Programación III

Guía de ejercicios propuestos

Parte 1: Análisis de eficiencia temporal

1. Calcular la eficiencia de los siguientes procedimientos, mostrando la justificación del cálculo:

   a.

```
Metodo_1 (var conjunto[1..n], int n){
   bool variableCond = calcularCondicion( conjunto)
   Si (variableCond){
      Procesar(conjunto)
      Metodo_1(conjunto, n/3)
   }sino{
      Metodo_1(conjunto, n/3)
   }
   fin sino
}
```

Donde el tiempo de "`calcularCondicion`" es $O(log\ n)$, y el "`Procesar`" es $O(n)$

   b.

```
Metodo_2 (var conjunto[1..n], int n){
   bool variableCond = calcularCondicion( conjunto)
   Si (variableCond){
      Procesar(conjunto)
   }sino{
      Metodo_2(conjunto, n/2)
      Metodo_2(conjunto, n/2)
   }
   fin sino
}
```

Donde el tiempo de "`calcularCondicion`" y de "`Procesar`" son constantes

   c.

```
Metodo_3 (var conjunto[1..n], int n){
   int val = calcularValor( n)
   mientras (val < n){
      Procesar(conjunto)
   }fin mientras
   Metodo_3(conjunto, n-1)
   Metodo_3(conjunto, n-1)
}
```

Donde el tiempo de "`calcularValor`" es constante y "`Procesar`" es *O(n log n)*

2. Dado el siguiente algoritmo DyC y conociendo que su costo es de *O(n)*, ¿Qué orden tiene el método *P*?

```
DyC(V[inicio..fin])
   si(fin>inicio)
      mitad = (inicio+fin)/2
      DyC(V[inicio..mitad])
      DyC(V[mitad+1..fin])
      P(V)
   finsi
finDyC
```

Parte 2: Divide y conquista

A continuación se presenta una lista de ejercicios. Para cada uno de ellos

- Esbozar el algoritmo que resuelve el problema
- Identificar las partes que componen la solución y su relación con el esquema modelo de la técnica elegida
- Detallar las estructuras que se utilizarán
- Escribir el método en *Java* para resolverlo
- Analizar la complejidad de la solución en tiempo y espacio.

1. Determinar si una secuencia de ***n*** caracteres está ordenada alfabéticamente.

2. Dado un vector de ***n*** números naturales ordenados crecientemente, determinar si un número ***x*** dado pertenece al vector.

3. Calcular a^n cuando **n** es una potencia de 2.

4. Dado un vector ***A*** de números enteros, calcular elemento mayoritario. Si se tiene un vector ***A*** de ***n*** enteros, un elemento ***x*** se denomina mayoritario de ***A*** si ***x*** aparece en el vector ***A*** más de ***n/2*** veces. Considerar que no puede haber más de un elemento mayoritario.

5. Dadas dos matrices cuadradas ***M1*** y ***M2*** de ***NxN***, calcular el producto de ambas, donde ***N*** es una potencia de ***2***.

6. Sea ***A[1..n]***, $n \geq 1$, un vector de enteros diferentes y ordenados crecientemente, tal que algunos de los valores pueden ser negativos. Diseñar un algoritmo que devuelva un índice natural ***k***, $1 \leq k \leq n$, tal que ***A[k] = k***, siempre que tal índice exista.

7. Dado un Vector ***A*** de números enteros, ordenarlo en forma creciente. Utilizar el método de ordenamiento Merge-Sort, pero dividiendo el vector en 3 subvectores y analizar el costo.

8. Un organismo ha decidido organizar un torneo de fútbol con **n** equipos participantes. Cada equipo ha de competir exactamente una vez con todos los

demás equipos oponentes. Además, se ha decidió que cada equipo juega exactamente un partido cada jornada, con la posible excepción de un solo día en el cual no juega. Si **n** es una potencia de 2, diseñar un algoritmo que permita que el torneo concluya en **n-1** jornadas.

