Guía de Ejercicios Prácticos - Ficha 13

Sitio: Universidad Virtual UTN FRC
Curso: Algoritmos y Estructuras de Datos (2020)
Libro: Guía de Ejercicios Prácticos - Ficha 13
Imprimido por: Luciana Lisette Montarce
Día: lunes, 23 de noviembre de 2020, 21:23

Descripción

Esta guía contiene enunciados de algunos ejercicios para aplicar los conceptos de programación en ***Python*** que se analizan en la ***Ficha 13***. Los alumnos no deben subir nada al aula virtual: la guía se propone como fuente de ejercicios generales. Se sugiere intentar resolver cada uno de estos problemas ya sea trabajando solos o en grupos de estudio, y cuando las soluciones se publiquen, controlar lo hecho con las sugerencias propuestas por sus docentes. Utilice el foro del curso para plantear dudas y consultas, que cualquier alumno puede intentar responder.

Tabla de contenidos

1. Ordenar y buscar

Se pide un programa que cargue n elementos numéricos aleatorios entre 1 y 100 inclusive (pueden existir duplicados). A partir de ese arreglo:

1. Ordenarlo de forma ascendente y mostrarlo
2. Buscar un elemento x dentro del arreglo (x se ingresa por teclado). Si no existe, informarlo. Si existe, determinar cuántos valores impares mayores a x se encontraron en el arreglo.

1.1. principal.py

```
import random


def validar_mayor_que(minimo, mensaje):
    n = int(input(mensaje))
    while n <= minimo:
        print('INVALIDO! INGRESE OTRO')
        n = int(input(mensaje))
    return n


def binary_search(v, x):
    # busqueda binaria... asume arreglo ordenado...
    izq, der = 0, len(v) - 1
    while izq <= der:
        c = (izq + der) // 2
        if x == v[c]:
            return c
        if x < v[c]:
            der = c - 1
        else:
            izq = c + 1
    return -1


def selection_sort(v):
    # ordenamiento por seleccion directa
    n = len(v)
    for i in range(n - 1):
        for j in range(i + 1, n):
            if v[i] > v[j]:
                v[i], v[j] = v[j], v[i]


def contar_impares(v):
    print('Vector recortado:', v)
    cant = 0
    for i in range(len(v)):
        if v[i] % 2 != 0:
            cant += 1
    return cant


def contar_impares_no_tan_buena(v, x):
    # Recorre todos los elementos, cuando por estar ordenado el vector
    # se sabe que los mayores a x estan despues de ella
    cant = 0
    for i in range(len(v)):
        if v[i] % 2 != 0 and v[i] > x:
            cant += 1
    return cant


def contar_impares_tambien_buena(v, pos):
    # El recorrido empieza en la posicion de  + 1
    cant = 0
    for i in range(pos + 1, len(v)):
        if v[i] % 2 != 0:
            cant += 1
    return cant


def principal():
    n = validar_mayor_que(0, 'Ingrese cantidad de elementos del vector: ')
    v = [0] * n
    for i in range(n):
        v[i] = random.randint(1, 100)
```

```
    selection_sort(v)
    print(v)
    x = int(input('Ingrese elemento a buscar:'))
    pos = binary_search(v, x)
    if pos == -1:
        print('El elemento no existe')
    else:
        cant = contar_impares_no_tan_buena(v, x)
        print('Hay', cant, 'elementos impares mayores que', x)
        cant = contar_impares(v[pos + 1:])
        print('Hay', cant, 'elementos impares mayores que', x)
        cant = contar_impares_tambien_buena(v, pos)
        print('Hay', cant, 'elementos impares mayores que', x)


if __name__ == '__main__':
    principal()
```

2. Alumnos

Ingresar (o generar de manera aleatoria) los legajos de los n alumnos de un curso, siendo n un valor que se carga por teclado, y almacenarlos en un arreglo unidimensional. Se pide para ello:

a - Ordenar el arreglo de menor a mayor. Mostrar por pantalla como quedó.

b - Buscar en el arreglo el alumno con el legajo x, x se ingresa por teclado. Si existe mostrarlo, si no mostrar un mensaje de error.

