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

Examen de Programación 2

Febrero de 2022

Problema 1 (50 puntos)

Considere la siguiente especificación del TAD ***Tabla*** acotada de *unsigned int* (dominio) en *char* (codominio):

```
struct RepTabla;

typedef RepTabla * Tabla;

typedef unsigned int nat;

// POS: Devuelve la Tabla vacía, sin correspondencias, donde max es la cantidad máxima de elementos que se puede almacenar en la Tabla
Tabla crear(nat max);

// POS: En el caso de que d no tiene imagen en t, agrega la correspondencia (d,c).
// En el caso contrario solo actualiza la imagen de d con c.
// La operación no tiene efecto si t está llena y d no tiene imagen en t.
void insertar (nat d, char c, Tabla & t);

// POS: Devuelve true si y sólo si d tiene imagen en t.
bool definida (nat d, Tabla t);

// POS: Devuelve la cantidad de correspondencias en t. En particular, 0 si t es la tabla vacía.
int cantidad (Tabla t);

// PRE: definida(d,t). POS: Retorna la imagen de d en t.
char recuperar (nat d, Tabla t);

// POS: Elimina de t la correspondencia que involucra a d, si d está definida en t. En otro caso la operación no tiene efecto.
void eliminar (nat d, Tabla & t);
```

a) Proponga y justifique una representación para el TAD ***Tabla*** de tal manera que *cantidad* tenga O(1) en el peor caso e *insertar*, *definida, recuperar* y *eliminar* tengan O(1) en el caso promedio asumiendo que la distribución de los elementos del dominio es uniforme.

b) Implemente la representación del TAD (***RepTabla***) y el código de las operaciones ***crear***, ***cantidad*** e ***insertar***. Omita el código del resto de las operaciones del TAD, que puede asumir implementadas. Si utiliza operaciones auxiliares debe implementarlas.

Solución:

a) La representación propuesta consiste en representar la tabla mediante un hash abierto, utilizando la cantidad máxima (max) de elementos como tamaño de la tabla, y la función de hash $h: uint \rightarrow [0, max)$, tal que $h(x) = x \% max$, es decir, h(x) es el resto de dividir x entre el tamaño de la tabla.

De esta forma, como la cantidad esperada de elementos no supera el tamaño de la tabla y la función de hash distribuye de manera uniforme (ya que los elementos del dominio están distribuidos de manera uniforme por hipótesis), podemos afirmar que las operaciones *insertar*, *recuperar*, *definida* y *eliminar* tendrán O(1) en el caso promedio, ya que todas ellas consisten esencialmente en acceder a la posición correspondiente del hash en tiempo O(1) y luego realizar una búsqueda lineal en una lista de tamaño O(1) promedio.

Luego, para poder garantizar que la función *cantidad* tenga O(1) en el peor caso será necesario además mantener en la representación de la Tabla una variable de tipo *int* que contenga la cantidad de elementos de la tabla, la cual se mantendrá actualizada modificándola en las operaciones de inserción y remoción cuando corresponda.

b) A continuación se implementan la representación del TAD y las operaciones solicitadas.

Examen de Programación 2

Febrero de 2022

```
typedef unsigned int nat;

struct nodo {
	nat d;
	char c;
	nodo* sig;
};
struct RepTabla {
	nodo ** hash;
	nat max; // cantidad máxima de elementos en la tabla
	int cant; // cantidad actual de elementos en la tabla
};
typedef RepTabla * Tabla;

Tabla crear(nat max) {
	Tabla res = new RepTabla;
	res->max = max;
	res->cant = 0;
	res->hash = new (nodo*)[max];
	for ( int i = 0; i < max; i++)
		res->hash[i] = NULL;
	return res;
}

int cantidad(Tabla t) {
	return t->cant;
}

void insertar(nat d, char c, Tabla &t) {
	// Si no está llena o es una actualización, avanzo
	if( (cantidad(t) < t->max) || definida(d,t)) {
		int pos = d % t->max;
		if(definida(d,t)) { // es una actualización
			// busco el par en el bucket correspondiente y actualizo c
			nodo* iter = t->hash[pos];
			while (iter->d != d) iter = iter->sig;
			iter->c = c; // sé que lo voy a encontrar porque definida(d,t) es true
		} else { // es una nueva inserción
			// creo un nuevo nodo y lo inserto al inicio del bucket correspondiente
			nodo* nuevo = new nodo;
			nuevo->d = d;
			nuevo->c = c;
			nuevo->sig = t->hash[pos];
			t->hash[pos] = nuevo;
			t->cant++; // actualizo cantidad de elementos en la tabla
		}
```

Examen de Programación 2

Febrero de 2022

```
      } // else: cantidad(t) >= t->max && !definida(d,t),
            // la tabla está llena y no es una actualización
}
```

Problema 2 (50 puntos)

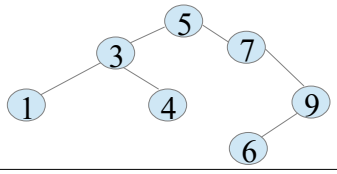
Implemente un procedimiento que recibe como parámetro de entrada un árbol binario de enteros ***a*** e imprime la suma de los valores de cada nivel de ***a***. El tiempo de ejecución en el peor caso del procedimiento debe ser O(n), siendo ***n*** la cantidad de elementos ***a***. Brinde la especificación de el/los TADs auxiliares que necesite para resolver el problema (no es necesario implementarlos).

```
void sumasNiveles(ABEnt a);
```

```
 struct nodoABEnt {
  int dato;
  nodoABEnt * izq, * der;
  };
typedef nodoABEnt * ABEnt;
```

Ejemplo

Entrada:



Salida: 5 10 14 6

Solución:

Para la solución será necesario contar con un TAD Cola de ABEnt. Se especifica a continuación.

```
struct RepCola;
typedef RepCola * Cola;

/ * Devuelve la cola Cola vacía. * /
Cola crearCola ();

/ * Agrega el árbol a a la cola c. * /
void encolar (ABEnt a, Cola &c) ;

/ * Devuelve 'true' si c es vacía , 'false' en otro caso . * /
bool esVaciaCola (Cola c) ;

/ * Devuelve el elemento en el frente de la cola c
PRE: !esVaciaCola (c) . * /
ABEnt frente (Cola c);

/ * Remueve el elemento del frente de la cola c.
PRE: !esVaciaCola (c). * /
void desencolar (Cola &c);

/ * Libera toda la memoria ocupada por c. * /
void destruirCola (Cola &c);
```

```
void sumasNiveles (ABEnt a)
{
  if (a!= NULL){  //Si el árbol es vacío no se imprime nada
    Cola c = crearCola();
```

```
    encolar(a, c);  //encolo el primer árbol
    encolar(NULL, c);  //utilizo NULL como marca de fin de nivel
    int suma = 0;
    while !esVaciaCola(c){
       ab = frente(c);  //obtengo el primer elemento de la cola
       desencolar(c);
       if (ab != NULL){  //No llegué al final de un nivel, sumo y encolo los hijos
         suma = suma + ab→dato;
         if (ab→izq != NULL)
           encolar (ab→izq, c);
         if (ab→der != NULL)
           encolar (ab→der, c);
       }else{
         printf("%d ", suma);
         suma = 0;
         if !esVaciaCola(c)
           encolar (NULL, c);  //Agrego marca para el siguiente nivel
       }
     }
     destruirCola(c);
  }
}
```