Arboles

Dr. J.B. Hayet

CENTRO DE INVESTIGACIÓN EN MATEMÁTICAS

Noviembre 2009

Outline

1. Arboles
2. Propiedades de arboles binarios
3. Recorridos

Previously en la clase

- Las recursiones y la estructura de arboles están fundamentalmente ligadas.
- Dos estrategias recursivas clásicas: DaC y programación dinámica.
- Vamos a ver mas elementos sobre estructuras de arboles.

Outline

Arboles

Arboles son un concepto muy corriente de organización de datos:

- Arboles genealógicos, arboles de torneos.
- Organización jerárquica en empresas.
- Temario de un libro.
- Organización de los archivos en una computadora.

Modelo abstracto de una estructura jerárquica.

Arboles, definiciones

- Un vértice, o nodo, es un objeto simple, que puede estar designado por un nombre, y llevar eventualmente información.
- Un arista es una conexión entre dos vértices.
- Un camino es una lista de vértices distintos en que vértices sucesivos están conectados por un arista.
- Un grafo no orientado es un conjunto de vértices y de aristas.
- Un árbol es un grafo no orientado tal que existe uno y uno solo camino entre dos nodos.

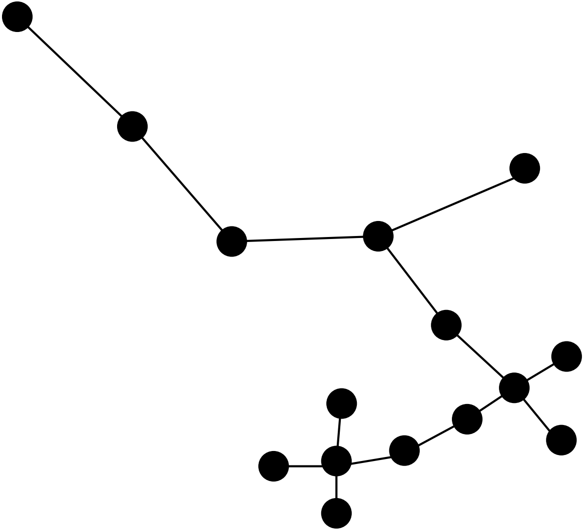
Arboles, definiciones

Para un grafo no orientado G de N nodos, hay equivalencia entre las siguientes proposiciones que definen un árbol:

- G tiene $N - 1$ aristas y no ciclos.
- G tiene $N - 1$ aristas y es conectado.
- G es conectado pero si se quita cualquier arista, ya no lo es.
- G es sin ciclos pero si se añade cualquier arista, ya tiene.
- entre dos nodos de G hay uno y uno solo camino conectándolos.

Arboles, definiciones

Arbol genérico:

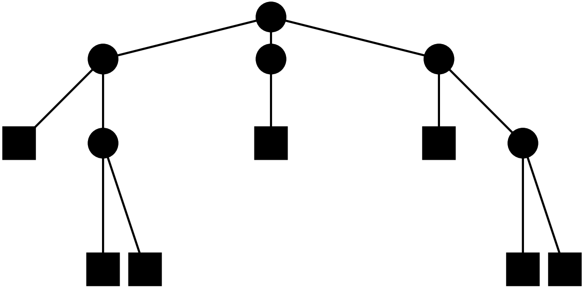


Arboles, definiciones

- Un árbol es enraizado cuando se distingue una raíz.
- La raíz puede:
  1. tener 0 nodos conectados, y en este caso es un nodo hoja (o terminal),
  2. tener un numero finito no nulo de nodos conectados, tal que cada uno es raíz de un sub-arbol.
- En general, en computación, árbol = árbol enraizado.
- Un nodo m es arriba (resp. abajo) de un nodo n si m se encuentra en el único camino que conecta la raíz a n (resp. n se encuentra en el único camino que conecta la raíz a m).
- Cada nodo (excepto la raíz) tiene exactamente un nodo justo arriba de el: su padre, y los que están justo abajo de el son sus hijos.
- Nodos hermanos (*sibling*), abuelos...

Arboles, definiciones

Arbol enraizado:



Arboles, definiciones

- El grado de un nodo es el numero de hijos que tiene.
- La profundidad de un nodo es el tamaño del camino de la raíz hasta este nodo.
- La altura de un nodo es el tamaño del camino mas largo que vaya de este nodo a un nodo hoja.
- La altura de un árbol es la altura de la raíz (o equivalentemente el máximo de las profundidades).
- Un árbol ordenado es un árbol enraizado en que se especifica el orden de los nodos hijos de cada nodo.
- Un conjunto ordenado de arboles ordenados es un bosque.

En computación, un árbol es generalmente un árbol ordenado.

