NOMBRE DE ALUMNO: VICTOR HUGO LÓPEZ MORENO

NOMBRE DEL PROFESOR (A): VIOLETA MABRIDIS MÉRIDA VELAZQUEZ

NOMBRE DEL TRABAJO: SUPERNOTA

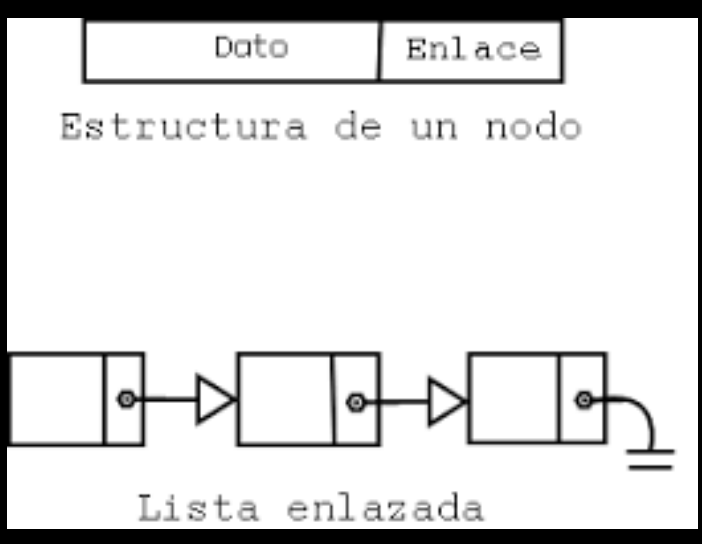
MATERIA: ALGORITMOS Y ESTRUCTURA DE DATOS

GRADO: 5°

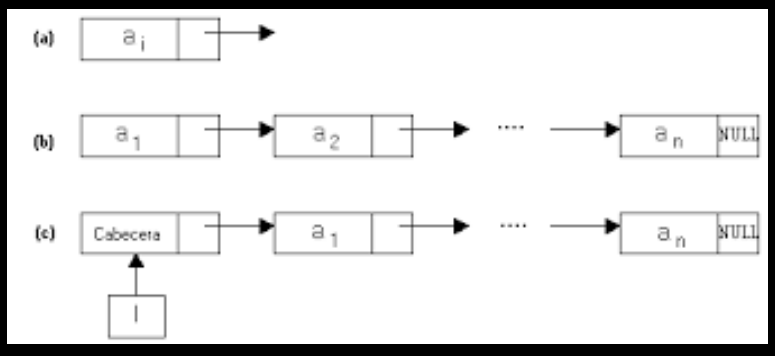
EL TAD LISTA

Las listas constituyen una de las estructuras lineales más flexibles, porque pueden crecer y acortarse según se requiera, insertando o suprimiendo elementos tanto en los extremos como en cualquier otra posición de la lista.

Por supuesto esto también puede hacerse con vectores, pero en las implementaciones más comunes estas operaciones son O(n) para los vectores, mientras que son O(1) para las listas.



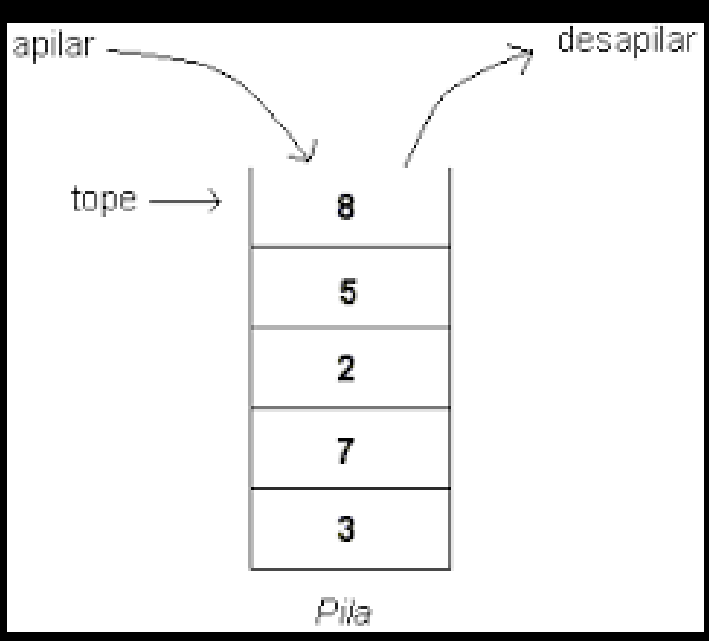
DESCRIPCIÓN MATEMÁTICA DE LAS LISTAS



Desde el punto de vista abstracto, una lista es una secuencia de cero o más elementos de un tipo determinado, que en general llamaremos elem_t, por ejemplo, int o double. A menudo representamos una lista en forma impresa como una sucesión de elementos entre paréntesis, separados por comas

