Configurando un algoritmo de ordenación posicional para Push Swap

Este algoritmo es muy eficiente. Sus resultados sobrepasan con creces los límites que impone push\_swap. Se garantiza un 100% si está correctamente implementado y es relativamente fácil de entender y de implementar.

1. Configurando listas enlazadas:

Usaremos listas enlazadas para implementar este algoritmo. Cada elemento contendrá varias variables:

typedef struct s\_bloques//Estructura de los bloques a y b

{

    struct s\_bloques    \*siguiente;//El puntero al siguiente nodo

    int                 numero;//El número a ordenar

    int                 indice;//El índice de la lista

    int                 posicion;//Posición del número en la lista

    int                 posicion\_objetivo;//Posicion donde vamos a insertar el número a ordenar

    int                 coste\_a;//El coste de mover los elementos al Stack a

    int                 coste\_b;//El coste de mover los elementos en el Stack b

}   t\_bloques;

La importancia de estas variables quedará más clara durante la descripción del algoritmo

1. Asignando Índices

Una vez que hemos rellenado la pila A con los valores a ordenar, necesitamos asignar un índice a cada elemento, desde el más pequeño al más grande. El número de índice irá desde el 1 hasta el número más alto de valores a ordenar. Si tenemos 10 números a ordenar. El índice irá desde el 1 al 10:

|  |  |
| --- | --- |
| **Value** | **Index** |
| 1900 | 9 |
| 42 | 8 |
| 18 | 7 |
| -146 | 2 |
| -30 | 3 |
| 2 147 483 647 | 10 |
| 3 | 6 |
| 0 | 5 |
| -2 147 483 648 | 1 |
| -2 | 4 |

Asignando índices es más fácil saber el orden en el que deberían estar los números aleatorios que hemos metido.

1. Eligiendo un algoritmo de ordenación dependiendo del número de valores a ordenar

La mayoría de algoritmos de ordenación del Push\_Swap tienen distintos métodos para 5, 100 y 500 valores. Este no es el caso. Funciona para 500 valores y para 5. Sin embargo, tenemos un algoritmo diferente para ordenar 3 números.

**Si la pila A no está ordenada,** procederemos de manera diferente dependiendo del número de valores a ordenar:

* 2 valores: Hacemos “**sa”**.
* 3 valores: Algoritmo para más de 3 valores[.](https://web.archive.org/web/20220802162832/https:/www.codequoi.com/en/push_swap-efficient-positional-sorting-algorithm/#sorting_algorithm_for_3_numbers)
* > 3 values: Algoritmo para más de 3 valores[.](https://web.archive.org/web/20220802162832/https:/www.codequoi.com/en/push_swap-efficient-positional-sorting-algorithm/#sorting_algorithm_for_over_3_numbers)
  1. **Algoritmo de ordenación para 3 números**

Para 3 valores hay 6 casos posibles:

Case actions

1 2 3 => no action

1 3 2 -> rra -> 2 1 3 -> sa -> 1 2 3 => 2 actions

2 1 3 -> sa -> 1 2 3 => 1 action

2 3 1 -> rra -> 1 2 3 => 1 action

3 1 2 -> ra -> 1 2 3 => 1 action

3 2 1 -> ra -> 2 1 3 -> sa -> 1 2 3 => 2 actions

Nuestro pequeño algoritmo de ordenación de 3 números sólo necesita coger estas 3 acciones: **ra**, **rra** and **sa**, dependiendo de la posición del número más grande.

No necesitamos usar la pila B

* 1. **Algoritmo de ordenación para más de 3 números**

Para una serie de números mayor, necesitamos programar más acciones. Para decidir qué movimiento hacer para ordenar cada valor, necesitamos calcular la posición de cada elemento en su pila. Así, podemos comparar el número de acciones que requiere llevar cada valor al principio de la pila. Sólo entonces elegimos la serie de movimientos más eficiente.

Nuestro algoritmo de ordenación tiene 8 pasos, por supuesto, siempre que no esté ordenado ya:

1. **pb** todos los elementos de la pila A excepto los 3 últimos. Mete en el stack b los valores más pequeños primero y luego los más grandes para ayudar con la eficiencia al ordenar.
2. **Ordena los 3 valores que quedan en el bloque A**.
3. Mientras haya elementos en el Bloque B:
   1. **Encontrar la posición actual de cada elemento** del Bloque A y del Bloque B.
   2. **Calculate the target position** in stack A where each element in stack B should be.
   3. **Calculate the number of actions** (the cost) to put each element in stack B at its target position in stack A and **choose the element that’s cheapest to move**.
   4. **Execute the sequence of actions** needed to move the element from stack B to stack A.
4. If stack A is not sorted, chose between **ra** and **rra** to rotate it into ascending order.

### Paso 1: Enviar todo a la pila B menos los 3 últimos

El primer paso es enviar todos los elementos al bloque b, excepto 3. Mantenemos estos 3 elementos en el bloque A para evitar movimientos extra. Después de esto, sólo tenemos que aplicar el algoritmo simple de ordenación de 3 números del bloque A.

