PONTIFICIA UNIVERSIDAD CATÓLICA DE CHILE
ESCUELA DE INGENIERÍA
DEPARTAMENTO DE CIENCIA DE LA COMPUTACIÓN

IIC2133 Estructuras de datos y algoritmos
1° **semestre 2019**

Solución Ayudantía Examen

Hashing

1. Cuando existe eliminación de los datos que se ingresan a la tabla de hash podrían haber problemas. Podríamos dejar de encontrar elementos en la tabla producto de la eliminación de uno de ellos. Este problema se puede solucionar agregando un *flag* a cada entrada de la tabla de hash, el cual indicará si en ella se eliminó o no un elemento. Esta solución no es muy recomendable en la práctica ya que podría aumentar en gran medida el tiempo de búsqueda de elementos dentro de la tabla. Es por esta razón que si se sabe de ante mano que habrá eliminación de algunas de las entradas que se ingresarán en la tabla, no se debe usar direccionamiento abierto.

2. A continuación se muestra la tabla luego de ingresar las claves. En general, el único método que podría presentar problemas es el doble hashing. Imaginemos ahora tuvieramos $h_2(k) = k\,\%(m-1)$, para el caso en que $k = 10$, por ejemplo, $h_2(10) = 0$ y la función de hash nos entregaría siempre el mismo valor, y en consecuencia no podríamos encontrar nunca una celda vacía con doble hashing.

	0	1	2	3	4	5	6	7	8	9	10
Encadenamiento	22 → 11 → 33	1	13 → 24					18		42 → 31	
Sondeo lineal	22	1	13	11	24	33		18		42	31
Doble hashing	22	1	13		11	18	31	24	33	42	

Figura 1: Tabla de hash

Árboles Rojo-Negro

1. El árbol resultante sería

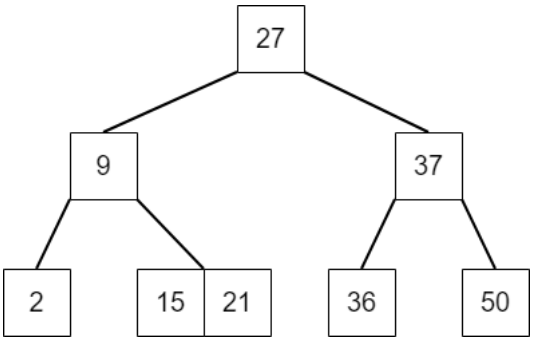

Figura 2: Árbol 2-3 resultante

2. El árbol resultante sería

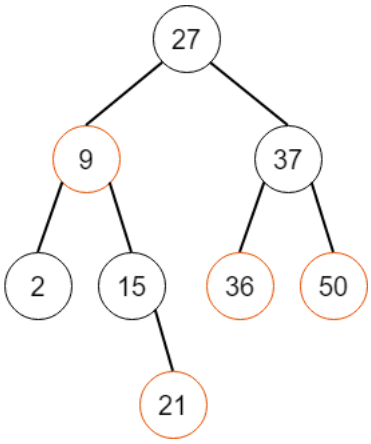

Figura 3: Árbol rojo-negro resultante

3. No es posible, dado que por definición, al ingresar un nuevo nodo este ingresa de color rojo (la única excepción a esto es cuando se ingresa el primer nodo, el cual es raíz también y debe ser negro).

   Ahora, para que el árbol resultante posea solo nodos negros, basta incluir eliminaciones en el proceso. Supongamos tenemos el árbol generado tras la inserción de las claves 5, 6, 4 y 3 (en ese orden). Si al árbol resultante le eliminamos el nodo 3, tendríamos un árbol con únicamente nodos negros.

Kosaraju

Kosaraju (algoritmo visto en clases) busca las SCC (componentes fuertemente conectadas) de un grafo. Para ello se debe ejecutar una vez DFS sobre el grafo original (G), almacenando tiempos de visita y finalización en cada uno de los nodos. Luego, se toma el grafo transpuesto (G') y se ejecuta DFS sobre este, pero se van tomando los nodos de mayor a menor tiempo de finalización. Este último DFS entregará un bosque de árboles en donde cada árbol corresponde a una SCC. Teniendo esto en cuenta, el algoritmo que determina si G es semiconectado es el que se muestra a continuación

Algorithm 1 : SEMI-CONNECTED(G)

Ejecutar una vez Kosaraju, para encontrar el grafo de componentes, llamémoslo C.
Encontrar el orden topológico de C. Esto se puede realizar utilizando el algoritmo visto en clases (*topsort*). Supongamos el orden obtenido de los nodos es $n_1, n_2, ..., n_k$.
for $i = 1, ..., k$ **do**
  if *no hay arista desde* n_i *hasta* n_{i+1} **then**
    return FALSE
  end
end
return TRUE

TopSort

Para resolver este problema se puede utilizar tanto el algoritmo visto en clases como el algoritmo de Kahn (visto en la ayudantía), el cual básicamente consiste en ir agregando al orden topológico aquellos nodos que no poseen aristas entrantes y cada vez que agregamos dicho nodo borramos sus aristas salientes. Un posible orden topológico para el grafo es $I \to E \to K \to B \to F \to C \to A \to H \to D \to J \to G$

Grafos

1. Supongamos tenemos u, v que pertenecen a una de las componentes conexas del grafo. Entonces, por definición de SCC, sabemos que existe un camino que va de u a v, y existe otro camino que va de v a u. Ahora, la unión de estos dos caminos dará origen a un ciclo. Al tener un ciclo en el grafo, no es posible tener un orden topológico (esto por definición de orden topológico, ya que no es posible tener un ordenamiento lineal de los nodos).

2. Supongamos que no existiese un orden topológico en G'. La única forma de que no exista un orden topológico en un grafo es cuando hay ciclos. Entonces, agregaremos una arista que vaya desde el nodo D hasta el EF (sin pérdida de generalidad). Ahora existe un ciclo, pero también disminuye el número de componentes conexas, ya que se cumple $d \rightsquigarrow e \wedge e \rightsquigarrow d, d \in D, e \in EF$. Luego tenemos solo dos componentes conexas: ABC y DEF, llegando a una contradicción.