Programación 2

La previa
Estructuras Arborescentes
(parte 2)

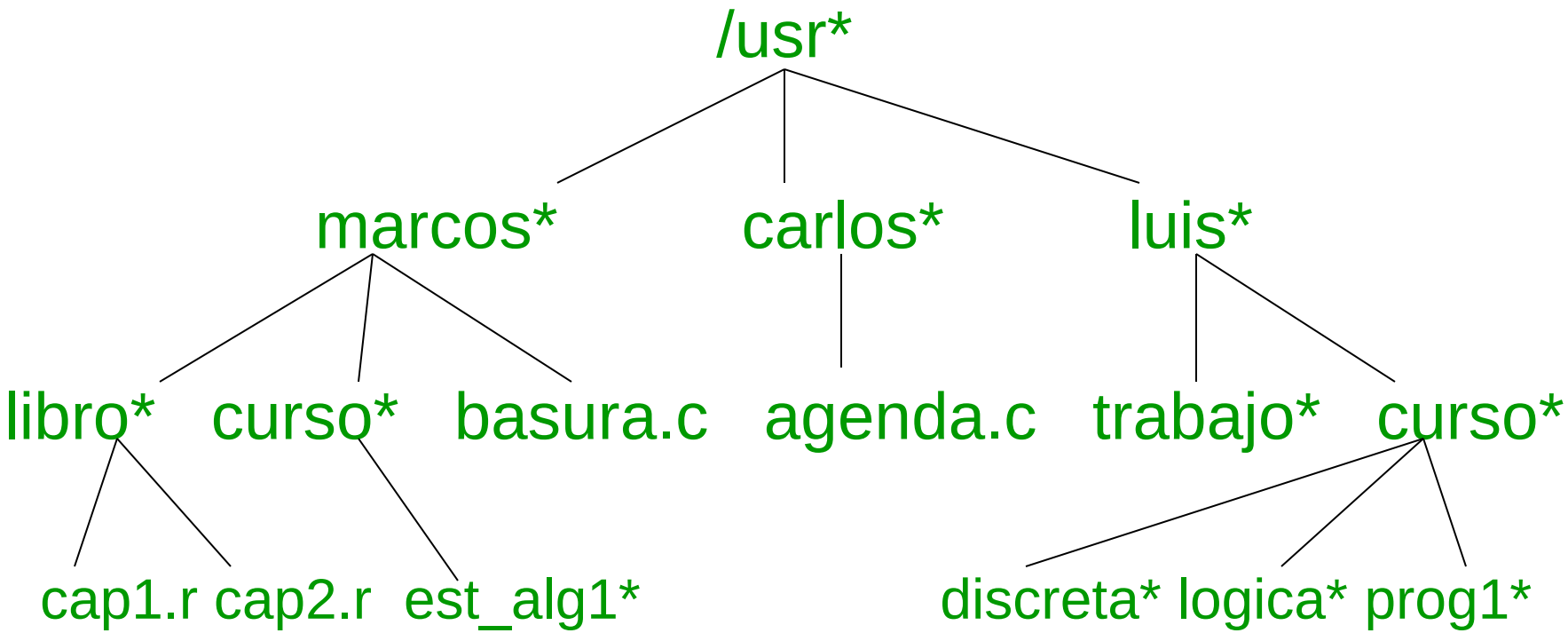
Arboles Generales.
Cómo implementarlos?

Una estructura *árbol* (*árbol general o finitario*) con tipo base T es,

1. O bien la estructura vacía
2. O bien un nodo de tipo T, llamado raíz del árbol, junto con un número finito de estructuras de árbol, de tipo base T, disjuntas, llamadas *subárboles*

¿cómo representamos árboles generales?

