Carlos Figueredo – 201813445
Camilo Otalora - 201732760

Parcial 3

Resuelva formalmente y envíe su solución en PDF de los siguientes problemas

Importante: Resolver los problemas 1 y 2 usando técnicas de cambio de representación en grafos. Recuerde en cada problema definir formalmente un grafo, definir el tamaño del problema, y dar los órdenes de complejidades temporales y espaciales en términos de estos últimos.

[30 puntos] Problema 1)
La empresa productora de zapatos de los Alpes fabrica estos artículos en la ciudad de Cúcuta, y los debe distribuir en las ciudades de Bogotá, Cartagena y Medellín. Actualmente la flota de distribución de la empresa es limitada, por lo que el gerente toma la decisión de diseñar un circuito de tal forma que el camión distribuidor salga de la fábrica en Cúcuta, visite las ciudades en secuencia (una ciudad después de la otra), y una vez visite las tres ciudades, se devuelva a la fábrica en Cúcuta. El gerente quiere que este circuito se recorra en el menor tiempo posible y le encarga a usted definir cómo realizar el circuito.

1a) Suponiendo que tiene un mapa de los municipios y ciudades de Colombia, y la información de los tiempos promedio que se tarda en movilizarse de un municipio a otro INMEDIATAMENTE CONTINUO. Diseñe un algoritmo que determine, cuál es el mejor orden para recorrer las tres ciudades que se deben visitar.

Es un grafo no dirigido y con pesos.

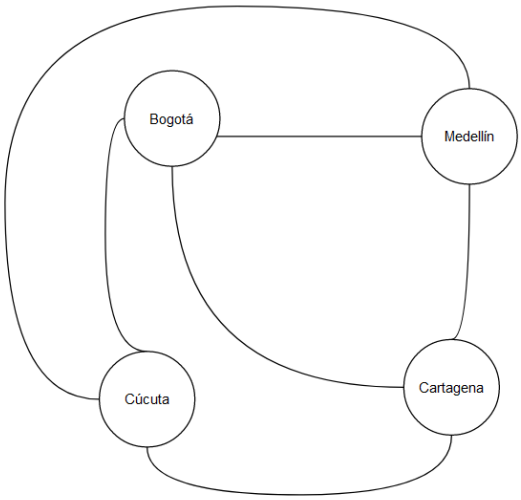
$V = "Conjunto\ de\ ciudades\ o\ municipios"$

$E = (u, v) \in E \equiv "u\ y\ v\ son\ municipios\ o\ ciudades\ continuos'$

$c: E \to R^+, \qquad c(u, v) \equiv "Tiempo\ que\ se\ tarda\ en\ viajar\ desde\ u\ a\ v'$

$Tamaño\ del\ problema := (|V|, |E|)$

Se usa el algoritmo de Dijkstra para camino más corto entre las 4 ciudades y con estructura de datos Fibonacci Heap, realizándolo un total de 6 veces. Una vez se tienen los tiempos entre esas 4 ciudades, se hace un grafo con los costos de tiempo mínimos entre las ciudades.



Carlos Figueredo – 201813445
Camilo Otalora - 201732760
Posterior a eso, se realiza programación dinámica (se vuelve un problema similar al ciclo Hamiltoniano) donde se tiene a Cúcuta como punto inicial y punto final. Sea C (S, i) el camino de menor costo que pasa por todos los vértices del conjunto S exactamente una sola vez y el i es cualquier vértice que no sea el inicial. El algoritmo resolverá esto desde sus subconjuntos.

Al terminar, usará una lista para devolver en orden las ciudades por las cuales debe pasar el camión.

$Al\ principio, |E|\ equivale\ a\ logV\ porque\ es\ un\ grafo\ poco\ denso$

$Después, el\ algoritmo\ del\ ciclo\ se\ realiza\ sobre\ unos\ datos\ de\ tamaño\ constante$

Complejidad temporal: $T = 6 * T_{Dijkstra}(|V|, |E|) + T_{CH}(4) = 6 * O(VlogV + logV) + O(1)$
$T = O(VlogV)$

Complejidad espacial: $S = 6 * S_{Dijkstra}(|V|, |E|) + S_{CH}(4) = 6 * O(V) + O(1) =$
$S = O(V)$

1b) ¿Qué modificaciones debería hacer en el algoritmo si se desea conocer la ruta explícita del circuito de distribución? ¿Cómo cambiarían las complejidades en tal escenario?

