Parcial de Programación 3

27 de noviembre de 2017

En recuadros con este formato aparecerán aclaraciones que cumplen una función explicativa pero que no eran requeridos como parte de la solución.

Ejercicio 1 (20 puntos)

Dada una secuencia $A = a_1, \ldots, a_n$ con n caracteres.

Considere las siguientes definiciones:

Subsecuencia: A' es una subsecuencia de A, si es una secuencia formada por caracteres pertenecientes a A, y además se cumple que el orden en que aparecen en A se respeta en A'. Por ejemplo: Si A=f**ot**og**ra**fo, entonces **otra** es una subsecuencia de A, y **rato** no lo es.

Substring: A' es un substring de A, si es una subsecuencia de A que además cumple que los caracteres en A' ocurren de forma consecutiva en A. Por ejemplo, si A=foto**grafo**, **grafo** es un substring de A, y otra no lo es.

Considere el siguiente pseudocódigo,

```
int f (char* A, char* B) {
    n = length(A)
    m = length(B)
    int Mt [n+1][m+1];
    int Mp [n+1][m+1];

    for(int i=0; i<=m; i++){
        Mt[0][i] = 0;
        Mp[0][i] = 0;
    }

    for(int i=0; i<=n; i++){
        Mt[i][0] = 0;
        Mp[i][0] = 0;
    }

    for(int i=1; i<=n; i++){
        for(int j=1; j<=m; j++){
            if (A[i] == B[j]){
                Mp[i][j] = 1+ Mp[i-1][j-1];
            }
            else{
                Mp[i][j] = 0;
            }
            Mt[i][j] = max(Mp[i][j], Mt[i-1][j],Mt[i][j-1]);
        }
    }

    return Mt[n][m];
}
```

Listado 1: Algoritmo 1

(a) Realice el proceso de ejecución para la entrada $A = la$ $B = ala$, escriba en su respuesta solamente el estado final de las matrices Mt y Mp.

(b) Indique qué se almacena en cada estructura auxiliar Mt y Mp.

(c) ¿Qué problema resuelve?

(d) Plantee la recurrencia a partir de la cual se construyó el pseudocódigo.

(e) I. Escriba el pseudocódigo una solución recursiva del problema. Considere la siguiente firma para el pseudocódigo: `int f (int n, int m, int** Mp)`

II. Indique qué invocaciones serían necesarias para resolver el problema Mt(2,3). Sugerencia: utilice el árbol de recursión.

III. ¿Qué inconveniente le encuentra a esta solución?

(f) Determine la complejidad en espacio de memoria del Algoritmo 1. ¿Hay alguna forma de reducirla? Justfique.

Solución:

(a) $A = la$ $B = ala$

Mp :

i \ j		a	l	a
	0	0	0	0
l	0	0	1	0
a	0	1	0	2

Mt :

i \ j		a	l	a
	0	0	0	0
l	0	0	1	1
a	0	1	1	2

(b) Los sufijos de una cadena $a_1, \ldots, a_i$ son los substrings $a_h, \ldots, a_i$, con $1 \leq h \leq i$.

$Mt[i][j]$ representa el largo del substring común más largo entre $a_1, \ldots, a_i$ y $b_1, \ldots, b_j$.

$Mp[i][j]$ representa el largo del sufijo común más largo entre $a_1, \ldots, a_i$ y $b_1, \ldots, b_j$.

(c) Resuelve el problema de encontrar el largo del substring común más largo entre A y B.

(d) Se considera un máximo total, representado por Mt, que es donde se almacena la solución al problema y un máximo parcial, representado por Mp, que almacena el substring de largo máximo utilizando los caracteres en la posición actual.

