UNIVERSIDAD DE GRANADA

Departamento de Ciencias de la Computación e Inteligencia Artificial

Reto 3: TDA Lineales

J. Fdez-Valdivia
Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Estructuras de Datos

Grado en Ingeniería Informática
Doble Grado en Ingeniería Informática y Matemáticas
Doble Grado en Ingeniería Informática y ADE

Resolver 5 de los siguientes ejercicios:

1. Implementar una función:

   int ppt(list<int> &H, int n);

   que reciba una lista de enteros que representan todas las elecciones previas del jugador oponente en el juego piedra-papel-tijera y devuelve la siguiente más probable nextplay. La lista solo tiene los enteros 1, 2 y 3, que se corresponden con las elecciones de “piedra”, “papel” y “tijera” respectivamente. El último elemento de la lista corresponde a la última jugada. El objetivo es tratar de predecir la próxima jugada nextplay del oponente. Para ello debe buscar en su historial H todas las veces que el jugador jugó la misma secuencia que en las últimas n partidas y retornar el valor siguiente a dicha vez que apareció la secuencia.

   Por ejemplo, si el historial es H=(1,1,2,2,1,2,3,3,2,1,2,3,1,2) y n=2 la función debería retornar 3. Las últimas dos jugadas fueron last=(1,2), y la anterior vez que jugó (1,2) eligió a continuación nextplay=3.

   En caso de que la secuencia last no aparezca debe retornar nextplay=0.

   Sugerencia:

   Copiar en una lista auxiliar last las últimas n jugadas. Recorriendo H para cada posición p verificar si a partir de p se encuentra una copia de last. Si se encuentra y hay al menos un elemento adicional en H quedarse con ese candidato nextplay. De todos los posibles nextplay quedarse con el último.

   Notas:

   Tener en cuenta que dos apariciones de la secuencia buscada pueden solaparse, por ejemplo si H=(1 2 1 2 1 2 3 1 2 1 2) y n=4 tenemos las secuencias (1 2 1 2 1) y (1 2 1 2 3). Se recomienda utilizar un iterador p que va recorriendo la lista y para cada p recorrer con otro iterador q desde p hacia atrás. Tener en cuenta que al final de H aparece la secuencia buscada, pero no debe tenerse en cuenta porque no tiene un elemento siguiente.

2. Implementar una función:

   bool contenido(list <list<int> > &LL, list <int> &L);

   que dadas una lista de listas LL y una lista L, devuelva true si cada elemento de L está contenido en una y sólo una lista de LL. Si un elemento de L no se encuentra en ninguna lista contenida en LL, deberá retornar false.

   Ejemplos: Considerando LL=((1 2 3 4) (1 2 3 5) (1 2 3 6)).

   Si L=(6 5 4) -> true ya que 6 está sólo en la lista L2, 5 en L1 y 4 en L0

   Si L=(6 5) -> true ya que 6 está sólo en la lista L2, 5 en L1. No hay ningún elemento único para la lista L0 pero no importa.

   Si L=(6 5 1) -> false, ya que 1 está en todas las listas.

   Si L=(6 5 3) -> false, ya que 3 está en todas las listas.

   Si L=(6 5 3 7) -> false, ya que 3 está en todas las listas y 7 en ninguna.

3. Considere el problema de generar todas las **subsecuencias ordenadas** de la secuencia X = (1, 2, ..., n). Por ejemplo, si n = 4 las subsecuencias ordenadas de X = (1, 2, 3, 4) son: (1), (12), (123), (124), (13), (134), (14) (2), (23), (234) (24), (3), (34) y (4). Esta construcción se puede implementar mediante el uso de una pila S bajo las siguientes reglas:

- Inicializar la pila con el elemento 1.

- Si el tope t de la pila verifica t < n entonces apilamos t + 1.

- Si t = n, entonces lo desapilamos y, a continuación, si la pila no quedara vacía

  incrementamos el nuevo tope de la misma.

El algoritmo termina cuando la pila queda vacía.

Ejemplo:

```
        4
     3 3 4    4              4
   2 2 2 2 3 3 4          3 3 4    4
 1 1 1 1 1 1 1 1        2 2 2 2 3 3 4
```

4. Implementar una función:

void lexico_stack(int & n);

que dado un número natural n imprime todos los conjuntos ordenados de (1, 2, ..., n). **Sugerencia:** Implementar el algoritmo descripto, llamando a una función auxiliar

void imprime_pila(stack<int> &S)

que imprime la pila S en forma no-destructiva.

Restricciones: Usar la STL para pilas. En lexico_stack(): usar una sola estructura auxiliar. En imprime_pila(): usar una sola estructura auxiliar. No usar otros algoritmos de la STL.

5. Se necesita llenar una mochila, incapaz de soportar más de un peso determinado K, con todo o parte de un conjunto de objetos, cada uno con un peso y valor específicos. Los objetos colocados en la mochila deben maximizar el valor total sin exceder el peso máximo. En esta versión simplificada, se supone que cada objeto tiene el mismo valor, pero pueden tener diferentes pesos. Por lo tanto se trata de elegir la mayor cantidad de objetos sin exceder K. El peso de los objetos (enteros positivos) está en una lista P.

Implementar una función

list<int> mochila(list<int> &P, int K);

que reciba una lista P con el peso de cada uno de los objetos que se podrían llevar en la mochila y un peso máximo permitido K y que devuelva una lista M con una lista con un subconjunto de los elementos de P cuya suma es <=K y tiene la máxima cantidad de elementos. Si hay varias posibilidades debe retornar cualquiera de ellas.

