Integrador TP N° 1

Tomás Vidal
Arquitectura de Computadoras
Facultad de Ingeniería, UNLP, La Plata, Argentina.
26 de Abril, 2024.

I. Bubble Sort

Para resolver el problema se empleó el algoritmo de **Bubble Sort**, debido a que cumple con los requerimientos, es simple, facil de implementar y, solo necesita un espacio extra de memoria. Aunque no todo es perfecto, este algoritmo implica un coste de media de $\mathbf{O}(n^2)$, en el mejor de los casos es $\mathbf{O}(n)$, y en el peor nuevamente es $\mathbf{O}(n^2)$; esto quiere decir que si se tiene un vector con n elementos, se tienen que realizar de media n^2 operaciones para ordenar este vector con el algoritmo.

I-A. *Espacio extra de memoria*

La variable **aJData** (con dirección de memoria **aJAddr**), en el código (I), es un espacio de memoria extra requerido (mencionado previamente en I) que se emplea para almacenar temporalmente el dato del vector que está siendo comparado, ya que la idea del algoritmo es utilizar las mismas posiciones de memoria y no crear nuevas; por ejemplo: si se tiene el vector $[a_0, a_1, a_2]$, $a_0 = 1$, $a_1 = -1$ y $a_2 = 10$ y se está ordenando a_0, entonces almaceno temporalmente a_0 y copio el valor de a_1 en la posición de memoria de a_0 (ya que quiero ordenar de menor a mayor), y luego copio el valor almacenado en la memoria temporal en la posición de memoria del dato a_1. Si no se hubiera empleado un espacio extra de memoria, no se hubiera podido recuperar el valor de a_0 para copiarlo en a_1.

I-B. *Algoritmo implementado*

El algoritmo de Bubble Sort implementado se puede observar en el diagrama de flujo I-B, en el mismo se puede apreciar que hay dos bucles independientes (el de **i** y el de **j**), estos fueron creados en el código de *Ensamblador* “iterando” las *variables* de **iIndex** y **jIndex** (ver)

Listing 1. Fragmento del código *“69854_1_0.asm”*

```
/ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
/ Entry Point del algoritmo de ordenamiento
/ es un bucle 'while' que itera cada posicion del vector de datos
/ y los va ordenando en la misma posicion de memoria
/ -> Esto replica tal cual el diagrama de flujo provisto
BubbleSortInit, Load  Zero
                Store iIndex / i = 0
                Store jIndex / i = 0

                / DisplayRtrnPath++
                Load      DisplayRtrnPath
                Add       One
                Store     DisplayRtrnPath

                / i < n ?
                / si -> ir a (j<n-i-1?)
                / no -> terminar bucle
StartILoop,     LoadI      DataLengthPtr
                Subt       iIndex
                Skipcond   800
                Jump       SortEnded

                / j = 0
                Load       Zero
                Store      jIndex

                / j < n-i-1 ?
                / si -> ver si permutar datos
                / no -> i++
AfterStartILoop,LoadI      DataLengthPtr
                Subt       iIndex
                Subt       One
                Subt       jIndex
                Skipcond   800
                Jump       IncrementI

                / Compruebo si se tienen que permutar los datos
                / para eso debo guardo temporalemente el dato a[j]
                / y la direccion de a[j] (&a[j])
                Load       DataPtr
                Add        jIndex
                Store      aJAddr

                / incremento la direccion &a[j+1]
                Add        One
                Store      aJ1Addr

                / temp = a[j]
                LoadI      aJAddr
                Store      aJData

                / a[j] > a[j+1] ?
                / si -> permutar
                / no -> j++
                LoadI      aJ1Addr      / a[j+1]
                Subt       aJData       / a[j]
                Skipcond   000
                Jump       IncrementJ

                / Se permutan los datos

                / almaceno en &a[j] el dato de a[j+1]
                LoadI      aJ1Addr
                StoreI     aJAddr

                / almaceno en &a[j+1] el dato temporal (antiguo a[j])
                Load       aJData
                StoreI     aJ1Addr

                / j++
IncrementJ,     Load       jIndex
                Add        One
                Store      jIndex

                Jump       AfterStartILoop


                / i++
IncrementI,     Load  iIndex
                Add   One
                Store iIndex
                Jump  StartILoop


/ - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Fig. 1. Diagrama de flujo