

Universidad de Santiago de Chile

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA EN INFORMÁTICA

TAREA 2: BRANCH AND BOUND Y "JOBS ASSIGNMENT"

TALLER DE PROGRAMACIÓN

Sección: 13315-0-A-1

Profesor: Pablo Roman Asenjo

Ayudante: Ricardo Hasbun

Autor: John Serrano Carrasco

Mayo 2022

Contents

1	El problema	2
2	La solución	2
2.1	Pruebas y errores	5
3	Complejidad y eficiencia	5
4	Conclusiones	5
5	ANEXOS	6

1 El problema

Se dispone de N procesos y N CPUs. Con esto en mente, se forma una matriz cuadrática de dimension M, donde cada posicion de la matriz corresponda a un costo asociado a resolver el proceso P utilizando la CPU C. Se busca utilizar el algoritmo "Branch and Bound" para encontrar cual es la combinación mas optima tal que el costo obtenido sea el menor. Para ello, se pide encontrar la matriz binaria tal un 1 represente aquella asignacion seleccionada para ese conjunto de proceso y CPU. A continuación se muestra mediante imaganes lo comentado hasta el momento:

Por ejemplo, se tiene la siguiente matriz de costo:

		Procesos					
		7	5	3	2	4	8
		8	5	1	3	2	4
CPUs		5	1	6	8	1	4
		1	7	5	2	4	8
		1	3	7	5	4	2
		7	5	1	8	2	3

Figura 1: Matriz de costo de 6 x 6.

La matriz binaria de la matriz de costo anterior, es:

0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	0	0	0
1	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	1	0

Figura 2: Matriz de binaria asociada a la matriz de costo anterior.

2 La solución

Para resolver este problema utilizando Branch Bound, necesitamos cinco clases importantes:

- Clase Nodo
- Clase Heap
- Clase Cost
- Clase Container
- Clase Assignment

La solución es la siguiente: Primero, se debe generar la matriz de costo. Por ordenes de la pauta, tenemos que la dimensión de la matriz de costo es de 500 x 500 (esto puede ser modificado como se

desea). De esta forma, tendremos una matriz de costo generada al utilizar el constructor de la clase Assignment.

Luego, viene el metodo solve(). Primero se parte creando dos Heap, que será de tamaño N. Una de ellas será utilizada para recorrer todos los nodos correspondientes a una fila y la segunda será utilizada para guardar solo el nodo "importante" de la fila (Es decir, el nodo que tiene el menor costo).

Para este problema, un nodo se compone de los siguientes atributos: Una CPU asociada (representado por un entero), un procesador asociado (representado por un entero), un "costo total" el cual corresponde el costo que tiene desde el nodo raiz hasta el nodo actual, un "costo principal" el cual corresponde a la suma del costo total con el costo estimado hacia abajo (Lo anterior se calcula utilizando el algoritmo voraz), un arreglo de booleanos, el cual marca con verdadero aquella columna que ya está utilizada en alguna fila (Recordemos que una CPU solo se puede encargar de un procesador y que un procesador solo puede ser manejada por una CPU) y finalmente, hay una referencia al nodo padre del nodo.

Volviendo a la solución se crea un nuevo nodo, al cual llamaremos "root" (raíz), al cual se le asignará la posición (-1,-1) y esto nos permite realizar la búsqueda del nodo mínimo en toda la primera fila. Para ello, el nodo se inserta en la Heap "nodesHeap" y se ingresa a un ciclo while. Mientras no esté vacía la heap, vamos a sacar el nodo clasificado como "mínimo" (La heap automáticamente nos hace el trabajo de dejar al nodo de menor costo en el tope de la Heap). Como solo hay un nodo, se va a clasificar a este como el mínimo. Teniendo esto en consideración, se saca la componente i del nodo mínimo (CPU) y se le suma 1, esto para poder recorrer la fila que le sigue al nodo mínimo. Se tiene una verificación, ya que si CPU + 1 resulta ser igual a N, eso significa que ya se recorrieron todas las filas y por lo tanto, se llegó a una solución. Como aún no sucede eso, se van a recorrer todos los procesadores para esa CPU específica. Se verifica que no hayan procesadores ya asignados (El arreglo de booleanos del nodo mínimo nos ayuda con esto, ya que si hay un procesador ya asignado, entonces el arreglo en esa posición del procesador será "true") y se procede a crear un nuevo nodo, donde se toman los datos correspondiente a la posición del procesador, y se calcula el costo total y principal. Luego se coloca en la heap de "nodesHeap" y se repite el proceso hasta que se hayan recorrido todos los procesadores disponibles para esa CPU.