Ejemplos:

Si P=(3,2,7,11,2,5,9,4), K=11 -> M=(3,2,2,4)

Si K=10 -> M=(3,2,2) o M=(4,2,2) o M=(5,2,2)

Si K=12 -> M=(2,2,3,4)

Sugerencia:

Escribir el algoritmo en forma recursiva. Si K<=0 o P=() entonces la solución es la lista vacía. Tomar el primer elemento a0 de P y formar la lista Pm0=P-(a0) es decir P pero sin el elemento a0. Considerar dos casos, que a0 esté o no en M. Si a0>K no hace falta considerar la

posibilidad de que a0 esté en M. En el caso que a0 NO esté en M llamar a mochila(Pm0,K). En el caso que a0 SI esté en M la suma de los elementos de Pm0 debe sumar K-a0 de manera que se debe llamar a mochila(Pm0,K-a0). A la M resultante debe agregársele a0. De las dos posibilidades quedarse con la que tiene la mochila con más elementos.

Notas:

Los elementos en M pueden estar en cualquier orden.

Puede haber elementos iguales en P.

Puede darse K==0 en ese caso debe retornar M=().

6. Dado un vector de listas VL y una lista L, implementar una función:
   bool iscomb (vector<list<int>>> VL, list<int> L);

   que devuelva true si L es una combinación de las listas de VL en alguna permutación dada. Cada una de las VL[j] debe aparecer una y sólo una vez en L. Por ejemplo:

   si VL=[(1,2,3),(4,5,6),(7,8)], y L=(7,8,4,5,6,1,2,3) entonces iscomb(VL,L)->true. Pero si

   L=(3,2,1,7,8,4,5,6)->false.

   Sugerencia:

   Escribir una funcion bool isprefix(Lpref,L); que retorna true si la lista Lpref es prefijo de L. Primero chequear que la longitud de L debe ser igual a la suma de las longitudes de los VL[j]. Si pasa este test se procede en forma recursiva. Para cada VL[j] se determina si es prefijo de L. Si ninguna de ellas es prefijo entonces retorna false. Si una (o varias de ellas) lo es entonces se aplica recursivamente iscomb() para determinar si L-VL[j] es una combinación de VL-VL[j]. Es decir se quita VL[j] del comienzo de L y la posicion j de VL. La recursión se corta cuando L es vacía.

7. Implementar una función:
   bool is_sublist(list<int> &L1, list<int> &L2, list<list<int>::iterator> &iters)

   que dadas dos listas listas de enteros L1,L2 devuelve true si la lista L2 es una sublista de L1, es decir que L2 se puede obtener a partir de L1 sólo eliminando algunos de sus elementos. Por ejemplo:

   si L1=(3,2,6,5,4,1,8) y L2=(3,6,5,1), entonces is_sublist(L1,L2,iters) debe retornar true, mientras que si L2=(3,6,1,5) entonces debe retornar false ya que los elementos 1 y 5 están invertidos con respecto a su posición en L1.

   Adicionalmente, en el caso que si sea sublista debe devolver a través de iters las posiciones (iteradores) que corresponden a los elementos de L1 que están en L2. En el ejemplo L1=(3,2,6,5,4,1,8) y L2=(3,6,5,1) debe devolver true y en iters 4 posiciones en la lista L1 correspondientes a los elementos 3,6,5,1. El algoritmo debe ser O(n)

   Sugerencia: Para cada elemento de L1 verificar si se encuentra en L2 siempre partiendo desde la última posición en que se encontró el elemento anterior.

8. Implementar una función:
   bool nullsum(list<int> &L, list<int> &L2);

   que dada una lista L, devuelve true si hay un rango de iteradores [p,q)) tal que su suma de los elementos en el rango sea 0. En caso afirmativo debe retornar además el rango (uno de ellos, porque puede no ser único) a través de L2. En caso negativo L2 debe quedar vacía. El algoritmo debe ser como mucho cuadrático.

Por ejemplo: si L=(-10, 14, -2, 7, -19, -3, 2, 17, -8, 8) entonces debe retornar true y

L2=(14, -2, 7, -19), ó L2=(-8, 8). Si L=(1,3,-5) entonces debe retornar false y L2=().

9. Implementar una función:

void reordena(stack<int> &P);

que reordena los elementos de una pila de tal forma que quedan los elementos impares queden en el fondo y los que si lo satisfacen arriba.

Por ejemplo si P = {1, 3, 4, 2, 3, 5, 7, 6, 8, 2, 9} entonces debe quedar P = {4, 2, 6, 8, 2, 1, 3, 3, 5, 7, 9}.

Restricciones: el algoritmo debe ser estable, es decir los elementos pares deben quedar en el mismo orden entre sí, y lo mismo para los impares. Se puede solo una estructura auxiliar de tipo cola.

10. Implementar una función

int stablepartition (int x, vector<int> & A);

que, dado un entero x y un vector de n enteros A, A= (A0,A1,...,An−1) reordene el contenido del vector alrededor de un índice k de forma que los elementos del subvector A[0..k] sean menores o iguales que x, y los elementos del subvector A[k+1..n−1] sean mayores que x, y devuelva ese índice k (k=-1 si todos los elementos de A son mayores que x). Además, ha de respetarse el orden relativo de los elementos en el vector, es decir:

• Si a[i] y a[j] with i<j son ambos menores o iguales que x, entonces en el vector final A[i] estará antes que a[j], y ambos estarán situados en la "parte" a[0..k] que contiene los elementos ≤x;

• Si a[i] y a[j] with i<j son ambos mayores que x, entonces en el vector final A[i] estará antes que a[j], y ambos estarán situados en la "parte" A[k+1..n−1] que contiene los elementos >x;

Por ejemplo si x=2 y el vector es A= (1,5,3,0,4), la función lo modifica y convierte en A=(1,0,5,3,4), y devuelve k=1. El costo de la función ha de ser como mucho O(n).

