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

Segundo Parcial Programación 2

Noviembre de 2022

Problema 1 (31 puntos)

Considere la especificación del TAD ***Lista*** no acotada de enteros con las operaciones:

1. ***Lista crear*()** // Crea una lista vacía
2. ***void insertar(int x, Lista & l)*** // Agrega el entero *x* a la lista *l*, que puede contener elementos repetidos
3. ***bool esVacia(Lista l)*** // Retorna true si y solo si la lista *l* es vacía
4. ***int primero(Lista & l)*** // Elimina y retorna el primer entero ingresado en *l*. Precondición: !esVacia(*l*)
5. ***int ultimo(Lista & l)*** // Elimina y retorna el último entero ingresado en *l*. Precondición: !esVacia(*l*)
6. ***Lista copia(Lista l)*** // Retorna una copia idéntica de la lista *l* sin compartir memoria con ésta
7. ***void destruir(Lista & l)*** // Destruye la lista *l*, liberando toda su memoria.

a) Implementar iterativamente la operación ***esLifoFifo*** que dada una lista *l*, de tipo *Lista*, retorne true si y sólo si la secuencia de elementos que se obtienen de *l* siguiendo la política LIFO (el último en ingresar es el primero en salir) coincide con la secuencia de elementos que se obtiene de *l* siguiendo la política FIFO (el primero en ingresar es el primero en salir). En el caso de la lista vacía, *esLifoFifo* debe retornar true. La función no debe modificar la lista parámetro. Solo se puede usar el TAD *Lista* (no use estructuras de datos auxiliares, como arreglos), sin acceder a la *representación* de una implementación de dicho TAD.

bool esLifoFifo (Lista l)

b) Implementar el TAD ***Lista*** de tal manera que las operaciones 1–5 sean O(1) peor caso. Omita el código de las operaciones 3, 5, 6 y 7, pero **implemente las restantes (1, 2 y 4)** luego de definir una ***representación*** del TAD que cumpla los requerimientos de eficiencia establecidos.

Problema 2 (22 puntos)

Considere un TAD ***Conjunto*** de elementos de un tipo T que tiene, entre otras, las siguientes operaciones:

bool pertenece (Conjunto c, T e), que retorna true si y solo si el elemento *e* está en el conjunto *c*
T minimo (Conjunto c), que retorna el mínimo elemento de un conjunto *c* no vacío (precondición)

Considere además una implementación del TAD ***Conjunto*** usando *hashing* abierto, que tiene la siguiente representación:

```
struct nodoHash {
        T dato;
        nodoHash * sig;
}
```

```
struct representacionConjunto {
        nodoHash ** tabla; // tabla de hash
        int cota; // tamaño de la tabla de hash
        int cantidad; // cantidad actual de elementos en el conjunto
}
typedef representacionConjunto * Conjunto
```

a) Implemente las operaciones ***pertenece*** y ***minimo***, considerando la representación previa de un conjunto con *hashing* abierto. Considere la existencia de una función de hash: ***unsigned int h (T e)*** y asuma que los elementos de tipo *T* se pueden manipular con los operadores básicos: =, ==, <. Tener en cuenta que la función ***h*** retorna un número natural que puede exceder el valor definido para el tamaño de una tabla de hash.

b) ¿Cuál es el orden de tiempo de ejecución en el caso promedio de las operaciones ***pertenece*** y ***minimo***, asumiendo que la función ***h*** y el tamaño de la tabla de hash son los adecuados? Adicionalmente, ¿Cuál es el tiempo de ejecución para el peor caso de ambas operaciones? Explique muy brevemente.

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

Segundo Parcial Programación 2

Noviembre de 2022

SOLUCIONES

1-a)

```
bool esLifoFifo (Lista l){
        Lista fifo = copia(l);
        Lista lifo = copia(l);
        bool res = true;
        while (res && !esVacia(fifo)){ // si fifo es vacía, también lo será lifo
                res = (primero(fifo) == ultimo(lifo));
        }
        destruir(fifo);
        destruir(lifo):
        return res;
}
```

Nota: Alternativamente se podría usar una sola copia de *l*, chequeando sobre ésta el primero con el último, el segundo con el penúltimo, y así sucesivamente.

1-b)

```
struct nodoLista {
        int dato;
        nodoLista * ant;
        nodoLista * sig;
} // doble encadenada
```

```
struct representacionLista {
        nodoLista * inicio; /* donde se hacen las inserciones y
        está el último ingresado (lifo) */
        nodoLista * final; // donde está el primero ingresado (fifo)
}
typedef representacionLista * Lista
```

```
Lista crear(){
    Lista l = new representacionLista;
    l->inicio = l->final = NULL;
    return l;
}
```

```
void insertar(int x, Lista & l){
    nodoLista * nuevo = new nodoLista;
    nuevo->dato = x;
    nuevo->ant  = NULL;
    nuevo->sig = l->inicio;
    l->inicio = nuevo;
    if (l->final==NULL)
        l->final = nuevo;
    else     l->inicio->sig->ant = nuevo;
}
```

```
int primero(Lista & l){
    nodoLista * aBorrar = l->final;
    int ret = aBorrar->dato;
    if (l->inicio==l->final){
        l->inicio = l->final = NULL;
    }
    else{
        l->final->ant->sig = NULL;
        l->final = l->final->ant;
    }
    delete aBorrar;
    return ret;
}
```

2-a)

```
bool pertenece (Conjunto c, T e){
        NodoHash * l = c->tabla[h(e)%c->cota];
        while (l!=NULL && l->dato!=e)
                l = l->sig;
        return l!=NULL;
}
```

Segundo Parcial Programación 2

Noviembre de 2022

```
T mínimo (Conjunto c){ // Pre: c no vacío
        nodoHash * min = NULL;
        nodoHash * l;
        for (int i=0; i<c->cota; i++){
                l = c->tabla[i];
                while (l!=NULL){
                        if (min==NULL || l->dato<min->dato)
                                min = l;
                }
        }
        return min->dato;
}
```

2-b)

Bajo las condiciones establecidas, la operación **pertenece** tiene tiempo de ejecución promedio O(1) (la lista en cada posición de la tabla tiene en promedio un nodo), en tanto que la operación **minimo** tiene O(*n*), siendo *n* la cantidad de elementos, ya que debe recorrer toda la tabla (todas las listas) para encontrar el menor (en la tabla de hash no hay un orden). Por otra parte, ambas operaciones tienen O(*n*) peor caso: **minimo** por las razones ya expuestas y, **pertenece** si se tiene un mal diseño del hash (la función ***h*** no dispersa bien y/o la tabla de hash posee un valor muy pequeño en relación a la cantidad de elementos).