Arboles, representación

El ADT Tree; operaciones sobre nodos individuales v

- Parent(v): regresa el padre de v, error si root.
- Children(v): regresa el conjunto de los niños de v.
- FirstChild(v): regresa el primer hijo (o un árbol vacío).
- LeftSibling(v): regresa el hermano precedente (o un árbol vacío).
- RightSibling(v): regresa el hermano siguiente (o un árbol vacío).
- IsLeaf(v): true si es un nodo hoja.
- IsInternal(v): true si es un nodo interno (no es hoja).
- IsRoot(v): true si es el nodo raíz.
- Depth(v): regresa la profundidad.
- Height(v): regresa la altura.
- Degree(v): regresa el grado.

Arboles, representación

El ADT Tree; operaciones sobre el árbol T

Size(): regresa el numero de nodos dentro de T.

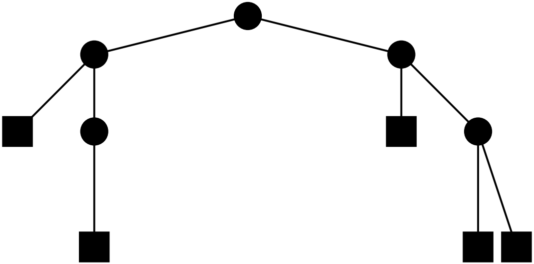
Root(): regresa el nodo raíz de T.

Height(): regresa la altura de T.

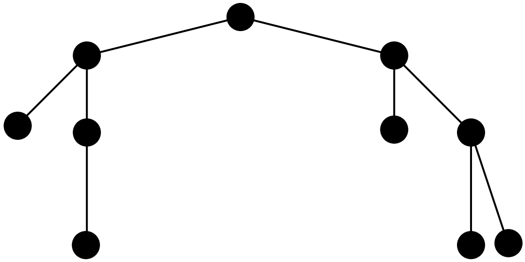
Arboles, definiciones

- Un árbol (estrictamente) M-ario es un árbol ordenado en que cada nodo n es de grado $g(n) = M$ o $g(n) = 0$. Se habla de nodos externos para nodos "especiales" que tienen cero hijo y sirven como referencia para completar hijos de nodos $g(n) < M$.
- Un árbol binario, es un árbol ordenado en que cada nodo n es de grado $g(n) = 2$ o $g(n) = 0$. Se habla de hijo izquierda y de hijo derecha.

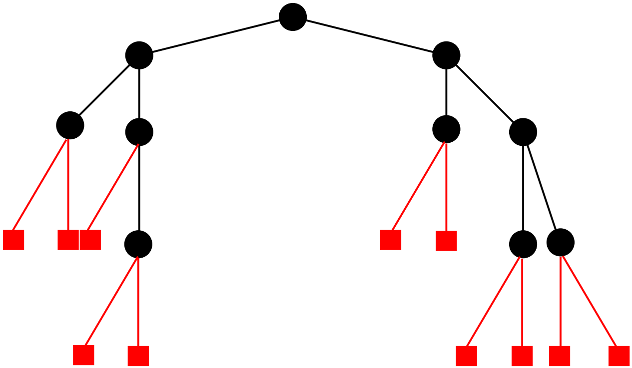
Arboles binario, definiciones



Arboles binario, definiciones



Arboles binario, definiciones



Arboles binario, aplicaciones

- Sirve como estructura de datos fundamental para problemas:
  - de implementación de filas de prioridad (montículos).
  - de estructuras adaptadas a las búsquedas (BSTs).
- Estudiarles ayuda a analizar los problemas con su estructura subyacente.

Arboles binario, definiciones

- Todo nodo interno tiene necesariamente dos hijos, izquierda y derecha.
- Recursivamente, se puede definir un árbol binario como :
  - o un nodo externo,
  - o un nodo interno conectado a un par de arboles binarios, el sub-arbol de izquierda y el sub-arbol de derecha.
- Concepto matemático abstracto, que puede ser representado de varias maneras:
  - estructura informática,
  - representación gráfica,
  - representación binaria: 111001010011001100100.

Arboles binario, definiciones

- Arbol binario completo: todos los niveles están completados por nodos internos, excepto el ultimo.
- Una propiedad importante para esos,

$$2^{h-1} < N + 1 \leq 2^h.$$

- Arbol binario perfecto: todas las hojas tienen una profundidad igual.

Outline

Arboles binario, definiciones

Es importante estudiar en particular los arboles binarios: existe una correspondencia 1-1 entre los arboles ordenados y los arboles binarios.