11. Un tipo de dato pila_doble es un tipo que permite insertar y borrar por ambos extremos. Implementar las operaciones push y pop de este tipo de dato usando como representación un vector. Las operaciones tienen un parámetro que indica la pila sobre la que se hacen las operaciones:

a) **void pila_doble<T>::push(int numpila, const T & elem)**

b) **void pila_doble<T>::pop(int numpila)**

12. Implementar una función:

void expand(list<int> &L,int m);

que transforme los elementos de una lista L de tal forma que todos los elementos de L resulten ser menores o iguales que m, pero de tal forma que su suma se mantenga inalterada. Para esto, si un elemento x es mayor que m entonces, lo divide en tantas partes como haga falta para satisfacer la condición; por ejemplo si m=3 podemos dividir a 10 en 3,3,3,1. Es decir si L=(7,2,3,1,4,5), entonces después de hacer expand(L,2) debe quedar L=(2,2,2,1,2,2,1,1,2,2,2,2,1).

13. Implementar una función:

stack<int> mergeSortedStacks(stack<int> A, stack<int> B);

que dadas 2 pilas ordenadas (el mínimo valor está en el tope) devuelva una pila ordenada (el mínimo en el tope) que contenga todos los elementos de A y B. No pueden usarse estructuras auxiliares salvo pilas.

14. Implementar una función:

void compacta(list<int> &L,stack<int> &S);

que va tomando un elemento entero n de de la pila S y, si es positivo, saca n elementos de L y los reemplaza por su suma. Esto ocurre con todos los elementos de S hasta que se acaben, o bien se acaben los elementos de L. No pueden usarse estructuras auxiliares.

Por ejemplo: Si L=(1,3,2,1,4,5,3,2,4,1) y S=(3,2,-1,0,2,5,2,-3) entonces L debe quedar así L=(6,5,8,7), y S=(2,-3) (es decir, sobran elementos de S).

Otro ejemplo: Si L=(1,3,2,1,4,5,3,2,4,1,3,2,1,4,7) y S=(3,2,-1,0,2,5) entonces L debe quedar así L=(6,5,8,12,1,4,7), y S=() (es decir, sobran elementos de L).

15. Implementar una función:

void mayorar(list<int> &L1,list<int> &L2);

que modifica las listas L1 L2 de tal manera que si a1j , a2j son los elementos de L1 y L2 antes de aplicar la función y a′1j , a′2j los elementos después de aplicar la función, entonces a′1j = max(a1j , a2j), a′2j = mın(a1j , a2j). Si las listas no tienen la misma longitud, entonces los elementos restantes quedan inalterados.

Ejemplo:

si L1=(14, 0, 6, 13, 11, 12, 3, 17, 14, 18) y

L2=(6, 4, 4, 11, 12, 15, 8, 17, 18, 11, 23, 1, 2, 5, 15)

entonces después de hacer mayorar(L1,L2) debe quedar:

L1=(14, 4, 6, 13, 12, 15, 8, 17, 18, 18) y L2=(6, 0, 4, 11, 11, 12, 3, 17, 14, 11, 1, 2, 5, 15).

Sugerencia: Recorrer ambas listas con dos posiciones e ir intercambiando los elementos si a1j < a2j .

Restricciones: Usar la STL para listas. No usar el operador --. No usar ninguna estructura auxiliar. El algoritmo debe ser O(n). Cuidado de no usar posiciones inválidas al iterar sobre las listas.

16. Implementar las funciones de insertar y borrar elementos del TDA **list<int> a partir de** las funciones del TDA **stack <int>**

17. Implementar una función:

bool encuentra(list<int> &L1, list<int> &L2, list<int> &indx);

que devuelve true o false dependiendo de si L1 es una sublista o no de L2. En caso de que si lo sea, retorna en indx los índices de los elementos de L2 que forman L1, si no indx debe retornar vacía, independientemente de lo que contenía previamente.

Por ejemplo, si L2={13, 9, 8, 12, 9, 6, 12, 2, 9, 14, 18, 10} y L1={13, 9, 9, 6, 2, 14} entonces encuentra debe retornar true, e indx={0, 1, 4, 5, 7, 9}. Si L1={8, 9, 13} debe retornar false e indx={}.

Nota: Los índices en indx deben ser estrictamente crecientes. No pueden usarse estructuras auxiliares. El tiempo de ejecución del algoritmo debe ser O(n).

18. Pasar (especificando los pasos seguidos) la siguiente expresión de postfijo a prefijo usando el TDA lineal que creas más apropiado: { a b c + / e f * g h * - - }
19. Dada una pila P de enteros implementar una función

void transformarpila(stack<int> & p);

que transforme una pila p en otra en la que los elementos aparezcan en el mismo orden 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> pasaría a quedar como P = <1,2,3,4,5,1,9,8,7,3>

20. Implementar una función

void rotación(queue<int> & C);

que saque una cierta cantidad de enteros del frente de la cola C y los vuelve a insertar al final de cola, de forma que quede en el frente el primer número par que haya en la cola. Por ejemplo, si

C={1,3,5,**2**,4} ====> C={2,4,1,3,5}

21. Una matriz NxN de enteros se dice que está bi-ordenada si cada columna tiene los elementos ordenados de forma no decreciente de arriba abajo, y cada fila tiene los elementos también ordenados de forma no decreciente de izquierda a derecha .