Un ejemplo de árbol general: “estructura de directorios”

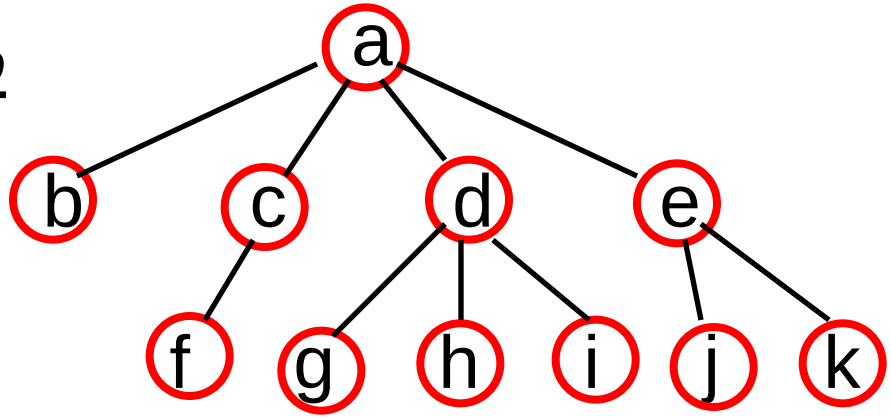


Una aplicación: listar un directorio

(recorridos de árboles)

Arbol general - Arbol binario

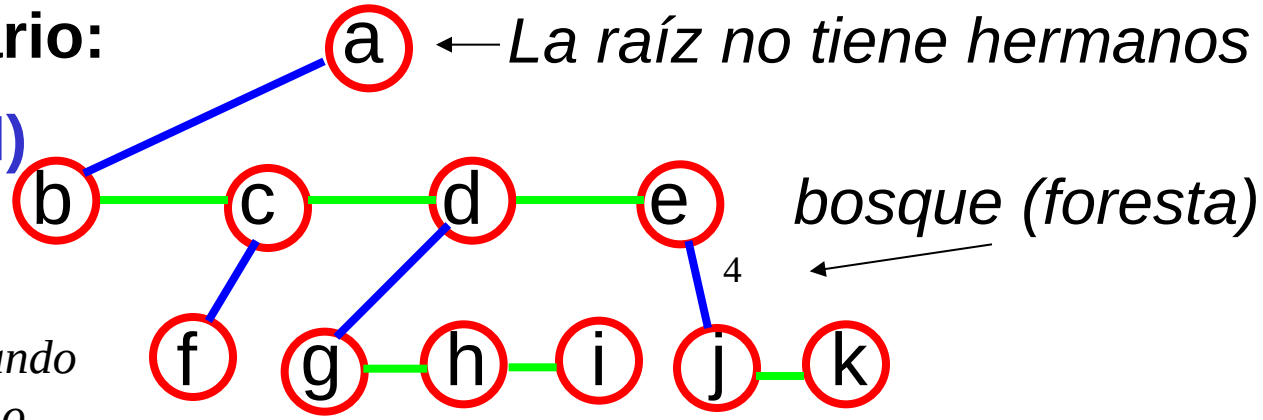
Cada nodo tiene un núnero *finitio* de subárboles



. . . . . .

Representación como árbol binario:

*** primer hijo (pH)**
*** siguiente hermano (sH)**

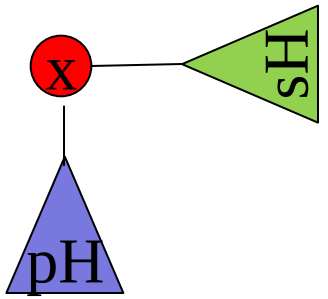


. . . . . .

Notar que estamos representando tanto bosques de árboles como árboles (bosques de un sólo árbol; la raíz).

Ejemplo 1

- La función contar nodos de un árbol general (recordar que la raíz no tiene hermanos).



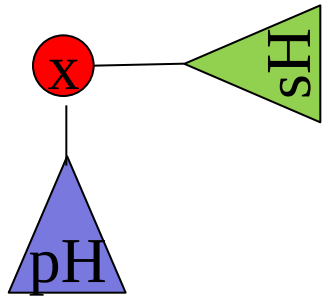

```
typedef NodoAG* AG;
struct NodoAG { T item; AG pH; AG sH; };

int nodos(AG t){
   if (t == NULL) return 0;
   else return nodos(t->pH)+nodos(t->sH)+1;
}
```

Notar que:

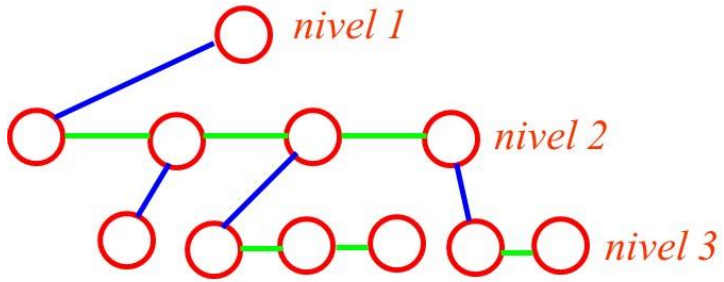
- **el código es idéntico al de contar nodos en un árbol binario tradicional.**
- **si se invoca a *nodos* con un bosque *t* (*t->sH!=NULL*), se tiene la cantidad de nodos del bosque.**

Ejemplo 2



- La función altura de un árbol general (recordar que la raíz no tiene hermanos).

```
int altura(AG t){
   if (t == NULL) return 0;
   else return MAX(1+altura(t->pH),
                   altura(t->sH));
}
```



Notar que el código NO es idéntico al de la altura de un árbol binario tradicional:

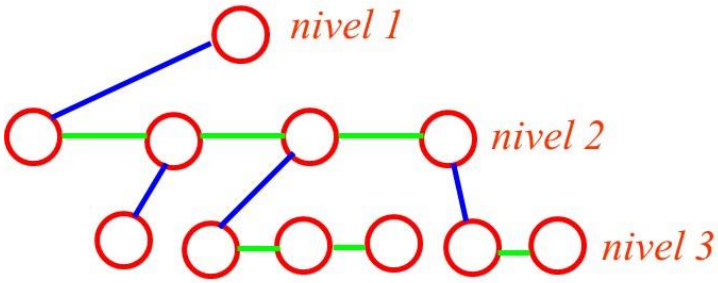
- "primer hijo" (**pH**) aumenta la altura.
- "siguiente hermano" (**sH**) no aumenta la altura, la mantiene (ver dibujo).

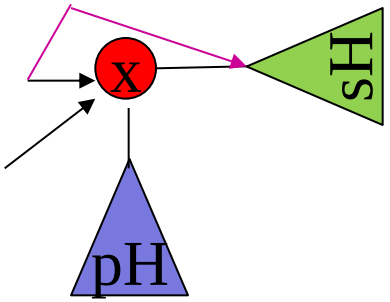
Si se invoca a la función *altura* con un bosque *t* (*t->sH!=NULL*), se tiene la altura del bosque.

Ejemplo 3

- Elimina cada subárbol que tiene a x como raíz de un árbol general (recordar que en la representación *pH-sH*, el nodo raíz principal no tiene hermanos).

...

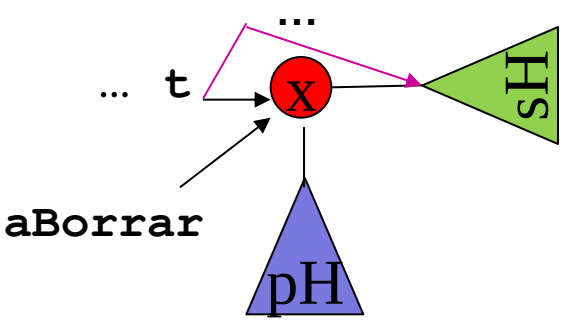




Ejemplo 3

- Elimina cada subárbol que tiene a x como raíz de un árbol general (recordar que en la representación *pH-sH*, el nodo raíz principal no tiene hermanos).

```
void elim(AG & t, T x){
   if (t != NULL){
        if (t->item != x){
               elim(t->pH, x);
               elim(t->sH, x);
        }else{ AG aBorrar = t;
               t = t->sH;
               elimTodo(aBorrar->pH);
               delete aBorrar;
               elim(t, x);
        }
   }
}
```




```
void elimTodo(AG & t){
  if (t != NULL){
    elimTodo(t->pH);
    elimTodo(t->sH);
    delete t;
    t = NULL;
  }
}
```

