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

Examen de Programación 2

19 de diciembre de 2024

*Escribir en todas las hojas **nombre completo**, **cédula, número de hoja** y **cantidad de hojas que entrega**. Escribir de manera clara y legible (con lapiz o lapicera). Empezar cada problema en hoja nueva y no escribir las hojas de ambos lados.*

Problema 1 (50 puntos)

Considere la siguiente definición del tipo *ABB* de los árboles binarios de búsqueda, en memoria dinámica, que almacenan productos ordenados según su código (único), de tipo *int*, y el precio de cada uno, de tipo *float*:

```
typedef nodoABB * ABB;
struct nodoABB {int codigo; float precio;  ABB izq, der;};
```

a) Implemente una función ***combinar*** que dados dos *ABB t1* y *t2*, que contienen productos y sus precios, genere y retorne un nuevo *ABB*, que no comparta memoria, que sea el resultado de combinar los datos de ambos. En caso de encontrar el mismo producto en ambos árboles, pero con valores de precio diferentes, se debe guardar el precio mayor. Si los árboles son vacíos retorna NULL. Si usa operaciones auxiliares, deberá implementarlas, indicando también sus pre y postcondiciones.

```
ABB combinar (ABB t1, ABB t2)
```

b) Indique el orden de tiempo de ejecución para el peor caso de su solución. Justifique muy brevemente.

Problema 2 (50 puntos)

Se sabe que el Programa Nacional de Vivienda (PNV), tiene un registro de familias que desean acceder a una vivienda en una Cola de Prioridad **colaFamilias**, en la que las familias tienen un *Índice de prioridad* calculado en base a sus necesidades (int), un *Identificador de familia* (string) y un *Departamento* en donde desean vivir ([1..19]).

Además el PNV posee un catálogo de cantidad de viviendas disponibles para cada *Departamento* ([1..19]), de tipo Mapping llamado **vivDisp** de viviendas disponibles. El Mapping **vivDisp** tiene al menos las siguientes operaciones:

```
//Crea y devuelve un vivDisp vacío para registrar viviendas para 19 departamentos
vivDisp crearVD();
//Agrega una vivienda disponible al departamento depto en vD. Pre: 1<=depto<=19.
void agregarVivienda(int depto, vivDisp &vD);
//Devuelve la cantidad de viviendas disponibles en depto en vD. Pre: 1<=depto<=19.
int cantViviendas(int depto, vivDisp vD);
//Resta una vivienda disponible al departamento depto en vD.
//Pre: cantViviendas(depto, vD)>0 y 1<=depto<=19.
void restarVivienda(int depto, vivDisp &vD);
```

(a) Especifique el TAD Cola de Prioridad **colaFamilias**, con operaciones para:

- **crear** una cola vacía con hasta n familias
- **ingresar** una familia con *indicePrio*, *idFamilia*, *departamento*
- **obtenerFlia** devuelve el *id* de la familia con mayor prioridad según *indicePrio*
- **obtenerDpto** devuelve el *Departamento* de la familia con mayor prioridad según *indicePrio*
- **quitar** la familia con mayor prioridad según *indicePrio*
- saber si una cola es **vacía**

(b) Implemente una función **noTieneVivienda**, que recibe una Cola de Prioridad **colaFamilias** y un Mapping **vivDisp** de viviendas disponibles, y que devuelve el identificador de la familia con más prioridad a la que no se puede otorgar una vivienda en el departamento deseado. En caso de poder satisfacer a todas las familias devuelve "exito". Se asume que los strings se pueden asignar y comparar como los tipos básicos (=, ==, !=, etc.). Se puede modificar los parámetros de tipo **colaFamilias** y **vivDisp.**

```
string noTieneVivienda(colaFamilias cF, vivDisp vD);
```

(c) ¿Qué implementación de los TADs **colaFamilias** y **vivDisp** permite que **noTieneVivienda** sea O(n log n) peor caso? Explique brevemente.

Examen de Programación 2

19 de diciembre de 2024

SOLUCIÓN

Problema 1

a)

```
ABB combinar (ABB t1, ABB t2) {
      ABB res = NULL;
      agregarEnABB (res, t1);
      agregarEnABB (res, t2);
      return res;
}

/* PRE:- POS: agrega en t los productos de t2, manteniendo el precio mayor para los
productos comunes */
void agregarEnABB (ABB & t, ABB t2) {
      if (t2 != NULL) {
            insABB (t, t2->codigo, t2->precio);
            agregarEnABB (t, t2->izq);
            agregarEnABB (t, t2->der);
      }
}

/* PRE:- POS: inserta en t un código de producto con su precio. Si ya existe el pro-
ducto, se considera el mayor precio */
void insABB (ABB & t, int codigo, float precio) {
      if (t == NULL) {
            t = new nodoABB;
            t->codigo = codigo;
            t->precio = precio;
            t->izq = t->der = NULL;
      }
      else if (t->codigo < codigo)
            insABB (t->der, codigo, precio);
      else if (t->codigo > codigo)
            insABB (t->izq, codigo, precio);
      else if (t->precio < precio)
            t->precio = precio;
}
```

b) Es O(n*m) peor caso, siendo n y m la cantidad de elementos/nodos de los árboles t1 y t2, ya que se recorren ambos realizando inserciones en un nuevo árbol binario de búsqueda nuevo. Tener presente que la inserción en un árbol binario de búsqueda recorre en el peor caso todos sus nodos.
Puede haber otras soluciones con otros tiempos, por ejemplo aplanando los árboles.

Examen de Programación 2

19 de diciembre de 2024

Problema 2

a)

```
//Crea y devuelve una colaFamilias vacía
colaFamilias crearCF(int maxFamilias);

//Agrega la familia con id idFam, prioridad iP y departamento dep, a cF.
void ingresarFamilia(string idFam, int iP, int dep, colaFamilias &cF);

//Devuelve el id de la familia con mayor prioridad según indicePrio en cF.
string obtenerFlia(colaFamilias cF);

//Devuelve el departamento de la familia con mayor prioridad según indicePrio en cF.
int obtenerDpto(colaFamilias cF);

//Quitar  la familia con mayor prioridad según indicePrio de cF.
void quitarFamilia(colaFamilias &cF);

//Devuele true si cF está vacía
bool esVaciaCF(colaFamilias cF)
```

b)

```
string noTieneVivienda(colaFamilias cF, vivDisp vD)
{
      bool encontreFam= false;
      string famSinViv= "exito";

      while (!esVaciaCF(cF) && !encontreFam){
            string idFam= obtenerFlia(cF);
            int deptoFam= obtenerDpto(cF);

            if (cantViviendas(deptoFam, vD)>0){
                  restarVivienda(deptoFam, vD);
                  quitarFamilia(cF);
            }
            else{
                  famSinViv= idFam;
                  encontreFam= true;
            }
      }
      return  famSinViv;
}
```

c)

El mapping se puede implementar con un arreglo indexado por departamento [1..19], para tener acceso en O(1) peor caso; y la cola de prioridad como un Heap, de esta forma las operaciones con mayor tiempo de ejecución dentro del while son O(log n) y en total `noTieneVivienda` será O(n log n) en peor caso.