Ejemplo:

1	4	9
9	9	9
12	14	15

(a) Implementar una función:

bool esmatrizbiordenada (vector<vector<int>> & M);

que devuelva true si M es ua matriz bi-ordenada
(b) Implementar de forma eficiente una función

22. Implementar una función

void intercambia_sec (list<int> L);

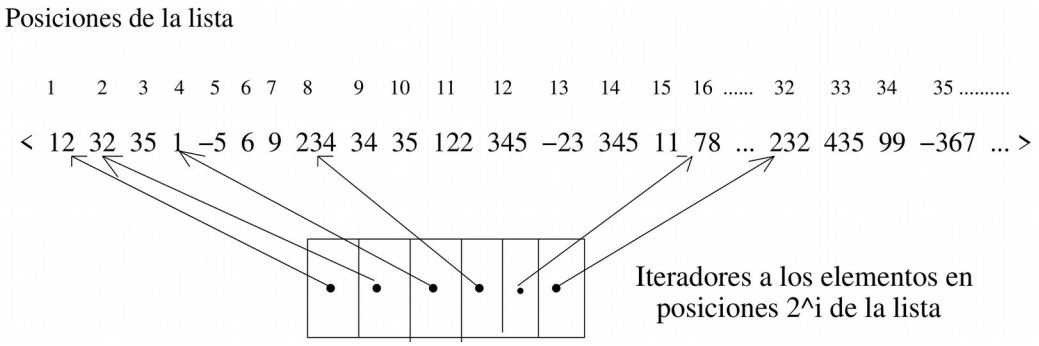
que dada una lista L, intercambia el grupo de los primeros elementos consecutivos impares por el siguiente grupo de elementos consecutivos pares y así sucesivamente.

Por ejemplo si L={1,2,4,5,6,8,7,9,13,2} despues de llamar a intercambia_sec (L) debe quedar L={2,4,1,6,8,5,2,7,9,13}

Restricciones: Puede usarse una sola estructura auxiliar de tipo cola y el algoritmo debe ser O(n)

23. Se define una **skiplist** como una lista que además ofrece un mecanismo para acceder rápidamente a los elementos de una forma indexada usando un entero. El mecanismo podría consistir en una estructura de datos asociada a la lista que almacena cada uno de los

iteradores correspondientes a una serie de posiciones concretas de la lista (p.ej. posiciones que corresponden a los elementos que se encuentran en lugares que ordinalmente sean potencias de 2). Un ejemplo de skiplist con los iteradores asociados a las posiciones {1, 2, 4, 8, 16, ….} de la lista podría ser:



(a) Definir una representación eficiente para el TDA skiplist
(b) Implementar la siguiente función:

void insert (skiplist<int> L, int pos, const int & elem);

que inserta un dato *elem* en una posición *pos* (0 <= pos <= L.size()) de la skiplist *L*.
(c) Implementar una clase iteradora que itere sobre los elementos asociados a las posiciones {1, 2, 4, 8, 16, ….} de la lista. Se deben implementar también las funciones begin() y end().

24. Implementar una función

void postintercalar(list<int> & L1, list<int> & L2);

que permita "postintercalar" dos listas de enteros (intercalar alternativamente todos los elementos que las integran **de final a principio**), con los siguientes requisitos:
(a) Hay que comprobar que las dos listas L1 y L2 no son vacías
(b) Se empieza por el primer elemento de la primera lista L1
(c) La segunda lista L2 contendrá el resultado final y la primera quedará vacía.
(d) Si una de ellas tiene un menor número de elementos que la otra, el exceso de nodos se incorporará a la lista resultante.
Ejemplo:
Primera lista, L1 : (100,200); Segunda lista. L2: (1,2,3,4,5,6)
Despues de aplicar la función:
L2: (1,2,3,4,**200**,5,**100**,6); L1: vacia

25. Supongamos que representamos una lista usando un vector de la siguiente forma:

```
class listacursores {
   private:
     struct dato {
                 char elem;
                 int siguiente
                };
     vector<dato> elementos;
     int primero;
     int nelems;
   public:
.......
 };
```

donde el campo "elem" es el elemento de cada posición de la lista, y "siguiente" indica la posición dentro del vector en que está el siguiente elemento de la lista.
Ejemplo:

El vector:

0	1	2	3	4
a,4	**d,5**	**b,3**	**e,1**	**c,2**

con:

nelems=5; primero=0 representa la lista (en orden) L: **<a, c, b, e, d>**

Dada dicha representación de listas donde la posición de cada elemento viene determinada por un número entero, construir una función que imprima los elementos de la lista. Han de usarse iteradores, de forma que los elementos listados por el iterador deben aparecer en el orden en que están en la lista (independientemente de como estén almacenados en el vector) y para usarlos correctamente, deben construirse los operadores * y ++ junto con las funciones begin() y end(). No pueden usarse estructuras de datos auxiliares.

26. Implementar una funcion:

void flota_pares(stack<int> &P);

que reordena los elementos de una pila P de tal manera que los pares quedan arriba de los impares. Los elementos pares deben quedar en el mismo orden entre sı, y lo mismo para los impares. Por ejemplo, si P=(4,17,9,7,4,2,0,9,2,17) entonces después de hacer flota_pares(P) debe quedar P=(4,4,2,0,2,17,9,7,9,17). Usar dos pilas auxiliares.

