TABLOIDE DE ÁRBOLES Y CAMINO MÍNIMO DE GRAFOS

CON SOLUCIONES Y EXPLICACIONES

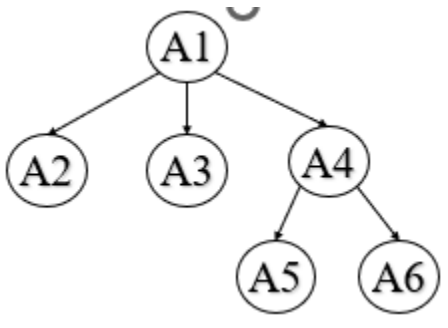
Prof. Leonardo Fundora Luis

Ingeniería Informáica, 2do año, CPE

PREGUNTA TIPO: Teoría

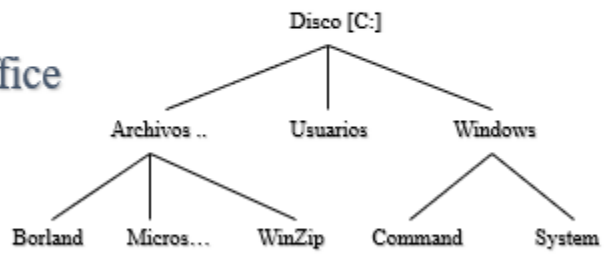
AFIRMACIONES SOBRE ÁRBOLES

Un árbol es una colección de elementos, uno de los cuales es distinguido como raíz. Entre ellos existe una relación (“paternidad”) que define una estructura jerárquica.

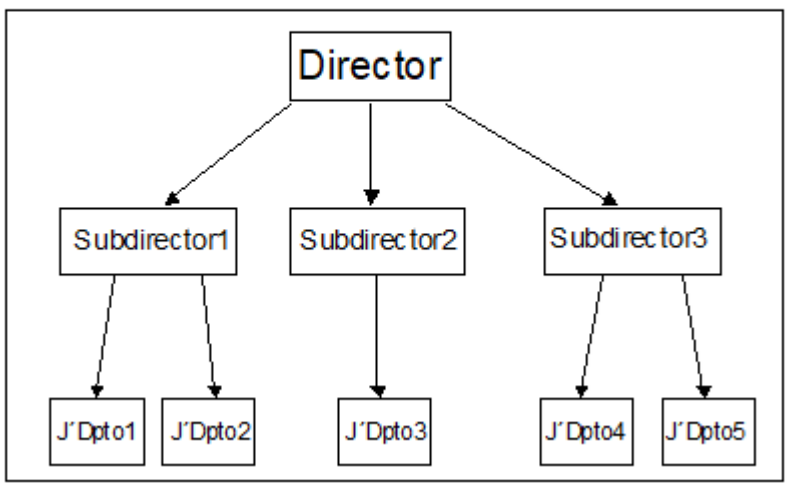


Definir un TDA para representar la siguiente estructura de directorios.

Disco C:
- Archivos de programa
  - Borland
  - Microsoft Office
  - WinZip
- Usuarios
- Windows
  - Command
  - System



Definir un TDA para representar la siguiente estructura de empleados de una empresa.



Un elemento por sí mismo es un árbol. Este elemento se considera también la raíz del árbol.

Un caso particular de árbol es el árbol nulo o vacío, un árbol sin elementos.

Especificaciones de los árboles:

- Sólo la raíz no tiene un elemento padre. Los elementos restantes tienen padre único.
- Para todo elemento k, distinto de la raíz, existe una única secuencia de la forma:

  k0, k1, k2, k3, ..., kn; k0 = raiz; kn = k

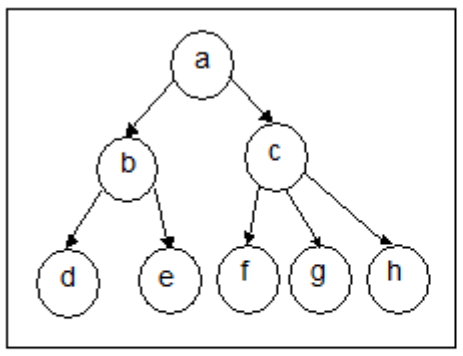
  con n >= 1, donde ki es hijo de ki-1, para

  1 <= i <= n.

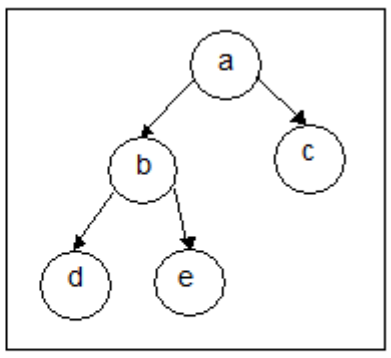
- Cada elemento ki de la secuencia es la raíz de otro subárbol.

