UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Algorítmica (11593)
VA, Voraces y RP–13/enero/2020

Escuela Técnica Superior de Ingeniería Informática

etsinf

NOMBRE: _____ **NÚM.:** _____

1 **3.5 puntos**

Hay N tareas a ejecutar (numeradas de 1 a N) y M procesadores idénticos (numerados de 1 a M), con $M \leq N$, que trabajan en paralelo. El tiempo necesario para procesar la tarea i-ésima en cualquiera de los procesadores es t_i. Teniendo en cuenta que el orden en el que se ejecuten las tareas dentro de un mismo procesador no es significativo, diseña un ***algoritmo de Ramificación y Poda*** que determine qué tareas se deben ejecutar en cada procesador de forma que se minimice el tiempo de finalización de la tarea que termine más tarde.

a) Expresa el problema en términos de optimización: expresa formalmente el conjunto de soluciones factibles y la función objetivo a minimizar. Explica brevemente cómo expresas una solución x del conjunto X. Pon un ejemplo de solución factible para una instancia con 5 tareas y 3 procesadores.
b) Describe los siguientes conceptos sobre los estados que serán necesarios para el algoritmo:
   1) Representación de un estado (no terminal). Pon un ejemplo para la instancia con 5 tareas y 3 procesadores.
   2) Condición para que un estado sea solución. Pon un ejemplo para la instancia con 5 tareas y 3 procesadores.
   3) Identifica el estado inicial que representa todo el conjunto de soluciones factibles.
c) Define una función de ramificación. Contesta a las siguientes cuestiones:
   1) Explica la función.
   2) Define la función (en python o en lenguaje matemático).
   3) Pon un ejemplo para una instancia con 10 tareas y 5 procesadores.
d) Diseña una cota optimista no trivial. Contesta a las siguientes cuestiones:
   1) Explica la cota (caso general, de un estado intermedio).
   2) Explica el cálculo de la cota del estado inicial.
   3) Define la cota (en python o en lenguaje matemático).

Solución: Las soluciones factibles serán todas las posibles asignaciones de tareas a procesadores. Se deberá asignar cada tarea a un procesador, pudiendo expresarse una solución como una N-tupla $(x_1, x_2, \ldots, x_N)$ tal que x_i es el procesador asignado a la tarea i-ésima, es decir, un valor de 1 a M. Para este problema no hay ninguna restricción adicional:

$$X = \{(x_1, \ldots, x_N) \mid x_i = [1..M], 1 \leq i \leq N\}$$

Calculemos ahora el tiempo total empleado en la solución $(x_1, x_2, \ldots, x_N)$ para el procesador m:

$$P_m(x_1, x_2, \ldots, x_N) = \sum_{1 \leq i \leq N : x_i = m} t_i$$

La función objetivo de una solución $(x_1, x_2, \ldots, x_N)$ será el tiempo necesario para procesar las N tareas, que vendrá determinado por el máximo tiempo entre todos los procesadores:

$$\max_{1 \leq m \leq M} P_m(x_1, x_2, \ldots, x_N) = \max_{1 \leq m \leq M} \sum_{1 \leq i \leq N : x_i = m} t_i$$

Se desea encontrar la asignación tareas a procesadores que minimiza esta función objetivo. El espacio de búsqueda X, tal y como se ha definido, es de tamaño M^N. Pero, como se nos indica que todos los procesadores son iguales, hay muchas soluciones equivalentes en ese espacio de búsqueda. Por ejemplo, si hay 5 tareas y 3 procesadores, la solución (1,1,2,3,2) es equivalente a la solución (2,2,3,1,3), ya que el tiempo de finalización sería el mismo en ambos casos, Para evitarlo (esto es análogo al planteamiento de la solución para el algoritmo del empaquetado en cajas o bin-packing) asumiremos, al ramificar un estado $(x_1, x_2, \ldots, x_k, ?)$, lo siguiente:

- Siempre se asigna la tarea 1 al primer procesador, por lo que el estado incial que representa implícitamentea todo el conjunto de soluciones factibles será (1,?).
- El número de estados generados al ramificar otro está acotado por el número de procesadores ya asignados: la nueva tarea $k+1$-ésima se podrá asignar a cada uno de los procesadores ya asignados a las tareas precedentes o bien al siguiente libre (si es que queda alguno libre):

$$branch((x_1, ..., x_k, ?)) = \{(x_1, ..., x_k, x_{k+1}, ?) | x_{k+1} \in [1..\,\text{mín}((\max_{1 \leq i \leq k} x_i) + 1, M)]\}$$

  Por ejemplo, para una instancia con 10 tareas y 5 procesadores:

$$branch((1, 2, 1, ?)) = \{(1, 2, 1, 1, ?), (1, 2, 1, 2, ?), (1, 2, 1, 3, ?)\}$$

  No tiene sentido asignar la nueva tarea a ningún otro procesador.

