Enunciado 3, Problema de Fuerza Bruta

Análisis e Implementación

Dany Efraín Rubiano Jiménez
Departamento de Ingeniería Informática
Universidad de Santiago de Chile
Santiago de Chile, Chile
dany.rubiano@usach.cl

***Abstract*— En el presente documento se analiza la aplicación del método de fuerza bruta, con el fin de resolver una problemática dada aplicando su definición y explicando cómo se contextualiza en el desarrollo de esta.**

Index Terms—Fuerza Bruta, Algoritmo, C.

I. Introducción

A pesar de que existen incontables algoritmos, se han logrado identificar un número de técnicas de diseño suficiente mente generales como para poder aplicarse a un gran número de problemas. Estas técnicas dan como resultado algoritmos eficientes (a veces...) y que son de muy diversos dominios de a plicación.

Para esta ocasión se aborda un problema dado, para el cual se aplicará la técnica de resolución de la fuerza bruta, buscando encontrar la solución óptima a partir de la generación de todas las posibilidades de solución que se puedan dar.

II. Descripción del Problema

El problema dado al cual se aplicará la técnica de resolución de la fuerza bruta se contextualiza en un operativo de ayuda en una zona de catástrofe, donde se tiene un grupo de personas, las cuales tienen distintas habilidades y un tiempo asociado al desarrollo de cada tarea cubierta por una habilidad. Se pide asignar las tareas a las distintas personas con el objetivo de terminar en el menor tiempo posible el trabajo de ayuda.

Para ello se entrega la lista de tareas y de tiempos asociados para completar cada tarea por cada persona, un ejemplo de ello, se encuentra en la Tabla 1. Esta Entrada está contenida en un archivo de nombre “entrada.in” que contiene una matriz que representa la asociación entre el número de tarea y la persona que lo puede cubrir. El valor de esta relación es el tiempo. Las filas indican las personas y las columnas las tareas.

TABLE I. Tabla Complejidad Algorítmica

Personas	**Tareas**				
	T_1	T_2	T_3	T_4	T_5
P_0	2	1	1	1	0
P_0	0	0	1	1	0
P_0	0	1	1	3	1
P_0	0	1	0	1	1
P_0	0	3	0	1	1

Para el resultado, se debe generar un archivo de nombre “salida.out” donde se escriba un arreglo del mismo tamaño que la cantidad de tareas en la entrada, la cual mostrará la persona asignada. Esta sería la representación de la solución óptima de la asignación de las tareas en función del tiempo y habilidades de cada persona. Un ejemplo para la salida se representa en la Tabla II.

TABLE II. Tabla Complejidad Algorítmica

Tareas				
T_1	T_2	T_3	T_4	T_5
1	5	2	3	4

III. Explicación de la Solución

En el problema dado, se requiere encontrar la asignación óptima de tareas, es decir, encontrar la combinación que obtenga el mayor valor en la relación persona/tarea, para esto debemos buscar en la matriz de entrada los mayores valores para cada tarea y ordenarlos de manera que no se repitan las personas asignadas a una tarea y la combinación entregue el mayor valor posible.

La técnica requerida en la resolución del problema se enmarca en la fuerza bruta, por lo que siguiendo el método que sigue esta técnica, se deben generar todas las combinaciones posibles de solución. Como a cada persona se le asigna una única tarea, entonces para generar todas las combinaciones posibles se usa la permutación, la noción de permutación

menciona a los posibles ordenamientos de aquellos elementos que forman parte de un conjunto no infinito. Esto quiere decir que una permutación es un cambio de la manera en la que se disponen los elementos. Puede considerarse como una función de tipo biyectiva dentro del conjunto, ya que señala distintas correspondencias entre los elementos. Para el número de elementos "n", la cantidad de combinaciones posibles a través de la permutación es correspondiente al factorial del número de elementos.

De esta manera, se obtienen todas las soluciones posibles para la asignación de las tareas, después basta con buscar la solución que dé el mejor resultado, teniendo en cuenta las condiciones que impone el problema.

IV. Método Utilizado

Para la resolución del problema, tal como se mencionó anteriormente se utiliza la técnica de la fuerza bruta, generando todas las combinaciones posibles a través de la permutación de los elementos, en este caso los elementos serán representados por los trabajadores.

En la literatura existen multitud de algoritmos para realizar permutaciones. Un buen resumen se puede encontrar en *Permutations Generation Methods de R. Sedgewick, ACM Computing Surveys, Vol. 9, No. 2, Junio 1977*. A partir de este, se genera un algoritmo que genera todas las combinaciones a través de la permutación.