9. El problema del par más cercano consiste en encontrar dos puntos dentro de un conjunto de puntos cuya distancia sea menor que la que existe entre cualquier otro par de puntos del conjunto. Suponiendo que los puntos vienen ordenados por sus coordenadas **(x; y),** y que han sido clasificados en orden ascendente de la coordenada **x**, resolver el problema.

Parte 3: Greedy

A continuación, se presenta una lista de ejercicios. Para cada uno de ellos

- Esbozar el algoritmo que resuelve el problema
- Identificar las partes que componen la solución y su relación con el esquema modelo de la técnica elegida
- Demostrar su correctitud
- Detallar las estructuras que se utilizarán
- Escribir el método en *Java* para resolverlo
- Analizar la complejidad de la solución en tiempo y espacio.

1. Cambio de monedas: Dado un conjunto ***C*** de ***N*** tipos de monedas con un número ilimitado de ejemplares de cada tipo, se requiere formar, si se puede, una cantidad ***M*** empleando el mínimo número de ellas. Por ejemplo, un cajero automático dispone de billetes de distintos valores: $100, $25, $10, $5 y $1, si

se tiene que pagar $289, la mejor solución consiste en dar 10 billetes: 2 de $100, 3 de $25, 1 de $10 y 4 de $1.

2. Problema de la Mochila: Se tienen ***n*** objetos y una mochila. Para ***i = 1,2,..n***, el objeto ***i*** tiene un peso positivo p_i y un valor positivo v_i. La mochila puede llevar un peso que no sobrepase ***P***. El objetivo es llenar la mochila de tal manera que se maximice el valor de los objetos transportados, respetando la limitación de capacidad impuesta. Los objetos pueden ser fraccionados, si una fracción x_i ($0 \leq x_i \leq 1$) del objeto ***i*** es ubicada en la mochila contribuye en $x_i * p_i$ al peso total de la mochila y en $x_i * v_i$ al valor de la carga. Formalmente, el problema puede ser establecido como:

maximizar $\sum_{i=1}^{n} x_i * v_i$, con la restricción $\sum_{i=1}^{n} x_i * p_i \leq P$

donde $v_i > 0$, $p_i > 0$ y $0 \leq x_i \leq 1$ para $1 \leq i \leq n$.

Por ejemplo, para la instancia $n = 3$ y $P = 20$

$(v1, v2, v3) = (25, 24, 15)$

$(p1, p2, p3) = (18, 15, 10)$

Algunas soluciones posibles son:

(x1, x2, x3)	$x_i * p_i$	$x_i * v_i$
(1/2, 1/3, 1/4)	16.5	24.25

(1, 2/15, 0)	20	28.2
(0, 2/3, 1)	20	31
(0, 1, 1/2)	20	31.5

puede observarse que ***(0, 1, 1/2)*** produce el mayor beneficio.

3. Planificación de tareas con plazo fijo: Se deben procesar ***n*** tareas en un único procesador. Cada tarea se procesa en una unidad de tiempo y debe ser ejecutada en un plazo no superior a t_i. La tarea ***i*** produce una ganancia $g_i > 0$ si se procesa en un instante anterior a t_i. Una solución es factible si existe al menos una secuencia ***S*** de tareas que se ejecuten antes de sus respectivos plazos. Una solución óptima es aquella que maximiza la ganancia ***G*** tal que :

$$G = \sum_{s \in S} g_s$$

Diseñar un algoritmo para encontrar la solución óptima.

Por ejemplo, para la instancia ***n = 4*** y los siguientes valores:

(g1, g2, g3, g4) = (50, 10, 15, 30)

(t1, t2, t3, t4) = (2, 1, 2, 1)

Las planificaciones que hay que considerar y los beneficios correspondientes son:

Secuencia	***Beneficio***	***Secuencia***	***Beneficio***
1	50	***2,1***	60
2	10	***2,3***	25
3	15	***3,1***	65
4	30	***4,1***	80
1,3	65	***4,3***	45

Para maximizar el beneficio en este ejemplo, se debería ejecutar la secuencia ***4,1.***

4. Minimizar tiempo de espera: Un procesador debe atender ***n*** procesos. Se conoce de antemano el tiempo que necesita cada uno de ellos. Determinar en qué orden el procesador debe atender dichos procesos para minimizar la suma del tiempo que los procesos están en el sistema.