Como hemos visto, una solución parcial puede representarse como un prefijo de una solución completa. Es decir, mediante una tupla $(x_1, \ldots, x_k, ?)$ de longitud $k < N$. Para que un estado o solución parcial $(x_1, \ldots, x_k, ?)$ sea solución basta con que $k = N$.

Para el caso que nos ocupa de reparto de carga en procesadores paralelos, como sabemos a priori el tiempo de ejecución de todas las tareas, con un tiempo total $T = \sum_{1 \leq i \leq N} t_i$, el reparto óptimo sería aquel que asignara el mismo tiempo a cada procesador: $\lceil T/M \rceil$. Generalmente, este reparto no será posible porque exigiría fraccionar algunas de las tareas en más de un procesador, lo que no es contemplado en esta situación. Sin embargo, este valor nos proporciona una cota inferior al tiempo mínimo requerido, ya que la solución óptima no podrá ser mejor. Calculemos, para cada estado intermedio $(x_1, \ldots, x_k, ?)$, la carga del procesador con más carga:

$$maxP = maxP((x_1, \ldots, x_k, ?)) = \max_{1 \leq m \leq M} \sum_{1 \leq i \leq k : x_i = m} t_i$$

y lo comparamos con el óptimo teórico $\lceil T/M \rceil$, siendo el máximo de ambos valores una cota inferior a la mejor solución alcanzable a partir de ese estado:

$$cota\_inferior((x_1, \ldots, x_k, ?)) = \max(maxP, \lceil T/M \rceil)$$

De esta forma se explorarán primero los estados que no sobrepasen el coste óptimo teórico y, agotados estos, los que lo sobrepasen en menor medida. Ello forzará la exploración a empezar repartiendo lo más posible las tareas entre los procesadores (recorrido en anchura), al menos en los primeros niveles.

Otra cota inferior un poco más elaborada consistiría en asumir que las tareas sin asignar se pueden fraccionar de forma arbitraria, por lo que, en primer lugar, cubriríamos el tiempo sobrante en cada procesador hasta alcanzar el tiempo del procesador con la máxima carga, y el tiempo restante de las tareas (si es que lo hay) se dividiría entre los M procesadores. La cota inferior de la mejor solución alcanzable a partir del estado $(x_1, \ldots, x_k, ?)$ sería el máximo valor entre el tiempo máximo que llevamos $maxP$ y ese reparto:

1. Suma de tiempo "libre" en cada procesador: $Tp = \sum_{m=1}^{M} (maxP - \sum_{1 \leq i \leq k : x_i = m} t_i)$
2. Suma de tiempo de tareas aún no asignadas: $Tt = \sum_{i=k+1}^{N} t_i$

3. Si $Tt \leq Tp$, una cota inferior para ese estado es $maxP$. En caso contrario, la cota sería $maxP + \lceil (Tt - Tp)/M \rceil$

$$cota\_inferior((x_1, \ldots, x_k, ?)) = \text{máx}\,(maxP, maxP + \lceil (Tt - Tp)/M \rceil)$$

2 **1.5 punto**

Podemos mejorar el algoritmo anterior incializando la variable "mejor solución vista hasta el momento" a una solución factible arbitraria. Pero sería mejor usar una estrategia voraz en lugar de usar una cualquiera. Para ello, vuelve al planteamiento original:

"Hay N tareas a ejecutar (numeradas de 1 a N) y M procesadores idénticos (numerados de 1 a M), con $M \leq N$, que trabajan en paralelo. El tiempo necesario para procesar la tarea i-ésima en cualquiera de los procesadores es t_i. Teniendo en cuenta que el orden en el que se ejecuten las tareas dentro de un mismo procesador no es significativo, diseña un ***algoritmo Voraz*** que determine qué tareas se deben ejecutar en cada procesador de forma que se intente minimizar en lo posible el tiempo de finalización de la tarea que termine más tarde."

1. Explica en lenguaje natural la estrategia voraz a seguir.
2. Indica el coste del algoritmo a desarrollar.
3. Escribe una función Python que reciba la lista de los tiempos de las N tareas y el número de procesadores M y que devuelva una lista con la asignación de tareas a procesadores.