Se puede modificar de tal manera que en cada ejecución de Dijkstra, se guarde una lista con las ciudades por las que se pasa entre la ciudad A y B en el menor tiempo posible, después en el ciclo se podrá guardar el orden en el que se pasará por las ciudades solicitadas, usando los anteriores datos se pueden unir y al final retornar todas las ciudades por las que se pasa, en orden y con el menor costo de tiempo posible, las complejidades aumentarían en |V| debido a las listas, pero no cambiaría en términos de Big-O.

[20 puntos] Problema 2)
Un monitor de ingeniería de sistemas está diseñando un compilador de java para encontrar posibles DeadLocks en el código de proyectos de los estudiantes. Un posible DeadLock lo definiremos en este contexto como el escenario en el cual un método de Java m1 depende de un método m2, y un método m2 dependen a su vez del método m1. Adicionalmente, el monitor ha desarrollado una función Dependencia(.), en la cual, para un método m de Java, Dependencia(m) retorna un listado con todos los métodos los cuales dependen de la ejecución de m. El monitor desesperado por no saber qué más hacer, acude a usted.

2a) Suponiendo que tiene acceso al desarrollo hecho por el monitor y a la lista de métodos presentes en el proyecto de un estudiante. Diseñe un algoritmo que determine si existe o no un posible DeadLock en el código del proyecto Java.

Es un grafo dirigido sin peso, si x apunta a y, x depende de y

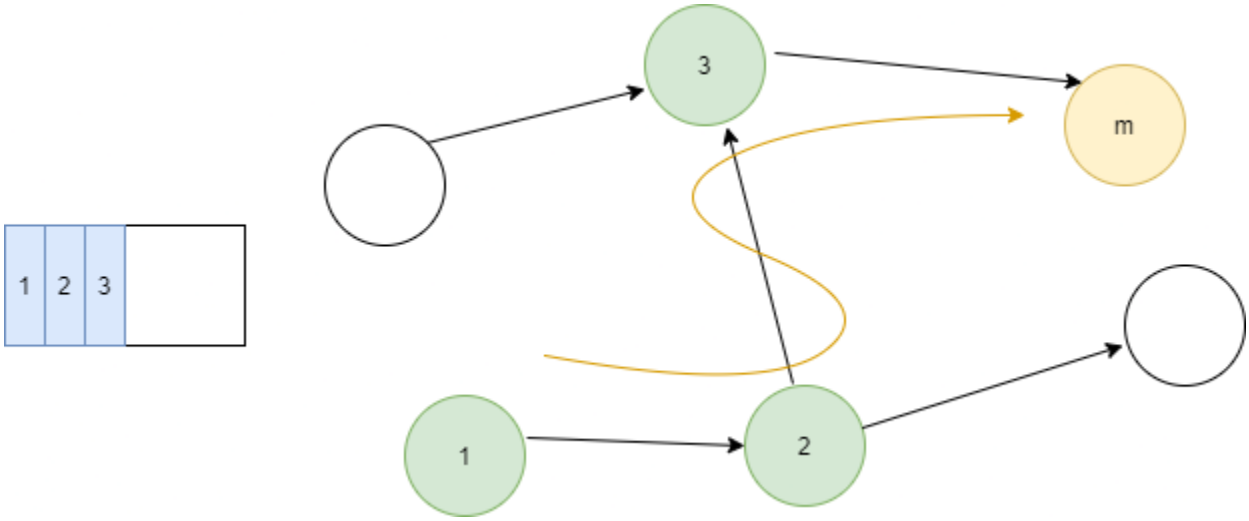
$V = "Conjunto\ de\ métodos'$

$E = (u, v) \in E \equiv "u\ está\ dirigido\ a\ v\ y\ depende\ del\ mismo'$

$Tamaño\ del\ problema: (|V|, |E|)$

Se usa un algoritmo DFS tal que empiece por un nodo fuente, luego empiece a recorrer si puede llegar al nodo m y va metiendo en una pila lo que haya recorrido, si logra llegar a m, todo lo que esté en la pila cumple dependencia(m).