Una vez se haya completado esto, se realizará un pull, donde el nodo que saldrá será automáticamente el nodo de menor costo principal. Como tenemos muchos nodos que no nos sirven, utilizamos la otra Heap, donde temporalmente insertamos el nodo mínimo, ya que lo necesitamos guardado para la próxima iteración del while. Se vuelve a crear la Heap de nodesHeap y se vuelve a agregar el nodo mínimo. Luego el ciclo while se repite, sacando nuevamente el nodo mínimo y utilizando para calcular los siguientes nodos de la CPU que le procede. El proceso continua hasta que el número de la CPU coincide con N, o bien, la Heap quede vacía, en cuyo caso significa que no se pudo encontrar una solución óptima.

Si sucede lo primero, entonces se van a imprimir N mensajes donde cada mensaje corresponde a qué CPU se le asignó que procesador. Posterior a eso, se imprime la matriz binaria y el costo asociado a la solución.

En el main del programa, se hace un ciclo for con N repeticiones de distintas matrices de costo para poder calcular el tiempo de resolución de las matrices y el tiempo promedio. Se recomienda probar con 25 iteraciones ya que puede que la memoria colapse dependiendo de dónde se esté probando el código.

El proceso de creación de la solución fue basada en lo enseñado en clases, como también en algunos tutoriales encontrados en la Web e información de páginas como GeeksforGeeks. Una mención honrosa es a la página de GeeksForGeeks titulada "Job Assignment Problem using Branch And Bound video", donde se tiene mucha información tanto para comprender el problema, como para saber cómo llevar a cabo una solución del problema utilizando Branch Bound.

A continuación, se tienen algunas imágenes que pueden complementar lo dicho anteriormente.