Con el algoritmo de permutaciones de realizado, es posible generar fácilmente las combinaciones de los trabajadores asignadas para todas las tareas que se tengan en el problema y, a medida que son generadas, evaluarlas según el valor que corresponda en la matriz que relaciona la tarea y la persona para ser comparadas con el valor máximo parcial alcanzado hasta el momento, guardando un registro de la combinación que generó el valor máximo.

Finalmente se entrega en la salida la combinación que generó un mayor valor terminando el algoritmo.

V. Descripción de la Solución

Para dar paso a establecer la solución del problema, primero se debe pasar la lista de entrada a una estructura que almacene todos los datos del archivo dado, y así poder operarlos. Esta estructura corresponde a una matriz cuadrada, puesto que a cada trabajador se le asignará una única tarea. Para ahorrar memoria, se establece una función que cuenta el número de líneas que tiene el archivo y con este establecer el tamaño de la matriz.

Una vez se tiene el número de líneas que tiene el archivo se procede a crear la matriz que guardarán los datos contenidos en el archivo.

Ahora se da paso para capturar estos datos y pasarlos a la matriz. Para esto, se procede a crear una función que guarde en las filas y columnas los datos, según su separación, cuando los datos están separados por espacios, los elementos se guardan en la misma fila en las distintas columnas, y cuando hay un salto de línea, se pasa a la siguiente fila.

Ahora es el turno de generar todas las combinaciones posibles, y tal como se mencionó anteriormente estas se harán con el uso de permutaciones, de esta manera, se puede distribuir a los trabajadores en las tareas con todas sus posibilidades para luego encontrar la solución óptima. Estas combinaciones se guardan en una matriz, la cual posee un tamaño de filas igual al número de posibilidades, dada por el factorial del número de elementos a permutar, y el tamaño de las columnas será el número de tareas, más un espacio para establecer una suma de las relaciones para cada combinación de los trabajadores.

Entonces, se calcula el factorial del número de trabajadores, para establecer las variables con las dimensiones adecuadas en la resolución del problema. Este cálculo se determina con una función recursiva, la cual recibe como parámetro el número de elementos, y en cada llamada este número disminuye en uno, así hasta cumplir el caso base y obtener el resultado final.

Una vez calculado el factorial, se procede a crear la matriz que guardará todas las combinaciones resultantes.

Ahora se procede a utilizar el algoritmo de permutación sacado del libro mencionado anteriormente. Se utiliza el algoritmo en una función que recibe como parámetros el número de combinaciones posibles dado por el factorial, el número de elementos a permutar y la matriz donde se guardarán todas estas combinaciones. Esta función a su vez llama a otras funciones para su proceso, entre estas funciones se encuentra una función que realiza el intercambio entre dos elementos, otra que realiza un proceso de reverso a los elementos, y una última que se encarga de escribir los elementos en la matriz susodicha.

La función que hace la permutación, está implementada suponiendo que el primer elemento está en la posición 1 del arreglo, en vez de la posición 0, como es habitual en C. De este modo, al definir el arreglo de los elementos a permutar que se le va suministrar a la función, hay que tener en cuenta que el elemento en la posición 0 no se utiliza en las permutaciones.

Para establecer la solución óptima, se utiliza una función para sumar los tiempos que cada uno de los trabajadores ocupa para cada tarea en cada combinación, para ello, se recorre la matriz de las combinaciones recuperando los datos de cada posición, los cuales corresponderían a los trabajadores, según estos, se recuperan los datos asociados al trabajador en la matriz de entrada para así sumar los tiempos de cada combinación en la relación de tareas y trabajador. Esta suma se guarda en la última columna de la matriz de combinaciones y se genera para cada una de ellas.

Con los resultados de las sumas de cada combinación, se debe obtener la solución óptima. Para ello, se establece un procedimiento, la cual busca la mayor de las sumas. Cuando se encuentra el mayor, se guarda la posición en que está este, para luego poder generar el archivo de salida en que se muestre la

mejor combinación de asignación de cada trabajador a cada tarea en función del tiempo y las habilidades.

Luego en el main se escribe el archivo de salida recorriendo los elementos de la matriz de combinaciones según la posición donde está el mejor caso y que refleja la solución óptima.

De esta manera, mediante el uso de la fuerza bruta, se puede llegar a tener la solución ´optima al problema dado.

VI. Complejidad Algorítmica

La complejidad algorítmica representa la cantidad de recursos (temporales) que necesita un algoritmo para resolver un problema y por tanto permite determinar la eficiencia de dicho algoritmo. En esta ocasión, se representa el orden de cada una de las funciones utilizadas, calculado por el número de ciclos. El cálculo del orden para cada una de las funciones establecidas en la resolución del problema se encuentran en la Tabla III.

