postdata9
www.wuolah.com/student/postdata9

41636

stlejercicios.pdf

Exámenes resueltos

2° Estructuras de Datos

Grado en Ingeniería Informática

Escuela Técnica Superior de Ingenierías Informática y de Telecomunicación
Universidad de Granada

WUOLAH

El más PRO del lugar puedes ser Tú.

¿Quieres eliminar toda la publi de tus apuntes?

¡Fuera Publi!
Concéntrate al máximo

Apuntes a full.
Sin publi y sin gastar coins

Para los amantes de la inmediatez, para los que no desperdician ni un solo segundo de su tiempo o para los que dejan todo para el último día.

Quiero ser PRO

4,95 / mes

EJERCICIOS EXAMEN STL

1. [Junio 2003] Dada una pila P de enteros, escribir una función (usando el TDA pila) que transforme dicha pila en otra en la que los elementos aparezcan en orden inverso al original, y habiendo sido eliminados los elementos que siendo consecutivos aparezcan repetidos. Ejemplo:

P = <1,1,2,3,3,4,5,5,1,1,9,8,7,7,3>
P* = <3,7,8,9,1,5,4,3,2,1>

```
void func1(stack<int> &p1){

	stack<int> p2;

	while(!p1.empty())
		//cuando p2 esté vacía, inserta el top de p1
		if(p2.empty())
			p2.push(p1.top());
			cout << p2.top() << ' ';  // la muestra

		//si el tope de p1 != el tope de p2, lo inserto en p2
		if(p1.top() != p2.top())
			p2.push(p1.top());
			cout << p2.top() << ' ';  // la muestra
		p1.pop();
	p1 = p2;
}
```

2. [Septiembre 2003] Construir una función Reemplazar que tenga como argumentos una pila con tipo de elementos int y dos valores (nuevo y viejo) de forma que si el segundo valor aparece en la pila, este se cambiará por el nuevo valor.

```
bool Reemplazar(stack<int> &p, int nuevo, int viejo){
	bool rem = false;
	stack<int> aux;

	// cout << "Primer while: " << endl;

	while(!p.empty()){
		if(p.top() == viejo){
			// cout << "p.top del if " << p.top() << endl;
			p.pop();
			aux.push(nuevo);
			rem = true;
		}else{
			// cout << "p.top " << p.top() << endl;
			aux.push(p.top());
			p.pop();
		}
	}

	// cout << "Segundo while: " << endl;
	while(!aux.empty()){
		// cout << "aux.top " << aux.top() << endl;
		p.push(aux.top());
		aux.pop();
	}

	return rem;
}
```

3. [Junio 2004] Implemente una función que compruebe si hay en una pila el mismo número de paréntesis abiertos que cerrados.

```
bool mismoNum(stack<char> p1){
	bool igual_num = false;
	int tam = p1.size(), p_a = 0, p_c = 0;
	stack<char> p2;

	for(int i = 0; i < tam; i++)
		if(p1.top() == ')')
			p_a++;
		if(p1.top() == '(')
			p_c++;
		p1.pop();

	return p_a == p_c ? true : false;
}
```

