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

Examen de Programación 2

Julio de 2022

Problema 1 (30 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; nodoAG *pH, *sH; }
```

Implemente la función ***bool existeNodo (AG t, int n)*** que retorne true si y solo si en el árbol *t* existe al menos un nodo que tiene *n* o más hijos, asumiendo *n>0* y t->sH==NULL. Si utiliza operaciones auxiliares, deberá implementarlas. La función ***existeNodo*** debe tener O(*m*) en el peor caso, siendo *m* la cantidad de nodos de *t*.

Problema 2 (30 puntos)

Considere la siguiente definición del tipo *ABB* de los árboles binarios de búsqueda de enteros sin elementos repetidos , en memoria dinámica:

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

Implemente un procedimiento ***insertarMenores*** que dados dos ABB *t1* y *t2*, y dado un entero *x*, inserte en *t2* los elementos de *t1* que sean menores estrictos a *x*. Si no hay elementos menores a *x* en *t1* (en particular si *t1* es vacío), el procedimiento no tendrá efecto. La función deberá evitar recorrer elementos de *t1* que no sean estrictamente necesarios. Si usa operaciones auxiliares, deberá implementarlas.

```
void insertarMenores(ABB t1, ABB & t2, int x)

Ejemplo: Si t1 es:    6    y  x=8 entonces en t2 deberían incorporarse los
                    /  \   siguientes elementos de t1: 1, 3, 4, 5, 6, 7
                  4    9
                 / \   /
                1  5  7
                 \     \
                  3     8
```

Problema 3 (40 puntos)

Considere la siguiente especificación, con pre y postcondiciones, de un TAD *cola de prioridad* no acotado de elementos de un tipo *T* con prioridades que toman valores enteros:

```
// PRE: -
// POS: retorna una nueva cola de prioridad vacía
ColaPrioridad crearColaPrioridad ();

// PRE: -
/* POS: inserta un elemento t de tipo T con prioridad p a la cola de prioridad. Los
elementos con igual prioridad se consideran en orden FIFO. */
void encolar(ColaPrioridad &cp, T t, int p);
```

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

Examen de Programación 2

Julio de 2022

```
// PRE: -
// POS: retorna la cantidad de elementos presentes en la cola de prioridad
unsigned int cantidad (ColaPrioridad cp);

// PRE: cantidad(cp)!=0
/* POS: retorna y quita el elemento con mayor prioridad (mayor valor entero) de la
cola de prioridad. Ante elementos de igual prioridad máxima, retorna y elimina el más
antiguo (orden FIFO) */
T obtener(ColaPrioridad &cp);

// PRE: -
// POS: retorna una copia de la cola de prioridad parámetro sin compartir memoria
ColaPrioridad copia (ColaPrioridad cp);

// PRE: -
// POS: destruye la cola de prioridad, liberando su memoria
void destruir (ColaPrioridad &cp);
```

Se pide:

a) Defina una representación del TAD anterior en la que las operaciones ***crearColaPrioridad, obtener*** y ***cantidad*** tengan O(1) de tiempo de ejecución en el peor caso. Escriba el código de las operaciones ***crearColaPrioridad*** y ***encolar.*** Asuma que las restantes están implementadas. Considere que los elementos de tipo T pueden manipularse usando los operadores habituales de asignación y comparación (como si fuera un tipo numérico básico).

b) Implemente la función ***bool indistinguibles (ColaPrioridad cp1, ColaPrioridad cp2)*** que dadas dos colas de prioridad retorne true si y solo si son indistinguibles. Esto es, si los elementos de tipo *T* que se obtienen de ambas son los mismos y en el mismo orden. La función no deberá acceder a la representación del TAD (su implementación), ni modificar las colas de prioridad parámetro, ni dejar memoria colgada.

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

Examen de Programación 2

Julio de 2022

SOLUCIONES

PROBLEMA 1

```
bool existeNodo (AG t, int n){
	if (t == NULL)
		return false;
	else
		return (hijos(t) >= n) || existeNodo(t->pH, n) || existeNodo(t->sH, n);
}

// Retorna la cantidad de hijos de t en un árbol pH-sH. Precondición: t != NULL.
int hijos (AG t){
	int cantHijos = 0;
	t = t->pH;
	while (t != NULL){
		cantHijos++;
		t = t->sH;
	}
	return cantHijos;
}

