Problemas de Diseño y Análisis de Algoritmos

Problema 1: Tareas y Expertos

Equipo: 3

Miembros:
- Laura Brito Guerrero
- Sheyla Cruz Castro
- Rodrigo García Gómez

Enunciado del Problema:

Imagine una cantidad n de tareas, cada una de ellas con determinada complejidad *ci*, que deben ser realizadas. Para hacer esto se cuenta con un grupo *m* de expertos, caracterizados por sus respectivas capacidades *di*. Se sabe que el experto i-ésimo puede ejecutar la tarea j-ésima si $di >= cj$.Cada experto consume un día en hacer una tarea y ha de recibir por su trabajo una cantidad *ki* de pesos, que siempre es la misma independientemente del número de tareas que resuelva. El objetivo en este problema es determinar el menor tiempo posible en que se pueden resolver las *n* tareas sin pagar más de *k* pesos. Se asume que las tareas se pueden hacer en cualquier orden y que distintos expertos pueden trabajar al mismo tiempo en sus respectivas tareas.

Abstract

Primeramente lo atacamos con un algoritmo *fuerza bruta* exponencial que analiza todas las combinaciones posibles que resultan de distribuir los distintos expertos a tareas, y da la mejor solución posible. Este algoritmo es ineficiente, por lo cual lo intentamos por otras vías. Una de ellas consistía en hacer una búsqueda binaria sobre un predicado de la forma $no, no, no...si, si, si...$ buscando el primer si que responda a la menor cantidad de días que se requieren para realizar las *n* tareas, pero resultó ser un fracaso.Otra solución consiste en combinar una *mochila* para obtener subconjuntos de elementos, con un algoritmo cuadrático para determinar la mejor ganancia; este resulta $n^3 * k$, también de costo exponencial ya que depende del valor de k. Una cuarta idea sería implementar un algoritmo para, dado un grupo de científicos fijo válido y las tareas a completar por estos, encontrar la menor cantidad de días en que pueden lograrlo, de forma polinomial; esta es costosa, pues para determinar dicho grupo tengo que generar todas las combinaciones.

Aclaración:

Para cada uno de los tres problemas se implementó un programa *testergenerator* que se encarga de generar de forma aleatoria una inmensa cantidad de ficheros con casos de prueba y un *tester* con el cual se implementan estos casos con dos o más métodos distintos. De esta forma es posible comprobar con una altísima seguridad la correctitud de un algoritmo a partir de la comparación de sus resultados con los resultados de otro algoritmo enfocado en la *fuerza bruta* con una complejidad temporal muy superior, pero cuya correctitud es más evidente. Además, de esta forma es fácil comprobar los casos específicos en los que determinado algoritmo falla, ayudando así la búsqueda y arreglo de errores. Estas demostraciones no producen las demostraciones formales necesarias para la correcta resolución de los ejercicios, pero son muy útiles para dirigir correctamente los esfuerzos, ya sea para demostrar uno en el que se tenga una fuerte convicción de correctitud, o para comprobar los motivos por el que otro es incorrecto.

Idea Fuerza Bruta:

(brute1)

La idea consiste en generar todas las posibles asignaciones de expertos a tareas. Serían todas las formas de tomar n tareas del conjunto de tareas(como es condición necesaria que todas las tareas se realicen, se toman todas estas) donde cada una puede tomar m formas distintas,o sea pueden ser realizadas por alguno de los m expertos. De aquí nos quedamos con la combinación que consuma la menor cantidad de días entre todas las posibles distribuciones.

Inicialmente se esperan tres líneas: la primera con la cantidad de tareas a realizar,la cantidad de expertos y el presupuesto del que se dispone, $n, m,$ y k respectivamente; luego una segunda con las capacidades de los expertos; y finalmente las complejidades de cada una de las tareas. Una vez que contamos con todos los datos creamos un array con los números en el intervalo de 0 a n-1,o sea de tamaño n,sobre el cual trabajaremos. Por consiguiente generamos todas las posibles distribuciones a través del método *VariacionesConRepeticiones*.

Toda tarea tendrá asignada un experto y este será único pues una tarea solo puede ser realizada una vez. La distribución admite elementos repetidos, o sea un experto puede realizar más de una tarea.Una vez que se genera una posible distribución se analiza su factibilidad,de esto se encarga el método *Analiza*. Para ello se comprueba que el experto i-ésimo asignado a la tarea j-ésima tenga capacidad mayor o igual que la complejidad de dicha tarea; y se comprueba además que la suma del dinero que recibe cada experto

es menor o igual que el presupuesto con que se cuenta. Una distribución bajo estas restricciones se considera válida. De todas ellas nos quedamos con la que requiera menor cantidad de días. La cantidad máxima de días que requiere cada disposición es igual al máximo de la cantidad de tareas que realiza cada experto, pues este requiere de todo un día para la realización de la tarea y estos pueden trabajar simultáneamente, luego el que más tareas tenga requerirá más días, y dicha cantidad será cota máxima para el tiempo total requerido. Por último nos quedamos con el mínimo de los máximos(duraciones máximas de las distintas posibilidades) pues lo que se quiere es minimizar el tiempo de realización de las n tareas.