Carlos Figueredo – 201813445
Camilo Otalora - 201732760



Complejidad temporal: Es un DFS, lo cual hace que recorra todos los nodos y revise todos los arcos, es decir T = O(V+E)

Complejidad espacial: Usa una pila y tiene nodos marcados, en el peor caso contendría todos los nodos,

es decir: S=O(2V) =O(V)

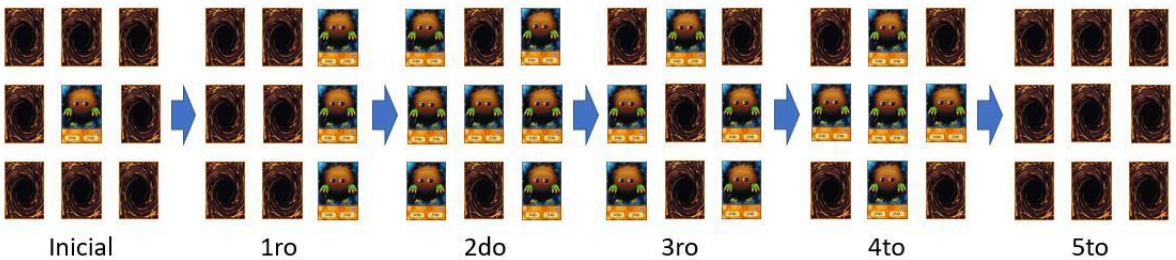
[30 puntos] Problema 3)

Considere el siguiente juego, se tienen $n \times n$ cartas (de su juego favorito), organizadas de forma matricial, de las cuales unas cartas están boca arriba y otras cartas están boca abajo. El objetivo del juego es dejar todas las cartas del tablero boca abajo. Para esto, se consideran las siguientes reglas.

• Cambiar la posición de una carta es, voltearla boca abajo en caso de que esté boca arriba. O voltear boca arriba la carta en caso de que esté boca abajo.

• Puede seleccionar cualquier carta del tablero para cambiar su posición, pero en caso de hacerlo, también debe cambiar la posición de las cartas inmediatamente arriba, abajo, a la izquierda, y a la derecha de la carta seleccionada. Este proceso completo es considerado 1 movimiento.

Se desea diseñar entonces un algoritmo de agenda que determine cuál es la mínima cantidad de movimientos necesarios para resolver un tablero de cartas de tamaño $n \times n$ que llegue por parámetro. Por ejemplo, para un tablero inicial de n=3 configurado de la siguiente forma:



Se puede resolver usando como mínimo 5 movimientos (NO hay forma de resolverlo en 4 movimientos o menos). A continuación, se muestran los movimientos que se deben hacer como mínimo para resolver este tableo:

Carlos Figueredo – 201813445
Camilo Otalora - 201732760

3a) Solucione el problema planteado definiendo formalmente $BusQ$, $SOLPOS$, sat, SOL, s, suc

$BusQ = \{x \in M_{nxn} \wedge 0 \leq k \leq n^2 | "x\ tiene\ k\ cartas\ boca\ abajo'\}$

$(\forall i,j | 0 \leq i,j < N : v[i][j] = Z)\ Siendo\ Z\ un\ booleano\ tal\ que\ false\ represente\ boca\ abajo\ y$

$true\ represente\ boca\ arriba$

$SOLPOS = BusQ$

$sat : SOLPOS \rightarrow bool$

$sat(v) \equiv (\forall i,j | 0 \leq i,j < N : v[i][j] = false)$

$SOL = \{x \in SOLPOS | sat(x)\}$

$s = Matriz\ dada\ por\ parámetro$

$suc(v) = \{Cartas\ medias\}[(\forall i,j | 0 < i,j < N : v[i][j] = \neg c[i][j] \wedge v[i+1][j] = \neg c[i+1][j]$

