Profesora: Elba Karen Sáenz García

Práctica 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*.

Al final de la práctica el estudiante habrá implementado los algoritmos en algún lenguaje de programación.

Actividades

Actividad 1

Se proporciona el procedimiento dado en clase del pseudocódigo para el algoritmo ***Counting Sort*** en Python. Se requiere utilizarla para elaborar un programa que ordene una lista de 20 enteros generada de forma aleatoria con valores comprendidos entre 0 y k. El valor de k es propuesto pero debe ser mayor a 5 y menor a 20 .

```
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
```

```python
import random
def countingSort(A,k):
    C=[0 for i in range(k+1)]

    B=[0 for i in range(len(A))]

    for j in range(len(A)):
        C[A[j]]+=1

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

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

La salida del programa debe contener la información indicada en la figura 1.

```
Los valores a ordenar son
[6, 0, 9, 7, 2, 0, 1, 10, 6, 3, 4, 9, 3, 10, 0, 2, 0, 8, 8, 0]

La lista C inicializada es:  [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
La lista C que indica cuantos elementos hay de cada clase:
0 - 5
1 - 1
2 - 2
3 - 2
4 - 1
5 - 0
6 - 2
7 - 1
8 - 2
9 - 2
10 - 2
La lista C que indica cuantos elementos son menores o iguales:
0 - 5
1 - 6
2 - 8
3 - 10
4 - 11
5 - 11
6 - 13
7 - 14
8 - 16
9 - 18
10 - 20

La lista C que indica Al final:
0 - 0
1 - 5
2 - 6
3 - 8
4 - 10
5 - 11
6 - 11
7 - 13
8 - 14
9 - 16
10 - 18
La lista B que contiene los elementos ordenados:
[0, 0, 0, 0, 0, 1, 2, 2, 3, 3, 4, 6, 6, 7, 8, 8, 9, 9, 10, 10]
```

Figura 1

Actividad 2

Ordenar un secuencia numérica de entre 2 y 5 digitos mediante el algoritmo Radix Sort. Mostrar como se va ordenando de acuerdo a cada digíto.

```python
def radixSort(arr):
    # Encontrar el número máximo para conocer el número de dígitos
    max_val = max(arr)
    # counting sort para cada dígito
    #(exp es 10^i donde i es el dígito actual)
    exp = 1
    while max_val // exp > 0:
        counting_sort(arr, exp)
        exp *= 10
```

```python
def counting_sort(A, exp):
    n = len(A)
    B = [0] * n
    C= [0] * 10
    # Contar las ocurrencias de cada dígito en exp
    for i in range(n):
        index = (A[i] // exp) % 10
        C[index] += 1

    for i in range(1, 10):
        C[i] += C[i - 1]
    # Construir el arreglo B
    i = n - 1
    while i >= 0:
        index = (A[i] // exp) % 10
        B[C[index] - 1] = A[i]
        C[index] -= 1
        i -= 1
    # Copiar el arreglo ordenado en arr
    for i in range(n):
        A[i] = B[i]
```

Actividad 3 reporte

Para la explicación del algoritmo *Radix Sort* se utiliza 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, donde como algoritmo de ordenación estable se utiliza el *Counting Sort* visto antes.

Se muestra también una función main() para probar el código proporcionado con la secuencia $\{XI7FS6, PL4ZQ2, JI8FR9, XL8FQ6, PY2ZR5, KV7WS9, JL2ZV3, KI4WR2$. Se pide probar y colocar la impresión correspondiente para ver las listas parcialmente ordenadas como su muestra en la figura 2:

```
Lista ordenada con caracter 6
['PL4ZQ2', 'KI4WR2', 'JL2ZV3', 'PY2ZR5', 'XI7FS6', 'XL8FQ6', 'JI8FR9', 'KV7WS9']
Lista ordenada con caracter 5
['PL4ZQ2', 'XL8FQ6', 'KI4WR2', 'PY2ZR5', 'JI8FR9', 'XI7FS6', 'KV7WS9', 'JL2ZV3']
Lista ordenada con caracter 4
['XL8FQ6', 'JI8FR9', 'XI7FS6', 'KI4WR2', 'KV7WS9', 'PL4ZQ2', 'PY2ZR5', 'JL2ZV3']
Lista ordenada con caracter 3
['PY2ZR5', 'JL2ZV3', 'KI4WR2', 'PL4ZQ2', 'XI7FS6', 'KV7WS9', 'XL8FQ6', 'JI8FR9']
Lista ordenada con caracter 2
['KI4WR2', 'XI7FS6', 'JI8FR9', 'JL2ZV3', 'PL4ZQ2', 'XL8FQ6', 'KV7WS9', 'PY2ZR5']
Lista ordenada con caracter 1
['JI8FR9', 'JL2ZV3', 'KI4WR2', 'KV7WS9', 'PL4ZQ2', 'PY2ZR5', 'XI7FS6', 'XL8FQ6']
La Lista ordenada:  ['JI8FR9', 'JL2ZV3', 'KI4WR2', 'KV7WS9', 'PL4ZQ2', 'PY2ZR5', 'XI7FS6', 'XL8FQ6']
```

Figura 2

El código es el siguiente:

```python
def formaListaConClaves(B,numCar):
    Btmp=[]
    D=formaCodigo()

    for i in range(len(B)):
        Btmp.append([B[i]]*2)
        A3=list(B[i])
        Btmp[i][1]=D[A3[numCar-1]]
    return Btmp
```

```python
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
```

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

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

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

main()
```

Profesora: Elba Karen Sáenz García

Actividad 4

Implementar un programa que ordene por bucket sort la secuencia

.78 , .17 .39, .72 , .94 , .21, .12 , .23 .68

Utilizar el algoritmo en pseudocódigo siguiente y mostrar en pantalla los buckets antes de ordenarlos, después de ordenarlos y como queda la secuencia ordenada.

```
BucketSort(A)
Inicio
   n=número de elementos en A
   Formar arreglo B [0, … n-1]
   Para i=0 hasta n-1
      Crear en cada B[i] una lista vacía
   Fin Para
   Para i=1 hasta n
      Insertar A[i] en la lista B[  ⌊n A[i]⌋ ]
    Fin Para
    Para i=0 hasta n-1
        Ordenar B[i] con insertion Sort
    Fin Para
    Colocar en orden los elementos de cada lista B[0] , B[1].  … B [n-1] en A
Fin
```