Examen de Programación 3
20 de diciembre de 2024

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

Ejercicio 1 (50 puntos)

Se nos pide decidir si un conjunto de n tareas es *realizable* dada la existencia de dependencias entre algunos pares de tareas. Tenemos un conjunto T_1 de pares (i, j) de tareas en los que la tarea j solo puede empezar a realizarse si la tarea i ya está terminada. Hay otro conjunto T_2 de pares de tareas que comparten un supervisor al que solo se lo puede convocar una vez para ambas tareas. Por lo tanto, si $\{i, j\}$ es un par de tareas de T_2 los lapsos en que se realizan las tareas tienen que tener algún instante común.

La instancia (n, T_1, T_2) es realizable si es posible construir una secuencia S de largo $2n$ en la que por cada i perteneciente a $\{1, \ldots, n\}$ se incluyan elementos s_i y f_i que representan respectivamente el inicio y el fin de la tarea i, y los segmentos $s_i, \ldots, f_i$ de S cumplan las restricciones definidas en T_1 y T_2.

Por ejemplo, sean $n = 3$, $T_1 = \{(1, 2)\}$ y $T_2 = \{\{1, 3\}, \{2, 3\}\}$. Esta es una instancia realizable porque, entre otras, las secuencias $s_1, s_3, f_1, s_2, f_3, f_2$ y $s_3, s_1, f_1, s_2, f_2, f_3$ cumplen todas las restricciones:

1. s_i precede a f_i para $1 \leq i \leq 3$.
2. f_1 precede a s_2, por lo que se cumple la restricción definida en T_1.
3. el segmento $s_3, \ldots, f_3$ se intersecta con los segmentos $s_1, \ldots, f_1$ y $s_2, \ldots, f_2$, por lo que se cumplen las dos restriccones definidas en T_2.

Como ejemplo de instancia no realizable consideremos $n = 4$, $T_1 = \{(1, 2), (3, 4)\}$ y $T_2 = \{\{1, 4\}, \{2, 3\}\}$. Para cada i, s_i debe preceder a f_i por definición. Además, por T_1, f_1 debe preceder a s_2, y f_3 debe preceder a s_4. Por lo tanto se debe cumplir que s_1, f_1, s_2, f_2 estén en ese orden, y también que s_3, f_3, s_4, f_4 estén en ese orden. Además, por T_2, $s_2, \ldots, f_2$ y $s_3, \ldots, f_3$ deben tener algún instante común, por lo que s_2 debe preceder a f_3. Entonces tenemos que f_1, s_2, f_3, s_4 tienen que estar en ese orden. Como f_1 está antes que s_4 los segmentos $s_1, \ldots, f_1$ y $s_4, \ldots, f_4$ no pueden tener instantes en común.

Se pide:

1. Escriba un algoritmo que dados n, T_1 y T_2 informe si es una instancia realizable, y en el caso de que lo sea, devuelva una secuencia S que cumpla las restricciones definidas por T_1 y T_2. El algortimo debe admitir una implementación cuyo tiempo de ejecución sea $O(m + n)$, donde $m = |T_1| + |T_2|$.
2. Demuestre que su algoritmo es correcto.
3. Demuestre que su algoritmo admite una implementación cuyo tiempo de ejecución es $O(m + n)$.

Solución:

En este ejercicio se espera:

Una solución que obtenga lo que se pide, o sea, un algoritmo que indique si la instancia es realizable, y en caso de serlo, devuelva una secuencia de largo $2n$ que cumpla las restricciones pedidas, la demostración de que el algoritmo es correcto, y la demostración de que puede implementarse en el orden de tiempo pedido.

(a) La solución típica consiste en construir un grafo dirigido, indicar si la instancia es realizable si y solo si el grafo es acíclico, y en caso de serlo, obtener un ordenamiento topológico que se interpreta como la secuencia pedida. Pueden usarse, citando, pero sin tener que reescribir, algoritmos vistos en el teórico.

