Examen de Programación 2

16 de Diciembre 2020

Problema 1 (50 puntos)

```
typedef nodo* Lista;
struct nodo {int dato; Lista sig;};
```

a) Implemente un procedimiento iterativo ***insFinal*** que dados una lista de enteros ***l*** y un entero ***e***, agregue a ***e*** al final de ***l***. Si la lista es vacía, lo insertará como único elemento de ésta.

void insFinal (Lista & l, int e)

b) Dada la siguiente función iterativa ***copia*** en C++ que, dada una lista de enteros ***l*** devuelve una copia idéntica de ***l*** sin compartir memoria con ésta:

```
Lista copia (Lista l){
      Lista resultado = NULL;
      while (l != NULL){
            insFinal(resultado, l->dato);
            l = l->sig;
      }
      return resultado;
}
```

Calcule el orden de tiempo de ejecución en el peor caso de la función ***copia***. Justifique brevemente.

c) Sin usar el procedimiento ***insFinal*** ni otras estructuras auxiliares, implemente la función copiaEficiente para construir la copia de una lista, sin compartir memoria, en un menor orden de tiempo de ejecución en el peor caso que la función copia. Calcule el orden del peor caso y justifique brevemente.

Lista copiaEficiente (Lista l)

Solución:

a)
```
void insFinal (Lista & l, int e){
      Lista nuevo = new nodo;
      nuevo->dato = e;
      nuevo->sig = NULL;
      if (l == NULL)
            l = nuevo;
      else{
            Lista iter = l;
            while (iter->sig != NULL)
                  iter = iter->sig;
            iter->sig = nuevo;
      }
}
```

b) El tiempo de ejecución en el peor caso del procedimiento ***insFinal*** es $T(n) <= \Sigma_1^n 1 = n$, por lo que ***insFinal*** es O(n), siendo ***n*** el largo de la lista.

Luego tiempo de ejecución en el peor caso de la función ***copia*** es $T(n) <= \Sigma_1^n T_{insFinal}(n) = \Sigma_1^n i = n(n+1)/2$, por lo que ***copia*** es $O(n^2)$, siendo ***n*** el largo de la lista.

c)
```
Lista copiaEficiente (Lista l){
      Lista result = NULL;
      if (l != NULL){
            Lista nuevo = new nodo;
            nuevo->dato = l->dato;   //copiamos el primer elemento
            nuevo->sig = NULL;
            result = nuevo;

            Lista iter = result;    // usamos iter para no perder el comienzo
            while (l != NULL){       // recorremos con l porque pasa por copia
                  Lista nuevo = new nodo;
                  nuevo->dato = l->dato;
                  nuevo->sig = NULL;
                  iter->sig = nuevo;
                  iter = iter->sig;
                  l = l->sig;
            }
      }
      return result;
}
```

Examen de Programación 2

16 de Diciembre 2020

Problema 2 (50 puntos)

Considere la siguiente especificación del **TAD *Multiset*** no acotado de números enteros en el rango **1..K**, para una constante **K** dada:

```
// POS: Devuelve el multiset vacío.
Multiset crear();

// POS: Agrega n ocurrencias del elemento x al multiset m. PRE: n>0, 1<=x<=K.
void insertar (Multiset & m, int x, int n);

// POS: Devuelve la suma total de ocurrencias de todos los elementos del multiset m (0 si m está vacío).
int cantidad (Multiset m);

// POS: Devuelve la cantidad de ocurrencias del elemento x del multiset m (0 si x no está en m).
int ocurrencias (Multiset m, int x);

/* POS: Elimina a lo sumo n ocurrencia del elemento x del multiset m. Si ocurrencias(m, x)<=n entonces en m no quedarán ocurrencias del elemento x. */
void eliminar (Multiset & m, int x, int n);

// POS: Devuelve el mínimo elemento del multiset m (independientemente del número de ocurrencias). PRE: m no vacío.
int min (Multiset m);
```

Se quiere implementar el TAD ***Multiset*** anterior de tal manera que las operaciones *insertar*, *ocurrencias* y *eliminar* tengan O(log(*n*)) de tiempo de ejecución en el caso promedio, siendo *n* la cantidad de elementos diferentes del multiset, y las operaciones *crear*, *cantidad* y *min* tengan O(1) peor caso. Además, *eliminar* deberá liberar la memoria correspondiente. La implementación debe hacerse en espacio O(n).

Se pide:

a) Defina en C++ la **representación del TAD** y escriba los códigos de las operaciones ***crear***, ***insertar*** *y* ***min***. Omita el código del resto de las operaciones, que puede asumir están implementadas.

```
struct RepMultiset;
typedef RepMultiset * Multiset;
```

b) Implemente el procedimiento ***vaciar*** que dado un multiset *m*, según la especificación dada previamente, elimine todos sus elementos, dejando a éste vacío. El procedimiento *vaciar* no debería acceder a la representación del TAD ni dejar memoria colgada (sin liberar).

```
void vaciar (Multiset & m)
```

Examen de Programación 2

16 de Diciembre 2020

Solución:

a)

```
typedef nodoABB* ABB;
struct nodoABB { int elem; int ocurr; ABB izq, der;};
struct RepMultiset {ABB multi; int cant; int Mmin;};
typedef RepMultiset * Multiset;

Multiset crear(){
     Multiset m = new RepMultiset;
     m->multi = NULL;
     m->cant = 0;
     return m;
}

void insertar (Multiset & m, int x, int n){
      insABB(x, n, t->multi));

      if (m->cant == 0) || (m->Mmin > x)      // si no hay elementos o
            m->Mmin = x;                      //si x es menor que el mínimo
      m->cant = m->cant + n;
}

void insABB (int e, int o, ABB & a)
{
   if (a==NULL)
    {a = new nodoABB;
     a->elem = e;
     a->ocurr = o;
     a->izq = NULL;
     a->der = NULL;}
   else
     {if (a->elem == e)
        {a->ocurr = a->ocurr + o;}
      else
        {if (a->elem < e)
            result = insABB(e, o, a->der);
         else
            result = insABB(e, o, a->izq);    // a->elem > e
        }
     }
}

int min (Multiset m){
      return (m->Mmin);
}
```

b)

```
void vaciar (Multiset & m){
      while (cantidad(m) > 0){
            eliminar(m, min(m), ocurrencias(m, min(m)));
      }
}
```