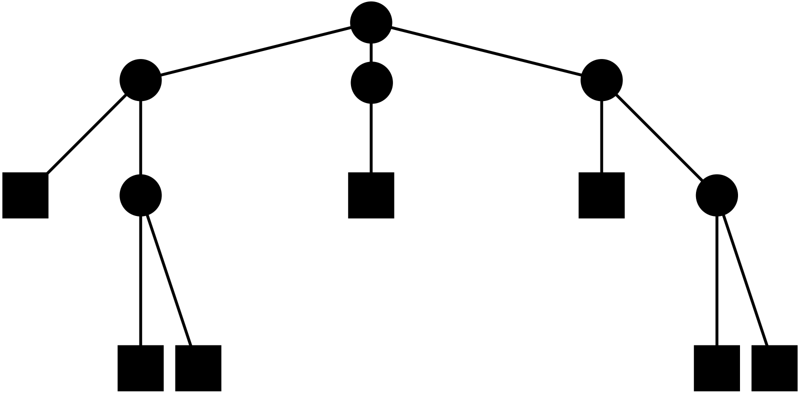
Se puede definir muy fácilmente un árbol binario en que cada nodo tiene por hijos su hijo izquierdo en el árbol genérico, su hermano a la derecha en el árbol genérico!

Arboles binario, definiciones

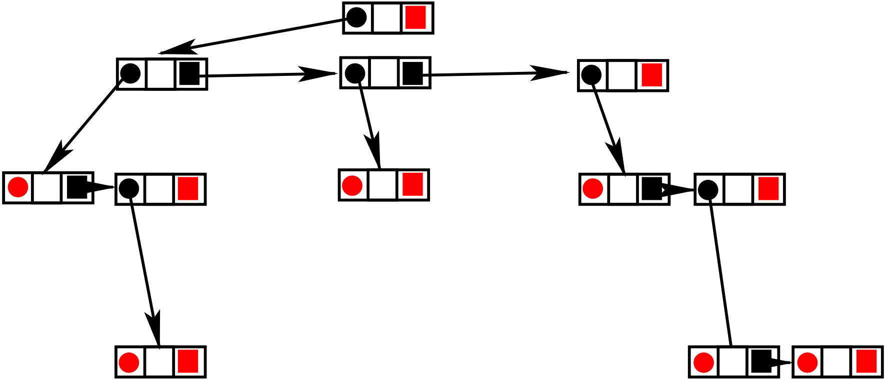
Es importante estudiar en particular los arboles binarios: existe una correspondencia 1-1 entre los arboles ordenados y los arboles binarios.

Se puede definir muy fácilmente un árbol binario en que cada nodo tiene por hijos su hijo izquierdo en el árbol genérico, su hermano a la derecha en el árbol genérico!

Arboles binario, definiciones



Arboles binario, definiciones



Arboles, representación

El ADT BinaryTree incluye todas las operaciones de Tree, a las que se añade

left(v): regresa el hijo izquierdo de v, error si externo.
right(v): regresa el hijo derecho de v, error si externo.

Arboles, representación

Representación informática:

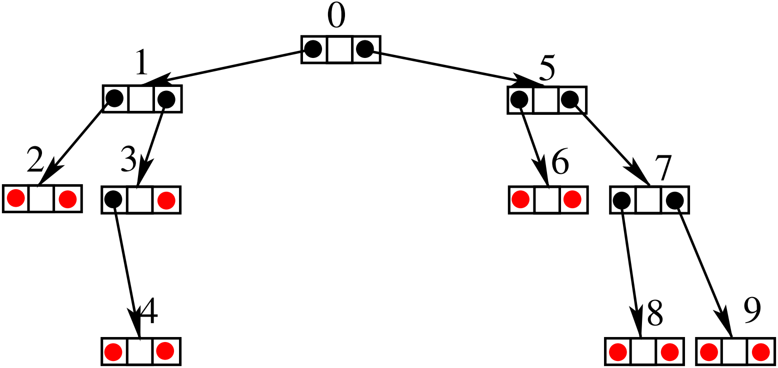
```
class Node {
 public:
   Node();
   Node(const Item &o);
   ...
 private:
  Item item;
  Node *l,*r;
};
typedef Node *Link;
```

Nodos con items y pares de apuntadores hacia nodos... Mover hacia el sub-arbol de derecha:

```
if (x!=NULL)
   x = x->getRight();
```

Arboles, representación

Para el árbol binario precedente:



Eficiente para representar las operaciones *top-down*. ¿Qué hacer si necesitamos usar **subir** el árbol?

Arboles, representación

Representación informática:

```
class Node {
 public:
   Node();
   Node(const Item &o);
   ...
 private:
  Item item;
  Node *l,*r,*p;
};
typedef Node *Link;
```

Similar a listas doblemente ligadas. . .

```
if (x!=NULL)
  x = x->getFather();
```

Arboles, representación

Representación informática:

- Generalización para arboles M-arios:
  - M ligas (apuntadores) explícitos ("left", "middle", "right"),
  - arreglo de M apuntadores.
- Generalización para cualquier árbol ordenado: usar contenedor dinámico (lista, vector, deque) para almacenar las ligas hacia los hijos.
- Arboles no ordenados? Representado con arboles ordenados o binarios, pero con problema de representación múltiple (y determinar que dos arboles ordenados diferentes corresponden al mismo árbol no ordenado).

Arboles binarios, representación

Representación informática: otra posibilidad es encodificar todas las legas (nodo-hijo, nodo-padre) en arreglos!