Restricciones: No usar más estructuras auxiliares que pilas y solo pilas y la función debe ser O(n).

27. Escribir una función

void creciente(queue<int> &Q);

que elimina elementos de Q de tal manera de que los elementos que quedan estén ordenados en forma creciente.

Por ejemplo, si Q={5,5,9,13,19,17,16,20, 19, 21} tras aplicar la función, debe quedar Q={5,5,9,13,19,20,21}

Idea: Ir eliminando los elementos de la cola Q y poniéndolos en una cola auxiliar Q2 solo si el elemento es mayor o igual que el máximo actual. Finalmente, se trasladan los elementos de Q2 a Q

28. Supongamos que tenemos el siguiente segmento de código:

```
{int n;
 stack<int> p(n);
 for (i=1; i<=n; ++i)
       if (test(i)) cout << i << " ";
       else p.push(i);
 while(!p.empty())
       i=p.top();
       p.pop();
       cout <<i;
 }                    // test es una función booleana.
```

Dada una secuencia de valores entre 1 y n ¿Cómo identificarías si es una secuencia que ha salido de la ejecución del código anterior?

29. Dada una lista de enteros L con elementos **repetidos**, implementar una función:

list<list<int>> agrupariguales <const list<int> & L);

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.
Ejemplo:
L={1,3,4,5,6,3,2,1,4,5,5,1,1}, **salida=**{ {1,1,1},{2},{3,3},{4,4},{5,5,5,},{6} }

30. Implementar una función:
bool detectaErrores(string pal);

que tome una cadena de caracteres en la que aparecen paréntesis, corchetes y llaves junto con otros caracteres. Se trata de comprobar la correcta colocación balanceada de dichos símbolos. La función devuelve true si la cadena está balanceada y falso si no. Por ejemplo, la cadena {[a*(b+4)] / (c*5)} está balanceada.

31. Implementar una función:

template <class T> void duplicarlista (list<T> & L);

que permita "duplicar" una lista L (intercalar alternativamente tras cada elemento en la posición i, el elemento que está en la posición n-i-1 (i=0,1,...,n-1).
Ejemplos:
Lista inicial: (a,b,c,d) Lista final: (a,d,b,c,c,b,d,a)
Lista inicial: (1,2,3,4,5) Lista final: (1,5,2,4,3,3,4,2,5,1)

32. Implementar una función:
void rotalista(list<int> & L, int n);

que traslada los primeros n elementos de la lista al final de la misma. Por ejemplo, si L={1,3,5,4,2,6}, entonces rotalista(L,2) la deja en L={5,4,2,6,1,3}.

33. Implementar una función

int dminmax (list<int> & L);

que, dada una lista L devuelva la distancia entre las posiciones del mínimo y del máximo de la lista. La distancia debe ser positiva si el mínimo está antes del máximo y negativa en caso contrario. Por ejemplo, si L={5,1,3,2,4,7,6} entonces dminmax(L) debe retornar 4, ya que las posiciones del mínimo y máximo son 2 y 6, respectivamente. Para L={5,9,3,2,4,1,6} debe retornar -4. Si el valor del mínimo aparece repetido se debe tomar la primera posición, mientras que para el máximo se debe tomar la ultima. Por ejemplo, para L={7,5,1,3,2,1,4,7,6} entonces debe retornar 5. No pueden usarse estructuras auxiliares.

34. Dadas dos listas de enteros L1 y L2, implementar una función

bool lexicord (list<int> L1, L2);

que devuelve true si la lista L1 es mayor que la L2 en sentido “lexicográfico”, y false en caso contrario.

El orden lexicográfico es el orden alfabético usado para ordenar las palabras en el diccionario, si consideramos que la lista de enteros es una “palabra” cuyas “letras” son cada uno de los enteros de la lista, se van comparando los números en las posiciones correspondientes hasta encontrar uno diferente. La lista que tiene el

número (“letra”) mayor es la lista “mayor” en el sentido lexicográfico. Asi, por ejemplo,

L1={1,3,2,**4**,6} L2={1,3,2,**5**} ====> L2 > L1 (devuelve false)

Si una lista está totalmente contenida en otra, e smayor la lista más larga:

L1={1,3,3,4,5} L2={1,3,3} ====> L1 > L2 (devuelve true)

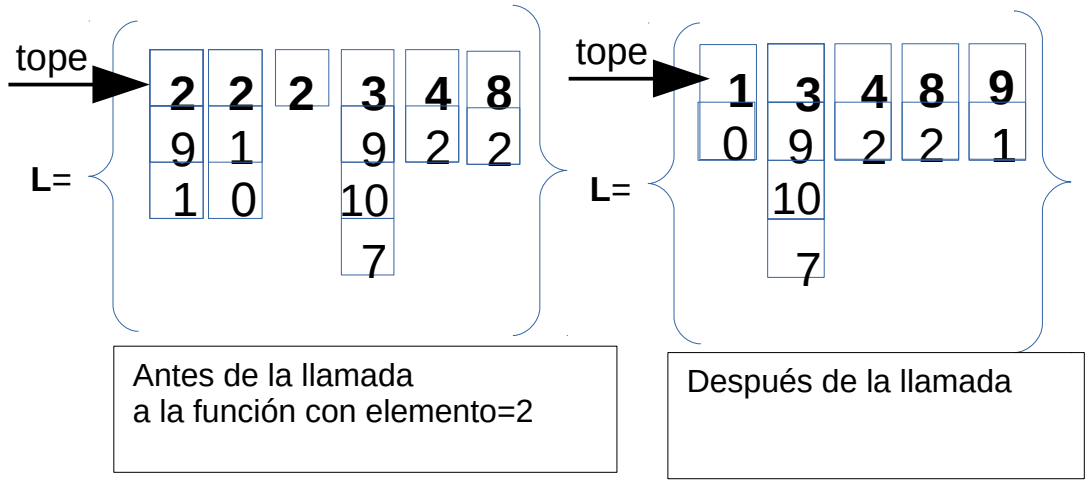
2 listas son iguales solo si tienen la misma cantidad de elementos y los elementos en las posiciones correspondientes son iguales:

L1={1,2,3,4} L2={1,2,3,4} ====> L1 = L2 (devuelve false)

35. Dada una lista que contiene pilas con enteros y que se encuentra ordenada de menor a mayor por el tope de cada pila, Implementar una función **borrar**, que elimina el tope de cada pila en la lista, con valores iguales al dado como parámetro. La cabecera de la función sería:

void Borrar(list<stack<int> > & L, int elemento);

Ejemplo:



Nota: la lista tiene que quedar ordenada por el tope de las pilas trás la ejecución de la función.

36. Implementar una función

bool anagrama (list<int> & L1, list<int> & L2)

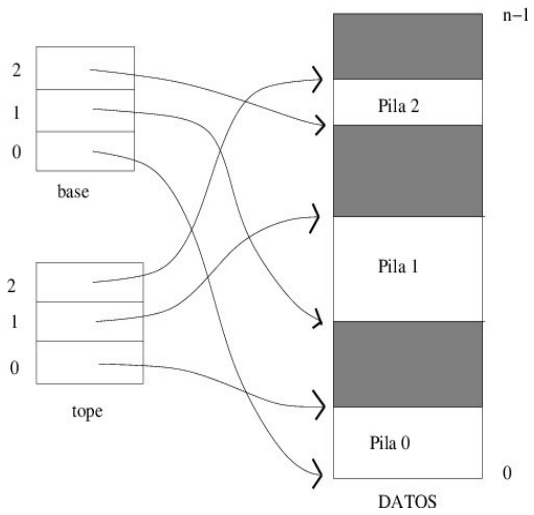
que devuelva true si L1 y L2 tienen la misma cantidad de elementos y los elementos de L1 son una permutación de los de L2

P.ej: si L1={1,23,21,4,2,3,0} y L2={21,1,3,2,4,23,0} devolvería TRUE pero si L1={1,3,5} y L2={4,5,4} devolvería FALSE

Si hay elementos repetidos tienen que estar el mismo número de veces en las 2 listas para poder ser TRUE. No pueden usarse estructuras auxiliares, la función como mucho debe ser $O(n^2)$, el algoritmo puede ser destructivo y no conservar las listas iniciales y no puede usarse ningun algorítmo de ordenación (incluyendo l.sort())

Sugerencia: Para cada elemento de L1, se recorre L2. Si el elemento está en L2, se borra y se continúa con el siguiente elemento hasta recorrer todos los elementos de L1. Si el elemento no está en L2, se devuelve directamente false.

37. Se pretende almacenar k pilas en un único vector utilizando la estructura de datos sugerida en la figura siguiente (para el caso k=3). Implementa las funciones **push** y **pop** que insertan/borran un determinado entero en una de las pilas



38. Implementar un **TDA cola** usando **dos pilas**.

39. Implementar una función

void sign_split(list<int> &L,vector< list<int> > &VL);

que dada una lista L, devuelva en el vector de listas VL las sublistas contiguas del mismo signo (el caso 0 se considera junto con los positivos, es decir separa negativos de no-negativos). El algoritmo puede ser destructivo sobre L es decir puede modificar a L

Ejemplos:

L:[4,-3,-5,-4,-5,-1,4,-1,-5,-5,1,-5,3,3,0,2,2],

=> **VL:[[4],[-3,-5,-4,-5,-1],[4],[-1,-5,-5],[1],[-5],[3,3,0,2,2]],**

L:[0,4,-2,4,1,-1,-4,-4,-3,-1,-4,4,1],

=> **VL:[[0,4],[-2],[4,1],[-1,-4,-4,-3,-1,-4],[4,1]],**

L:[2,-1,3,-3,3,-3,0,-1,-2,0],

=> **VL:[[2],[-1],[3],[-3],[3],[-3],[0],[-1,-2],[0]],**

Sugerencia: Chequear el signo s del primer elemento y extraer tantos elementos de ese signo como se pueda para ponerlos en VL[0]. Invertir el signo (s=-s) y repetir, para poner la siguiente sublista en VL[1]. Seguir así, invirtiendo s en cada iteración hasta que se acaben los elementos de L. Posiblemente la solución más simple es la destructiva, extrayendo elementos de L a medida que se insertan en los VL[i]

40. Implementar una función

int max_sublist_m(list<int> &L, int m);

que dada una lista de enteros, L y un entero m tal que 0<m<=L.size() , y encuentre y devuelva el valor de la mayor suma de entre todas las posibles sumas de sublistas de longitud m.