   Por ejemplo, para ***n = 3*** se tienen, los procesos ***(p1, p2, p3)*** y tiempos de proceso ***(5, 10, 3)***

Orden de atención	***Tiempo de espera***
p1, p2, p3	5 + (5+10) + (5+10+3) = 38
p1, p3, p2	5 + (5+3) + (5+3+10) = 31
p2, p1, p3	10 + (10+5) + (10+5+3) = 43
p2, p3, p1	10 + (10+3) + (10+3+5) = 41
p3, p1, p2	3 + (3+5) + (3+5+10) = 29
p3, p2, p1	3 + (3+10) + (3+10+5) = 34

   La ordenación que produce el tiempo de espera mínimo es ***(p3, p1, p2)***.

5. Dado un conjunto de **n** cintas con n_i registros ordenados en cada una de ellas, se pretende mezclarlas de a pares hasta lograr una única cinta ordenada. La

secuencia en la que se realiza la mezcla determinará la eficiencia del proceso. Diseñar un algoritmo que busque la solución óptima minimizando el número de movimientos.

Por ejemplo: 3 cintas: A con 30 registros, B con 20 y C con 10:

A. Mezclamos A con B (50 movimientos) y el resultado con C (60 movimientos), con lo que realizamos en total 110 movimientos.
B. Mezclamos C con B (30 Movimientos) y el resultado con A (60). Total = 90 movimientos

6. Un mecánico necesita llevar a cabo **n** reparaciones urgentes conociendo el tiempo que le va a llevar cada una de ellas, es decir, en la tarea *i*-ésima tardará t_i minutos. Debido a que el pago que recibe el mecánico depende del nivel de satisfacción del cliente, necesita decidir el orden en el que atenderá cada reparación para minimizar el tiempo medio de espera de los clientes. Es decir, si llamamos E_i a lo que espera el cliente *i*-ésimo hasta ver finalizada su reparación, se necesita minimizar la expresión:

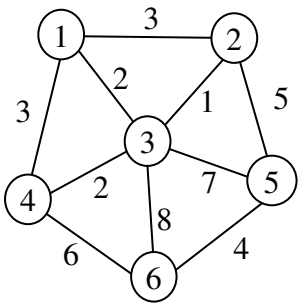
$$T(n) = \sum_{I=1}^{n} E_i$$

Parte 3. Anexo Algoritmos sobre Grafos

1. Dado un grafo **G** dirigido, cuyos nodos representan las escuelas y gimnasios de un barrio en forma numérica, y sus aristas la distancia en cuadras para llegar entre los diferentes lugares representados en el grafo. Determinar cuántas son las

cuadras mínimas para llegar de la escuela **X** al gimnasio **Y** (ambos puntos están representados en el grafo).

2. Aplicar el algoritmo de Kruskal sobre el siguiente grafo, mostrando el orden en que son añadidas las aristas a la solución.



Si aplicáramos el algoritmo de Prim, ¿podemos asegurar que se obtendría siempre la misma solución? ¿Podemos asegurar que el coste de la solución sería el mismo? ¿Por qué?

3. Dado un GrafoTDA **G** dirigido, cuyos nodos representan la ciudad de Buenos Aires y las ciudades capitales de las provincias de la Argentina, y las aristas entre esos nodos los tiempos de vuelo entre las ciudades. Determinar cuáles ciudades convienen ser alcanzadas desde la ciudad de Buenos Aires a través de otra ciudad, y no en forma directa. Suponer que el tiempo de demora de aterrizaje es despreciable.