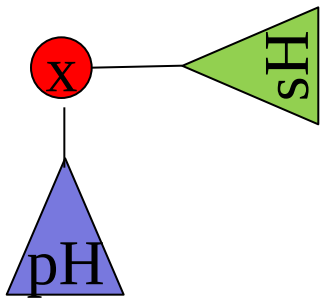
Buscar dato y dar su ubicación

Considere un árbol general de enteros representado mediante un árbol binario de enteros con la semántica puntero al primer hijo (pH), puntero al siguiente hermano (sH).

```
typedef nodoAG * AG;
struct nodoAG { int dato; AG pH; AG sH; };
```

Pensar en la especificación del problema...

AG buscar (AG t, int x)



Buscar dato y dar su ubicación

Considere un árbol general de enteros representado mediante un árbol binario de enteros con la semántica puntero al primer hijo (pH), puntero al siguiente hermano (sH).

```
typedef nodoAG * AG;
struct nodoAG { int dato; AG pH; AG sH; };
```

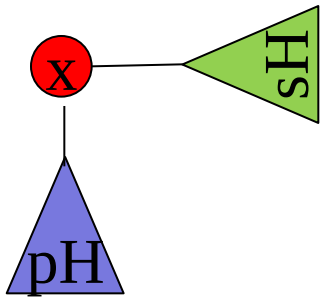
```
AG buscar (AG t, int x)
    if (t==NULL) return NULL;
    else if (t->dato==x) return t;
    else {      AG ret = buscar(t->pH, x);
                if (ret!=NULL) return ret;
                else return buscar(t->sH, x);
    }
}
```

Nivel dato

```
typedef nodoAG * AG;
struct nodoAG { int dato; AG pH; AG sH; };
```

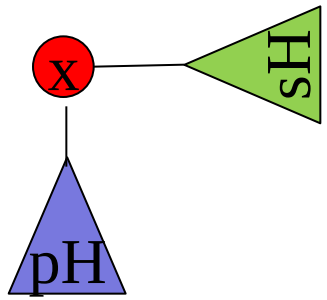
PRE: t no tiene elementos repetidos

int nivel(AG t, int x)



Nivel dato

```
int nivel(AG t, int x){
    if (t==NULL) return 0;
    else if (t->dato==x) return 1;
    else {      int ret = nivel(t->pH, x);
                if (ret>0) return 1+ret;
                else retun nivel (t->sH, x);
    }
}
```



Repaso de ABB

a) Considere la siguiente definición para árboles binarios de búsqueda de enteros (*ABB*):

```
typedef nodoABB * ABB
struct nodoABB { int dato; ABB izq; ABB der; }
```

Implemente una función recursiva (sin usar iteración) ***delMin*** que dado un árbol binario de búsqueda *t* de tipo *ABB* no vacío, elimine de *t* el mínimo elemento y retorne su valor. El árbol resultante deber ser también binario de búsqueda. No implemente ni asuma la existencia de operaciones auxiliares para implementar *delMin*.

```
PRE: t no vacío
int delMin(ABB & t)
```

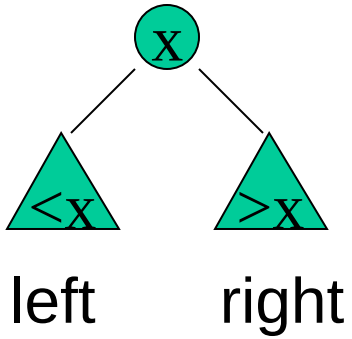
b) Explique muy brevemente el orden de tiempo de ejecución para el peor caso y para el caso promedio de ***delMin***.