(b) Estructurar correctamente una demostración del tipo pedido:

- Si el algoritmo devuelve una secuencia (y, por lo tanto, indica que la instancia es realizble) esa secuencia debe cumplir con todas las restricciones pedidas.
  Se debe demostrar por qué el algoritmo obtiene un ordenamiento topológico OT (puede citarse, sin necesidad de reescribir, resultados conocidos del curso), y, haciendo valer la definición de ordenamiento topológico y la construcción del grafo, por qué OT es una secuencia que cumple con las restricciones pedidas.
- Si la instancia es realizable el algoritmo devuelve una secuencia que cumple las restricciones pedidas.
  Se debe demostrar que si la instancia es realizable el grado es acíclico y que en ese caso el algortimo obtiene un ordenamiento topológico (puede citarse, sin necesidad de reescribir, resultados conocidos del curso).

(c)
- Se debe mencionar qué estructuras de datos se usan para poder cumplir el orden de tiempo pedido.
- Se debe relacionar los parámetros n y m del problema con $|V|$ y $|E|$ (los que habitualmente se usan para análisis de algoritmos de grafos).
- Analizar el tiempo de cada parte del algoritmo (puede citarse, sin necesidad de reescribir, resultados conocidos del curso).
- Aplicar herramientas básicas de análisis asintótico para combinar resultados (por ejemplo, suma de funciones).

(a) El algoritmo tiene dos pasos:

1. construir un grafo dirigido G, y
2. obtener un ordenamiento topológico de G y devolverlo como resultado, o informar que la instancia no es realizable.

El grafo se construye con $2n$ vértices que se etiquetan con los elementos de la secuencia que se quiere construir, esto es, $s_1, f_1, \ldots, s_n, f_n$. Se agregan aristas de acuerdo a las condiciones que deben cumplir los elementos de la secuencia S.

1. una arista (s_i, f_i) para $i \in \{1, \ldots, n\}$, para que el inicio de cada tarea preceda a su fin,
2. una arista (f_i, s_j) para cada $(i, j) \in T_1$, para que la tarea j empiece después que termina la tarea i,
3. dos aristas, (s_i, f_j) y (s_j, f_i) para cada $(i, j) \in T_2$, para que las tareas i y j tengán algún instante común.

Para el segundo paso se adapta el algoritmo del libro de referencia para detectar la existencia de un ciclo.

```
1 Algoritmo OT (G)
2     si G es vacío entonces
3         devolver la secuencia vacía
4     si no, si en G no hay vértices con grado de entrada 0 entonces
5         TERMINAR informando que la instancia no es realizable
6     en otro caso
7         Elegir un vértice v con grado de entrada 0
8         Remover v de G
9         devolver la secuencia formada por v seguida de OT (G)
```

(b) Veamos que si el algoritmo devuelve una secuencia S, esta cumple con las condiciones requeridas para que la instancia sea consistente.

La secuencia tiene que cumplir

1. El inicio, s_i, de cada tarea precede a su final, f_i.
2. El final de la tarea i precede al inicio de la tarea j para cada $(i, j) \in T_1$.
3. Los segmentos $s_i \dots f_i$ y $s_j \dots f_j$ tienen elementos en común para cada $\{i, j\} \in T_2$. Una condición necesaria y suficiente para esto es que los dos inicios, s_i, s_j precedan a los dos finales f_i, f_j.

Para cada relación de precedencia que debe ocurrir entre elementos de la secuencia en cada uno de estos puntos, se agregó una arista en el punto correspondiente del paso de construcción del grafo. El algoritmo OT es, excepto por la condición de la línea 4, el descrito en el libro de referencia. Entonces, por un resultado teórico, si el algoritmo devuleve una secuencia, esa secuencia es un ordenamiento topológico del grafo G. Por definición, esto implica que para cada arista (u, v) de G el vértice u preceda al vértice v en el ordenamiento. Y esto implica que cada relación de precedencia se cumple, por lo que la instancia es realizable.