Definiciones:

- *Grado de un elemento* cantidad de hijos.
- *Grado de un árbol* mayor de los grados de todos sus elementos.
- *Elemento hoja* elemento sin hijos.
- *Elemento rama* elemento que tiene hijos, o sea, la raíz de un subárbol.
- *Nivel de un Elemento* nivel de su padre más uno. Por definición, la raíz del árbol tiene nivel 0.
- *Árbol completo de nivel n* árbol en el que cada elemento de nivel n es una hoja y cada elemento de nivel menor que n tiene, al menos, un subárbol no vacío.



Árbol completo de nivel 2

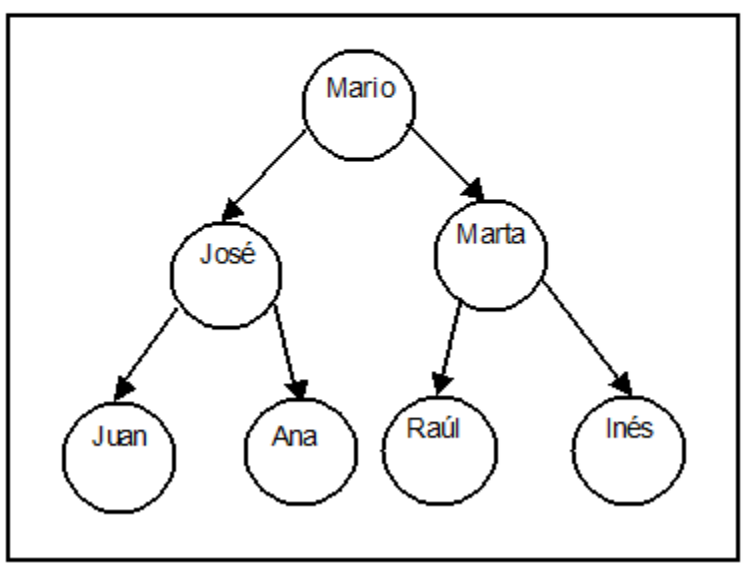


Árbol no completo de nivel 2

- *Padre de un elemento* elemento raíz del subárbol más pequeño que contiene a dicho elemento y en el cual él no es raíz.
- *Hijo de un elemento* elemento raíz de uno de los subárboles.
- *Predecesor de un elemento* elemento que le antecede en un recorrido del árbol.
- *Hermano de un elemento* otro elemento hijo de su padre.
- *Árbol ordenado* árbol para el que se considera el orden relativo de los subárboles de cualquier elemento.

Ejemplo de árbol ordenado

Árbol genealógico de una persona



- *Altura de un elemento* en un árbol es 1 + la altura del hijo que mayor altura posea. Por definición la altura de un elemento hoja es 0. La altura de un árbol es la altura del elemento raíz.
- *Profundidad(nivel) de un elemento* 1 + la profundidad del padre. Por definición la profundidad del elemento raíz es 0.

Recorridos en árboles

Existen diferentes formas de ordenar todos los nodos de un árbol de forma secuencial.

Los tres más importantes: **preOrden**, **postOrden** e **inOrden** (simétrico). Estos ordenamientos se definen recursivamente de la siguiente forma:

Si el árbol T es nulo, entonces la lista vacía ([]) es el preOrden, postOrden e inOrden de T.

Si T está formado por un único nodo n, entonces [n] (lista formada por n) es el preOrden, postOrden e inOrden de T.

En otro caso, si **T** es un árbol con raíz ***n*** y subárboles ordenados izquierda-derecha $T_1, T_2, \ldots T_k$.

preOrden

La lista en *preOrden* (o recorrido en *preOrden* o *preOrden(T)*) de **T** es la lista formada por el nodo raíz **n** seguido de

preOrden(T1), preOrden(T2), ..., preOrden(Tk)

([n]+*preOrden(T1)+preOrden(T2)+...+ preOrden(Tk)*).

postOrden

La lista en *postOrden* (o recorrido en *postOrden* o *postOrden(T)*) de **T** es la lista formada por *postOrden(T1),postOrden(T2),...,postOrden(Tk)* seguido del nodo raíz **n** (*postOrden(T1)+postOrden(T2)+...+postOrden(Tk)* +[n]).

inOrden

La lista en *inOrden* (o recorrido en *inOrden* o *inOrden(T)*) de **T** es la lista formada por *inOrden(T1)*, seguido del nodo raíz **n** y a continuación *inOrden(T2),...,inOrden(Tk)* (*inOrden(T1)*+[n]+*inOrden(T2)*+...+ *inOrden(Tk)*).