Repaso de ABB

```
typedef nodoABB * ABB;
struct nodoABB { int dato; ABB izq; ABB der; };
```

PRE: t no vacío

int ***delMin****(ABB & t)*

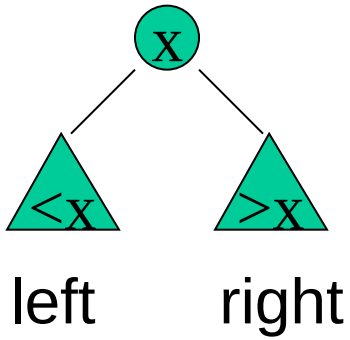


Repaso de ABB

```
typedef nodoABB * ABB;
struct nodoABB { int dato; ABB izq; ABB der; };
```

PRE: t no vacío

```
int delMin(ABB & t){
    if (t->izq==NULL){
        int min = t->dato;
        ABB aBorrar = t;
        t = t->der;
        delete aBorrar;
        return min;
    }
    else return delMin(t->izq);
}
```



Repaso de ABB

Para trabajar con una agenda de contactos telefónicos, considere la siguiente definición del tipo ***ABB*** de árboles binarios de búsqueda, que contienen números telefónicos (de tipo ***int***) y nombres completos de contactos (de un tipo ***T***):

```
struct nodoABB{
   unsigned int telefono; // número telefónico de un contacto
   T nombre; // nombre completo del contacto
   nodoABB * izq, * der;
}
typedef nodoABB * ABB;
```

Una agenda de contactos está modelada con un árbol de tipo ***ABB***, organizado (ordenado) por números telefónicos, que NO se pueden repetir. Los nombres completos asociados a los teléfonos en el árbol si podrían repetirse. Asuma que los nombres de tipo ***T*** pueden asignarse y compararse con los operadores habituales (=, ==).

a) Implemente un procedimiento recursivo ***void agendar(ABB & t, int num, T nom)*** que, dado un árbol *t* de tipo ***ABB***, un número de teléfono *num* y un nombre completo *nom*, inserta el contacto *num* con nombre *nom* en *t*, si *num* no estaba en *t*. En caso contrario, reemplaza el nombre completo asociado a *num* en *t* con *nom*. No defina operaciones auxiliares.

Indique el orden de tiempo de ejecución en el peor caso y en el caso promedio de ***agendar***.

Repaso de ABB

```
void agendar(ABB & t, int num, T nom){
    if (t == NULL){ // ingreso de un nuevo contacto
            t = new nodoABB;
            t->telefono = num;
            t->nombre = nom;
            t->izq = t->der = NULL;
    }
    else{
    if (num < t->telefono)
            agendar(t->izq, num, nom);
    else (num > t->telefono)
            agendar(t->der, num, nom);
    else t->nombre = nom; // reemplaza (actualiza) el nombre asociado al contacto num
    }
}
```

Indique el orden de tiempo de ejecución en el peor caso y en el caso promedio de **`agendar`**.

Repaso de ABB

Para trabajar con una agenda de contactos telefónicos, considere la siguiente definición del tipo ***ABB*** de árboles binarios de búsqueda, que contienen números telefónicos (de tipo ***int***) y nombres completos de contactos (de un tipo ***T***):

```
struct nodoABB{
   unsigned int telefono; // número telefónico de un contacto
   T nombre; // nombre completo del contacto
   nodoABB * izq, * der;
}
typedef nodoABB * ABB;
```

Una agenda de contactos está modelada con un árbol de tipo ***ABB***, organizado (ordenado) por números telefónicos, que NO se pueden repetir. Los nombres completos asociados a los teléfonos en el árbol si podrían repetirse. Asuma que los nombres de tipo ***T*** pueden asignarse y compararse con los operadores habituales (=, ==).

b) Implemente una función recursiva ***int contactos(ABB t, T nom)*** que, dado un árbol *t* de tipo ***ABB*** y un nombre completo *nom*, retorna la cantidad de contactos (números telefónicos) en *t* que coinciden con *nom*. Si *nom* no está en *t*, el resultado debe ser 0. No defina operaciones auxiliares.

Indique el orden de tiempo de ejecución en el peor caso de ***contactos***.

Repaso de ABB

```
int contactos(ABB t, T nom)
    if (t == NULL)
        return 0;
    else if (t->nombre == nom)
        return 1 + contactos(t->izq, nom) + contactos(t->der, nom);
    else return contactos(t->izq, nom) + contactos(t->der, nom);
}
```

Indique el orden de tiempo de ejecución en el peor caso de **`contactos`**.