OPERACIONES ABSTRACTAS SOBRE LISTAS

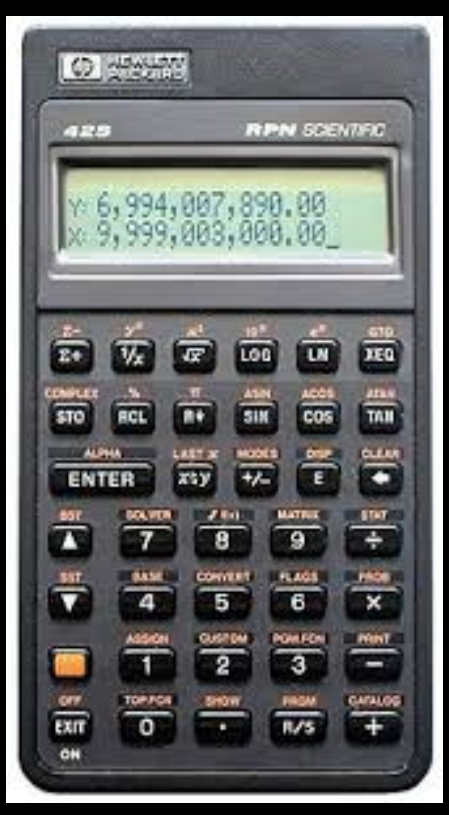
Consideremos una operación típica sobre las listas que consiste en eliminar todos los elementos duplicados de la misma. El algoritmo más simple consiste en un doble lazo, en el cual el lazo externo sobre i va desde el comienzo hasta el último elemento de la lista. Para cada elemento i el lazo interno recorre desde i + 1 hasta el último elemento, eliminado los elementos iguales a i. Notar que no hace falta revisar los elementos anteriores a i (es decir, los elementos j con j < i), ya que, por construcción, todos los elementos de 0 hasta i son distintos.



EL TAD PILA

Básicamente es una lista en la cual todas las operaciones de inserción y borrado se producen en uno de los extremos de la lista. Un ejemplo gráfico es una pila de libros en un cajón. A medida que vamos recibiendo más libros los ubicamos en la parte superior. En todo momento tenemos acceso sólo al libro que se encuentra sobre el “tope” de la pila.

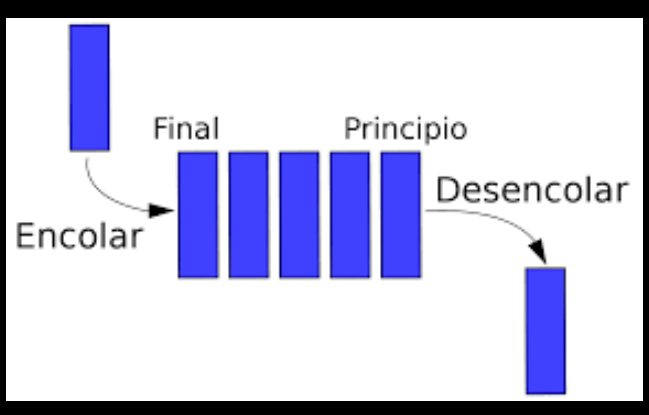
UNA CALCULADORA RPN CON UNA PILA



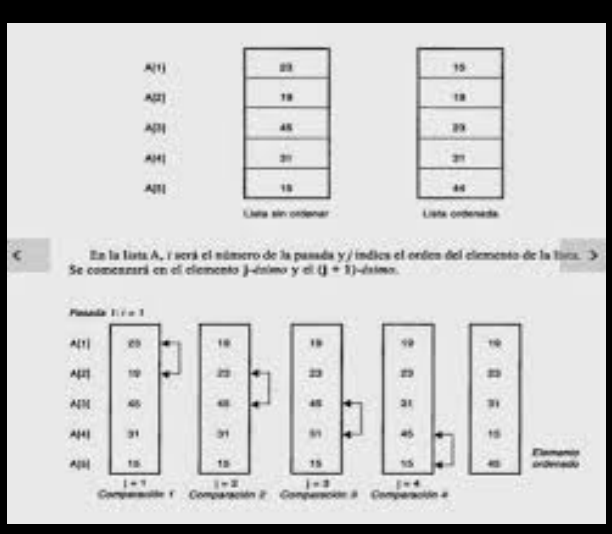
La forma de implementar una calculadora RPN es usando una pila. A medida que el usuario entra operandos y operadores se aplican las siguientes reglas Si el usuario ingresó un operando, entonces simplemente se almacena en la pila. Si ingresó un operador θ se extraen dos operandos del tope de la pila, digamos t el tope de la pila y u el elemento siguiente, se aplica el operador a los dos operandos (en forma invertida) es decir u θ t y se almacena el resultado en el tope de la pila.