Solución: En cuanto a usar una solución inicial en el algoritmo de Ramificación y Poda, cualquier reparto "ciego" de las N tareas a los M procesadores sería válido. Para mejorar el reparto se puede usar el siguiente algoritmo voraz: ordenar las tareas por duración decreciente y asignarlas siguiendo una ruta circular, es decir, considerando el primer procesador como adyacente al último (por ejemplo, para la instancia presentada con 5 tareas y 3 procesadores, y asumiendo que las tareas ya están ordenadas por duración decreciente, la solución voraz sería: (1,2,3,1,2). También se podría hacer una asignación "en zig-zag", para el ejemplo sería: (1,2,3,3,2). El coste de este algoritmo voraz está dominado por la ordenación de las tareas, con un coste $O(N \log N)$.

Otra estrategia voraz sería asignar la siguiente tarea al procesador que tenga el menor tiempo acumulado hasta el momento. Esta idea se puede implementar eficientemente manteniendo en un minheap el tiempo acumulado en cada procesador (inicialmente todo a 0s) e ir recorriendo las tareas (en cualquier orden): se asigna esa tarea al procesador que aparece como mínimo en el minheap, se elimina ese elemento del minheap, se le suma la duración de la tarea que se está procesando y se inserta de nuevo en el minheap. El coste sería: $O(N \log M)$.

3 **2.5 puntos**

Dado un tablero de ajedrez de tamaño $N \times N$ donde cada casilla tiene asignado un valor, haz una traza de un algoritmo Ramificación y Poda que sitúe N torres en dicho tablero de modo que las torres no se amenacen entre sí y que la suma de las casillas ocupadas por las torres sea máxima. (Nota: las torres se mueven libremente en horizontal y vertical.) Haz la traza para este ejemplo con $N = 4$.

4	5	2	1	2
3	4	6	3	2
2	7	3	1	7
1	3	2	4	6
	1	*2*	*3*	*4*

Ten en cuenta:

a) Sigue una estrategia por primero el mejor (en caso de empate, el estado más cercano a una solución) y con poda implícita.
b) Explica brevemente cómo vas a representar: el estado inicial, un estado incompleto y un estado solución. Pon un ejemplo sobre la instancia de cada tipo de estado.
c) Para el ejemplo de solución factible que has dado, calcula su valor de función objetivo.
d) Explica brevemente la cota optimista que vas a utilizar.
e) Para la traza sobre la instancia presentada, ten en cuenta lo siguiente: La traza se debe mostrar el *conjunto de estados activos* de cada iteración del algoritmo indicando la cota optimista de cada estado (ej: como superíndice) y subrayando el estado que se selecciona para la siguiente iteración. Hay que indicar también si se actualiza la variable mejor solución y la poda implícita u otras podas.
f) Si para la traza utilizas el esquema que inicializa la variable mejor solución $\hat{x}$ a una solución factible o a una cota pesimista, describe qué algoritmo o método utilizas para calcularla y cuál es su coste temporal.

Solución: La solución vendrá dada por una configuración válida de las torres en el tablero, que se puede representar como una tupla de 4 elementos (x_1, x_2, x_3, x_4) que indica la posición de la torre de la columna i en la fila x_i. La restricción es que no puede haber dos torres en la misma fila (en la misma columna está implícito por la representación elegida), por lo que, en realidad, las soluciones factibles serán el conjunto de permutaciones de 1 a N. Podemos hacer una inicialización temprana de la variable mejor solución con una solución cualquiera, por ejemplo, la permutación $(1, 2, 3, 4)$, que tiene asociado un valor de $3 + 3 + 3 + 2 = 11$. El coste del cálculo de esta solución y su valor de función objetivo es $O(N)$. Otros ejemplos de soluciones factibles para esta configuración serían: $(1, 3, 2, 4)$ con un valor $3 + 6 + 1 + 2 = 12$ y $(4, 3, 2, 1)$ con un valor $5 + 6 + 1 + 6 = 18$.

Una posible cota superior para este problema vendría dada por el máximo valor de las filas en cada columna y asumir que las torres restantes se colocan ahí, sin tener en cuenta si la fila que tiene esa puntuación ya está ocupada. Estos valores se pueden preprocesar con un coste $O(N^2)$ y tienen un valor para la instancia: $maxv = (7, 6, 4, 7)$. Si el cálculo de la cota se realiza de forma incremental, el coste será constante:

$$F((x_1, x_2, \ldots, x_k, ?)) = F((x_1, x_2, \ldots, x_{k-1}, ?)) + v(k, x_k) - maxv_k.$$

Inicializamos el conjunto de estados activos con el estado que representa todo el espacio de búsqueda (ninguna torre posicionada), con un valor de cota igual a $7 + 6 + 4 + 7 = 24$. En cada iteración seleccionaremos como siguiente a ramificar el más prometedor de acuerdo a su valor de cota. La variable mejor solución es $(1, 2, 3, 4)$ con un valor de 11. Aunque en el algoritmo de ramificación y poda el conjunto de estados activos se representaría con un maxheap, en la traza lo representaremos como un conjunto sin ordenar.