De esta forma, el problema se formaliza como sigue:

$$Mt(i,j) = \begin{cases} 0, & \text{si } i = 0 \text{ o } j = 0 \\ max\{Mp(i,j), Mt(i-1,j), Mt(i,j-1)\}, & \text{en caso contrario} \end{cases}$$

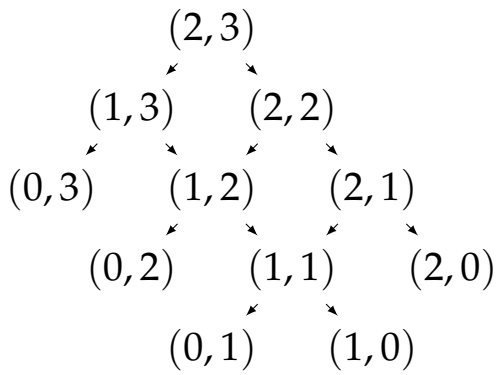
$$Mp(i,j) = \begin{cases} 0, & \text{si } i = 0 \text{ o } j = 0 \\ \mathbb{1}_{a_i,b_j} \times (Mp(i-1,j-1)+1), & \text{en caso contrario} \end{cases}$$

$$\mathbb{1}_{a_i,b_j} = \begin{cases} 0, & \text{si } a_i \neq b_j \\ 1, & \text{en caso contrario} \end{cases}$$

(e) I.

```
int f (int n, int m, int** Mp) {
    if (n == 0 || m ==0){
        return 0    ;
    }
    else{
        return max{f(n-1,m, Mp), f(n,m-1,Mp), Mp[n][m]};
    }
}
```

II.



III. El problema que tiene lo anterior es que se invoca muchas veces al mismo subproblema, calculando más de una vez lo mismo. Esta superposición de problemas es lo que evita la técnica Programación Dinámica.

(f) La complejidad en espacio del algoritmo es $O(n \times m)$ porque se utilizan dos matrices de tamaño $(n + 1) \times (m + 1)$.

Como para resolver el problema $Mt(i, j)$ solo hay que mirar los subproblemas $Mt(i - 1, j)$, $Mt(i, j - 1)$ alcanza con mantener, además de la fila actual, la fila anterior con un arreglo de tamaño $m + 1$ para Mt. Como además $Mt(i, j)$ utiliza $Mp(i, j)$ y para mantener esta información solo se necesita el valor $Mp(i - 1, j - 1)$ alcanzaría con tener un arreglo de tamaño $m + 1$. Por lo tanto la complejidad en espacio del algoritmo sería $O(m)$.

De manera análoga se puede resolver el problema manteniendo dos columnas para cada matriz. Con esta versión la complejidad en espacio del algoritmo sería $O(n)$.

Ejercicio 2 (10 puntos)

Para las siguientes afirmaciones sobre los Algoritmos Aleatorizados indicar si son verdaderas o falsas, justificando brevemente y dando un ejemplo:

(a) Solamente son útiles para diseñar algoritmos para problemas intratables.

(b) La aleatoriedad sirve, entre otras cosas, para diseñar algoritmos eficientes incluso ante el peor caso, y correctos con cierta probabilidad.

(c) Son deterministas en función de la entrada que reciben.

(d) La aleatoriedad sirve, entre otras cosas, para diseñar algoritmos correctos que en el caso promedio el orden de tiempo de ejecución es distinto al del peor caso.

(e) La idea de los Algoritmos Aleatorizados es estudiar cómo se comporta un algoritmo cualquiera en el caso promedio (no dar un ejemplo para esta parte).

Solución:

(a) **Falso**. En problemas tratables son útiles para mejorar la eficiencia (en tiempo o en memoria) o introducir simplicidad. Por ejemplo, Quicksort resuelve un problema tratable con aleatoriedad.

(b) **Verdadero**. Por ejemplo, el algoritmo que elige aristas no cubiertas al azar para resolver el problema *Vertex Cover* da el óptimo con cierta probabilidad y es polinomial.

(c) **Falso**. Por ejemplo, si en Quicksort el pivot se elige al azar, dada la misma entrada el algoritmo puede tomar distintas decisiones.

(d) **Verdadero**. Un ejemplo es Quicksort, que en promedio es $O(n \log n)$ pero en el peor caso es $O(n^2)$.