4. [Septiembre 2005] Dada una lista de enteros, especificar e implementar una función que modifique la lista invirtiendo el orden relativo de los elementos que sean primos.
Entrada: <1,4,5,6,3,12,15,17>
Salida: <17,4,3,6,5,12,15,1>

```
bool esPrimo(int n){
    int i = 2;      //para i = 1 siempre es primo
    while(i < n)
        //si encuentra algún número por el cual sea divisible → NO es primo
        if(n % i == 0)
            return false;
        i++;
    return true;
}

void fun(list<int> &l){

    list<int>::iterator it;
    it = l.begin();
    list<pair<int,list<int>::iterator>> lista_auxiliar;

    //inserto en una lista los números primos y su iterador en la lista l
    while(it != l.end())
        if(esPrimo((*it)))
            lista_auxiliar.push_back(make_pair((*it),it));
        ++it;

    //iterador que va a iterar desde el principio de la fila hasta el final
    list<pair<int, list<int>::iterator>>::iterator it_p;
    it_p = lista_auxiliar.begin();

    //iterador que va a iterar desde el final hasta el principio de la lista
    list<pair<int, list<int>::iterator>>::iterator it_f;
    it_f = lista_auxiliar.end();
    --it_f;
    list<int>::iterator iterad;

    //mientras it_p no llegue al final
    while(it_p != it_f)

        //iterad es el iterador de la posición en la que se inserta el nuevo elemento
        iterad = l.insert((*it_f).second, (*it_p).first);

//tenemos que borrar el elemento siguiente, ya que no se ha sustituido, sino insertado
        ++iterad;
```

```
        l.erase(iterad);
        ++it_p; --it_f;
}
```

5. [Septiembre 2001] Construir un método de la clase list<T> que permita "peinar" dos listas (intercalar alternativamente todos los elementos que las integran) con los siguientes requisitos:
a) Se empieza por el primer elemento de la lista que invoca el método.
b) La lista que invoca contendrá el resultado final y la que se pasa como argumento quedará vacía.
c) Si una de ellas tiene un número menor de elementos, el exceso de elementos se incorporará a la lista resultante.

```
template <class T>
void list::peinar(list<T> &l2){

	//en esta lista voy a ir guardando el resultado de peinar ambas listas
	list<T> l3;

	int tam1 = this.size(), tam2 = l2.size();
	int tam3 = tam1 <= tam2 ? tam1 : tam2;
	//tam3 va a ser igual al tamaño más pequeño de las 2 listas

	for(int i = 0; i < tam3*2; i++){
		if(i%2 == 0){
			l3.push_back(this.front());
			this.pop_front();
		}else{
			l3.push_back(l2.front());
			l2.pop_front();
		}
	}

	//si tam3 ha tomado como valor más pequeño tam1, l2 es más grande que l1
	if(tam1 == tam3){
		int tam = l2.size();
		for(int i = 0; i < tam; i++){
			l3.push_back(l2.front());
			l2.pop_front();
		}
	}
	else if(tam2 == tam3){
		int tam = this.size();
		for(int i = 0; i < tam; i++){
			l3.push_back(this.front());
			this.pop_front();
		}
	}

	this = l3;
}
```

6. [Junio 2003] Escribir una función usando el TDA Conjunto de enteros para determinar si un conjunto está incluido dentro de otro. La función toma como entrada dos conjuntos c1, c2 y devuelve un booleano indicando si c1 está incluido en c2.

```
template<class T>
bool incluido(const Conjunto<T> &c1, const Conjunto<T> &c2){
	Conjunto<T>::iterator it;

	for(it = c1.begin(); it != c1.end(); ++it){
		if(c2.find((*it)) == c2.end()){
			return false;
		}
	}
	return true;
}
```

7. [Junio 2012] Implementar una función que mezcle dos listas ordenadas de elementos del tipo T en una única lista ordenada, donde no existen elementos repetidos. La mezcla debe hacerse con una eficiencia de O(n) siendo n el número de elementos de la lista.

```
//Supongo que las listas pasadas como argumento están ordenadas y no tienen elementos repetidos
template<class T>
list<T> mezclar(const list<T> &l1, const list<T> &l2){

	list<T> l3;

	list<T>::const_iterator it1;
	it1 = l1.begin();

	list<T>::const_iterator it2;
	it2 = l2.begin();

	//cuando uno de los 2 llegue a end, para (halt)
	while(it1 != l1.end() && it2 != l2.end()){
		if((*it1) < (*it2)){
			l3.push_back((*it1));
			++it1;
		}
		else if((*it1) > (*it2)){
			l3.push_back((*it2));
			++it2;
		}else{
			l3.push_back((*it2));
			++it1;
			++it2;
		}
	}

	//si ha sido el it1 el que ha terminado el while, sigo push l2
	if(it1 == l1.end()){
		while(it2 != l2.end()){
			l3.push_back((*it2));
			++it2;
		}
	}
	else if(it2 == l2.end()){
		while(it1 != l1.end()){
			l3.push_back((*it1));
			++it1;
		}
	}

	return l3;
}


//otra forma mucho más sencilla
list<int> mezclar2(list<int> l1, list<int> l2){

    list<int> l3;

    l3.merge(l1);
    l3.merge(l2);
    l3.unique();

    return l3;
}
```