EL TAD COLA

Por contraposición con la pila, la cola es un contenedor de tipo “FIFO” (por “First In First Out”, el primero en entrar es el primero en salir). El ejemplo clásico es la cola de la caja en el supermercado. La cola es un objeto muchas veces usado como buffer o pulmón, es decir un contenedor donde almacenar una serie de objetos que deben ser procesados, manteniendo el orden en el que ingresaron. La cola es también, como la pila, un subtipo de la lista llama también a ser implementado como un adaptador.



INTERCALACIÓN DE VECTORES ORDENADOS



Ejemplo 2.4: Un problema que normalmente surge dentro de los algoritmos de ordenamiento es el intercalamiento de contenedores ordenados. Por ejemplo, si tenemos dos listas ordenadas L1 y L2, el proceso de intercalamiento consiste en generar una nueva lista L ordenada que contiene los elementos en L1 y L2 de tal forma que L está ordenada, en la forma lo más eficiente posible.

ORDENAMIENTO POR INSERCIÓN

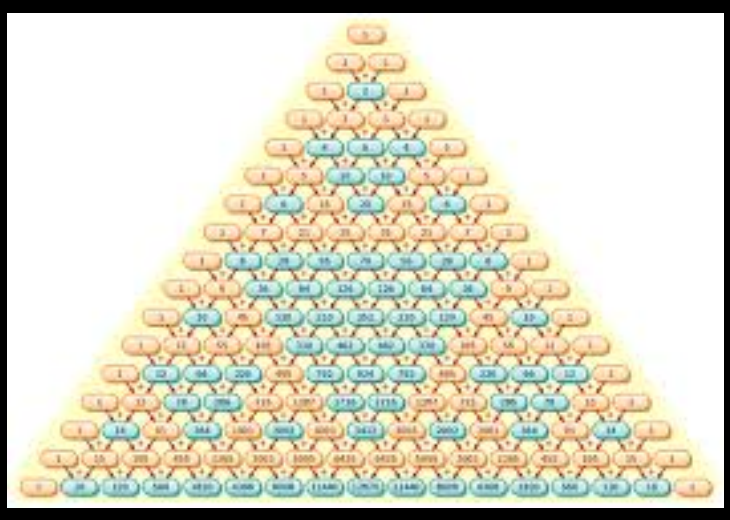
Verificamos que los elementos en las posiciones pares (10 12 14 16 . . .) están ordenados entre sí, como también los que están en las posiciones impares (1 3 5 7 . . .). Consideremos primero el algoritmo de “ordenamiento por inserción” (ver código 2.19, los algoritmos de ordenamiento serán estudiados en más detalle en un capítulo posterior).

TIEMPO DE EJECUCIÓN

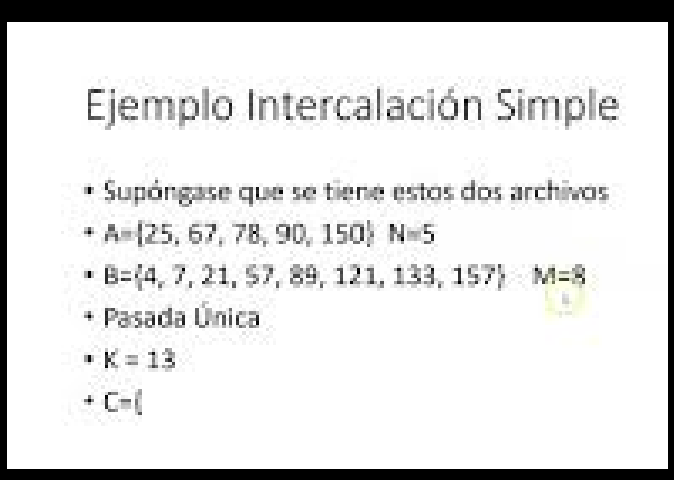
Consideremos ahora el tiempo de ejecución de este algoritmo. El lazo sobre j se ejecuta n - 1 veces, y el lazo interno se ejecuta, en el peor caso j - 1 veces, con lo cual el costo del algoritmo es, en el peor caso.
En el mejor caso, el lazo interno no se ejecuta ninguna vez, de manera que sólo cuenta el lazo externo que es O(n). En general, el tiempo de ejecución del algoritmo dependerá de cuantas posiciones deben ser desplazadas (es decir p - j) para cada j

PARTICULARIDADES AL ESTAR LAS SECUENCIAS PARES E IMPARES ORDENADAS

Como la intercalación de listas ordenadas es O(n) surge la incógnita de si el algoritmo para arreglos puede ser mejorado. Al estar las posiciones pares e impares ordenadas entre sí puede ocurrir que en promedio el desplazamiento sea menor, de hecho, generando vectores en forma aleatoria, pero tales que sus posiciones pares e impares estén ordenada se llega a la conclusión que el desplazamiento promedio es O(√ n), de manera que el algoritmo resulta ser O(n 3/2). Esto representa una gran ventaja contra el O(n 2) del algoritmo de ordenamiento original.