Por ejemplo: Si

L={1,2,-5,4,-3,2}

m=1 ⇒ 4 (por {4})

m=2 ⇒ 3 (por {1,2})

m=3 ⇒ 3 (por {4,-3,2})

m=4 ⇒ 2 (por {1,2,-5,4})

m=5 ⇒ 0 (por {2,-5,4,-3,2})

m=6 ⇒ 1 (por{1,2,-5,4,-3,2})

41. Sea el triángulo de Tartaglia, que comienza en nivel=0, Implementar una función que almacene en un vector dinámico todos los números del triángulo para un valor de nivel dado como parámetro de la función. Ejemplo:

1	nivel=0
1 1	nivel=1
1 2 1	nivel=2
1 3 3 1	nivel=3
1 4 6 4 1	nivel=4

Suponiendo que la llamada a la función fuese Tartaglia(4), el vector dinámico debería quedar: v={1 1 1 1 2 1 1 3 3 1 1 4 6 4 1}
La función tendrá el siguiente prototipo:

void tartaglia (int nivel, vector <unsigned int> &v);

42. Implementar una función

int max_sublist_par(list<int> &L, list<int> &subl);

que dada una lista de enteros, L encuentre y devuelva la sublista más larga de elementos consecutivos todos ellos pares. Si hay varias con el mismo tamaño, debe devolver cualquiera de ellas.

Por ejemplo: Si

L={5,4,5,8,6,2,10,9,3,1} ⇒ subl={8,6,2,10}

L={1,2,4,6,5,0,2,4} ⇒ subl={2,4,6}

Sugerencia: Guardar la lista actual más larga en subl. Recorrer la lista L y cada vez que se encuentre un elemento par, copiarlo junto con todos los elementos pares siguientes en una lista temporal tmp y si tmp.size() > subl.size(), reemplazarla.

43. Implementar una función

list<int> max_sublist(list<int> &L);

que dada una lista de enteros, L, encuentre y devuelva la sublista consecutiva con mayor suma entre sus elementos. Los elementos pueden ser negativos. Si hay varias sublistas con la misma suma máxima, se debe devolver la más corta.

Por ejemplo: Si

L={1,2,-5,4,-3,2} ⇒ {4} L={5,-3,-5,1,7,-2} ⇒ {1,7}

L={4,-3,11,-2} ⇒ {4,-3,1} L={4,-4,2,2} ⇒ {4}

44. Implementar una función:

void large_even_list(vector< list<int> >&VL,list<int>&L);

que dado un vector de listas de enteros VL, devuelva en L aquella lista VL[j] que contiene la máxima cantidad de elementos pares en el mismo orden. Solo debe devolver los elementos pares

Ejemplo:

V[0]={**0**,1,**2**,3,**4**,5,7},
V[1]={**0**,1,**2**,3},
V[2]={**2,2,2**,1,**0**} devuelve L=VL[2]={2,2,2,0}

45. Dada una lista de enteros L y un entero positivo m, implementar una función:

void interlaced_split(list<int>&L, int m,vector< list<int> >&VL);

que divida L en m sublistas (en un vector de listas) en forma entrelazada, es decir

a0,a1,a2,...,an van insertándose en las listas VL[o], VL[1], VL[2],...,VL[m-1], VL[0], VL[1], VL[2]....es decir, el elemento aj va a la lista VL[k], k=j%m

Por ejemplo:

Si L={0,1,2,3,4,5,6,7,8} ⇒ VL[0]={0,4,8}, VL[1]={1,5}, VL[2]={2,6} VL[3]={3,7}

46. Usando el tDA list<int>, implementar 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.
Ejemplos:
entrada={**1**,3,4,1,4} y (k=1) ==> entrada= {1,1,3,4,4,}
entrada={3,**1**,4,1,4,1,1} y (k=1) ==> entrada= {3,1,1,1,1,4,4}

47. Implementar una función:

queue<int> mergeSortedqueues(queue<int> A, queue<int> B);

que dadas 2 colas ordenadas (el mínimo valor está en el frente) devuelva una cola ordenada (el mínimo en el frente) que contenga todos los elementos de A y B. No pueden usarse estructuras auxiliares salvo colas.

48. Dadas dos colas con elementos enteros, implementar la función:
queue<int> multi_interseccion (const queue<int> & q1, const queue<int> &q2)

que calcula la multi-intersección de dos colas: elementos comunes en los dos colas repetidos tantas veces como aparezcan en la cola con menor número de apariciones del elemento.
Ejemplos:
si q1={2,2,3,3} y q2={1,2,3,3,3,4} entonces q1∩ q2={2,3,3}
si q1={2,2,2,3,3} y q2={1,2,2,2,2,3,3,3,4} entonces q1∩ q2={2,2,2,3,3}

49. Implementar una función

int ascendente(list <int> &L, list<list<int> > &LL);

que, dada una lista L, genera una lista de listas LL de tal forma de que cada sublista es ascendente.

Por ejemplo, si la lista es L=(0,5,6,9,4,3,9,6,5,5,2,3,7), entonces hay 6 corridas ascendentes, a saber: (0,5,6,9), (4), (3,9), (6), (5,5) y (2,3,7). Por lo tanto la función debe retornar LL=((0,5,6,9,4,3,9,6,5,5,2,3,7), (0,5,6,9), (4), (3,9), (5,5), (2,3,7)).

50. Implementar una función

void deja1solo(list<int> &L);

51. de tal forma que. si vamos dividiendo a L en rangos de pares e impares consecutivos, deja el primero de cada rango.

Por ejemplo:

Si L=(5 4 5 3 4 9 10 4 9 3 2 10 3 7 5 3) entonces debe dejar L=(5 4 5 4 9 10 4 9 3 2 3). Para L=(10 10 1 9 7 4 4 10 7 1 1 4 10 1 1 7) debe dejar L=(10 1 4 7 4 1).