4. Las emisiones de los automóviles privados son una de las principales fuentes de contaminación en las grandes ciudades. Las autoridades universitarias han decidido contribuir seriamente a la mejora de la calidad del aire prohibiendo el paso de automóviles por el campus y creando una serie de líneas de autobuses eléctricos que permitan acceder a distintos puntos de la Ciudad Universitaria. Para ello se han estudiado los principales flujos de vehículos entre distintos puntos seleccionados

(facultades, estación de metro, etc.) de la Ciudad Universitaria. Los resultados del estudio contienen el número de vehículos que transitan al día entre estos puntos seleccionados (sin tener en cuenta el sentido en el que circulan). El objetivo de este estudio es localizar los trayectos entre puntos seleccionados que utilizan más vehículos y que permiten conectar todos los puntos seleccionados. Diseñar un algoritmo que proporcione estos trayectos. Validar que el algoritmo sea óptimo en base a algoritmos conocidos.

Parte 4: Programación Dinámica

A continuación, se presenta una lista de ejercicios, para cada uno de ellos

- Escribir el planteo recursivo que representa la solución
- Analizar la optimalidad de la solución
- Esbozar el algoritmo que resuelve el problema
- Detallar las estructuras que se utilizarán
- Escribir el método en *Java* para resolverlo
- Analizar la complejidad de la solución en tiempo y espacio.

1. Cambio de monedas: Dado un conjunto ***C*** de ***N*** tipos de monedas, cada una con una denominación c_i (**1<=i<=n**), con un número ilimitado de ejemplares de cada tipo, se requiere determinar, si se puede, la cantidad mínima de ellas necesarias para formar una cantidad ***M***. Por ejemplo, un expendedor de boletos electrónico dispone de monedas de distintos valores: ***$100***, ***$25***, ***$10***, ***$5*** y ***$1***, si se tiene que pagar ***$289***, la mejor solución consiste en dar ***10*** monedas (***2*** de ***$100***, ***3***

de ***$25***, ***1*** de ***$10*** y ***4*** de ***$1***). Otro ejemplo sería si se disponen de monedas de ***$5***, ***$4*** y ***$1***, y se quieren entregar ***$8***, la mejor solución sería entregar ***2*** monedas de ***$4***. Y si además de saber cuántas monedas se requieren en total, se quisiera averiguar cuántas de cada denominación?

2. Problema de la Mochila: Se tienen ***n*** objetos y una mochila. Para ***i = 1, 2,..n***, el objeto ***i*** tiene un peso positivo p_i y un valor positivo v_i. La mochila puede llevar un peso que no sobrepase ***P***. Los objetos no pueden ser fraccionados. El objetivo es llenar la mochila de manera tal que se maximice el valor de los objetos transportados, respetando la limitación de capacidad impuesta. El algoritmo debe determinar el valor máximo que se podrá cargar. ¿Y si además se quiere saber qué objetos se deberá cargar para alcanzar dicho valor?

3. Problema del Viajante: Un viajante de comercio debe visitar ***n*** ciudades. Cada ciudad está conectada con las restantes mediante carreteras de longitud conocida. El problema consiste en hallar la longitud de la ruta que deberá tomar para visitar todas las ciudades retornando a la ciudad de partida, pasando una única vez por cada ciudad y de modo tal que la longitud del camino recorrido sea mínima.

4. Multiplicación de matrices: Sea $M = M_1 \times M_2 \times \ldots \times M_n$ una multiplicación encadenada de matrices de dimensiones conocidas. Como la multiplicación es asociativa, hay varias maneras de resolver secuencias de productos matriciales. Se desea conocer la menor cantidad posible de multiplicaciones necesarias para resolver el problema. Dadas dos matrices $\mathbf{M}_1$ y $\mathbf{M}_2$ de dimensiones $\mathbf{m}_1\mathbf{x}\mathbf{m}_2$ y $\mathbf{m}_2\mathbf{x}\mathbf{m}_3$ respectivamente, la cantidad necesaria de multiplicaciones es $\mathbf{m}_1\mathbf{x}\mathbf{m}_2\mathbf{x}\mathbf{n}_3$. Y si además quisiéramos determinar las asociaciones necesarias?