8. [Septiembre 2012] Se dispone de una secuencia de elementos enteros almacenados en una cola y se desea comprobar si dicha secuencia en orden inverso coincide con alguna subsecuencia de elementos de una pila P.

P.ej. Si la cola es C = <3,8,5>, C = <2,3,8> ó la Pila P = <5,8,3,2,4> sería cierto, pero si la cola fuese p.ej. C1 = <3,8> ó C2 = <4,8,5>, y tuviésemos la misma pila P, sería falso. Usando solamente los TDA Pila y Cola, diseñar una función bool condicion(stack<int> &P, queue<int> &Q) que compruebe si la condición se cumple. Únicamente se puede usar una pila adicional.

```
bool condicion(stack<int> &P, queue<int> &Q){

}
```

9. [Septiembre 2012] Usando el TDA list<int>, construir una función void agrupar elemento (list<int> &entrada, int k) que agrupe de forma consecutiva en la lista de entrada todas las apariciones del elemento k en la lista, a partir de la primera ocurrencia. P.ej. Si entrada = {1,3,4,1,4} y k = 1, entonces entrada = {1,1,3,4,4}, o si, entrada = {3,1,4,1,4,1,1} y k = 1, entonces entrada = {3,1,1,1,1,4,4}.

```
void agrupar_elemento(list<int> &entrada, int k){

    list<list<int>::iterator> lista;
    list<int>::iterator it;

    //almaceno en una lista los iteradores que encuentre
    for(it = entrada.begin(); it != entrada.end(); ++it){
        if((*it) == k){
            lista.push_back(it);
        }
    }

    it = lista.front();
    lista.pop_front();
    int tam = lista.size();     //el tamaño de la lista es el número de ocurrencias que
hay de k

    //insertamos en la posición de la primera ocurrencia,
    //todas las que hayamos encontrado y eliminamos las otras
    for(int i = 0; i < tam; i++){
        entrada.insert(it, k);
        entrada.erase(lista.front());
        lista.pop_front();
    }
}
```

10. [Septiembre 2013] Dada una lista de enteros con elementos repetidos, diseñar (usando el TDA lista) una función que construya a partir de ella una lista ordenada de listas, de forma que en la lista resultado los elementos iguales se agrupen en la misma sublista.
Por ejemplo, si entrada = {1,3,4,5,6,3,2,1,4,5,5,1,1,7},
entonces salida = {{1,1,1}, {2}, {3,3}, {4,4}, {5,5,5}, {6}, {7}}.

```
void func(list<int> l){

    l.sort();

    list<list<int>> ll;     //lista en la que se devuelve el resultado
    list<int> aux;          //lista auxiliar

    //añado en la lista auxiliar el primer elemento de l para poder empezar el while
    aux.push_back(l.front());
    l.pop_front();

    while(!l.empty()){
        //si son distintos, inserto en l1 la lista auxiliar y la vacío
        if(aux.front() != l.front()){
            ll.push_back(aux);
            aux.clear();
        }
        aux.push_back(l.front());
        l.pop_front();
    }
    //la última lista no se inserta, ya que cuando l está vacía sale del while
    //y no hace ll.push_back, por ello hay que realizarlo fuera del bucle
    ll.push_back(aux);

    list<list<int>>::iterator it_l;
    it_l = ll.begin();

    list<int> l_aux;    //lista auxiliar
    list<int>::iterator it_aux;

    cout << "Lista de listas: " << endl;
    while(it_l != ll.end()){
        l_aux = (*it_l);
        it_aux = l_aux.begin();

        cout << "{" << (*it_aux);
        ++it_aux;
        while(it_aux != l_aux.end()){
           cout << ", " << (*it_aux);
           ++it_aux;
        }
        cout << "} ";
        ++it_l;
    }
    cout << endl;
}
```