Sugerencia: El algoritmo puede ser **in-place**, de forma que sólo tiene que borrar elementos

52. Implementar una función:

bool sumaIgual (const list<list<int> > &L);

que devuelve true si cualquier suma por filas o columnas da el mismo resultado (si una lista tiene menos elementos se asumen que el resto son cero). Por ejemplo, en la siguiente lista si se suma cada fila o cada columna siempre se obtiene 3:

L=< < 0,1,1,1>
<2,1>
<1,0,2>
<0.1.0.2> >

53. Implementar una función:
void dividePila (stack<int> &p, int n, list< stack<int> > &l);

que toma una pila y un número que indica en cuantas partes queremos dividir dicha pila. Cada una de las pilas resultantes se devuelven dentro de la lista l. El número pasado es mayor que 0 y un divisor del tamaño de la pila. Por ejemplo, para n=3:

Ejemplo: p={1,2,3,4,5,6} P1=> {5,6} P2=>{3,4} P3=>{1,2}
tope tope tope tope

Nótese que el orden de los elementos en las minipilas es el mismo que el que tenían en la pila original. Si se desea, la pila p puede quedar vacía tras el proceso.

54. Escriba una función:
void construyeCola (list< queue<int> > &l, queue<int> &c);
que a partir de una lista de colas construye una única cola. Por ejemplo:

Ejemplo: c1= {1,2} c2={3,4} c3={5,6} ==> c={1,2,3,4,5,6}
frente frente frente frente

55. Implementar una función
void subsecuencia (list<int> &L);
que dada una lista de enteros, elimine todos aquellos que no sean mayores que todos los anteriores.
Ejemplo: L={1,3,4,2,4,7,7,1} ==> L={1,3,4,7}

56. Dadas dos listas L1 y L2 de intervalos cerrados de valores enteros [ini, fin], con ini<=fin, donde en cada lista los intervalos se encuentran ordenados, verificándose que si un intervalo intervalo1 está antes que otro intervalo2 en la lista entonces intervalo1.fin < itervalo2.ini.
Implementar una función que seleccione un (sub)intervalo x de L1 y lo pase a L2. En este proceso podría ser necesario tanto el dividir un intervalo de L1 como fusionar intervalos en L2 cuando ocurran solapamientos. Tras seleccionar el intervalo x, los valores inicial y final de x no pertenecerán a ningún intervalo en L1. El prototipo sería:

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) y false en caso contrario. Tras seleccionar el intervalo x, los valores comprendidos entre inicial y final de x no pertenecerán a ningún intervalo en 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],[10,11],[18,20],[25,26]> 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]> L2:<[0,1],[12,23]>.

57. Implementar una función

stack<int> coctel (stack<int> &p1, stack<int> &p2);

que dadas dos pilas de enteros las vuelque en una única pila de enteros. El volcado se realiza tomando un entero de la primera, un entero de la segunda, un entero de la primera, etc. Si alguna se acaba antes que otra, se seguirán volcando únicamente los enteros de la no vacía. Al final del volcado, ambas pilas iniciales han de quedar vacías. Tener en cuenta que las pilas p1 y p2 pueden estar inicialmente vacías.

58. Implementar las funciones de insertar y borrar elementos del TDA **queue<int> a partir de** las funciones del TDA **stack <int>**

59. Implementar una función:

void peinarlistas (list<char> & L1, list<char> & L2);

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 L1
(b) La lista L1 contendrá el resultado final y L2 quedará vacía.
(c) Si una de ellas tiene un menor número de elementos que la otra, el exceso de elementos se incorporará a la lista L1

Ejemplo 1:
Antes de invocar a la función
L1: (a,b)
L2: (z,y,x,w,v,u)
Despues de invocar a la función
L1: (a,z,b,y,x,w,v,u)
L2: vacia

Ejemplo 2:
Antes de invocar a la función
L1: (a,b,c,d,e,f)
L2: (z,y,x)
Despues de invocar a la función
L1: (a,z,b,y,c,x,d,e,f)
L2: vacia

60. Sobrecargar el operador [ ] en la clase list<T>para devolver el elemento de la lista que ocupa la posición dada por el parámetro. Se debe tener en cuenta que pos debe ser un número entero válido. El prototipo sería:

template <class T> T& list<T>::operator[](unsigned int pos) const

Consideraciones:

*1.- El reto es **individual***

2.- Los ejercicios que a cada estudiante toca implementar se asignan por la letra de inicio del primer apellido:

2.1.- De la A a la E (ambos incluidos): Elegir 5 ejercicios entre el 1 y el 15
2.2.- De la F a la L (ambos incluidos): Elegir 5 ejercicios entre el 16 y el 30
2.3.- De la M a la R (ambos incluidos): Elegir 5 ejercicios entre el 31 y el 45
2.4.- De la S a la Z (ambos incluidos): Elegir 5 ejercicios entre el 46 y el 60

3.- La solución deberá incluir los códigos de las soluciones y entregarse obligatoriamente en un fichero tar o zip (se sugiere como nombre reto3.tar o reto3.zip)

4.- Las soluciones deberán estar documentadas adecuadamente.

5.- Si se entrega algún código que no compile o no funcione correctamente el reto quedará invalidado.

6.- Si la solución es correcta, se puntuará con 0.2 para la evaluación contínua

7.- El plazo límite de entrega es el 22 de Noviembre a las 23.55h