TABLE III. Tabla Complejidad Algorítmica

Función	Orden
contarLineasDeCodigo( )	O (n)
factorial(int n)	O (n^2)
lecturaArchivo(int n, int matriz[n][n])	O (n^2)
process(int* P, int m, int matriz[ ][m+1], int l)	O (n)
swap(int *x, int *y)	O (n)
reverse (int *P, int m)	O (n)
permuta (int *P, int m, int matriz[ ][m+1])	O (n^2)
obtenerTiempos (int m, int f, int A[m][m], int B[f][m+1])	O (n^2)

VII. Traza

En la Tabla IV, se muestra una traza simple recortada en base a la entrada propuesta en la Tabla I.

TABLE IV. Tabla Traza Simple

Función	Resultado - Traza
contarLineasDeCodigo( )	5
factorial(int n)	120
lecturaArchivo(int n, int matriz[n][n])	Elementos Leidos: 2 1 1 1 0 0 0 1 1 0 0 1 1 3 1 0 1 0 1 1 0 3 0 1 1
permuta (int *P, int m, int matriz[ ][m+1])	Combinaciones dadas: 5 4 3 2 1 5 4 3 1 2 5 4 2 3 1 5 4 2 1 3 5 4 1 3 2 5 4 1 2 3 5 3 4 2 1 5 3 4 1 2 5 3 2 4 1 5 3 2 1 4 5 3 1 4 2 5 3 1 2 4 5 2 4 3 1 5 2 4 1 3 5 2 3 4 1 5 2 3 1 4 5 1 4 2 3 . . . Así para las 120 combinaciones
obtenerTiempos (int m, int f, int A[m][m], int B[f][m+1])	Suma de la relación trabajador tiempo de cada combinatoria: 3 3 5 4 5 4 2 2 3 4 3 4 3 5 . . . Así para las 120 combinaciones mejor = 10 posicionMejor = 101
main( )	15234

VIII. Eficiencia

La eficiencia de cada una de las funciones empleadas en la resolución del problema, es de alguna manera aceptable, ya que en su solución entregan el resultado esperado.

Si hablamos de cada una de ellas, en particular el factorial, al ser esta una función recursiva, hay que tener un cuidado especial, puesto que entre más grande el número a calcular se puede llegar a utilizar grandes cantidades de memoria en un instante, pues implementa una pila cuyo tamaño crece linealmente con el número de recursiones necesarias en el algoritmo. Si los datos en cada paso son muy grandes, podemos requerir grandes cantidades de memoria.

Refiriéndose a la función de permutación, también se debe tener un sumo cuidado, pues generar todas las permutaciones para un número muy grande, requiere de un tiempo de

ejecución muy grande que puede ser expresando en años. En la Tabla V, se muestra el tiempo de computación en función del tamaño n de conjunto. (1 microsegundo por operación)

TABLE V. TABLA TIEMPO DE EJECUCIÓN DE UNA PERMUTACIÓN SEGÚN EL NÚMERO DE ELEMENTOS

N	N!	Tiempo
1	1	
2	2	
3	6	
4	24	
5	120	
6	720	
7	5040	
8	40320	
9	362880	
10	3628800	3 segundos
11	39916800	40 segundos
12	479001600	8 minutos
13	6227020800	2 horas
14	87178291200	1 dia
15	1307674368000	2 semanas
16	20922789888000	8 meses
17	355689428096000	10 años

IX. CONCLUSIONES

Como conclusiones se puede afirmar que se logró de alguna manera llegar a los resultados correctos, si lo probamos con los ejemplos que están en el enunciado del laboratorio.

Las estructuras de datos utilizadas se usaron en base a los algoritmos planteados y cumplen la función para lo que fueron declaradas.

Las técnicas de desarrollo de algoritmos nos permiten encontrar la mejor solución a los problemas que se nos presentan y deben ser solucionados por el computador, estas técnicas están orientadas para utilizarse en cada uno de los niveles de complejidad y variedad o alternativas para las cuales se aplican los algoritmos, en el caso de la fuerza bruta, para la mayoría de problemas reales, el número de soluciones candidatas es prohibitivamente elevado, por lo cual es ineficiente ante casos muy grandes. El tiempo de ejecución y la eficiencia de la solución con este método, depende del tamaño del problema.

REFERENCIAS

[1] *R. Sedgewick, "Permutations Generation Methods", ACM Computing Surveys, Vol. 9, No. 2, Junio 1977*