11. [Febrero 2013] Dadas dos listas de intervalos cerrados [ini,fin], L1 y L2, donde para cada lista los intervalos se encuentran ordenados por orden de tiempo de inicio satisfaciendo que si un intervalo it1 está antes que otro it2 en la lista entonces it1.fin < it2.ini.
Diseña un método que permita seleccionar un (sub)intervalo de L1 y lo pase a L2. En este proceso podría ser necesario el dividir un intervalo de L1 así como fusionar intervalos en L2 cuando ocurran solapamientos.

```
Typedef pair<int,int> intervalo;
bool extraer(list<intervalo> &L1, intervalo x, list<intervalo> &L2);
```

Devuelve true si ha sido posible realizar la extracción (x debe pertenecer a un único intervalo de L1). Por ejemplo, si
L1 = [1,7], [10,14], [18,20], [25,26]; L2 = [0,1], [14,16], [20,23] y x = [12,14]
entonces las listas quedarán como L1=[1,7],[10,22],[25,26] y L2=[0,1],[12,16],[20,23].

De igual forma, si L1 = [1,7],[10,22],[25,26]; L2 = [0,1],[14,16],[20,23] y x = [12,20]
entonces las listas quedarán como L1=[1,7],[10,11],[21,22],[25,26] y L2=[0,1],[12,23].

12. [Diciembre 2010] Considérese el TDA vector_disperso<T>, que representa un vector matemático instanciado para un tipo de dato numérico T. En estos vectores, la mayoría de componentes del vector son nulas, y sólo unas pocas tienen valor distinto de cero, por lo que es recomendable utilizar una representación que sólo almacene de forma explícita las componentes no nulas. Se pide:

a) Dar una representación para el TDA vector_disperso<T> que tenga en cuenta las observaciones realizadas anteriormente.

Vector disperso: aquel que almacena lo importante de un vector. Ejemplo: si tiene todo ceros y 3 números, sólo guarda los números.

Dado el siguiente vector su

0	0	1	0	0	2
0	1	2	3	4	5

vector disperso sería:

<2,1>	<5,2>

es decir, en la posición 2 hay un 1, en a posición 5 hay un 2, todo lo demás es 0.

Una buena representación para el vector disperso, sería un vector de pares de entero y T, esto es:

```
template <class T>
class vector_disperso<T>{
      private:

            //el int es el índice del vector, y T lo que hay
            map<int, T> datos;

            //número de elementos del vector
            int n;
}
```

En el ejemplo anterior, el vector sería el mostrado y n = 6.

b) Implementar la operación vd<T> vd<T>::operator+(const vd<T> &v) que calcula y devuelve la suma de 2 vectores.

```
vd<T> vd<T>::operator+(const vd<T> &v){

	vd<T> w;
	map<int, T>::iterator it_d;
	map<int, T>::iterator it_v;
	int tam = this.n < v.n ? this.n : v.n;

	for(int i = 0; i < tam; i++){
		it_d = datos.find(i);	//buscamos la posición en el mapa en ambos vd
		it_v = v.datos.find(i);

		//si ambos son distintos de end, hemos encontrado en ambos
		if(it_d != datos.end() && it_v != v.datos.end())
			w[i] = (*it_d).second + (*it_v).second;

		//si it_d no es end, hay un elemento en la posición i
		else if(it_d != datos.end())
			w[i] = (*it_d).second;

		//si it_v no es end, hay un elemento en la posición i
		else if(it_v != v.datos.end)
			w[i] = (*it_v).second;
	}

	if(this.n > v.n){
		for(int i = tam; i < this.n; i++){
			it_d = datos.find(i);

			if(it_d != datos.end()){
				w[i] = (*it_d).second;
			}
		}
	}
	else if(v.n > this.n){
		for(int i = tam; i < this.n; i++){
			it_v = v.find(i);

			if(it_d != v.datos.end()){
				w[i] = (*it_v).second;
			}
		}
	}
}
```

