Universidad Rey Juan Carlos

Práctica 1: Recursividad

Diseño y Análisis de Algoritmos

- Los códigos tendrán que probarse con el juez automático **DOMjudge**
  - gibson.escet.urjc.es
  - El nombre de usuario será "team-XXX", donde XXX es un número único por cada alumno. Podéis ver qué número os corresponde en un documento subido al aula virtual que describe la relación entre el nombre de usuario y el nombre de un alumno.
  - La contraseña es vuestro DNI (incluida la letra final en mayúsculas).
- Además de probar vuestros códigos con DOMjudge debéis subir los ficheros fuente al aula virtual en un solo archivo (.zip o .rar)
- No se entregará una memoria
- Fecha límite: Se especificará en el campus virtual
- 5 % de la nota final

Índice

1. Raíz cuadrada mediante el método de Newton-Raphson (2 puntos)

1.1. Introducción

En este ejercicio el objetivo consiste en implementar el método (recursivo) de Newton-Raphson, el cual sirve para hallar ceros de una función $f(x)$.

1.2. Enunciado del problema

El método consiste en generar una secuencia de posibles valores que converja a un cero de $f(x)$. La regla para generar la secuencia es:

$$x_n = x_{n-1} - \frac{f(x_{n-1})}{f'(x_{n-1})}$$

y partirá de un valor inicial x_0. El método debe parar cuando $|f(x_n)| < \varepsilon$, para un valor de ε pequeño. En este ejercicio consideraremos $\varepsilon = 10^{-6}$. Además, usaremos el método para hallar $\sqrt{a}$, donde a es un número real no negativo. Para ello se debe buscar un cero de la función $f(x) = x^2 - a$. Finalmente, usaremos el valor inicial $x_0 = a$.

1.2.1. Descripción de la entrada

Simplemente contiene a, seguida de un salto de línea.

1.2.2. Descripción de la salida

La salida contendrá $\sqrt{a}$, especificado siempre con cuatro cifras decimales, y terminado en un salto de línea.

1.2.3. Ejemplo de entrada

```
2↵
```

1.2.4. Salida para el ejemplo de entrada

```
1.4142↵
```

2. Insertar elemento en una lista ordenada (2 puntos)

2.1. Introducción

En este ejercicio el objetivo consiste en implementar una función recursiva que se usará en una versión recursiva del algoritmo *insert-sort*, el cual se pide en el tercer ejercicio.

2.2. Enunciado del problema

Se pide implementar un método recursivo que reciba una lista ordenada **a** de n números enteros, y otro número x como parámetro. El algoritmo deberá devolver una nueva lista ordenada que contenga x y los elementos de **a**. Es decir, deberá insertar x en la lista **a** de manera que ésta siga estando ordenada.

2.2.1. Descripción de la entrada

La primera línea de la entrada contendrá el número de elementos (n) de la lista **a**. La segunda línea contendrá los n elementos de la lista, los cuales estarán separados por espacios en blanco. La tercera línea contendrá el elemento x a insertar en la lista **a**.

2.2.2. Descripción de la salida

La salida contendrá una línea, con los elementos de la lista resultante separados por espacios en blanco. La línea terminará en un salto de línea (no habrá un espacio después del último número).

2.2.3. Ejemplo de entrada

```
5↵
-3 0 3 8 11↵
5↵
```

2.2.4. Salida para el ejemplo de entrada

```
-3 0 3 5 8 11↵
```

3. Tupla de listas de elementos impares y pares (2 puntos)

3.1. Introducción

Se pide implementar una función que reciba una lista y genere una tupla con dos listas, una con los elementos impares de la original, y otra con los pares.

3.2. Enunciado del problema

Se pide implementar una función que reciba una lista (array unidimensional) de números enteros **a**, y genere una tupla `T` = (**b**, **c**), donde **b** y **c** son las listas con los elementos impares y pares, respectivamente, de **a**.

3.2.1. Descripción de la entrada

La primera línea de la entrada contendrá el número de elementos (n) de la lista **a**. La segunda línea contendrá los n elementos de la lista, los cuales estarán separados por espacios en blanco.

3.2.2. Descripción de la salida

La salida contendrá una línea, con la tupla `T` = (**b**, **c**), y un salto de línea. Para imprimirla solo será necesario emplear la instrucción `print(T)`. Los elementos de **b** y **c** estarán ordenados de la misma manera que en **a**.

3.2.3. Ejemplo de entrada

```
13↵
3 6 5 8 7 0 1 4 3 2 4 5 6↵
```

3.2.4. Salida para el ejemplo de entrada

```
([3, 5, 7, 1, 3, 5], [6, 8, 0, 4, 2, 4, 6])↵
```

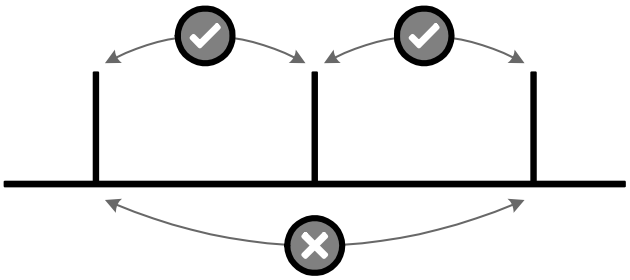
4. Variante de las torres de Hanoi (4 puntos)

4.1. Introducción

En este ejercicio el objetivo consiste en resolver una variante del problema de las torres de Hanoi.

4.2. Enunciado del problema

Asuma que las tres torres están dispuestas de izquierda a derecha. Las reglas del problema son las mismas que las del original, pero NO se permite mover discos directamente desde la torre de la izquierda a la de la derecha, y viceversa, como ilustra la siguiente figura:



Dados n discos ubicados en la torre de la izquierda, implementa un procedimiento recursivo que resuelva el problema, y mueva los n discos a la torre de la derecha.

4.2.1. Descripción de la entrada

En la primera línea se especifica $n \leqslant 9$, seguido de un salto de línea.

4.2.2. Descripción de la salida

La salida contendrá una línea de texto por cada movimiento que se realice de un disco. En concreto, se escribirá el texto:

```
Mueve disco X desde torre Y a torre Z↵
```

para mover el disco X desde la torre Y a la Z.

4.2.3. Ejemplo de entrada

```
2↵
```

4.2.4. Salida para el ejemplo de entrada

```
Mueve disco 1 desde torre 1 a torre 2
Mueve disco 1 desde torre 2 a torre 3
Mueve disco 2 desde torre 1 a torre 2
Mueve disco 1 desde torre 3 a torre 2
Mueve disco 1 desde torre 2 a torre 1
Mueve disco 2 desde torre 2 a torre 3
Mueve disco 1 desde torre 1 a torre 2
Mueve disco 1 desde torre 2 a torre 3
```