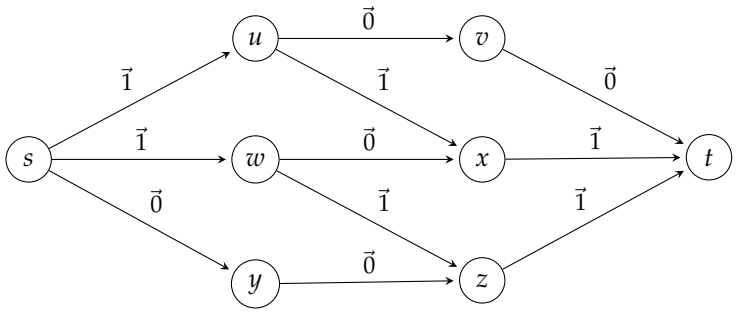
> Otros ejemplos: el algoritmo para encontrar el par de puntos más cercanos explicado en Kleinberg 13.7 y el algoritmo visto en monitoreos para hallar el Convex Hull.

(e) **Falso**. Esto no tiene que ver con la definición de los Algoritmos Aleatorizados, que es la de tener decisiones aleatorias para diseñar algoritmos.

Ejercicio 3 (10 puntos)

Sea $G(V, E)$ una red de flujo s-t con capacidades enteras en sus aristas.

(a) Sea la instancia de la red con capacidad constante de valor uno y flujo según aristas



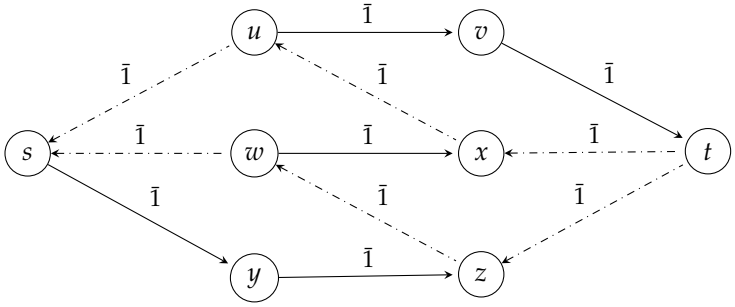
determinar su flujo máximo aplicando el algoritmo de Ford-Fulkerson a partir del flujo dado. [Mostrar los pasos del algoritmo mediante secuencias de grafos residuales y grafos con flujo.]

(b) Suponer que la red, con capacidad constante de valor uno, se construye a partir de un grafo bipartito al que se le agregan un nodo fuente, un nodo destino, aristas entre el nodo fuente y los nodos de una partición y aristas entre los nodos de la otra partición y el nodo destino. Formalmente, $G(V, E)$ se construye a partir de un grafo bipartito $G'(U \cup W, E' \subseteq U \times W)$ tal que $m = |U|$, $n = |W|$, y donde $V = U \cup W \cup \{s, t\}$ y $E = \{(u, w) | \{u, w\} \in E'\} \cup \{(s, u) | u \in U\} \cup \{(w, t) | w \in W\}$.

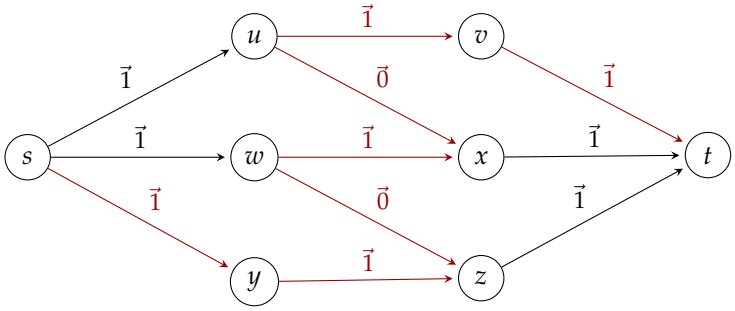
Determinar el largo máximo (cota superior) de los caminos de incremento (augmenting path) obtenidos al aplicar el algoritmo de Ford-Fulkerson en todo $E' \subseteq U \times W$ posible. [Establecer la cota en función de m y n.]