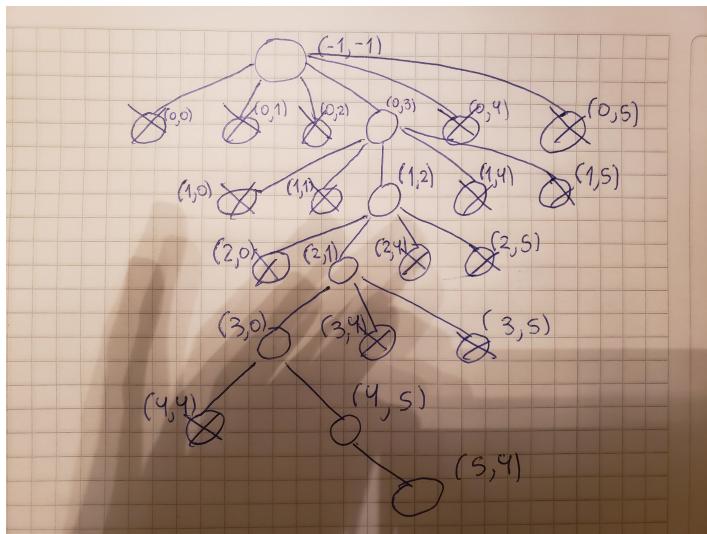


Figura 3: Boceto de la solución con Branch and Bound

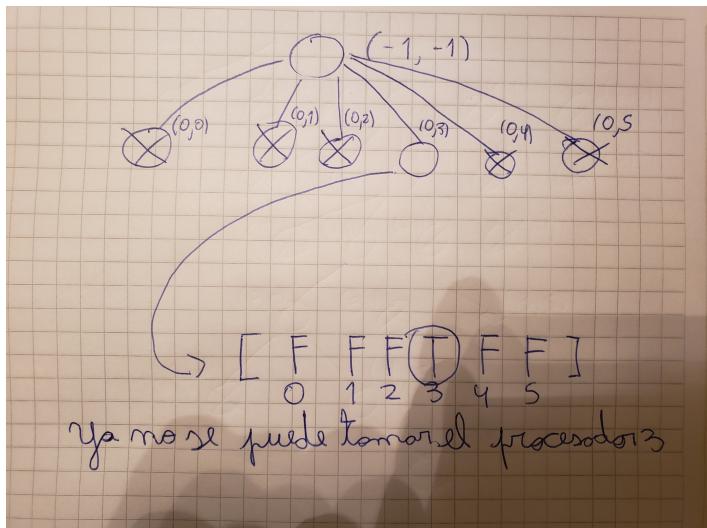


Figura 4: Boceto de AssignedArray

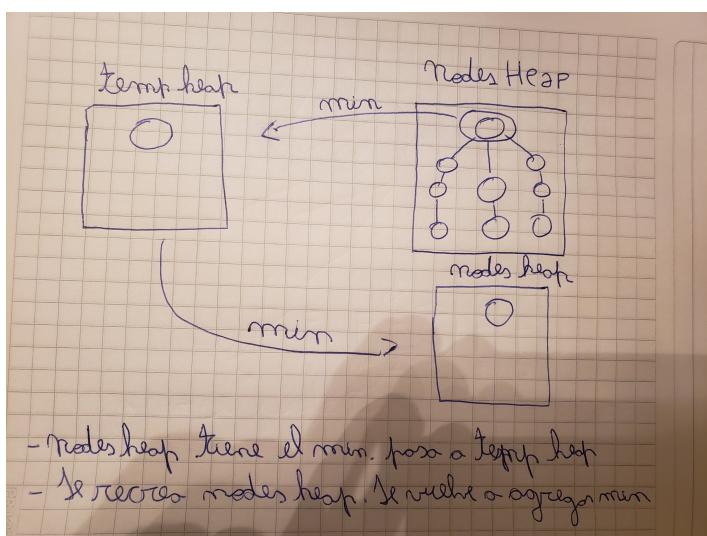


Figura 5: Boceto de uso de multiples heap para mantener el "nodo importante"

2.1 Pruebas y errores

Hubieron multiples pruebas y multiples errores durante el proceso de creación. Una de ellas fue mal interpretar el problema, ya que al principio se pensó que bastaba solo con sacar el minimo de cada fila y luego sumar el costo total. Sin embargo, tras hacer algunas pruebas, es claro de que el costo total no necesariamente se obtiene sumando los menores de las primeras filas, especialmente considerando el bloqueo de las columnas, lo que puede causar que en las filas mas abajo se tomen valores mayores a los que se podrian tomar. Otro problema fue el uso reiterado de destructores y de Heaps, lo cual causaba una demora de tiempo impresionante, a tal punto de que una de las ejecuciones de pruebas llegó a durar casi 7 minutos. Sin embargo, se logró solucionar el problema simplemente reubicando la generación de Heaps y el uso de destructores, lo cual redujo considerablemente el tiempo.

Hubieron problemas de memoria, problemas de implementación (mal uso de los nodos y heaps) junto con otros aspectos los cuales plantearon un gran desafio a la hora de llegar a una solución que funcionará de manera decente. Sin embargo, optimizando el uso de Heaps junto con su tamaño, se logró disminuir el tiempo a un intervalo de 70 - 90 segundos. Si se utilizan las FLAGS con -O2 para compilar, el tiempo disminuye a un intervalo de 25 - 40 segundos, esto teniendo en consideración una matriz de 500 x 500.

Entre los tiempos de prueba, considerando una matriz de 500 x 500 destacan los siguientes resultados:

- 150 segundos
- 118 segundos
- 89 segundos
- 70 segundos
- 38 segundos (Con -O2)
- 25 segundos (Con -O2)

3 Complejidad y eficiencia

A pesar de que hubieron muchos problemas, pruebas y errores con el proceso de creación, resulta que si se tiene un Algoritmo bastante eficiente, ya que es capaz de recorrer matrices de 500 x 500 en casi 1 minuto (25-38 segundos si consideramos la compilación con -O2). Si ignoramos las creaciones de Cost, Assignment y Container, es probable que tengamos una complejidad de O(n), debido a que es necesario inicialmente recorrer toda la fila de procesos para encontrar el minimo.

Si hacemos 25 pruebas de 500 x 1000, puede demorarse aproximadamente entre 10- 15 minutos, lo cual de todas formas es bastante eficiente, especialmente si consideramos que el estandar por matriz es de 3-5 minutos, lo cual implicaría que en terminar los 100 de 500 x 500 podria demorar desde 50 minutos hasta casi 1 hora, por lo que claramente se superó el estandar.

Aún asi y considerando lo mencionado en la parte de pruebas y errores, se logró disminuir considerablemente la complejidad, ya que incluso fue posible obtener una complejidad de $O(n) * O(n)$ debido a las malas implementaciones de las Heaps y Nodos, por lo que es correcto mencionar de que se tiene un código eficiente, que supera el estandar de tiempo promedio esperado.

Cabe destacar además, de que matrices bajo 300 x 300 pueden incluso demorarse menos de 10 segundos, dependiendo de si se tiene la opción de -O2 activada. En caso de problemas con matrices de 500 x 500, se recomienda hacer pruebas con matrices de menor tamaño.

