	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	25/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía Práctica de Estudio 3


Algoritmos de ordenamiento parte 3

Elaborado por:

M.I. Elba Karen Sáenz García

Revisión:

Ing. Laura Sandoval Montaña

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	26/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Guía práctica de estudio 3

Estructura de datos y Algoritmos II

Algoritmos de Ordenamiento. Parte 3.

Objetivo: El estudiante conocerá e identificará la estructura de los algoritmos de ordenamiento *Counting Sort* y *Radix Sort*.

Actividades

- Implementar el algoritmo *Counting Sort* en algún lenguaje de programación para ordenar una secuencia de datos.
- Implementar el algoritmo *Radix Sort* en algún lenguaje de programación para ordenar una secuencia de datos.

Antecedentes

- Análisis previo de los algoritmos en clase teórica.
- Manejo de arreglos o listas, diccionarios, estructuras de control y funciones en Python 3.

Introducción


Counting Sort

Es un algoritmo que no se basa en comparaciones y lo que hace es contar el número de elementos de cada clase en un rango de $0 - k$ para después ordenarlos determinando para cada elemento de entrada el número de elementos menores a él. Por lo tanto, la lista o arreglo a ordenar solo pueden utilizar elementos que sean contables (enteros).

Para la descripción del algoritmo se asumen 3 arreglos lineales:

- El arreglo de entrada A a ordenar con n elementos
- Un arreglo B de n elementos, para guardar la salida ya ordenada
- Un arreglo C para almacenamiento temporal de k elementos

El primer paso consiste en averiguar el rango de valores de los datos a ordenar ($0 - k$). Después se crea el arreglo C de números enteros con tantos elementos como valores haya en el intervalo $[0, k]$, y a cada elemento de C se le da el valor inicial de 0 (0 apariciones del elemento i donde toma valores de $i = 0, \dots, k$).

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	27/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

En el arreglo A de la figura 3.1 se tiene que el rango de valores de los elementos es de 0 a 5, por lo tanto, el arreglo C tendrá 6 elementos con índices de 0 a 5 el cual se inicializa en 0.

	1	2	3	4	5	6	7	8		0	1	2	3	4	5
A	2	5	3	0	2	3	0	3		C	0	0	0	0	0

Figura 3.1

Después se recorren todos los elementos del arreglo A a ordenar y se cuenta el número de apariciones de cada elemento para almacenarlo en C . Figura 3.2

	1	2	3	4	5	6	7	8
A	2	5	3	0	2	3	0	3
	0	1	2	3	4	5		
C	2	0	2	3	0	1		

Figura 3.2 [1]

Posteriormente se determina para cada elemento $C[i]$ ($i = 0, \dots, k$) cuántos elementos son menores o iguales a él.


En la figura 3.3 se puede observar que 2 elementos son menores o iguales a 1, 4 elementos son menores o iguales a 2, 7 elementos son menores o iguales a 3, etc.

	0	1	2	3	4	5
C	2	2	4	7	7	8

Figura 3.3

Para terminar, se coloca cada elemento del arreglo A , ($A[j]$, $j = n, \dots, 1$) en la posición correcta en el arreglo de salida B , de tal forma que cada elemento de entrada se coloca en la posición del número de elementos menores o iguales a él ($B[C[A[j]]]$). Cada vez que se coloca un elemento en B , se decrementa el valor de $C[A[j]]$.

Así el elemento $A[8] = 3$ se coloca en $B[C[A[8]]] = B[C[3]] = B[7]$. Figura 3.4.

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	28/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

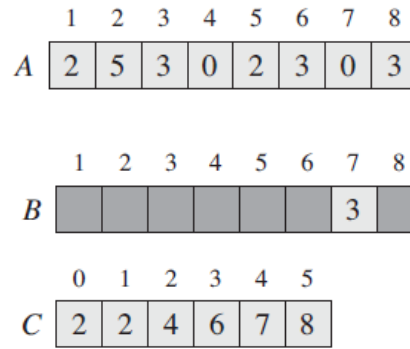


Figura 3.4

El elemento $A[7]$ se coloca en $B[C[A[7]]] = B[C[0]] = B[2]$. Figura 3.5.

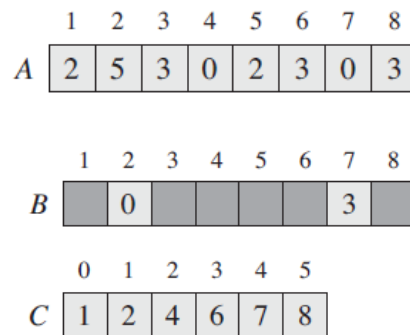



Figura 3.5

Una vez que se revisan todos los elementos de A tenemos que el arreglo ordenado en B queda como en la figura 3.6.