Nodo	0	1	2	3	4	5	6	7	8	9
Left	1	2	-1	4	-1	6	-1	8	-1	-1
Right	5	3	-1	-1	-1	7	-1	9	-1	-1
Parent	-1	0	1	1	3	0	5	5	7	7

Rápido, pero no muy flexible (destrucción de nodos,. . .). Ademas en general las operaciones son en función de apuntadores/referencias a nodos, no en términos de indices

Arboles binarios, representación

Representación informática: otra posibilidad es usar un arreglo de objetos de dimensión $2^{N+1}-1$ donde la N es la profundidad del árbol (funciona solo si N no varia) capa por capa.

	0	1	5	2	3	6	7	-	-	4	-	-	-	8	9

Ligas encodificadas por la estructura del arreglo:

- Hijos de $a[k]$ en $a[2k]$ y $a[2k+1]$.
- Todo por manipulación de índex: ¿padre? ¿profundidad?
- Pero, espacio desperdiciado: a usar en el caso que el árbol sí tiene propiedades de casi-completud (ejemplo: montículo).

Arboles binarios, más operaciones

Unas operaciones pueden llevar problemas:

- inserción o deleción de un nodo dentro de un árbol binario?

Otras que no:

- insertar un nodo abajo de un árbol binario,
- quitar una hoja,
- combinar dos arboles binarios y un nodo en un nuevo árbol binario,

y que se hacen en tiempo constante con estructuras ligadas.

Arboles binarios, propiedades

- un árbol binario con N nodos internos tiene $N+1$ nodos externos. ¿Prueba?
- un árbol binario con N nodos internos tiene $2N$ aristas, $N-1$ para los nodos internos, $N+1$ para los nodos externos. ¿Prueba?
- ¿cuántos arboles binarios posibles con N nodos internos?

$$\begin{cases} b_0 & = & 0 \\ b_n & = & \sum_{k=0}^{n-1} b_k b_{n-1-k} \sim \frac{4^n}{n\sqrt{n\pi}} \end{cases}$$

Números de Catalan.

Arboles binarios, propiedades

- un árbol binario con N nodos internos tiene $N+1$ nodos externos. ¿Prueba?
- un árbol binario con N nodos internos tiene $2N$ aristas, $N-1$ para los nodos internos, $N+1$ para los nodos externos. ¿Prueba?
- ¿cuántos arboles binarios posibles con N nodos internos?

$$\begin{cases} b_0 & = & 0 \\ b_n & = & \sum_{k=0}^{n-1} b_k b_{n-1-k} \sim \frac{4^n}{n\sqrt{n\pi}} \end{cases}$$

Números de Catalan.

Arboles binarios, propiedades

Para un árbol binario de N nodos internos, se puede definir esas dos cantidades:

- El largo de camino interno: suma de todos los largos de camino entre la raiz y los nodos internos, I_N.
- El largo de camino externo: suma de todos los largos de camino entre la raiz y los nodos externos, E_N.

Propiedad: $E_N = I_N + 2N$ (Prueba?)

Arboles binarios, propiedades

La altura a_N de un árbol binario de N nodos internos verifica:

$$\log N + 1 \leq a_N \leq N$$

Los dos casos extremos correspondiendo a arboles llenos (salvo la ultima linea) y a arboles degenerados lineales

Arboles binarios, propiedades

El largo de camino interno I_N verifica:

$$N \log \frac{N}{4} < I_N \leq \frac{N(N+1)}{2}$$

Esas propiedades son importantes para el uso concreto de las estructuras, y se nota que el caso que nos va a interesar mas es el de arboles equilibrados ya presente en unos algoritmos (MergeSort...)

Outline

Recorrer un árbol

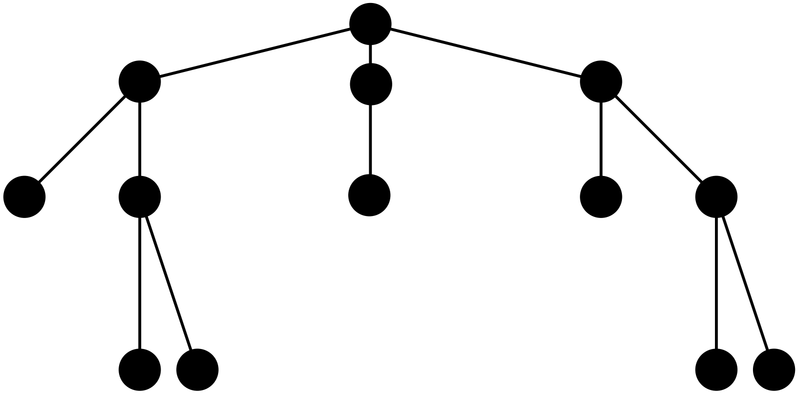
Como para toda estructura, el recorrido es la operación mas fundamental: se puede necesitar examinar todos los nodos del árbol para cumplir cierto tipo de operación.