La correctitud de esta solución se sustenta en que se analizan todas las posibilidades,garanti así que de existir una solución se va a encontrar y se queda en todo momento con la más óptima. Este algoritmo es exponencial,genera m^n distintas posibilidades, donde m es el número de expertos y n el número de tareas a realizar. Esta solución es muy ineficiente, se demora considerablemente para pequeños valores de la entrada.

Idea 2:

Intentamos hacer una búsqueda binaria sobre un predicado de la forma $no, no, no...si, si, si..$ buscando el primer sí que
responda a la menor cantidad de días que se requieren para realizar las n tareas. Asumimos que el predicado tendría esta forma pues supusimos que si se pueden realizar las n tareas en una cantidad i de días, con $i <= n$, en una cantidad mayor también se podría hacer, y como lo que se quiere es minimizar la cantidad de días lo que se busca es el primer *si*. Esto se puede hacer en $\log n$, el problema consiste en encontrar el experto capaz de realizar una cantidad de tareas i que coincide con la cantidad d días en cuestión, esta misma cantidad sería igual al máximo de tareas que se le puede asignar a un experto de modo que se puedan repartir el resto de las tareas entre los expertos que quedan, este experto ya no estaría disponible pues si a él se le pudiesen asignar más tareas aumentaría la cantidad de días. Se ordenan las tareas decrecientemente por complejidad, al igual que los costos. Se recorren intentando asignarle al experto de mayor capacidad el subconjunto de tamaño igual a la cantidad de días en cuestión o sea esa cantidad x de tareas, y estas se toman por orden de complejidad. Si es posible realizar esta asignación, porque tanto el presupuesto como las capacidades lo permiten, y se puede además repartir el resto de las tareas entre los expertos que quedan, esta es una asignación correcta o sea un *si* de mi predicado. De no ser posible asignarles esas tareas al experto de mayor capacidad

fue o bien porque dicha capacidad no fue suficiente, con lo cual esa combinación no será válida, pues nadie en adelante será capaz de realizarlas; o porque el presupuesto no fue suficiente, en dicho caso se prueba con el segundo subconjunto de mayor complejidad y se repite el proceso.

Esta idea no funciona pues el predicado no es necesariamente de la forma $no, no, no...si, si...$ pues por ejemplo, se puede encontrar un caso en el que se pueda hacer con una cantidad i de días y no necesariamente con una cantidad mayor (por ejemplo cuando la única posibilidad de hacerlo con una cantidad mayor de días se vea obligada a asignarle una tarea de más complejidad a un experto cuya capacidad sea menor). Si el predicado respondiese en cambio a una cantidad menor o igual que la cantidad i-ésima, si se podría asegurar que si es solución para i lo es para todo $k >= i$, pero no tenemos forma de comprobar que la parte menor$(<)$de la desigualdad se cumple sin calcularla previamente.Por lo que dicha solución no es válida.

Idea 3:

(calculate-days)

Se implementó un algoritmo para, dado un grupo de científicos fijo válido y las tareas a completar por estos, encontrar la menor cantidad de días en que pueden lograrlo, de forma polinomial. Se tiene n = la cantidad de tareas y m = la cantidad de científicos. Primeramente, es necesario ordenar de forma creciente a los científicos por sus capacidades y las tareas por sus dificultades.

La idea consiste en separar a los científicos por grupos en dependencia de la cantidad de tareas que sean capaces de cumplir. Se mantiene un puntero para científicos (j) y otro para tareas (I). Se avanza el puntero i por las tareas hasta que el científico j no sea capaz de cumplir la tarea i, entonces se agrega al nuevo grupo al científico j y a todos los científicos siguientes que tampoco sean capaces de cumplir dicha tarea, avanzando el puntero j. Los grupos llevarán constancia de la cantidad de tareas que pueden cumplir y la cantidad de científicos que poseen. La conformación de los grupos se realiza en tiempo lineal $(m+n)$ pues es evidente que se pasa sólo una vez por cada tarea y una por cada científicos.