Ahora veamos que si la instancia es realizable el algoritmo devuleve una secuencia que cumple todas las restricciones. Si es realizable hay una secuencia S' que cumple las precedencias entre pares de los tres puntos anteriores. Como por cada una de esas dependencias se agrega una arista en el grafo y no se agrega ninguna otra, la secuencia, considerada como vértices del grafo es un ordenamiento topológico. Esto implica, por un resultado teórico, que el grafo es acíclico. También por un resultado teórico, esto implica que nunca se va a cumplir la condición de la línea 4 del algoritmo. Por lo tanto el algoritmo va a terminar devolviendo una secuencia S (no necesariamente igual a S'), que por lo demostrado antes sabemos que cumple todas las restricciones requeridas.

(c) El grafo tiene $2n$ vértices y $n + |T_1| + 2|T_2|$ aristas. Entonces, como $m = |T_1| + |T_2|$ la cantidad de aristas es $O(n + m)$.

El grafo se implementa con listas de adyacencia. Es un resultado conocido del curso que el tiempo de ejecución de la construcción de un grafo $G = (V, E)$ es $O(|V| + |E|)$. Por lo tanto el tiempo de ejecución del primer paso del algoritmo es $O(m + n)$.

El algoritmo visto en el curso que construye un orden topológico también tiene tiempo de ejecución $O(m + n)$.

Como tenemos dos pasos, ambos con tiempo de ejecución $O(m + n)$, el tiempo de ejecución del algoritmo es $O(m + n)$.

Ejercicio 2 (50 puntos)

Los reposteros de una pequeña fábrica de pastas se proponen planificar su cocina para los próximos n días. En su menú tienen ravioles y sorrentinos y cada día pueden fabricar solo uno de esos dos tipos de pasta. La ganancia que obtienen por cada unidad de ravioles es g_r, y por cada unidad de sorentinos es g_s.

Por temas de limpieza de los equipos, solo pueden fabricar sorrentinos cada tres días; o sea, si el día i fabrican sorrentinos, los días $i + 1$ e $i + 2$ no pueden fabricar sorrentinos, aunque sí pueden fabricar ravioles.

Para cada día y cada tipo de pasta tienen un estimado de la cantidad de unidades que pueden vender. Para el día i, $1 \leq i \leq n$, denotamos la cantidad de unidades de ravioles y sorrentinos que estiman vender como r_i y s_i respectivamente.

Se asume que cada día pueden fabricar las cantidades demandadas y que van a poder vender lo que fabriquen. Por razones bromatológicas cada día deben desechar lo que no se vende, por lo que no fabricarían más unidades que las que estiman pueden vender.

Se pide:

(a) Escriba una relación de recurrencia que permita determinar la máxima ganancia acumulada que pueden obtener en los n días. Explique el significado de la función que defina y de sus parámetros. Justifique cada término de la relación de recurrencia.

(b) Escriba un algoritmo iterativo eficiente que, implementando la recurrencia de la parte (a), calcule la máxima ganancia que se puede obtener en los n días.

(c) Escriba un algoritmo eficiente que devuelve la planificación de que pasta se fabricará cada día para obtener la máxima ganancia. El resultado es una secuencia P de largo n, en donde P_i, $1 \leq i \leq n$, indica si el día i se debe fabricar ravioles o sorrentinos. Puede usar estructuras de datos usadas en el algoritmo de la parte (b).

