Instituto de Computación. Facultad de Ingeniería. Universidad de la República

Examen de Programación 2

Julio de 2021

Problema 1 (35 puntos)

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

Implemente la función ***AG padre(AG a, int x)*** que retorne un puntero al nodo padre en *a* del nodo que tenga a *x* como dato. Asumimos que *a* no tiene elementos repetidos. Si *x* no está en *a* o si *x* es la raíz de *a*, la función *padre* deberá retornar el puntero NULL. No se pueden definir operaciones auxiliares para implementar *padre*.

Problema 2 (65 puntos: 5+25+20+15)

Considere el TAD Cola no acotada de enteros (***ColaInt***), que contempla la política FIFO, con operaciones exclusivamente para: crear una cola vacía; insertar; eliminar y retornar el elemento correspondiente; y chequear si una cola está vacía.

a) Especifique el **Tad *ColaInt***, incluyendo sus pre y postcondiciones.

b) Implemente la función: ***ColaInt camino (ABB t, int x)***, que dado un árbol binario de búsqueda (ABB) de enteros 't' y un entero 'x', retorne una cola de tipo *ColaInt* que contenga el camino a 'x' desde la raíz de 't'. En la cola resultado el primero debe corresponder a la raíz de 't' y el último elemento, según la política FIFO, debe ser 'x'. Si 'x' no está en 't', el resultado deberá ser la cola vacía. La operación *camino* no puede acceder a la representación del TAD ni modificar sus parámetros. Considere:

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

c) Sugiera una **implementación eficiente del TAD *ColaInt***, definiendo la representación del TAD e implementando solamente sus operaciones constructoras; asuma que el resto de las operaciones del TAD están implementadas eficientemente.

d) Indique el orden de tiempo de ejecución para el peor caso y el caso promedio de la función ***camino*** (de la parte b), considerando la implementación previa del TAD *ColaInt*. Justifique.

Instituto de Computación. Facultad de Ingeniería. Universidad de la República

Examen de Programación 2

Julio de 2021

SOLUCIONES

Problema 1

```
AG padre (AG t, int x){
        if (t == NULL || t->dato == x) \\ si es el árbol vacío o la raíz
                return NULL;
        else{
                AG hijo = t->pH; \\ se busca x entre los hijos del nodo apuntado por t
                while (hijo != NULL && hijo->dato != x)
                        hijo = hijo->sH;
                if (hijo != NULL) return t; \\ x es un hijo del nodo apuntado por t
                else {
                        AG enPH = padre (t->pH, x);
                        if (enPH != NULL) return enPH; \\ el resultado está en la búsqueda por t->pH
                        else return padre (t->sH, x); \\ sino, finalmente buscamos en t->sH
                }
        }
}
```

Problema 2

a)

```
struct RepresentacionCola;
typedef RepresentacionCola* ColaInt;
```

ColaInt crearCola ();
// Devuelve la cola vacia.
void encolar (int t, ColaInt &c);
// Agrega el elemento t al final de c.
bool esVaciaCola (ColaInt c);
// Devuelve 'true' si c es vacia; 'false' en otro caso.
	int frente (ColaInt c);
	/* Devuelve el primer elemento de c.
	Precondicion: ! esVaciaCola(c). */
	void desencolar (ColaInt &c);
	/* Remueve el primer elemento de c.
	Precondicion: ! esVaciaCola(c). */

Instituto de Computación. Facultad de Ingeniería. Universidad de la República

Examen de Programación 2

Julio de 2021

b)

```
ColaInt camino (ABB t, int x){
  ColaInt c = crearCola();
  if perteneceABB(t, x){
        while (t->dato != x){
              encolar(t->dato, c);
              if (x < t->dato) t = t->izq;
              else t = t->der;
        }
        encolar(t->dato, c);
  }
  return c;
}
```

```
// retorna true si y solo si x está en t
bool perteneceABB (ABB t, int x){
  while (t != NULL && t->dato != x){
        if (x < t->dato) t = t->izq;
        else t = t->der;
  }
  return t != NULL;
}
```

c)

Usaremos una lista simplemente encadenada, con puntero al inicio y al final. De esta manera no acotamos la cantidad de elementos y permitimos hacer ingresos y egresos en O(1) peor caso. En particular, todas las operaciones del TAD Cola pueden ser de O(1) con esta representación.

```
struct Nodo{
   int valor;
   Nodo * sig;
}
```

```
struct RepresentacionCola{
   Nodo * primero;
   Nodo * ultimo;
}
```

```
ColaInt crearCola(){
   ColaInt c = new RepresentacionCola;
   c->primero = c->ultimo = NULL;
   return c;
}
// O(1) peor caso
```

```
void encolar (int t, ColaInt &c){
   Nodo * nuevo = new Nodo;
   nuevo->valor = t;
   nuevo->sig = NULL;
   if (c->primero == NULL)
         c->primero = nuevo;
   else c->ultimo->sig = nuevo;
   c->ultimo = nuevo;
} // O(1) peor caso
```

d)

O(n) peor caso y O(log2(n)) en el caso promedio, siendo n la cantidad de elementos del ABB.

La función camino usa la función auxiliar perteneceABB, que tiene O(n) peor caso y $O(\log_2(n))$ en el caso promedio, siendo n la cantidad de nodos del árbol. Esto es así ya que se recorre un camino de un ABB realizando en cada paso acciones de O(1). Recordar que la altura de un ABB en el peor caso es O(n) y en el caso promedio es $O(\log_2(n))$. Luego, la función camino tiene en el peor caso y en el caso promedio el mismo orden que la función perteneceABB, ya que recorrer dos veces un camino no cambia el orden del peor caso ni del caso promedio.