ALGORITMO DE INTERCALACIÓN CON UNA COLA AUXILIAR

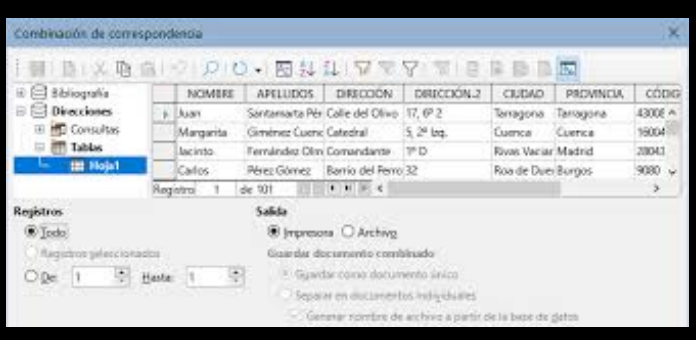


El algoritmo se muestra en el código 2.20, manteniendo el rango [p, q) en una cola auxiliar C. En (2.18) vemos el seguimiento correspondiente. Los elementos en la cola C son mostrados encerrados en una caja, en el rango [p, q). Notar que si bien, el número de elementos en la cola entra exactamente en ese rango, en la implementación el estado los elementos en ese rango es irrelevante y los elementos están en una cola auxiliar.

EL TAD CORRESPONDENCIA

La “correspondencia” o “memoria asociativa” es un contenedor que almacena la relación entre elementos de un cierto conjunto universal D llamado el “dominio” con elementos de otro conjunto universal llamado el “contradominio” o “rango”. Por ejemplo, la correspondencia M que va del dominio de los números enteros en sí mismo y transforma un número j en su cuadrado j 2 puede representarse como se muestra en la figura 2.15. Una restricción es que un dado elemento del dominio o bien no debe tener asignado ningún

INTERFAZ SIMPLE PARA CORRESPONDENCIAS

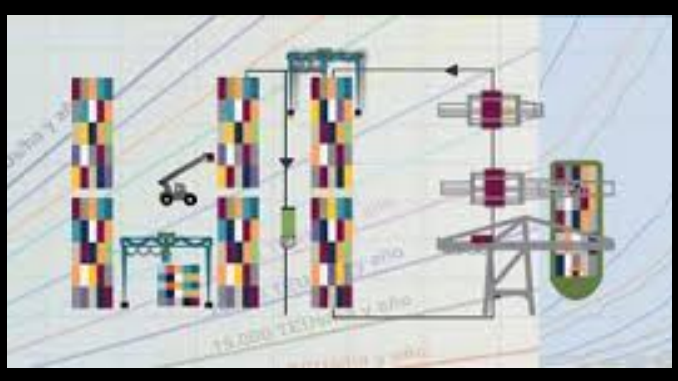


En el código 2.24 vemos una interfaz básica posible para correspondencias. Está basada en la interfaz STL pero, por simplicidad evitamos el uso de clases anidadas para el correspondiente iterator y también evitamos el uso de templates y sobrecarga de operadores. Primero se deben definir (probablemente via typedef’s) los tipos que corresponden al domininio (domain_t) y al contradominio (range_t).

IMPLEMENTACIÓN DE CORRESPONDENCIAS MEDIANTE CONTENEDORES LINEALES

Tal vez la forma más simple de implementar una correspondencia es guardando en un contenedor todas las asignaciones. Para eso simplemente definimos una clase elem_t que simplemente contiene dos campos first y second con la clave y el valor de la asignación (los nombres de los campos vienen de la clase pair de las STL).

Tal vez la forma más simple de implementar una correspondencia es guardando en un contenedor todas las asignaciones. Para eso simplemente definimos una clase elem_t que simplemente contiene dos campos first y second con la clave y el valor de la asignación (los nombres de los campos vienen de la clase pair de las STL).



REFERENCIAS

TODA LA INFORMACIÓN DE ESTE TRABAJO SE TOMÓ DE LA ANTOLOGIA DE LA MATERIA ALGORITMOS Y ESTRUCTURA DE DATOS