Árbol binario

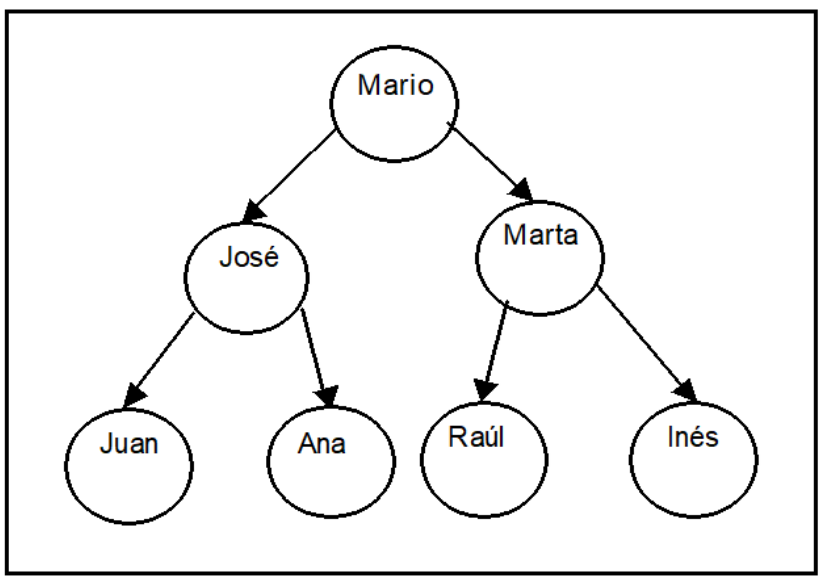
Definición: Un árbol binario (en inglés *binary tree*) es un árbol ordenado de, a lo sumo, grado 2.

Aclaraciones:

- A lo sumo grado 2: Cada elemento tiene a lo sumo dos hijos.
- Ordenado: importa el orden de los hijos, es decir, será necesario especificar de cada elemento cuál es el hijo izquierdo y cuál el hijo derecho.

Definición Recursiva:

- El árbol nulo es un árbol binario
- Un árbol binario está formado por un valor(raíz) un hijo izquierdo que es un árbol binario, y un hijo derecho que es un árbol binario.



RECORRIDOS Y CAMINO MÍNIMO EN EL GRAFO

Algoritmos greedy: También conocidos como algoritmos ávidos, glotones o voraces. Estos algoritmos trabajan por etapas, tomando en cada una de ellas la solución mejor (óptimo local) sin considerar las consecuencias futuras. El óptimo encontrado en una etapa puede posteriormente ser modificado si surge una solución mejor.

Algoritmo de Dijkstra

El algoritmo de Dijkstra, también llamado algoritmo de caminos mínimos, es un algoritmo para la determinación del camino más corto, dado un vértice origen, hacia el resto de los vértices en un grafo que tiene pesos en cada arista. La idea subyacente en este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen hasta el resto de los vértices que componen el grafo, el algoritmo se detiene.

```
función Dijkstra (Grafo G, nodo_salida s)
  //Usaremos un vector para guardar las distancias del nodo salida al resto
  entero distancia[n]
  //Inicializamos el vector con distancias iniciales
  booleano visto[n]
  //vector de boleanos para controlar los vértices de los que ya tenemos la distancia mínima
  para cada w ∈ V[G] hacer
     Si (no existe arista entre s y w) entonces
         distancia[w] = Infinito //puedes marcar la casilla con un -1 por ejemplo
     Si_no
         distancia[w] = peso (s, w)
     fin si
  fin para
```

```
  distancia[s] = 0
  visto[s] = cierto
  //n es el número de vértices que tiene el Grafo
  mientras que (no_estén_vistos_todos) hacer
     vértice = tomar_el_mínimo_del_vector distancia y que no esté visto;
     visto[vértice] = cierto;
     para cada w ∈ sucesores (G, vértice) hacer
         si distancia[w]>distancia[vértice]+peso (vértice, w) entonces
            distancia[w] = distancia[vértice]+peso (vértice, w)
         fin si
     fin para
  fin mientras
fin función.
```

Algoritmo de Floyd

Principio

Principio de Optimalidad de Bellman: Dada una secuencia óptima de decisiones, toda subsecuencia de ella es, a su vez, óptima. Un camino entre dos vértices será óptimo si sus caminos intermedios son óptimos también. Sean v y w dos vértices tales que el camino mínimo de v hacia w pasa por cierto vértice u. Por tanto: CostoCMin(v,w) = CostoCMin(v,u) + CostoCMin(u,w)

Características

- Cuenta con un bucle que examina cada vértice y lo toma como pivote.
- Cuenta con una matriz de pesos C.
- En cada iteración se actualiza C para todo par de vértices v y w, mediante la fórmula C[v, w] = min(C[v, w], C[v, u] + C[u, w]) donde u es el vértice pivote

```
1 /* Suponemos que la función pesoArista devuelve el coste del camino que va de i a j
2    (infinito si no existe).

3 También suponemos que      es el número de vértices y pesoArista(i,i) = 0
4 */
5
6 int camino[][];
7 /* Una matriz bidimensional. En cada paso del algoritmo, camino[i][j] es el camino mínimo
8 de i hasta j usando valores intermedios de (1..k-1). Cada camino[i][j] es inicializado a
9
10 */
11
12 procedimiento FloydWarshall ()
13    para k: = 0 hasta n - 1
14
15          camino[i][j] = mín ( camino[i][j], camino[i][k]+camino[k][j])
16
17    fin para
```