Todas las tareas realizadas por un grupo se podrán realizar también con los grupos superiores, por tanto se buscará repartir las tareas de forma que si el grupo i realiza sus tareas en un tiempo superior al grupo $i+1$, este aumente sus días en uno para ayudar al

grupo i. Este proceso se repetirá cada vez que el grupo i tenga mayor tiempo que el $i+1$. Al realizar esto para todos los grupos, cuando el algoritmo termine, entonces en el último grupo tendrá la mayor cantidad de días entre todos los demás, y por tanto, como todos los demás grupos finalizarán sus tareas en un tiempo menor o igual que él, esta será la cantidad de días a retornar. Este retorno tendrá la cantidad mínima de días de trabajo, puesto que los días de los grupos superiores aumentarán un día a la vez, mientras que los de grupos inferiores disminuirán en uno o más días, por tanto, si el grupo i se completa en más días que el $i+1$, la cantidad de días que se le aumentan a este, no será mayor que la que tenía el i antes de disminuirle.

A partir de este momento, solo se trabajará con la lista de grupos de científicos recién formada. El ciclo principal iterará por dicha lista de forma descendiente, a partir de la última posición (recordemos que esta lista está ordenada). Primeramente, en cada iteración (siendo la iteración i, que se está analizando el grupo $n-i$), si la cantidad de tareas es mayor que la cantidad de científicos, se divide tareas entre científicos y se guarda en una variable q (esto representa la cantidad de días necesaria para que el grupo termine sus tareas), en caso de haber resto se incrementa q en uno y en una variable r se almacena la diferencia entre los científicos del grupo y dicho resto (esto representa el sobrante de tareas del grupo para realizar las suyas en la cantidad de días q, o sea, si el grupo tiene 2 científicos y 5 tareas, en 3 días realiza sus tareas, y tiene uno de resto para almacenar). Además, si el grupo bajo análisis actual realiza sus tareas en una cantidad de días (q) mayor que el grupo inmediatamente superior, le pedirá a este una ayuda, incrementando en el grupo superior una variable que almacena las ayudas a prestar al grupo inferior; luego es necesario recalcular q y r del grupo actual pero retirando una cantidad de tareas similar a la cantidad de ayudas de su grupo superior multiplicada por la cantidad de científicos de este, de lo cual se encarga el ciclo principal, que se repite siempre que ocurra una *pedida de ayuda*. Si la cantidad de tareas es menor que la cantidad de científicos, el proceso es similar, con la diferencia de que en lugar de hacer un división, en q se almacenará 1, y en r, la substracción de los científicos por las tareas (si hay 5 científicos y 3 tareas, se cumplen en un día y se guarda 2 de resto para quitarle tareas al grupo inmediatamente menor) . Al terminar el ciclo principal, la solución quedará representada en la cantidad de días q más la cantidad de ayudas prestadas por el grupo de la última posición de la lista (el de las mayores capacidades).

Idea 4:

(knapsack)

Otra solución un poco más eficiente para este problema es combinarlo con una *mochila* y luego aplicarle un algoritmo cuadrático que garantice obtener la mejor ganancia, sobre los elementos que se encuentran escogidos en la *mochila*, para así garantizar obtener la mayor ganancia. Esta idea se lleva a cabo de la siguiente manera: inicialmente tenemos una mochila de peso k, el cual es el presupuesto destinado para la realización de las tareas. Como cada experto tiene una ganancia k_i
independientemente de la cantidad de proyectos que realice, esta misma representaría los pesos correspondientes a cada uno.

Como este algoritmo me garantiza que en cada momento exista una repartición válida de expertos, tenemos que ahora distribuir las tareas entre ellos tal que se cumpla que todas las tareas se tomen y de esta forma hallar la mejor combinación tal que el máximo de las tareas entre los expertos sea mínima. Esta combinación asignada resultará la ganancia de esta repartición, y en cada elección de expertos ir guardando este máximo de tareas con la intención de que la ganancia se minimice. Por lo que variaremos la salida del algoritmo *mochila* estándar, puesto que en vez de buscar la ganancia máxima (la cantidad de días máximos) se hallará la mínima.

Para hallar la combinación de las tareas por los expertos de manera eficiente se realizá el siguiente algoritmo: se tiene un arreglo de n tareas con sus respectivas complejidades, y se tienen m_k expertos escogidos. Las tareas estarán ordenadas decrecientemente por complejidad, y se ordenarán por capacidad en orden decreciente a los expertos escogidos. Se recorrerán las tareas en ese orden intentando otorgárselas a los distintos expertos, de manera tal que todos los expertos tengan tareas, y siempre se tratará de otorgar la menor cantidad de tareas por experto. Esto se realiza en dos ciclos anidados siempre analizando las tareas a entregar.