Solución:

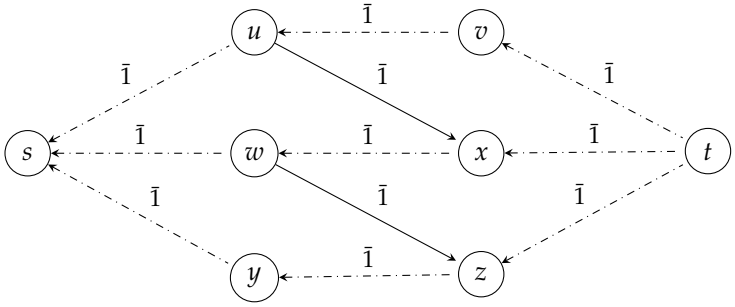
(a) Sea el grafo residual del flujo dado (con las aristas de retorno destacadas en punteado)



En el se determina el camino de incremento (s, y, z, w, x, u, v, t) con flujo de valor uno. Luego de aplicarlo en la red se tiene el nuevo flujo (con cambios respecto al anterior destacados en rojo)



Para este nuevo flujo se tiene el grafo residual



en el cual no hay camino de incremento. Por lo tanto el flujo es máximo y de valor tres.

(b) El camino de incremento de s a t más largo se obtendría al recorrer la máxima cantidad de nodos del grafo bipartito. El camino estaría conformado por el nodo s, al menos un nodo de U, al menos un nodo de W y el nodo t. Para que el camino incluya más de dos nodos del grafo bipartito tendría que haber aristas de retroceso en el grafo residual, tales que luego de acceder a un nodo de W se pueda "retornar" a un nodo de U. En el apartado anterior se tiene una instancia de la red con un flujo que determina un camino, con aristas de retorno, que recorre todos los nodos.

Si $m = n$ podría haber un camino que recorriera todos los nodos de G. Al recorrer todos los nodos, cantidad $2m + 2$, tiene largo $2m + 1$ (aristas).

Si $m \neq n$ el camino no puede recorrer todos los nodos de G', dado que no puede repetir nodos, por lo que a lo sumo puede recorrer $2\,\text{mín}\{m, n\}$ nodos. Entonces el camino más largo contiene $2\,\text{mín}\{m, n\} + 2$ nodos y su largo es $2\,\text{mín}\{m, n\} + 1$ (aristas).

Ejercicio 4 (10 puntos)

Para cada una de las siguientes preguntas, indique si la respuesta es “Sí”. “No”, o “No se sabe. Si se supiera resolvería el problema de decidir si $\mathcal{P} = \mathcal{NP}$”. Dé una explicación breve (una o dos oraciones) de su respuesta. Sugerencia: utilice su conocimiento de a que clase pertenece cada uno de los problemas mencionados en las preguntas.

Definamos la siguiente versión del problema de la *Programación de Intervalos* (Interval Scheduling) como en formato de problema de decisión: dada una colección de intervalos en una linea de tiempo, y una cota k, ¿la colección contiene un subconjunto de intervalos no superpuestos de tamaño al menos k?

(a) ¿Es verdad que *Programación de Intervalos* $\leq_P$ *Cubrimiento de Vértices* (Vertex Cover)?

(b) ¿Es verdad que *Conjunto Independiente* (Independent Set) $\leq_P$ *Programación de Intervalos*?

Solución:

(a) Si. Una solución puede ser: Interval Schedulling puede ser resuelto en tiempo polinómico, y por lo tanto tambien puede ser resuelto en tiempo polinómico con acceso a una caja negra que resuelva el problema Vertex Cover (llamándola 0 veces). Otra solución posible es: Interval Scheduling es NP y cualquier problema NP puede ser reducido al Vertex Cover.

(b) Esto es equivalente a decir que P = NP. Si P = NP, entonces Independent Set puede ser resuelto en tiempo polinómico y ser reducido a Interval Scheduling. Lo contrario también sería verdad ya que Independent Set en NP-Completo.