Figura 3.6

Un pseudocódigo del algoritmo es [1]:

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	29/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

CountingSort(A,B,k)

Inicio

Formar C de k elementos

Para i=0 hasta i=k

C[i]=0

Fin Para

Para j=1 hasta número de elementos de A

C[A[j]]=C[A[j]]+1

Fin Para

Para i=1 hasta i=k

C[i]=C[i]+C[i-1]

Fin para

Para j= número de elementos de A hasta 1

B[C[A[j]]]=A[j]

C[A[j]]=C[A[j]]-1

Fin Para

Fin


El tiempo de ejecución para este algoritmo es $\theta(n + k)$ dado que para el primer y tercer ciclo se toma un tiempo de $\theta(k)$ y para el segundo y último un tiempo de $\theta(n)$.

Counting Sort es un algoritmo estable, es decir, si el ordenamiento se hace con base en una relación de orden y en esa relación dos elementos son equivalentes, entonces se preserva el orden original entre los elementos equivalentes.

Radix Sort

El método de ordenamiento *Radix Sort* también llamado ordenamiento por residuos puede utilizarse cuando los valores a ordenar están compuestos por secuencias de letras o dígitos que admiten un orden lexicográfico.

El algoritmo ordena utilizando un algoritmo de **ordenación estable**, las letras o dígitos de forma individual, partiendo desde el que está más a la derecha (menos significativo) y hasta el que se encuentra más a la izquierda (el más significativo). **Nota:** a cada letra o dígito se le asigna una llave o código representado por un número entero, el cual se utiliza para el ordenamiento de cada elemento que conforma el valor original.

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	30/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Por ejemplo, en [1] se planea que una línea aérea proporciona números de confirmación diseñados con cadenas formadas con 2 caracteres donde cada carácter es un dígito o una letra que puede tomar 36 valores (26 letras y 10 dígitos) y así hay 36^2 posibles códigos.

Para cada carácter de los 36 se genera un código numérico entero de 0-36. Figura 3.7.

carácter	código
0	0
1	1
.	.
.	.
9	10
A	11
B	12
.	.
.	.
Z	35

Figura 3.7


Si se tienen los códigos de confirmación $\{F6, E5, R6, X6, X2, T5, F2, T3\}$ y se utiliza un algoritmo de ordenación estable en el carácter que se encuentra más a la derecha se obtiene la lista parcialmente ordenada de códigos $\{X2, F2, T3, E5, T5, F6, R6, X6\}$. Ahora si se ordena utilizando el mismo algoritmo de ordenamiento estable, pero sobre el carácter que se encuentra más a la izquierda se obtiene la lista $\{E5, F2, F6, R6, T3, T5, X2, X6\}$.

Procesa las letras o dígitos de forma individual partiendo desde el dígito menos significativo y hasta alcanzar el dígito más significativo.

Si los códigos de confirmación se forman con 6 caracteres y se tiene la lista de códigos $\{XI7FS6, PL4ZQ2, JI8FR9, XL8FQ6, PY2ZR5, KV7WS9, JL2ZV3, KI4WR2\}$, el ordenamiento del carácter más a la derecha hacia el de más a la izquierda se muestra en la Figura 3.2[1].

<i>i</i>	Resultado de las listas ordenadas con el i-ésimo carácter
1	$\langle PL4ZQ2, KI4WR2, JL2ZV3, PY2ZR5, XI7FS6, XL8FQ6, JI8FR9, KV7WS9 \rangle$
2	$\langle PL4ZQ2, XL8FQ6, KI4WR2, PY2ZR5, JI8FR9, XI7FS6, KV7WS9, JL2ZV3 \rangle$
3	$\langle XL8FQ6, JI8FR9, XI7FS6, KI4WR2, KV7WS9, PL4ZQ2, PY2ZR5, JL2ZV3 \rangle$
4	$\langle PY2ZR5, JL2ZV3, KI4WR2, PL4ZQ2, XI7FS6, KV7WS9, XL8FQ6, JI8FR9 \rangle$
5	$\langle KI4WR2, XI7FS6, JI8FR9, JL2ZV3, PL4ZQ2, XL8FQ6, KV7WS9, PY2ZR5 \rangle$
6	$\langle JI8FR9, JL2ZV3, KI4WR2, KV7WS9, PL4ZQ2, PY2ZR5, XI7FS6, XL8FQ6 \rangle$

Figura 3.8 [1]

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	31/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

El algoritmo en pseudocódigo del *Radix Sort* [2] es:

RadixSort(A,d)

Inicio

Para $i=1$ hasta $i=d$

Ordenamiento de A en el dígito i

Fin

Donde A es una lista de n elementos, d es el número de dígitos o caracteres que tienen los elementos de A , si $i = 1$ se refiere al dígito o carácter colocado más a la derecha y cuando $i = d$ al que está más a la izquierda. El ordenamiento se realiza con algún algoritmo estable como por ejemplo *Counting Sort*.

Desarrollo:

Actividad 1


Se proporciona la función mencionada en pseudocódigo para el algoritmo **Counting Sort** en Python. Se requiere utilizarla para elaborar un programa que ordene una lista de enteros con valores comprendidos entre 0 y k . El valor de k es propuesto, pero debe ser mayor a 10 y menor a 30.