5. Problema de la subsecuencia más larga: Dada una secuencia $X=\{x_1\ x_2\ \ldots\ x_m\}$, decimos que $Z=\{z_1\ z_2\ \ldots\ z_k\}$ es una subsecuencia de X (siendo $k \leq m$) si existe una secuencia creciente $\{i_1\ i_2\ \ldots\ i_k\}$ de índices de X tales que para todo $j = 1, 2, \ldots, k$ tenemos $x_{ij} = z_j$.

   Por ejemplo, $Z=\{BCDB\}$ es una subsecuencia de $X=\{ABCBDAB\}$ con la correspondiente secuencia de índices $\{2,3,5,7\}$.

   Dadas dos secuencias X e Y, decimos que Z es una subsecuencia común de X e Y si es subsecuencia de X y subsecuencia de Y.

   Se desea encontrar la longitud de la subsecuencia de longitud máxima común a dos secuencias dadas. Y si además se quisiera encontrar la subsecuencia propiamente dicha?

6. Dadas n funciones f_1, f_2, …, f_n y un entero positivo **M**, deseamos maximizar la función $f_1(x_1) + f_2(x_2) + \ldots + f_n(x_n)$ sujeta a la restricción $x_1 + x_2 + \ldots + x_n = M$, donde $f_i(0) = 0$ $(i=1,..,n)$, x_i son números naturales, y todas las funciones son monótonas crecientes, es decir, $x \leq y$ implica que $f_i(x) > f_i(y)$. Supóngase que los valores de cada función se almacenan en un vector.

7. Ana trabaja en una empresa de comunicaciones y le han encargado la compra del regalo de cumpleaños para Darío, un compañero de trabajo. Para ello Ana va a un negocio en donde hay diferentes obsequios que le pueden gustar a Darío. Cada obsequio tiene un precio determinado. Ana conoce el monto mínimo a gastar para la compra del regalo, y su objetivo es encontrar una combinación de objetos a comprar que cubra exactamente el monto a gastar o lo supere en forma

mínima, considerando que no puede repetir objetos en su compra. Diseñar un algoritmo que determine cuáles son los objetos que conforman el regalo y el dinero óptimo gastado.

8. Howard es un fanático de los comics. En la tienda de comics tienen un único ejemplar de cada revista. Cada revista tiene en la tapa el precio y la cantidad de páginas que contiene. Un día particular Howard decide comprar todas las revistas posibles de modo de maximizar la cantidad total de páginas sin superar su presupuesto disponible de $P. La tienda tiene listados con la información de precio y cantidad de páginas de cada revista ordenados por todos los criterios que sean necesarios. ¿Cuántas páginas en total tendrá si compra la mejor combinación posible de revistas?

Repaso de las tres técnicas anteriores

A continuación, se presenta una lista de ejercicios, para cada uno de ellos

- Justificar la elección de la técnica
- Esbozar el algoritmo que resuelve el problema
- Detallar las estructuras que se utilizarán
- Escribir el método en *Java* para resolverlo
- Analizar la complejidad de la solución en tiempo y espacio.

1. *Trasponer un vector*: Dados un vector **V[*0..n*]** y un número natural ***k*** entre ***1*** y ***n***, diseñar un algoritmo eficiente que trasponga los ***k*** primeros elementos de ***V*** en los

elementos en las ***n-k*** últimas posiciones, sin hacer uso de un vector auxiliar. Por ejemplo, si ***V*** es el siguiente vector:

3	5	12	8	9	12	4	7	13	21

Si ***K = 3***, el resultado debe ser:

8	9	12	4	7	13	21	3	5	12

2. *N embarcaderos*: A lo largo de un río hay **n** embarcaderos. En cada embarcadero se puede tomar una lancha que va hasta cualquier otro embarcadero río abajo (no es posible ir río arriba). Existe una tabla de tarifas de cada viaje entre un embarcadero **i** y un embarcadero **j** (***i<j***). El objetivo del problema es llegar desde un embarcadero **i** hasta un embarcadero **j** utilizando la menor cantidad de dinero posible, aunque haya que cambiar de lancha todas las veces que sea necesario, ya que este cambio no tiene costo adicional.