2.1. alumnos.py

```
__author__ = 'Cátedra de Algoritmos y Estructuras de Datos'

import random

def cargar(alumnos):
    for i in range(len(alumnos)):
        alumnos[i] = int(input("Ingrese el legajo del alumno " + str(i) + ": "))

def cargar_random(alumnos):
    for i in range(len(alumnos)):
        alumnos[i] = random.randint(70000, 75000)

def mostrar(alumnos):
    for i in range(len(alumnos)):
        print("Legajo del alumno", i, ":", alumnos[i])

def validar():
    n = -1
    while n <= 0:
        n = int(input("Ingrese la cantidad de alumnos: "))
        if n <= 0:
            print("Valor incorrecto!")
    return n

def ordenar(alumnos):
    n = len(alumnos)
    for i in range(n-1):
        for j in range(i+1, n):
            if alumnos[j] < alumnos [i]:
                alumnos[i], alumnos[j] = alumnos[j], alumnos[i]


def buscar(alumnos, x):
    pos = -1
    izq = 0
    der = len(alumnos)-1
    while izq <= der:
        c = (izq+der)//2
        if alumnos[c] == x:
            return c
        elif x > alumnos[c]:
            izq = c + 1
        else:
            der = c - 1
    return pos




def test():
    n = validar()
    alumnos = [0] * n
    cargar_random(alumnos)
    mostrar(alumnos)

    ordenar(alumnos)
    print("Listado de Alumnos ordenado")
    mostrar(alumnos)

    x = int(input("Ingrese el legajo del alumno a buscar: "))
    pos = buscar(alumnos, x)

    if pos == -1:
        print("El legajo NO se encuentra!")
    else:
        print("El alumno estaba en la posición:", pos)
```

```
if __name__ == "__main__":
    test()
```

3. Concurso de baile

Desarrollar un programa que permita procesar el puntaje obtenido por una pareja de bailarines en un concurso de TV.

Para ello, generar un vector de 7 elementos, representando a los miembros del jurado. Por cada celda, generar un valor aleatorio entre -1 y 10 (inclusive) indicando la puntuación recibida.

A continuación, informar:

•Los tres mejores puntajes recibidos.
•Si algún jurado los calificó con 6. En caso afirmativo, indicar cuántas notas mayores a esa recibieron.
•La diferencia entre el mayor y el menor puntaje.
•El puntaje total obtenido. Si es menor a 20, indicar que quedan descalificados.

3.1. concurso.py

```
__author__ = 'Cátedra de Algoritmos y Estructuras de Datos'

import random
from arreglos import *


def generar_vector(v, n):
    for i in range(n):
        v.append(random.randint(-1, 10))


def contar_mayores(v, x, pos):
    cant = 0
    for i in range(pos + 1, len(v)):
        if v[i] > x:
            cant += 1
    return cant


def sumar_vector(v):
    suma = 0
    for valor in v:
        suma += valor
    return suma


def principal():
    print('CONCURSO DE BAILE')
    puntos = list()
    generar_vector(puntos, 7)
    print(puntos)
    selection_sort(puntos)
    print('Los 3 mejores puntajes fueron:', puntos[len(puntos) - 3:])
    pos = binary_search(puntos, 6)
    if pos == -1:
        print('No los calificaron con 6')
    else:
        cant = contar_mayores(puntos, 6, pos)
        print('Puntajes mayores a 6:', cant)
    dif = puntos[-1] - puntos[0]
    print('La diferencia entre el mayor y el menor es:', dif)
    total = sumar_vector(puntos)
    print('El puntaje total es:', total)
    if total < 20:
        print('Quedaron descalificados :(')


if __name__ == '__main__':
    principal()
```

4. Usuarios (sin repetir!)

Desarrollar un programa que permita gestionar la creación de n nuevos usuarios de un sistema (n se ingresa por teclado).

Se deben cargar los nombres de usuario uno a uno y guardarlos en un vector. El programa no debe permitir que se ingrese un nombre repetido.

4.1. usuarios

```
__author__ = 'Catedra de Algoritmos y Estructuras de Datos'

from arreglos import *

def principal():
    print('CREACIÓN DE USUARIOS')
    n = int(input('Ingrese cantidad de usuarios: '))
    v = list()
    for i in range(n):
        nombre = input('Ingrese nombre de usuario: ')
        while linear_search(v,nombre) != -1:
            nombre = input('Ya existe! Ingrese otro: ')
        v.append(nombre)
    print('\nUsuarios:', v)

if __name__ == '__main__':
    principal()
```