Sin embargo, enviar todos los elementos al bloque B no significa que se haga sin un orden. De hecho, gracias a nuestro sistema de índices, podemos hacer una clasificación en dos pasos.

Primero, podemos enviar los valores más pequeños. Así, si un valor tiene un índice menor que el índice de la mitad de elementos (número total de elementos dividido por dos), podemos enviarlo a la pila B. De otra manera, rotamos en A. Después de esto, podemos hacer libremente “pb” a todos los elementos menos los 3 últimos que quedan en el bloque A.

De esta manera, el bloque B ya queda relativamente ordenado, lo que reduce el número de acciones que tendremos que hacer después.

### Paso 2: Ordenar los 3 números que quedan en el Bloque A

Los 3 números que quedan en el Bloque A tienen que ser ordenados. Usaremos el algoritmo de ordenación para 3 números anterior.

Con estos 3 valores ordenados, todo lo que necesitamos es insertar cada elemento del Bloque B en el lugar correcto del Bloque A. Con esta información, y teniendo en cuenta la posición, calcularemos cuál es la secuencia más eficiente de acciones y elegiremos qué elemento mover primero.

### Paso 3: Calculo de posiciones

Un paso clave en nuestro algoritmo es encontrar la posición de cada elemento en su pila, así como la posición objetivo en el bloque A de cada elemento del bloque B. Con esta información posicional, podremos calcular la secuencia más eficiente de acciones y elegir qué elemento mover primero.

#### Paso 3.1: Encontrando la posición de cada elemento

La posición de un elemento es una representación simple de la posición que ocupa en su pila:

value: 3 0 9 1

index: [3] [1] [4] [2]

position: <0> <1> <2> <3>

Todo lo que necesitamos hacer para encontrar la posición de cada elemento es escanear la pila de arriba abajo, y asignarle una posición a cada elemento que se incrementa con cada iteración. Por supuesto, tendremos que actualizar estas posiciones cada vez que queramos calcular la secuencia más eficiente de acciones.

#### Paso 3.2: Encontrando la posición objetivo de cada Elemento del Bloque B

Para cada elemento del Bloque B necesitamos encontrar su posición objetivo en el bloque A.

Podemos marcar esta posición objetivo simplemente escaneando el índice superior en la pila A de un elemento de la pila B. La posición del elemento de la pila A con ese índice será la posición objetivo de nuestro elemento del Bloque B.

Stack A contains

value: 8 0 1 3

index: [6] [1] [2] [4]

position: <0> <1> <2> <3>

Stack B contains

value: 2 6 9

index: [3] [5] [7]

position: <0> <1> <2>

target pos.: (3) (0) (1)

Aquí, un elemento al principio del bloque B tiene valor 2 y número de índice 3. Esto significa que es el tercer valor en orden ascendente de los números que necesitamos ordenar. Como este elemento ya está al principio del bloque B, podríamos enviarlo al principio del bloque A, pero no estaría en su marca correcta. Necesitamos que este elemento esté entre los índices 2 y 4. Si el elemento con el índice 4, actualmente en la posición 3 del bloque A, estuviera al principio de su bloque, podríamos enviarlo de manera segura del bloque B al bloque A. Así la posición objetivo del primer elemento del bloque B, es la posición la posición 3 del bloque A.

Si un elemento del bloque B tiene un índice superior a todos los elementos del bloque A, necesitamos usar el índice más pequeño como posición objetivo, para luego darle la vuelta con un “ra”

Con esto podremos determinar cuántas acciones llevará poner cada elemento en su sitio correcto de la pila, y podremos averiguar qué elemento del bloque B es más eficiente mover primero.

Obviamente, sólo necesitamos calcular la posición objetivo de los elementos del bloque B, porque los elementos del bloque A ya están en la posición correcta. Como varios elementos del bloque B pueden tener la misma posición objetivo en el bloque A, el coste variará dependiendo de la posición en la que estén del bloque B.

### Paso 4: Calculo del coste de la acción más eficiente

### Una vez que hemos determinado la posición y la posición objetivo de cada elemento de los dos bloques, podremos comparar sus costes en términos de número de acciones que llevaría poner cada elemento de arriba del bloque B en la marca correcta del bloque A.

### En ambos casos necesitamos distinguir entre rotar y rotar reverso para calcular correctamente el coste. Para determinar si necesitamos rotar reverso mejor que simplemente rotar, podemos medir el tamaño de la pila y dividirla por 2. Así es fácil chequear si la posición de nuestro elemento está en la mitad superior o la mitad inferior del bloque. Si la posición del elemento está en la mitad superior de la pila, significa que es mejor hacer “rb”. Si no, es mejor “rrb”. Sería muy últil marcar el coste rotar reverso como negativoNos será muy B, porque los elementos del bloque A ya están en la posición correcta. Como varios elementos del bloque B pueden tener la misma posición objetivo en el bloque A, el coste variará dependiendo de la posición en la que estén del bloque B.