// Versión puramente recursiva
struct bool_int {
	bool existe;
	int cantidad;
};

// Función auxiliar para existeNodo.
// En el campo existe del resultado devuelve true si y solo si alguno de los nodos
// del bosque que se accede desde b tiene n o más hijos.
// En el campo cantidad del resultado devuelve la cantidad de árboles del bosque
// que se accede desde b.
bool_int existeNodoAux (AG b, int n)
	bool_int res;
	if (b == NULL) {
		res = (bool_int){false,0};
	} else
		res = existeNodoAux (b->sH, n);
		res.cantidad++; // agrega el árbol
		if (! res.existe) {
			bool_int res_hijos = existeNodoAux (b->pH, n);
			res.existe = res_hijos.existe ||
			(res_hijos.cantidad >= n); // el nodo tiene n o más hijos
		}
	}
	return res;
}

bool existeNodo (AG t, int n){
```

Examen de Programación 2

Julio de 2022

```
      return  existeNodoAux(t,n).existe;
}
```

PROBLEMA 2

```
void insertarMenores(ABB t1, ABB & t2, int x){
      if (t1!=NULL){
            if (t1->dato >= x)
                  insertarMenores(t1->izq, t2, x);
            else{
            // insABB como en el teórico: inserta un entero en un ABB si no estaba
                  insABB(t1->dato, t2);
                  insertarMenores(t1->izq, t2, x);
                  insertarMenores(t1->der, t2, x);
            }
      }
}
```

PROBLEMA 3

Parte a)

```
struct nodoCola {
   T dato;
   int prioridad;
   nodoCola* sig;
};

struct representacionColaPrioridad {
   nodoCola* ppio;
   unsigned int largo;
};
typedef representacionColaPrioridad* ColaPrioridad;
/* Para cumplir los requisitos de orden se mantienen: un puntero al inicio de una
lista simplemente enlazada  y un entero con la cantidad de elementos presentes en la
lista. Las inserciones se hacen de manera ordenada, asegurando que el elemento de
mayor prioridad esté siempre al inicio. Ante igual prioridad, aparece en la lista el
primero que llegó (FIFO) */

ColaPrioridad crearColaPrioridad (){
   ColaPrioridad cp = new representacionColaPrioridad;
   cp->ppio = NULL;
   cp->largo = 0;
   return cp;
}

// procedimiento recursivo auxiliar de encolar que realiza inserciones ordenadas
void insLista(nodoCola* &l, T t, int p){
   if (l==NULL || p > l->prioridad) { // si tiene igual prioridad va después (FIFO)
       nodoCola * nuevo = new nodoCola;
       nuevo->dato = t;
       nuevo->prioridad = p;
```

```
      nuevo->sig = l;
      l = nuevo;
  }
  else insLista(l->sig, s, p);
}

void encolar(ColaPrioridad &cp, T t, int p){
  insLista(cp->ppio, t, p);
  cp->largo++;
}

// Versión iterativa
void encolar(ColaPrioridad &cp, T t, int p){
  nodoCola * nuevo = new nodoCola;
  nuevo->dato = t;
  nuevo->prioridad = p;
  cp->largo++;
  if ((cp->ppio == NULL) || (p > cp->ppio->prioridad)) {
      nuevo->sig =  cp->ppio;
      cp->ppio = nuevo;
  } else {
      nodoCola * cursor = cp->ppio;
      while ((cursor->sig != NULL) && (p <= cursor->sig->prioridad)) {
            cursor = cursor->sig;
      }
      nuevo->sig = cursor->sig;
      cursor->sig = nuevo;
  }
}
```

Parte b)

```
bool indistinguibles(ColaPrioridad cp1, ColaPrioridad  cp2){
  clon_cp1 = copia(cp1);
  clon_cp2 = copia(cp2);
  while (cantidad(clon_cp1)!=0 && cantidad(clon_cp2)!=0 &&
         obtener(clon_cp1) == obtener(clon_cp2)){}
  bool res = (cantidad(clon_cp1)==0 && cantidad(clon_cp2)==0);
  destruir(clon_cp1);
  destruir(clon_cp2);
  return res;
}
```