Se puede definir al menos dos recorridos en el caso de arboles genéricos:

- recorrido pre-orden, en que el nodo corriente esta examinado y luego los sub-arboles están recorridos,
- recorrido post-orden, en que los sub-arboles están recorridos primero y luego el nodo corriente esta examinado.

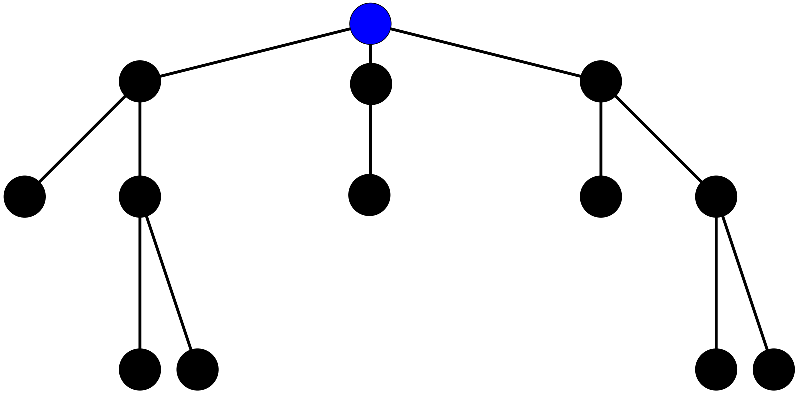
Recorrer un árbol

Recorrido pre-orden:



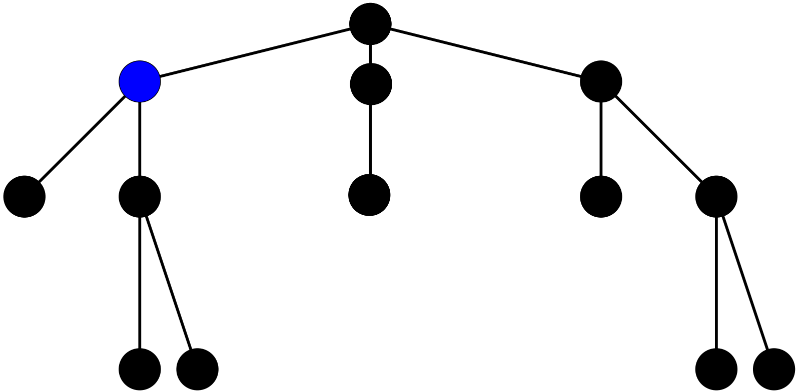
Recorrer un árbol

Recorrido pre-orden:



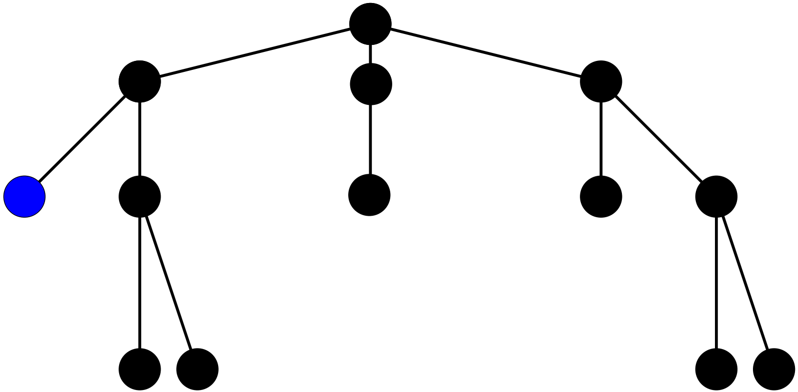
Recorrer un árbol

Recorrido pre-orden:



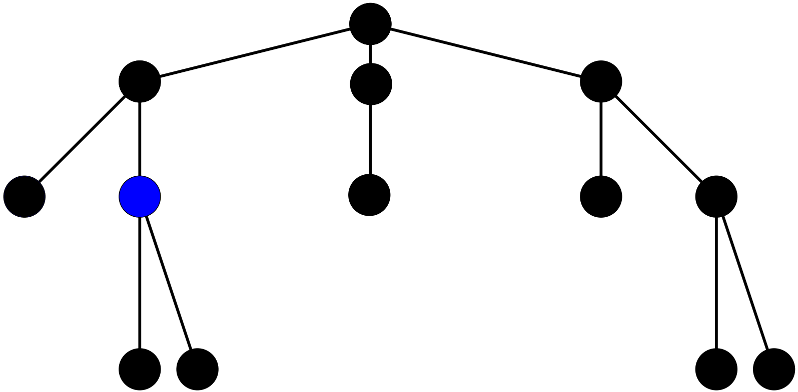
Recorrer un árbol

Recorrido pre-orden:



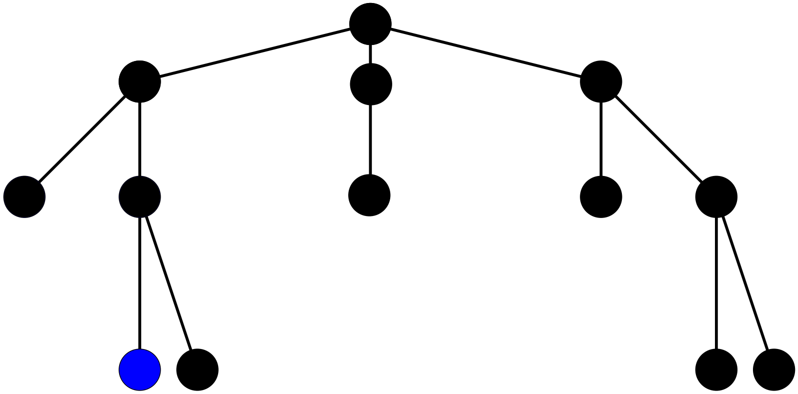
Recorrer un árbol

Recorrido pre-orden:



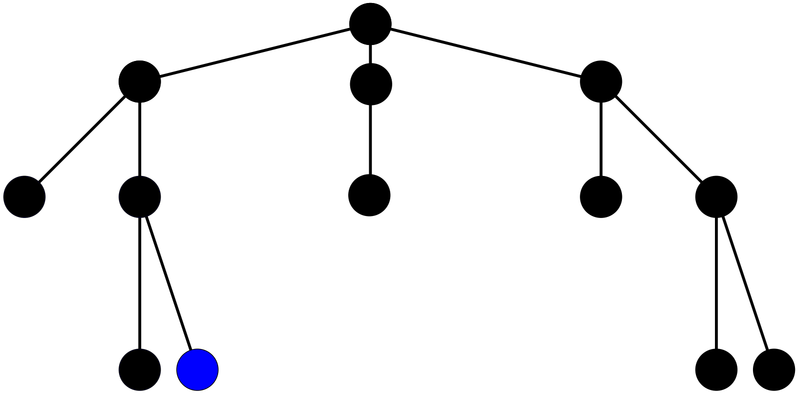
Recorrer un árbol

Recorrido pre-orden:



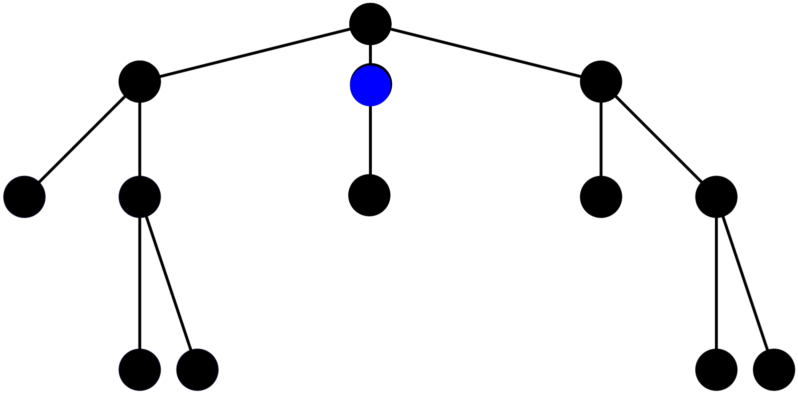
Recorrer un árbol

Recorrido pre-orden:



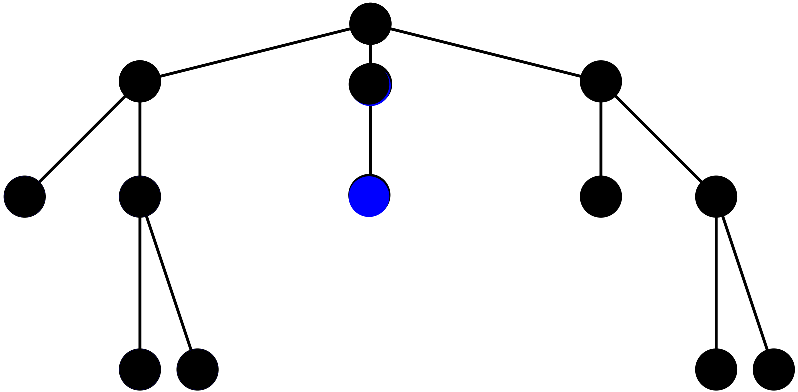
Recorrer un árbol

Recorrido pre-orden:



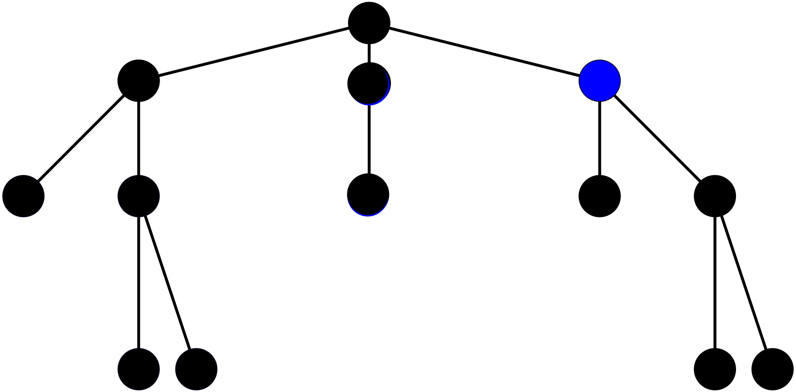
Recorrer un árbol

Recorrido pre-orden:



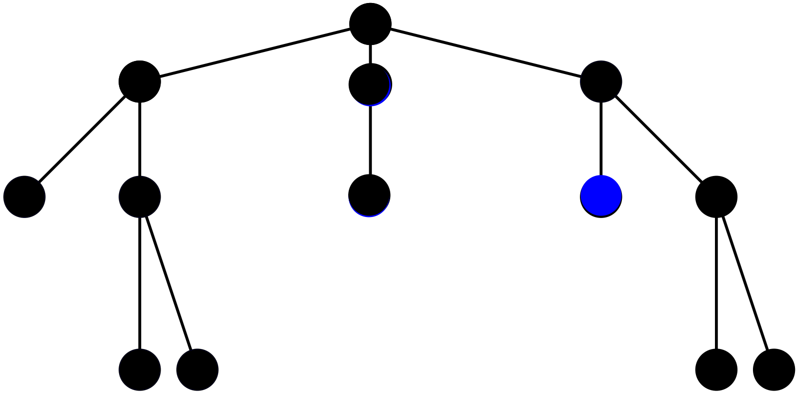
Recorrer un árbol

Recorrido pre-orden:



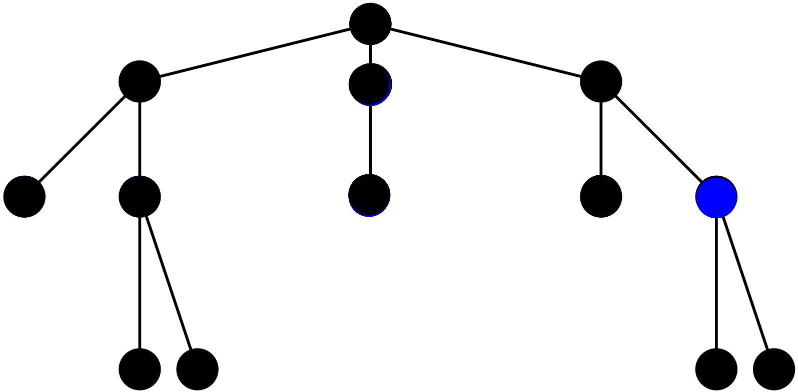
Recorrer un árbol

Recorrido pre-orden:



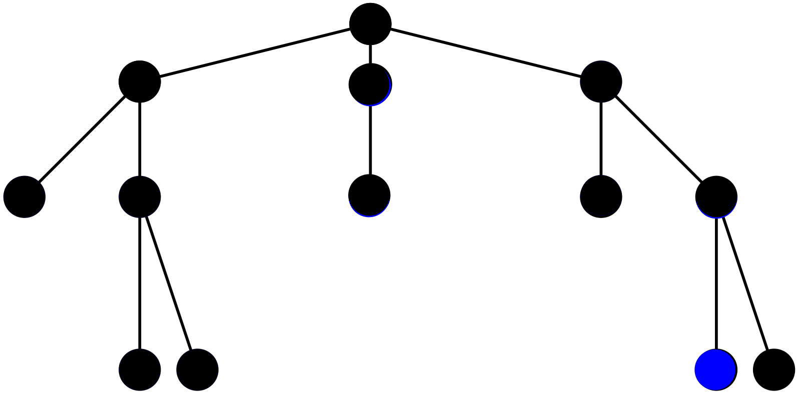
Recorrer un árbol

Recorrido pre-orden:



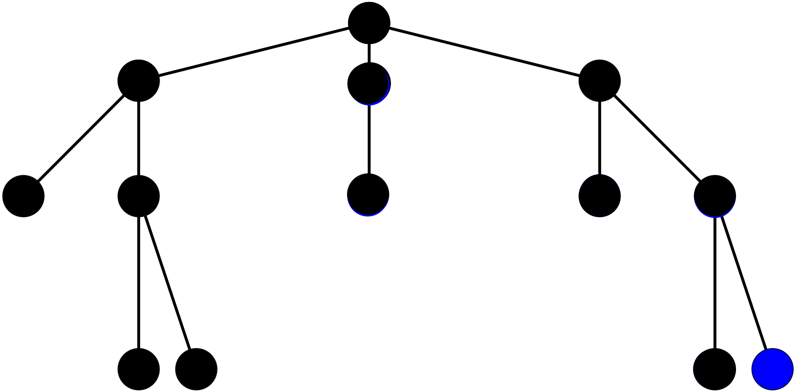
Recorrer un árbol

Recorrido pre-orden:



Recorrer un árbol

Recorrido pre-orden:



Recorrer un árbol

Recorrido pre-orden, caso binario:

```
void traverse(Link h, void visit(Link)) {
    if (h == NULL) return;
    visit(h);
    traverse(h->l, visit);
    traverse(h->r, visit);
  }
```

En el caso binario, se añade la posibilidad de visitar el nodo entre los dos recorridos del sub-arbol de izquierda y del de derecha: recorrido in-orden.