4 Conclusiones

Finalmente, se logró completar la tarea cumpliendo los objetivos propuestos, donde se tiene un algoritmo que es capaz de resolver un gran número de matrices de diferentes dimensiones, encontrando la gran mayoria de las veces la combinación optima.

A pesar de las complicaciones, se logró programar la tarea en C++ aprendiendo nuevas cosas en el proceso y se espera que este conocimiento sirva de experiencia para las futuras entregas de la Asignatura.

5 ANEXOS

```
Ingresar cantidad de procesos: 7
Matriz n: 1
7 5 3 2 4 8
8 5 1 3 2 4
5 1 6 8 1 4
1 7 5 2 4 8
1 3 7 5 4 2
7 5 1 8 2 3
A la CPU 0 se le asigna el procesador 3
A la CPU 1 se le asigna el procesador 2
A la CPU 2 se le asigna el procesador 1
A la CPU 3 se le asigna el procesador 0
A la CPU 4 se le asigna el procesador 5
A la CPU 5 se le asigna el procesador 4
Matriz Binaria (Solucion):
0 0 0 1 0 0
0 0 1 0 0 0
0 1 0 0 0 0
1 0 0 0 0 0
0 0 0 0 0 1
0 0 0 0 1 0

Costo minimo: 9
Solved in: 4e-05[s]

Suma de todos los tiempos: 4e-05
Tiempo promedio: 4e-05
```

Figura 6: Solución de una matriz de costo de de 6×6

Figura 7: Parte de la solución de una matriz de costo de 50×50

```

A la CPU 424 se le asigna el procesador 128
A la CPU 425 se le asigna el procesador 30
A la CPU 426 se le asigna el procesador 486
A la CPU 427 se le asigna el procesador 479
A la CPU 428 se le asigna el procesador 442
A la CPU 429 se le asigna el procesador 62
A la CPU 430 se le asigna el procesador 66
A la CPU 431 se le asigna el procesador 70
A la CPU 432 se le asigna el procesador 228
A la CPU 433 se le asigna el procesador 284
A la CPU 434 se le asigna el procesador 435
A la CPU 435 se le asigna el procesador 73
A la CPU 436 se le asigna el procesador 1
A la CPU 437 se le asigna el procesador 5
A la CPU 438 se le asigna el procesador 461
A la CPU 439 se le asigna el procesador 428
A la CPU 440 se le asigna el procesador 177
A la CPU 441 se le asigna el procesador 271
A la CPU 442 se le asigna el procesador 223
A la CPU 443 se le asigna el procesador 60
A la CPU 444 se le asigna el procesador 296
A la CPU 445 se le asigna el procesador 400
A la CPU 446 se le asigna el procesador 278
A la CPU 447 se le asigna el procesador 7
A la CPU 448 se le asigna el procesador 403
A la CPU 449 se le asigna el procesador 478
A la CPU 450 se le asigna el procesador 42
A la CPU 451 se le asigna el procesador 437
A la CPU 452 se le asigna el procesador 3
A la CPU 453 se le asigna el procesador 202
A la CPU 454 se le asigna el procesador 240
A la CPU 455 se le asigna el procesador 97
A la CPU 456 se le asigna el procesador 288
A la CPU 457 se le asigna el procesador 380
A la CPU 458 se le asigna el procesador 185
A la CPU 459 se le asigna el procesador 359
A la CPU 460 se le asigna el procesador 335
A la CPU 461 se le asigna el procesador 118
A la CPU 462 se le asigna el procesador 48
A la CPU 463 se le asigna el procesador 113
A la CPU 464 se le asigna el procesador 12

```

Figura 8: Parte de la solución de una matriz de costo de 500 x 500 (Es tan grande que no cabe todo en la pantalla)

```

Matriz n: 100
1 5 2 3 6
3 7 5 1 4
1 4 6 7 2
6 5 1 4 7
6 4 7 2 1
A la CPU 0 se le asigna el procesador 0
A la CPU 1 se le asigna el procesador 3
A la CPU 2 se le asigna el procesador 4
A la CPU 3 se le asigna el procesador 2
A la CPU 4 se le asigna el procesador 1
Matriz Binaria (Solucion):

1 0 0 0 0
0 0 0 1 0
0 0 0 0 1
0 0 1 0 0
0 1 0 0 0

Costo minimo: 9
Solved in: 4.7e-05[s]

Suma de todos los tiempos: 0.003588
Tiempo promedio: 3.588e-05

```

Figura 9: Promedio de tiempo tras 100 iteraciones de matrices de 5 x 5