Nombre: Christofer Rodríguez Rut: 20.239.786-7 Sección: B2

PROBLEMA 1

Diga si es Verdadero o Falso. Justifique.

A) Un aspecto relevante para considerar en el diseño de un algoritmo paralelo es el acceso a la memoria por parte de cada uno de los procesadores sobre los que se ejecutará el algoritmo.

R: Verdadero, ya que al tener múltiples procesadores accediendo a memoria de manera paralela pueden existir errores; un ejemplo de esto sería que 2 procesadores intenten escribir en la misma dirección a la vez.

B) Siempre es posible paralelizar un algoritmo secuencial que utiliza la técnica de backtracking.

R: Verdadero, ya que generalmente los algoritmos de backtracking se modelan como un árbol, por ejemplo, se podría dividir en 2 procesadores, uno para cada árbol hijo de la raíz y generalizando esta simple paralelización de un algoritmo de backtracking, podríamos decir que siempre se puede paralelizar.

PROBLEMA 2

Usted y su hermana gemela están planeando visitar, el próximo domingo, un centro comercial para realizar las siguientes actividades: ver una película, comprar ropa para ambos, comprar artículos en el supermercado y comer en un restaurante. Las actividades pueden realizarse en cualquier orden.

a) Construya un algoritmo para que usted y su hermana realicen todas las actividades en conjunto. Considere todas las actividades necesarias desde el momento que salen de su casa hasta que retornen a ella, por ejemplo: traslados, esperas para ser atendidos, etc.

Psudocódigo:

```
Buscar un medio de transporte hacia el centro comercial (Yo, Hermana)
Tomar un medio de transporte (Yo, Hermana)
Ir al cine (Yo, Hermana)
Comprar entradas para la película (Yo, Hermana)
Si (hay que esperar por la pelicula) hacer
     Esperar (Yo, Hermana)
Ir al supermercado (Yo, Hermana)
Buscar los productos (Yo, Hermana)
Ir a pagar por los productos (Yo, Hermana)
Si (hay fila) hacer
     Esperar (Yo, Hermana)
Pagar los productos
Ir a una tienda de ropa (Yo, Hermana)
Seleccionar ropa (Yo, Hermana)
Pagar la ropa (Yo, Hermana)
Ir a un restaurant (Yo, Hermana)
Pedir la comida (Yo, Hermana)
Mientras (yo tenga comida o hermana tenga comida) hacer
     Comer (Yo, Hermana)
Pagar la cuenta de la comida (Yo, Hermana)
Buscar un transporte para volver a la casa (Yo, Hermana)
Tomar el transporte hacia la casa (Yo, Hermana)
Entrar a la casa (Yo, Hermana)
```

b) Debido a que el lunes usted debe rendir un examen y desea destinar el mayor tiempo posible a repasar los contenidos después de su visita al centro comercial, decide que las únicas actividades que necesariamente realizará en conjunto con su hermana serán: ver una película y comer. Construya un algoritmo que le permita a usted y a su hermana realizar

todas las actividades descritas en el algoritmo anterior y que le permita disponer de un mayor tiempo para preparar su examen. Considere que deben salir y retornar juntos.

```
Buscar un medio de transporte hacia el centro comercial (Yo, Hermana)
Tomar un medio de transporte (Yo, Hermana)
Ir al cine (Yo, Hermana)
Comprar entradas para la película (Yo, Hermana)
Si (hay que esperar por la película) hacer
     Esperar (Yo, Hermana)
Paralelamente yo voy al supermercado a comprar y mi hermana va a comprar ropa
Reunirnos (Yo, Hermana)
Ir a un restaurant (Yo, Hermana)
Pedir la comida (Yo, Hermana)
Mientras (yo tenga comida o hermana tenga comida) hacer
     Comer (Yo, Hermana)
Pagar la cuenta de la comida (Yo, Hermana)
Buscar un transporte para volver a la casa (Yo, Hermana)
Tomar el transporte hacia la casa (Yo, Hermana)
Entrar a la casa (Yo, Hermana)
```

c) En caso de ser trillizos ¿dispondría de más tiempo para estudiar para su examen si se considera el algoritmo anterior? Analice.

Si consideramos que las actividades que se deben hacer en conjunto y aquellas que se pueden hacer por separado se mantienen, además pensando el hecho de “hacer cosas separado” como realizar esto de manera paralela, no tendría más tiempo para estudiar, ya que solo son 2 actividades las que se pueden realizar por separado, uno de los trillizos tendría que acompañar a un hermano/a o no hacer nada, por lo que el tiempo se mantendría. También se podría considerar el hecho de que por ejemplo, dos hermanos vayan al supermercado a comprar juntos y al volver a dividir este proceso paralelamente, el tiempo de respuesta reduciría, generando que efectivamente terminemos antes y tanga más tiempo para estudiar, pero ya que debemos esperar a que el otro hermano/a termine, el tiempo se mantendría.