3. *Maximizar número de actividades compatibles*. Se tienen ***n*** actividades que necesitan utilizar un recurso, tal como una sala de conferencias, en exclusión mutua. Cada actividad ***i*** tiene asociado un tiempo de comienzo c_i y un tiempo de finalización f_i de utilización del recurso, con $c_i < f_i$. Si la actividad ***i*** es seleccionada se llevará a cabo durante el intervalo $[c_i, f_i)$. Las actividades ***i*** y ***j*** son compatibles si los intervalos $[c_i, f_i)$ y $[c_j, f_j)$ no se superponen (es decir,

$c_i \geq f_j$ o ***$c_j \geq f_i$***). El problema consiste en encontrar la cantidad máxima de actividades compatibles entre sí.

4. *Mediana de dos vectores*: Dados dos vectores de enteros ***X[0..n]*** e ***Y[0..n]*** con ***n>=1***, ordenados de forma creciente, escribir un algoritmo para hallar la mediana del vector formado por el total de los ***2n*** elementos.

5. *Subsecuencia de suma máxima*: Dado un vector ***V[0..n]*** de enteros, encontrar los valores de ***i*** y ***j***, con ***0<=i <= j <= n***, tales que se $\sum_{k=i}^{j} V[k]$ maximice

   Diseñar un algoritmo que resuelva el problema en un tiempo en O(n log n).

6. *Traducciones*: Una empresa de traducciones desea realizar la traducción de textos de un idioma a otro. Para ello cuenta con algunos diccionarios bilingües. Cada diccionario sirve para hacer la traducción entre dos idiomas en cualquiera de los dos sentidos. Si se tienen **N** idiomas y **M** diccionarios, determinar la menor cantidad de traducciones que se deberán realizar para traducir un texto entre cualquier par de idiomas, siempre que sea posible realizarlo con los diccionarios disponibles.

7. *Maximización de programas*: Sean ***n*** programas ***P_1, ...,P_n*** que hay que almacenar en un disco. El programa ***P_i*** requiere ***S_i*** GB de espacio y la capacidad del disco es ***D*** GB. Realizar un algoritmo que ingrese la mayor cantidad posible de programas en el disco.

8. *Minimizar carga de combustible*: Un viajante realiza un recorrido desde la ciudad ***A*** a la ciudad ***B*** siguiendo una ruta dada y llevando un vehículo que le permite, con

el tanque de combustible lleno, recorrer ***N*** kilómetros sin parar. El viajante dispone de un mapa de rutas que le indica las distancias entre las estaciones de servicio que hay en su ruta. Como va con prisa, el viajante desea pararse a cargar combustible el menor número de veces posible. Diseñar un algoritmo para determinar en qué estaciones de servicio tiene que parar

Parte 5: Backtracking y Ramificación Y Poda

1. Se tiene un conjunto de salas comunicadas entre sí a través de puertas que se abren solamente en un sentido. Una de las salas se denomina entrada y la otra salida. Construir un algoritmo que permita ir desde la entrada a la salida atravesando la máxima cantidad de salas.

2. Dado un sistema de rutas aéreas modelado mediante un grafo, determine la ruta que contenga la mínima cantidad de escalas entre un par de ciudades dadas.

3. Ocho reinas: Construir un algoritmo que ubique ocho reinas en un tablero de ajedrez de modo tal que no se puedan capturar entre sí

4. Suma de subconjuntos: Dados n números positivos distintos, se desea encontrar todas las combinaciones de esos números tal que la suma sea **M**.

5. Partición de conjunto: Dado un conjunto de **n** enteros se desea encontrar, si existe, una partición en dos subconjuntos disjuntos, tal que la suma de sus elementos sea la misma.

6. Asignación de tareas a procesadores: Se tienen **m** procesadores idénticos y **n** tareas con un tiempo de ejecución dado. Se requiere encontrar una asignación de

tareas a procesadores de manera de minimizar el tiempo de ejecución del total de tareas.

