Árboles

Técnicas de Diseño de Algoritmos (Ex Algoritmos y Estructuras de Datos III)

Primer cuatrimestre 2024

Árboles

Definiciones:

- Un **árbol** es un grafo conexo sin circuitos simples.
- Una arista e de G es **puente** si $G - e$ tiene más componentes conexas que G.
- Un vértice v de G es **punto de corte** o **punto de articulación** si $G - v$ tiene más componentes conexas que G.

Árboles

Teorema: Dado un grafo $G = (V, X)$ son equivalentes:

1. G es un árbol.
2. G es un grafo sin circuitos simples, pero si se agrega una arista e a G resulta un grafo con exactamente un circuito simple, y ese circuito contiene a e.
3. Existe exactamente un camino entre todo par de nodos.
4. G es conexo, pero si se quita cualquier arista a G queda un grafo no conexo (toda arista es puente).

Lema 1: Sea $G = (V, X)$ un grafo conexo y $e \in X$. $G - e$ es conexo si y solo si e pertenece a un circuito simple de G.

Árboles

Definición:

- Una **hoja** es un nodo de grado 1.

Lema 2: Todo árbol no trivial tiene al menos dos hojas.

Lema 3: Sea $G = (V, X)$ árbol. Entonces $m = n - 1$.

Corolario 1: Sea $G = (V, X)$ sin circuitos simples y c componentes conexas. Entonces $m = n - c$.

Corolario 2: Sea $G = (V, X)$ con c componentes conexas. Entonces $m \geq n - c$.

Árboles

Teorema: Dado un grafo G son equivalentes:

1. G es un árbol.
2. G es un grafo sin circuitos simples y $m = n - 1$.
3. G es conexo y $m = n - 1$.

Árboles enraizados

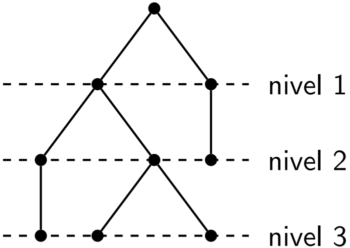
- Un *árbol enraizado* es un árbol que tiene un vértice distinguido que llamamos *raíz*.
- Explícitamente queda definido un árbol dirigido.
- El *nivel* de un vértice es la distancia de la raíz a ese vértice.
- La *altura* h de un árbol enraizado es el máximo nivel de sus vértices.
- Los vértices *internos* de un árbol enraizado son aquellos que no son ni hojas ni la raíz.
- Un árbol enraizado se dice *m-ario* si todos sus vértices internos tienen grado a lo sumo $m+1$ y su raíz a lo sumo m.
- Un árbol enraizado se dice *exactamente m-ario* si todos sus vértices internos tienen grado $m+1$ y su raíz m.

Árboles enraizados

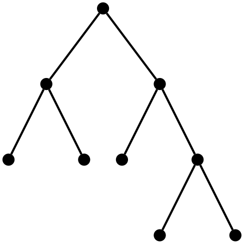
- Un árbol se dice *balanceado* si todas sus hojas están a nivel h o $h-1$.
- Un árbol se dice *balanceado completo* si todas sus hojas están a nivel h.
- Decimos que dos vértices adyacentes tienen *relación padre-hijo*, siendo el padre el vértice de menor nivel.

¿Cuántos nodos tiene en total un árbol exactamente m-ario que tiene i nodos internos?

Árboles enraizados

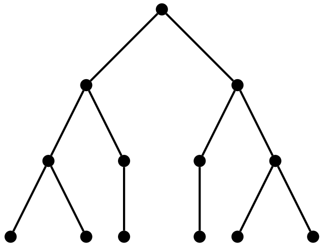


altura 3
2-ario
no exáctamente 2-ario

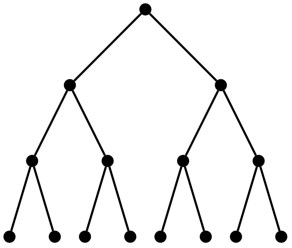


exáctamente 2-ario
balanceado
no balanceado completo

Árboles enraizados



2-ario
balanceado completo



exáctamente 2-ario
balanceado completo

Árboles enraizados

Teorema:

- Un árbol m-ario de altura h tiene a lo sumo m^h hojas. Alcanza esta cota si es un árbol exactamente m-ario balanceado completo con $h \geq 1$.
- Un árbol m-ario con l hojas tiene $h \geq \lceil \log_m l \rceil$.
- Si T es un árbol exactamente m-ario balanceado no trivial entonces $h = \lceil \log_m l \rceil$.

Árboles generadores

Definición:

- Un **árbol generador** (AG) de un grafo G es un subgrafo generador (que tiene el mismo conjunto de vértices) de G que es árbol.

Teorema:

- Todo grafo conexo tiene (al menos) un árbol generador.
- G conexo. G tiene un único árbol generador $\iff$ G es árbol.
- Sea $T = (V, X_T)$ un AG de $G = (V, X)$ y $e \in X \setminus X_T$. Entonces $T' = T + e - f = (V, X_T \cup \{e\} \setminus \{f\})$, con f una arista del único circuito de $T + e$, T' es árbol generador de G.