PROBLEMA 3

El problema de redistribución de carga se presenta en los principales puertos del mundo, este consiste en encontrar el mínimo número de contenedores de un conjunto con determinadas capacidades, de forma que, sumadas sus capacidades, sea posible redistribuir en ellos, en forma exacta, la carga total de un contenedor que está siendo desembarcado en el puerto.

Por ejemplo, si se busca el mínimo de contenedores para redistribuir la carga de un contenedor de 10 toneladas y se cuenta con tipos de contenedores con capacidades de 1 y 5 toneladas, y además se supone que siempre hay suficientes contenedores de cada tipo como para redistribuir toda la carga en contenedores de un solo tipo, las posibles formas de redistribuir la carga son:

Opción 1: 10 contenedores, distribuyéndose la carga de la siguiente forma 10 toneladas = 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1 + 1

Opción 2: 6 contenedores, distribuyéndose la carga de la siguiente forma 10 toneladas = 1 + 1 + 1 + 1 + 1 + 5

Opción 3: 2 contenedores, distribuyéndose la carga de la siguiente forma 10 toneladas = 5 + 5

Por lo tanto, la solución óptima del problema es la tercera opción que requiere solamente 2 contenedores.

Utilizando el enfoque de programación dinámica, describa un algoritmo que resuelva el problema general de redistribución de carga. Este es, dado un contenedor de capacidad igual a n (número entero) toneladas y disponiéndose una lista de x tipos de contenedores de distintas capacidades, encontrar el mínimo número de contenedores necesarios para redistribuir la carga. En la lista de tipos de contenedores, siempre habrá un tipo de contenedores con capacidad de una tonelada.

Considere para su descripción: la descomposición en subproblemas, el/los casos base y la estructura de datos para almacenar las soluciones parciales.

Para resolver este problema con el método de programación dinámica, lo modelaremos como el problema de la mochila, pero en vez de buscar el máximo beneficio en cada solución parcial, buscaremos el mínimo de contenedores, de cierta manera homologando el beneficio de cada elemento como que cada container tiene un beneficio de 1 y luego se decidirá si el óptimo lo obtenemos utilizando el container o no utilizándolo (casilla en la posición superior).

Si bien la mochila convencional vemos si podemos agregar el nuevo elemento y sumamos el beneficio que este aporta este elemento más el beneficio que aporta una mochila con la capacidad restante y con los elementos anteriores al actual; en esta implementación como podemos ocupar indefinidamente un mismo container, para saber el beneficio (cantidad container) utilizaremos la operación división entera entre la capacidad total sobre la capacidad del container que estamos analizando, de esta manera simularemos el hecho de utilizar más de una vez un mismo container y luego comprobaremos si la división entre estos números deja un resto con la operación modulo, en caso de que quede resto, se buscara en la tabla el optimo de container con el peso restante.

Teniendo una lista X con los tipos de contenedores, n que representa la capacidad del contenedor que deseamos almacenar. Además, tendremos un matriz A[X][N] con la misma cantidad de filas como tipos de container y la misma cantidad de columnas que peso del container a distribuir.

X/N	0	1	2	3	4	5	6	7	8	9	10
0											
1											
5											

*Sea // la operación división entera y mod la operación modulo.

Pseudo código:

```
Se llena toda la primera fila de A con 0 y la primera columna con 0
Mientras i = 1 <= largoX hacer
      Mientras j = 1 <= N hacer
            Num divEntera = j // X[i-1]
            num resto = j mod X[i-1]
            bActual = divEntera + A[i-1][resto]
            A[i][j] = Mínimo(bActual , A[i-1][j-1])
            j = j+1
      i = i+1
RETURN A[X][N]
```

Teniendo en cuenta de que el algoritmo anteriormente expuesto, técnicamente consiste en llenar una matriz, podemos decir que posee un orden de complejidad de $O(n^2)$. Si bien este algoritmo soluciona el problema con un orden de complejidad no tan elevado, este quizás podría ser mejorado al cambiar la manera en la que se almacenan las soluciones parciales, quizás si se pudieran almacenar en una tabla, el algoritmo podría llegar aproximadamente a un $O(n)$.

Ejemplo del estado final de la tabla con el ejemplo del enunciado:

X/N	0	1	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9	10
5	0	1	2	3	4	1	2	3	4	5	2