7. Asignación de tareas a empleados: Se tienen **n** empleados y **n** tareas, se conoce el tiempo que tardará el empleado **i** para realizar la tarea **j**. Se requiere encontrar una asignación de una tarea a cada empleado de manera de minimizar el tiempo de ejecución del total de tareas

8. Cambio de monedas: Dado un conjunto **C** de **N** tipos de monedas con un número ilimitado de ejemplares de cada tipo, se requiere formar, si se puede, una cantidad **M** empleando el mínimo número de ellas.

9. Dado un tablero de tamaño XxY y dado un rey en una casilla arbitraria (x0; y0). Cada casilla (x; y) del tablero tiene asignado un peso T(x; y), de tal forma que a cada recorrido R = {(x0; y0).....(xk ; yk)} se le puede asignar un valor que viene determinado por la siguiente expresión:

$$P(R) = \sum_{i=0}^{k} i \cdot T(x_i, y_i)$$

El problema consiste en diseñar un algoritmo que proporcione el recorrido de peso mínimo que visite todas las casillas del tablero sin repetir ninguna.

10. Un subconjunto independiente en un grafo no dirigido *G* es un subconjunto *W* de los vértices de *G* tal que para todo par de vértices en *W*, éstos no son adyacentes en G. Encontrar el conjunto independiente *W* de mayor tamaño en *G*. Por ejemplo, si V = {1, 2, 3, 4, 5} y A = {(1, 2), (1, 3), (3, 4), (2,

5), (3, 5)} los conjuntos independientes son {1},{2}, {3}, {4}, {5}, {1, 4}, {1, 5}, {2, 3}, {2, 4}, {4, 5} y {1, 4, 5}. Y el mayor es {1, 4, 5}.

11. Una empresa de traducciones ofrece servicios de traducción para eventos. La empresa tiene un grupo de traductores, cada uno de los cuales domina un conjunto de idiomas. Un día determinado tienen varios eventos simultáneos, cada uno en un idioma dado. Determinar si existe alguna asignación posible de traductores a eventos para cubrir la demanda. Por ejemplo, si tengo la siguiente información:

Traductor	Inglés	Francés	Alemán	Italiano	Árabe	Chino
Alberto	X		X	X		
Bruno	X	X	X		X	
Carlos				X	X	X
Daniel				X	X	X
Esteban		X		X		X

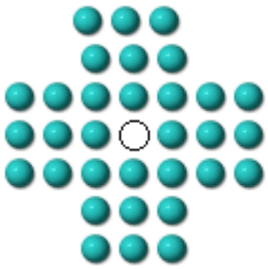
y los eventos son

1	2	3	4
Inglés	Francés	Árabe	Árabe

una posible asignación sería

1	2	3	4
Alberto	Bruno	Carlos	Daniel

12. Dado un tablero en forma de cruz de lado 3, en el cual se encuentran distribuidas 44 fichas quedando libre el centro del mismo, el juego consiste en dejar sólo una ficha en la posición central del tablero. Una ficha puede saltar sobre otra (en cualquiera de los cuatro sentidos) y caer en un lugar libre debiendo eliminarse la ficha sobre la que se saltó (Juego del Senku). Determinar una secuencia de pasos tal que logre el objetivo del juego.



13. Juego de los fósforos: Se disponen de **11** fósforos. Dos jugadores pueden retirar por turno **1**, **2** o **3** fósforos. Pierde aquel jugador que retira el último fósforo. Construir un algoritmo que determine para un jugador si existe una secuencia de jugadas ganadora.

14. Pilas de monedas: Dada una pila de **n** monedas, y **2** jugadores, quienes alternan movimientos, cada uno de ellos puede dividir una pila en otras **2**, siempre y cuando haya más de **2** monedas. Pierde el jugador que no puede dividir ninguna pila. Construir un algoritmo que determine para un jugador si existe una secuencia de jugadas ganadora.