Recorrer un árbol

Similaridad con los problemas de divide and conquer!

```
void rule(int l, int r, int h) {
    int m = (l+r)/2;
    if (h > 0) {
        rule(l, m, h-1);
        mark(m, h);
        rule(m, r, h-1);
    }
}
```

- pre-orden: poner una marca al centro, procesar las dos sub-partes consecutivamente después,
- in-orden: poner todas las marcas de la izquierda a la derecha,
- post-orden: poner las dos marcas mas chicas rodeando una marca mas grande y seguir...

Recorridos en profundidad!

Recorrer un árbol

Por las razones que ya hemos encontrado antes, puede ser interesante evitar usar funciones recursivas para la implementación de tal recorrido; una manera de hacerla no-recursiva es remarcar que en cada rama que empieza a ser explorada, se "memoriza" que hay que terminar trabajo acá (por ejemplo examinando la rama de derecha).

Recorrer un árbol

¿Cuál recorrido esta implementado en ese código?

```
void traverse(Link h, void visit(Link)) {
 stack<Link> s(max);
 s.push(h);
 while (!s.empty()) {
        visit(h = s.top());
        s.pop();
        if (h->r != NULL) s.push(h->r);
        if (h->l != NULL) s.push(h->l);
      }
  }
```

Recorrer un árbol

Para el in-orden, un poco mas complicado porque no se tiene que procesar el nodo ahora sino después de haber puesto en la pila su hijo izquierda (y de facto todo el sub-arbol correspondiente) para un procesamiento prealable

```
void traverse(Link h, void visit(Link)) {
 stack<LinkAndState> s(max);
 s.push(LinkAndState(h,0));
 while (!s.empty()) {
        LinkAndState hs = s.top();
        s.pop();
        if (hs.s==0) {
                s.push(LinkAndState(hs.h,1));
                if (hs.h->l != NULL)
                   s.push(LinkAndState(hs.h->l,0));
         } else {
                visit(hs.h);
                if (hs.h->r != NULL)
                   s.push(LinkAndState(hs.h->r,0));
        }
      }
  }
```

Recorrer un árbol

Dejo el post-orden en ejercicio pero el principio es similar: arreglarse para poner a procesar el nodo en la pila, luego los hijos izquierda y derecha (que tienen que estar procesados antes!) hasta llegar a nodos externos.

Esos recorridos se generalizan (excepto el in-orden) a arboles ordenados.

Costo en memoria: proporcional a la altura del arbol.

Recorrer un árbol

Alternativamente, se puede recorrer el árbol en el sentido de la anchura, nivel por nivel

Este recorrido puede estar expresado, no-recursivamente, usando una fila: procesas la raíz, añades a la fila los hijos, saldrán después en ese orden; luego para cada de los hijos añades los nietos. . .

Recorrer un árbol

Estructura de programa similar, pero recorrido muy diferente!

```
void traverse(Link h, void visit(Link)) {
    queue<Link> q(max);
    q.put(h);
    while (!q.empty()) {
        visit(h = q.get());
        if (h->l != 0) q.put(h->l);
        if (h->r != 0) q.put(h->r);
      }
  }
```

- Complejidad en espacio?
- Pila: Profundidad.
- Fila: Anchura.

Recorrer un árbol

- En una implementación C++, lo ideal sería proponer iteradores que implementen uno u otro de los recorridos.
- No hay contenedores "públicos" en la STL para arboles (aunque por ejemplo Map esta basado en un tipo particular de árbol equilibrado).

Recorrer un árbol

Cálculos sobre un árbol:

```
int count(Link h) {
    if (h == 0) return 0;
    return count(h->l) + count(h->r) + 1;
  }
int height(Link h) {
    if (h == 0) return -1;
    int u = height(h->l), v = height(h->r);
    if (u > v) return u+1; else return v+1;
  }
```

Construir un árbol

Para notación polaca

```
char *a; int i;
struct Node {
    Item item; Node *l, *r;
    Node(Item x) { item = x; l = NULL; r = NULL; }
};
typedef Node* Link;
Link parse() {
    char t = a[i++]; Link x = new Node(t);
    if ((t == '+') || (t == '*')) {
        x->l = parse(); x->r = parse();
    }
    return x;
}
```