando una gran confianza en sus ciudadanos, decide contratar un servicio de vigilancia. Todos los cuadros se encuentran situados en un pasillo, en la localización $X = \{x_1, x_2, ..., x_n\}$. $x_i \in \mathbb{R}$ representa el punto de colocación de cada cuadro a lo largo del pasillo. Cada vigilante puede proteger todos los cuadros a un metro (a izquierda y derecha) de la posición en la que se encuentre situado.

1. Diseñar un algoritmo mediante la técnica Greedy que coloque a los vigilantes a lo largo del pasillo minimizando el número de vigilantes a contratar.
2. Indicar si el heurístico utilizado encuentra siempre la solución óptima y, en su caso, demostrarlo. En caso contrario, proponer un contraejemplo.

Puntuaciones: [Ejercicio 1: 6 puntos; Ejercicio 2: 4 puntos]

Parcial Bloque 2, 2018/2019

Ejercicio 1 (Programación Dinámica)

1. Sea int[][] S una matriz de tamaño $n \times m$, siendo n el tamaño de w_1 y m el de w_2.

Entonces, $S(i,j)$ representa la cadena más corta que es supersecuencia de los primeros i caracteres de w_1, y los primeros j caracteres de w_2. Notación: $w_1[i]$ representa el carácter i-ésimo de w_1.

$$S(i,j) = \begin{cases} i, \text{ si } j=0 \\ j, \text{ si } i=0 \\ \min\left(S(i,j-1)+1,\ S(i-1,j)+1\right), \\ \quad \text{si } 0<i\leq n,\ 0<j\leq m,\ w_1[i] \neq w_2[j] \\ 1+S(i-1,j-1), \text{ si } 0<i\leq n, \\ \quad 0<j\leq m,\ w_1[i] = w_2[j] \end{cases}$$

2. Supongamos que $\alpha \in \mathbb{N}$ es la solución de $S(i,j)$, con $0\leq i\leq n$, $0\leq j\leq m$, y que además es óptima. Como necesitamos ver que cualquier subestructura es óptima, podemos asumir que $i,j \neq 0$, pues está claro que en estos casos la solución es la palabra en sí.

• $w_1[i] \neq w_2[j]$: tenemos que ver que $S(i, j-1)$ y $S(i-1, j)$ son soluciones también óptimas. Haremos la segunda, y la otra es análoga.

Supongamos, por reducción al absurdo, que $S(i-1, j) = \alpha$ no es óptima, esto es, que existe otra cadena de longitud $\beta < \alpha$, y que es supersecuencia de w_1 sin la última letra, y de w_2.

En este caso, podemos añadir la última letra de w_1 a la cadena de longitud β, y nos quedaría una cadena de longitud $\beta + 1 < \underline{\alpha + 1 = S(i, j)}$, y que además es solución a nuestro problema general que era óptimo, lo cual es una contradicción.

Lo subrayado se tiene por cómo hemos definido la ecuación de recurrencia para este caso.

• $w_1[i] = w_2[j]$, es completamente análogo, salvo por que le quitamos la última letra a w_1 y w_2, así que no lo hago :).

Por lo tanto, el problema cumple la propiedad de subestructura óptima.

3. Si trabajamos con las casillas $i, j > 0$, necesitaremos la solución de $i-1, j-1$, luego basta con recorrer de izquierda a derecha, y de arriba abajo.

```
public static int SSM (String w1, String w2){
    int n = w1.length();
    int m = w2.length();
    int[][] S = new int[n+1][m+1];

  for (int i=0; i<=n; i++){
    for (int j=0; j<=m; j++){
        if(j==0) S[i][0] = i+1;
        else if(i==0) S[0][j] = j+1;
        else if (i!=j) S[i][j] = 1+ S[i-1][j-1];
        else{
            S[i][j] = 1+ Math.min (S[i-1][j],
S[i][j-1]);
        }
    }
  } return S[n][m];
}
```

4.

		R	A	T	Ó	N	
		0	1	2	3	4	5
	0	0	1	2	3	4	5
G	1	1	2	3	4	5	6
A	2	2	3	3	4	5	6
T	3	3	4	5	4	5	6
O	4	4	5	6	5	5	6

Solución:
GRATÓN. (6)

Ejercicio 2 (Algoritmos Voraces)

1. Consideraremos como estructura de datos para almacenar la solución, una lista L de enteros, donde $L=\{l_i\}$, $l_i = j$ significa que el i-ésimo vigilante se encuentra en la posición j.

- $esFinal(L) = \forall x_i \in X, \exists \alpha, 0 \leq \alpha < L.size() /$
$x_i \in [l_\alpha - 1, l_\alpha + 1]$.

- $esVálida(L, k) = True$. (Siempre le puedo meter más vigilantes, para ser válida no tiene por qué ser eficiente ni final).

- Sea $X' = \{x_1', \dots, x_n'\}$ tal que X' es una permutación de X que verifica que $x_i' \leq x_j'$ $\forall i \leq j$. Entonces:

$$Sele(L) = x_i' + 1, \text{ donde } \exists_j \in \{0, \dots, L.size()-1\}$$

con $x_{i-1}' \in [l_j - 1, l_j + 1]$, pero $x_i' \notin [l_k - 1, l_{k+1}]$ $\forall k \in \{0, \dots, L.size()-1\}$, esto es, x_i' es el menor elemento todavía no cubierto por ningún vigilante.

- $ini = \{\}$.

No nos piden implementar el código, así que pasaré directamente al apartado 2.

Me refiero a nuestro profesor.

2. Nuestro heurístico siempre encuentra la solución óptima. Vamos a demostrarlo por inducción sobre el número de vigilantes, de la solución aportada por nuestro heurístico, que pertenecen a una solución óptima.

- Caso base: supongamos que queremos vigilar a x_1', esto es, el cuadro más a la izquierda, y sea $L = \{l_0, \dots, l_m\}$ la solución dada por nuestro heurístico, y $OPT = \{o_0, \dots, o_m\}$ una solución óptima, y sin pérdida de gen., ordenada.

  - Si $l_0 = o_0$, entonces $l_0 \in OPT$, luego tenemos un vigilante en una solución óptima, y hemos acabado.

  - Si $l_0 \neq o_0$, entonces $l_0 > o_0$, pues de lo contrario, $x_1' \notin [o_0 - 1, o_0 + 1]$, y como OPT está ordenada, OPT no sería solución.

    Pero entonces, como x_1' es el primer cuadro, $[o_0, x_1') \cap X = \emptyset$, y como $[x_1', o_0+1] \subset$ $\subset [x_1' = l_0 - 1, l_0 + 1]$, entonces $OPT' = \{l_0, o_1, \dots, o_n\}$ es una solución óptima (tiene el mismo número de vigilantes que OPT).

- El caso inductivo se razona de la misma forma, suponiendo que coinciden los k primeros vigilantes y viendo que el $k+1$ vigilante puede ser sustituido, pues nuestro heurístico selecciona el menor de los cuadros no vigilados.

Por lo tanto, si una solución óptima tiene tamaño m, nuestra función de selección devuelve una solución de tamaño m, luego también es óptima.