Solución:

En este ejercicio se espera:

(a) Diseñar una descomposición del problema en una estructura de subinstancias (subproblemas) de manera apropiada para resolver el problema mediante programación dinámica. La solución de cada instancia se debe poder obtener de manera eficiente mediante las soluciones de algunas de sus subinstancias. Se debe:

- Definir una función que calcula la ganancia máxima acumulad en la subinstancia, por ejemplo $OPT(i)$, donde i representa un conjunto de días. Explicar qué significa el valor de $OPT(i)$. Definir cuál es el rango de valores de i.
- Explicar claramente que significa i. Por ejemplo, i puede representar los primeros i días, o las días desde el i hasta el n.
- Obtener el valor de $OPT(i)$ de los valores de OPT para subinstancias en las que el conjunto de días esté contenido de manera estricta en las de $OPT(i)$. Por ejemplo, si el parámetro i representa los días desde la i hasta el n, el cálculo de $OPT(i)$ se puede obtener de $OPT(i+1)$ y $OPT(i+3)$.
- Incluir los casos base, aquellos en los que i representa solo uno, dos o tres días.
- Explicar por qué el planteo es correcto.

(b) Traducir la recurrencia en un algoritmo iterativo. Para eso se va a poblar una tabla con una celda por cada subinstancia. Se debe:

- Empezar el algortimo con los casos base.
- Recorrer la tabla de modo que al evaluar la expresión con la que se calcula el valor de una celda, ya estén calculados los valores de las celdas involucradas en la expresión. Se debe calcular el valor de cada celda de la tabla.
- Utilizar la expresión definida en la recurrencia para asignar el valor de la celda.
- Devolver el valor que representa la ganancia máxima acumulada en los n días.

(c) Escribir un algoritmo que obtiene la solución del problema, esto es, una secuencia de largo n en la que en cada posición se indica el tipo de pasta a fabricar en el día correspondiente para que la ganancia acumulada en los n días sea máxima. Se debe:

- Recorrer la tabla construida en el punto anterior en el sentido opuesto del parámetro i a como fue construida. Es decir, se empieza por las celdas que representan las n días y se termina por las que representan solo un día.
- En cada paso se procesa una celda que representa una menos días que en el paso anterior.
- La determinación de la pasta a fabricar debe ser consistente con la relación de recurrencia con la que se calculó el valor de la celda.

(a) El subproblema i corresponde a los días $i, \ldots, n$. Entonces $OPT(i)$ es la máxima ganancia que se puede obtener desde el día i hasta el día n.

En el día i se puede elegir fabricar sorrentinos o ravioles. Si se elige fabricar sorrentinos, en los días $i+1$ e $i+2$ se debe fabricar ravioles. A la ganancia de esos tres días se le debe sumar la máxima ganancia que se puede obtener desde el día $i+3$ hasta el día n, ya que lo que se fabrica esos días no está condicionado por haber fabricado sorrentinos el día i. Si se elige fabricar ravioles a la ganacia de ese día se le debe sumar la máximo que se puede obtener desde el día $i+1$ hasta el día n. Se debe elegir fabricar el producto que maximiza esas dos sumas.

Como casos base se puede tomar $OPT(n+d) = 0$ para $d \in \{1, \ldots, 3\}$.

Entonces la relación de recurrencia es

$$OPT(n+d) = 0, 1 \leq d \leq 3$$
$$OPT(i) = \text{máx}\{s_i \cdot g_s + (r_{i+1} + r_{i+2})g_r + OPT(i+3), r_i \cdot g_r + OPT(i+1)\}, 1 \leq i \leq n.$$

(b)

```
1  Algoritmo Ganancia
2      OPT[n+d] ← 0, d ∈ {1, ..., 3}
3      para i desde n decrenentando hasta 1
4          OPT(i) ← máx{s_i · g_s + (r_{i+1} + r_{i+2})g_r + OPT(i+3), r_i · g_r + OPT(i+1)}
5      devolver OPT(1)
```

(c)

```
1  Algoritmo Planificacion
2      i ← 1
3      mientras i ≤ n
4          si OPT[i] = r_i · g_r + OPT(i+1) entonces
5              P[i] ← 'R'
6              i ← i + 1
7          en otro caso
8              P[i] ← 'S', P[i+1] ← 'R', P[i+2] ← 'R'
9              i ← i + 3
10     devolver P
```

Una solución alternativa similar a esta se obtiene considerando que $OPT(i)$ representa la ganancia máxima que se puede obtener desde el día 1 hasta el i.