La función en Python del pseudocódigo descrito para **Counting Sort** es la siguiente, notar que aquí los índices de inicio de todas las listas es 0:

```
def countingSort(A,k):
    C=[0 for _ in range(k+1)]
    B=[0 for _ in range(len(A))]
    for j in range(0,len(A)):
        C[A[j]]=C[A[j]]+1

    for i in range(1,k+1):
        C[i]=C[i]+C[i-1]

    for j in range(len(A)-1,-1,-1):
        B[C[A[j]]-1]=A[j]
        C[A[j]]=C[A[j]]-1
    return B
```

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	32/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Una vez realizado el programa, indicar un procedimiento para obtener el valor de k , para saber el rango de valores de los elementos de la lista a ordenar. Además, contestar las siguientes preguntas

¿La lista se ordena en orden ascendente o descendente? ¿Por qué?

¿Es posible ordenar la lista en orden inverso?, ¿Cómo se haría? Realizar respectivos cambios en el programa.

Actividad 2

Para la explicación del algoritmo *Radix Sort* se utilizó el ejemplo de cómo ordenar claves de confirmación de una aerolínea, donde cada clave está formada con seis caracteres. Aquí se muestra una solución al problema en Python. Además, como algoritmo de ordenación estable se utiliza el *Counting Sort* visto al inicio del documento con unas pequeñas modificaciones.

Se pide probar el código proporcionado con la secuencia


{XI7FS6, PL4ZQ2, JI8FR9, XL8FQ6, PY2ZR5, KV7WS9, JL2ZV3, KI4WR2} y colocar la impresión correspondiente para ver las listas parcialmente ordenadas como en la Figura 3.8.

Una vez terminado lo anterior responder a la siguiente pregunta.

¿Qué cambios se harían para ordenar ahora del más significativo al menos significativo (izquierda a derecha)? Describir y realizar las modificaciones correspondientes al programa. _____

Cabe mencionar que para la implementación en Python de la solución al problema de la ordenación de claves de la aerolínea mediante el algoritmo *RadixSort* y uso de *Counting Sort*, se requirió contar el número de caracteres que conforman la clave y formar una lista donde cada elemento contiene información de la clave y el código correspondiente al carácter en análisis (figura 3.7). El código de la figura 3.7 se implementó con un diccionario.

Las funciones que conforman la solución y la llamada a la función que da la solución se muestran abajo:

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	33/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

```
def formalistaConClaves(B,numCar):
    Btmp=[]
    D={}
    j=0
    for i in range(48,58):
        D[chr(i)]=j
        j+=1
    for i in range(65,91):
        D[chr(i)]=j
        j+=1
    for i in range(len(B)):
        Btmp.append([B[i]]*2)
        A3=list(B[i])
        Btmp[i][1]=D[A3[numCar-1]]
    return Btmp
```

```
def countingSort2(A,k):
    C=[0 for _ in range(k+1)]
    B=[list(0 for _ in range(2)) for _ in range(len(A))]
    for j in range(0,len(A)):
        C[A[j][1]]=C[A[j][1]]+1


    for i in range(1,k+1):
        C[i]=C[i]+C[i-1]

    for j in range(len(A)-1,-1,-1):
        B[C[A[j][1]]-1][1]=A[j][1]
        B[C[A[j][1]]-1][0]=A[j][0]
        C[A[j][1]]=C[A[j][1]]-1
    return B
```

```
def obtenerElemSinClaves(E):
    Elem=[]
    for i in range(0,len(E)):
        Elem.append(E[i][0])
    return Elem
```

```
def radixSort(A):
    numCar=len(A[1])
    for i in range(numCar,0,-1):
        cc=formalistaConClaves(A,i)
        ordenado=countingSort2(cc,36)
        A=obtenerElemSinClaves(ordenado)

    return A
```

	Manual de prácticas del Laboratorio de Estructuras de datos y algoritmos II	Código:	MADO-20
		Versión:	01
		Página	34/180
		Sección ISO	8.3
		Fecha de emisión	20 de enero de 2017
Facultad de Ingeniería		Área/Departamento: Laboratorio de computación salas A y B	
La impresión de este documento es una copia no controlada			

Para observar el funcionamiento se crea una lista con las claves y después se llama a la función *radixSort()* como se observa en la definición de la siguiente función main().

```
def main():
    lista=[ 'XI7FS6', 'PL4ZQ2', 'JI8FR9', 'XL8FQ6', 'PY2ZR5', 'KV7WS9', 'JL2ZV3', 'KI4WR2' ]
    A=radixSort(lista)
    print('\n\nLista ordenada:\n',A)

main()
```

Actividad 3

Ejercicios propuestos por el profesor.

Referencias

[1] CORMEN, Thomas

Algorithms Unlocked

Cambridge MA, USA

The MIT Press, 2013

[2] CORMEN, Thomas, LEISERSON, Charles, et al.

Introduction to Algorithms

3rd edition

MA, USA

The MIT Press, 2009