$\wedge v[i-1][j] = \neg c[i-1][j] \wedge v[i][j+1] = \neg c[i][j+1] \wedge v[i][j-1] = \neg c[i][j-1])]$

$\vee \{Cartas\ i = 0\}[(\forall i,j | 0 < j < N \wedge i = 0 : v[i][j] = \neg c[i][j] \wedge v[i+1][j] = \neg c[i+1][j]$

$\wedge v[i][j-1] = \neg c[i][j-1] \wedge v[i][j+1] = \neg c[i][j+1])]$

$\vee \{Cartas\ i = N-1\}[(\forall i,j | 0 < j < N \wedge i = N-1 : v[i][j] = \neg c[i][j] \wedge v[i-1][j] = \neg c[i-1][j]$

$\wedge v[i][j+1] = \neg c[i][j+1] \wedge v[i][j-1] = \neg c[i][j-1])]$

$\vee \{Cartas\ j = 0\}[(\forall i,j | 0 < i < N \wedge j = 0 : v[i][j] = \neg c[i][j] \wedge v[i+1][j] = \neg c[i+1][j]$

$\wedge v[i-1][j] = \neg c[i-1][j] \wedge v[i][j+1] = \neg c[i][j+1])]$

$\vee \{Cartas\ j = N-1\}[(\forall i,j | 0 < i < N \wedge j = N-1 : v[i][j] = \neg c[i][j] \wedge v[i+1][j] = \neg c[i+1][j]$

$\wedge v[i-1][j] = \neg c[i-1][j] \wedge v[i][j-1] = \neg c[i][j-1])]$

$\vee \{Carta\ i = 0, j = 0\}[(\forall i,j | j = 0 \wedge i = 0 : v[i][j] = \neg c[i][j] \wedge v[i+1][j] = \neg c[i+1][j]$

$\wedge v[i][j+1] = \neg c[i][j+1])]$

$\vee \{Carta\ i = N-1, j = 0\}[(\forall i,j | j = 0 \wedge i = N-1 : v[i][j] = \neg c[i][j] \wedge v[i-1][j] = \neg c[i-1][j]$

$\wedge v[i][j+1] = \neg c[i][j+1])]$

$\vee \{Carta\ i = 0, j = N-1\}[(\forall i,j | j = N-1 \wedge i = 0 : v[i][j] = \neg c[i][j] \wedge v[i+1][j] = \neg c[i+1][j]$

$\wedge v[i][j-1] = \neg c[i][j-1])]$

$\vee \{Carta\ i = N-1, j = N-1\}[(\forall i,j | j = N-1 \wedge i = N-1 : v[i][j] = \neg c[i][j] \wedge v[i-1][j] = \neg c[i-1][j]$

$\wedge v[i][j-1] = \neg c[i][j-1])]$

Si se encuentra un nodo $x \in SOL$, se sumará a los pasos necesarios para llegar a la solución, luego se retorna el total de pasos mínimos necesarios para llegar a la solución.

Carlos Figueredo – 201813445
Camilo Otalora - 201732760

3b) ¿Necesita manejar nodos marcados? Argumente por qué si o por qué no

Sí necesita nodos marcados, esto debido a que es posible que en la búsqueda se encuentre varias veces con una misma configuración de cartas, lo cual sería un ciclo y afectaría la ejecución.

3c) ¿Necesita verificar Agenda vacía? Argumente por qué si o por qué no

Sí necesita agenda vacía, esto debido a que puede llegar a una configuración de las cartas tal que no pueda llegar a la solución (ya que no puede haber ciclos).

3d) Determine el orden Θ de complejidad temporal y complejidad espacial de su solución

$T_A = \theta(|BusQ| * T_{sat} + |suc|)$

$S_A = \theta(|BusQ| + |suc|)$

3f) Determine una heurística h para el algoritmo.

$h(v) = N^2 - \#(i,j|0 \leq i,j < N: v[i][j] = false)$

3e) Determine si se puede establecer o no una función domino para el algoritmo.

No se puede establecer una función de efecto domino ya que toca revisar todos los caminos posibles y no hay manera de “descartar” los siguientes pasos por lo que sí es igual es la única solución toca revisarla hasta el final.