c) Implementar la operación void asigna(int i, const T &c), que asigna el valor c a la componente i-ésima del vector).

```
void asigna(int i, const T &c){

	map<int,T>::iterator it;
	it = datos.find(i);

	datos.erase(it);
	datos[i] = c;
}
```

13. [Diciembre 2011] En un banco se almacena información actualizada sobre las cuentas bancarias, identificadas por un entero y su saldo real. El saldo de dichas cuentas se ve modificado por movimientos, identificados por un entero y que pueden ser de cantidades reales positivas o negativas. Realice las siguientes tareas:

a) Proponga la representación idónea para la ed banco de modo que las operaciones de consultar saldo de una cuenta, insertar/borrar/modificar un movimiento se realicen en un tiempo O(log2(n)) manteniendo el saldo de la cuenta actualizado en todo momento.

```
class Banco{
	private:
		map<int, ;

	public:
}
```

b) Implementa el método: void banco::modificar_movimiento(int nmov, float cant) considerando que el saldo de la cuenta asociada tiene que actualizarse.

14. [Diciembre 2011] Dado un tipo de dato vector_mejorado<T> almacenado en memoria como un vector<T> de tamaño fijo pero que permite inserciones y borrados del primer y último elemento en O(1). Realice las siguientes tareas:

a) proporcione la representación completa para dicha ed,
b) implemente el método vector_mejorado<T>::full() que indique si el vector está lleno,
c) proporcione una representación para vector_mejorado<T>::iterator e,
d) implemente el método vector_mejorado<T>::iterator::operator++ para dicha representación.

15. [Diciembre 2011] Implemente una función miembro de la clase palabras con cabecera void palabras::insertar(string dato) y que coloca juntos en una misma posición, de una lista de listas lis<list<string>> datos, todas aquellas cadenas que tengan la misma longitud. No se debe guardar más de una vez en la misma cadena.

16. [Septiembre 2012] Un departamento comercial dispone de N productos. Se pretende implementar un sistema capaz de gestionar a M usuarios. Para cada usuario se ha de almacenar los productos que este ha adquirido. Se supone que tanto el usuario, como el producto vienen identificados por dos etiquetas de tipo string así como que un usuario suele comprar un número de productos distintos mucho menor que N. Se pide:

a) Proponer dos representaciones distintas para un tipo de dato sistema capaz de gestionar dicha información de forma eficiente. Para cada representación indicar su función de abstracción e invariante de la representación. Analizar las ventajas y desventajas de las representaciones propuestas.

b) Seleccionar una de ellas e implementar los siguientes métodos,

void sistema::comprar(const string &user, const string &prod) que en orden O(log(M)) almacena que el usuario user ha adquirido el producto prod.

String sistema::superventas() que devuelve el nombre del producto que ha sido adquirido más frecuentemente por los usuarios del sistema.

17. [Febrero 2013] Una empresa mantiene datos sobre personas(DNI, apellidos, nombre, domicilio y teléfono, todos tipo string) y necesita poder hacer búsquedas sobre ellos tanto por dni como por apellidos. Define una representación eficiente (discutiendo alternativas) para manejar los datos de esta empresa e implementa las operaciones de inserción y de borrado de una persona en dicha estructura. Las cabeceras serían:

```
void datos::insertar(const persona &p)

void datos::borrar(const persona &p)
```