Recorrido de árboles, grafos o digrafos

- En muchas situaciones y algoritmos, dado un árbol (grafo o digrafo), queremos recorrer sus vértices exáctamente una vez.
- Para hacerlo de una forma ordenada y sistemática, podemos seguir dos órdenes:
  - **a lo ancho** (*Breadth-First Search - BFS*): se comienza por el nivel 0 (la raź) y se visita cada vértice en un nivel antes de pasar al siguiente nivel.
  - **en profundidad** (*Depth-First Search - DFS*): se comienza por la raíz y se explora cada rama lo más profundo posible antes de retroceder.

Recorrido de árboles, grafos o digrafos

```
recorrer(G)
    salida: pred[i] = padre de v_i, orden[i] = numero asignado a v_i
    next ← 1
    r ← elegir un vertice como raiz
    pred[r] ← 0                                   (marcar vertice r)
    orden[r] ← next
    LISTA ← {r}
    mientras LISTA ≠ ∅ hacer
        elegir un nodo i de LISTA
        si existe un arco (i, j) tal que j ∉ LISTA entonces
            pred[j] ← i                           (marcar vertice j)
            next ← next + 1
            orden[j] ← next
            LISTA ← LISTA ∪ {j}
        sino
            LISTA ← LISTA \ {i}
        fin si
    fin mientras
    retornar pred y orden
```

Recorrido de árboles, grafos o digrafos

BFS y **DFS** difieren en el elegir:

- **BFS**: *LISTA* implementada como cola.
- **DFS**: *LISTA* implementada como pila.

Cuando recorremos los vértices de un grafo conexo G, los valores de *pred* implícitamente definen un AG de G. ¿Qué ocurre con grafo no conexo, digrafos?

Estos dos procedimientos son la base de varios algoritmos:

- para encontrar todas las componentes conexas de un grafo,
- para encontrar todas las componentes fuertemente conexas de un digrafo,
- los puntos de corte (y las aristas puentes) de un grafo conexo,
- determinar si un grafo o digrafo tiene ciclos,
- en el problema de flujo máximo,
- camino mínimo de un grafo o digrafo no pesado,
- encontrar vértices que están a distancia menor que k.

BFS para calcular distancia

- Agregar una matriz $dist$ de tamaño $n \times n$.
- Inicializar $dist[i,j] \leftarrow \infty$ si $i \neq j$ sino $dist[i,i] \leftarrow 0$ para $1 \leq i \leq n$.
- Después de $pred[j] \leftarrow i$ dentro de la función $recorrer(G)$, se debe agregar $dist[r,j] \leftarrow dist[r,i] + 1$ (en caso de grafos, agregar también $dist[j,r] \leftarrow dist[r,j]$).
- Aplicar BFS para cada raíz $r : 1 \leq r \leq n-1$ en caso de grafos y $1 \leq r \leq n$ para digrafos.

DFS con más info.

Si aplicamos DFS para enumerar todos los vértices de un digrafo, se pueden clasificar sus arcos en 4 tipos:

- **tree edges**: arcos que forman el bosque DFS.
- **backward edges**: van hacia un ancestro.
- **forward edges**: van hacia un descendiente.
- **cross-edges**: van hacia a otro árbol (anterior) del bosque o al otra rama (anterior) del árbol.

Para grafos, solamente existen aristas tree edges y back edges !!!

Incorporamos algunos elementos adicionales para brindar más info.:

- una variable timer que se inicializa en 0.
- 2 arreglos: *desde* y *hasta*, ambos de dimensión n y se inicializan con -1.
- cada vez que un nodo nuevo i ingresa a *lista* que es una pila, se incrementa 1 el timer y se asigna dicho valor a $desde[i]$.
- cada vez que un nodo i sale de *lista*, se incrementa 1 el timer y se asigna dicho valor a $hasta[i]$.

DFS con más info.

El intervalo $(desde[i], hasta[i])$ representa el lapso de tiempo que el nodo i estuvo en la $lista - pila$. Para cualquier par de estos intervalos pasa una de las dos situaciones siguientes:

- hay una relación de contención entre ellos
- son disjuntos

Similitud con una secuencias de paréntesis !!!

Podemos usar esto y el arreglo *pred* para determinar el tipo de un arco $i \to j$

- tree edges: si $i = pred[j]$.
- backward edge: si $(desde[j], hasta[j])$ contiene a $(desde[i], hasta[i])$.
- **forward edges**: si $(desde[i], hasta[i])$ contiene a $(desde[j], hasta[j])$ y $i \neq pred[j]$.
- **cross-edges**: si $hasta[j] < desde[i]$.

Aplicaciones con DFS

Detección de ciclos

Teorema: Dado un grafo (digrafo) G.

G tiene un ciclo (ciclo orientado) $\Leftrightarrow$ existe un backward edge en G.

Sort topológico

Ordenar los nodos de acuerdo a su valor en el arreglo *hasta* de mayor a menor. ¿Funciona? ¿Realmente hay que ordenar?

Determinar las componentes fuertemente conexas de un digrafo

Determinar los puntos de cortes y las aristas puentes de un grafo