$$A^0 = \{\overbrace{(?)}^{24}\}.$$

Al seleccionar el estado (?) se obtienen 4 nuevos estados:

$$A^1 = \{\overbrace{(1, ?)}^{20}, \overbrace{(2, ?)}^{24}, \overbrace{(3, ?)}^{21}, \overbrace{(4, ?)}^{22}\}.$$

Seleccionamos para ramificar el estado $(2, ?)$:

$$A^2 = \{\overbrace{(2, 1, ?)}^{20}, \overbrace{(2, 3, ?)}^{24}, \overbrace{(2, 4, ?)}^{20}, \overbrace{(1, ?)}^{20}, \overbrace{(3, ?)}^{21}, \overbrace{(4, ?)}^{22}\}.$$

Seleccionamos para ramificar el estado $(2, 3, ?)$:

$$A^3 = \{\overbrace{(2, 3, 1, ?)}^{24}, \overbrace{(2, 3, 4, ?)}^{21}, \overbrace{(2, 1, ?)}^{20}, \overbrace{(2, 4, ?)}^{20}, \overbrace{(1, ?)}^{20}, \overbrace{(3, ?)}^{21}, \overbrace{(4, ?)}^{22}\}.$$

Al ramificar el estado $(2, 3, 1, ?)$ se obtiene la solución $(2, 3, 1, 4)$ que mejora $\hat{x}$ con un valor 19. Seguimos iterando y seleccionamos $(4, ?)$ para ramificar:

$$A^4 = \{\overbrace{(4, 3, ?)}^{22}, \overbrace{(2, 3, 4, ?)}^{21}, \overbrace{(2, 1, ?)}^{20}, \overbrace{(2, 4, ?)}^{20}, \overbrace{(1, ?)}^{20}, \overbrace{(3, ?)}^{21}\}.$$

Los estados $(4, 1, ?)^{18}$ y $(4, 2, ?)^{19}$ no se guardan. En esta iteración, el estado ramificado es el $(4, 3, ?)$:

$$A^5 = \{\overbrace{(4, 3, 1, ?)}^{22}, \overbrace{(4, 3, 2, ?)}^{20}, \overbrace{(2, 3, 4, ?)}^{21}, \overbrace{(2, 1, ?)}^{21}, \overbrace{(2, 4, ?)}^{20}, \overbrace{(1, ?)}^{20}, \overbrace{(3, ?)}^{21}\}.$$

Al ramificar $(4, 3, 1, ?)$ se obtiene una nueva y mejor solución de coste 22: $\hat{x} = (4, 3, 1, 2)$. En la siguiente iteración, el conjunto de estados activos es:

$$A^6 = \{\overbrace{(4, 3, 2, ?)}^{20}, \overbrace{(2, 3, 4, ?)}^{21}, \overbrace{(2, 1, ?)}^{20}, \overbrace{(2, 4, ?)}^{20}, \overbrace{(1, ?)}^{20}, \overbrace{(3, ?)}^{21}\}.$$

El mejor estado es $(2, 3, 4, ?)$ con una cota de 21, peor que la mejor solución solución obtenida hasta el momento. Esto significa que ninguno de los estados activos puede mejorar $\hat{x}$, con lo que se vacía A y el algoritmo termina devolviendo la solución $\hat{x} = (4, 3, 1, 2)$.

Podemos inicializar la variable mejor solución con una solución voraz más elaborada: elegir el máximo para la torre de la primera columna; el máximo para la torre de la segunda columna entre las casillas libre; y así sucesivamente se llegaría a la solución (2,3,1,4) con un valor asociado de 7+6+4+2=19. Este algoritmo tiene un coste $O(N^2)$. La traza no se vería modificada de forma substancial.

4 — 2.5 puntos

Disponemos de N objetos con pesos diferentes $w_1, w_2, ..., w_N$ que deseamos dividir en dos conjuntos de modo que ambos pesen exactamente lo mismo. Por ejemplo, si los pesos de los objetos son $\{7, 12, 1, 2, 8\}$, habría que repartirlos en dos conjuntos de forma que cada uno sume $(7 + 12 + 1 + 2 + 8)/2 = 15$. En este caso sí es posible: $\{7, 8\}$ y $\{12, 1, 2\}$.

Se pide que implementes un algoritmo que, utilizando la ***técnica de Búsqueda con Retroceso***, devuelva la solución encontrada, si es que existe.

1. Explica brevemente la estrategia a seguir.
2. Escribe un algoritmo (en seudo-código o python) que implemente la estrategia anterior.

Solución: El problema es análogo a la suma del subconjunto con el valor igual a la mitad de la suma de los pesos de todos los objetos.