

Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación
Segundo Semestre de 2014

IIC 1103 - Introducción a la Programación

Midterm ⋆ Tiempo: 180 minutos ⋆ Sin apuntes ⋆ Sin consultas

1. *Pierre de Fermat*, famoso matemático francés del siglo XVII, demostró que un número primo p es expresable como suma de dos cuadrados si y sólo si $p = 2$ ó $p - 1$ es divisible por 4.

   Por ejemplo:

   $$29 = 2^2 + 5^2 = 4 \cdot 7 + 1 \qquad 41 = 4^2 + 5^2 = 4 \cdot 10 + 1$$

   En esta pregunta debes escribir un programa que pida al usuario un entero n e imprima los primeros n números primos que satisfacen esta propiedad, indicando cuál es su descomposición como suma de cuadrados. Por ejemplo, si se ingresa $n = 3$ tu programa debe mostrar:

   ```
   > Ingrese un número: 3
   > 2 = 1^2 + 1^2
   > 5 = 1^2 + 2^2
   > 13 = 2^2 + 3^2
   ```

2. En esta pregunta representaremos un vector tridimensional como una lista de tres números. Deberás crear un programa que calcule la posición y la rapidez de un objeto en un instante de tiempo t, aplicando la segunda ley de Newton. La fuerza ($\overrightarrow{F}$), la aceleración ($\overrightarrow{a}$), la posición inicial ($\overrightarrow{r_0}$), la posición ($\overrightarrow{r}$), y la velocidad ($\overrightarrow{v}$) son vectores tridimensionales.

   Tu programa debe pedir al usuario la posición inicial, la fuerza aplicada al objeto, el tiempo en que se quiere realizar el cálculo y la masa del objeto, e imprimir como resultado las tres componentes del vector posición y el valor de la rapidez en el tiempo indicado. Debes suponer que el objeto está inicialmente en reposo.

   Para evitar repetir código en tu programa, debes implementar y usar en él tres funciones:

   - Multiplicar cada una de las componentes de un vector en $\mathbb{R}^3$ por un valor escalar.
   - Sumar dos vectores en $\mathbb{R}^3$, componente a componente.
   - Calcular la norma de un vector en $\mathbb{R}^3$ (para el cálculo de la rapidez).

   Las siguientes ecuaciones te pueden ayudar a resolver el problema:

   $$\begin{aligned} \overrightarrow{F} &= m \cdot \overrightarrow{a} \\ \overrightarrow{v} &= \overrightarrow{v_0} + \overrightarrow{a}\,t \\ \overrightarrow{r} &= \overrightarrow{r_0} + \overrightarrow{v_0}t + \frac{1}{2}\overrightarrow{a}\,t^2 \\ \textit{rapidez} &= \sqrt{v_x^2 + v_y^2 + v_z^2}, \text{ para } \overrightarrow{v} = (v_x, v_y, v_z) \end{aligned}$$

3. La distancia *Levenshtein* corresponde al menor número de caracteres que hay que **insertar**, **borrar** o **sustituir** para transformar un string a otro. Por ejemplo:

   - Las palabras *gato* y *gatito* están a distancia 2, pues para llegar de una a otra basta instertar `i` e insertar `t` (o bien eliminar `i` y eliminar `t`).
   - Las palabras *hola* y *ola* están distancia 1; para llegar de una a otra basta instertar `h` (o borrar `h`).
   - Las palabras *gallina* y *gallina* están a distancia 0, pues son iguales.
   - Las palabras *caro* y *cara* están a distancia 1; para llegar de una a otra basta sustituir `o` por `a`.

   En esta pregunta debes escribir un programa que pida dos strings al usuario e imprima si ellos están a distancia de Levenshtein 0, mayor que 1, o igual a 1. Si la distancia es igual a uno, se debe indicar la operación (insertar/borrar, o sustituir). A continuación se muestran tres diálogos que ejemplifican cómo debiese funcionar tu programa:

   - ```
     > Palabra 1? jaron
     > Palabra 2? jarron
     > Respuesta: 1 operación (insertar/borrar)
     ```
   - ```
     > Palabra 1? Limon
     > Palabra 2? limon
     > Respuesta: 1 operación (sustituir)
     ```
   - ```
     > Palabra 1? jarron
     > Palabra 2? melon
     > Respuesta: más de 1 operación
     ```

4. Una aerolínea nacional guarda sus vuelos en una lista llamada `vuelos`. Cada entrada de la lista es una tupla que contiene el número y la fecha de un vuelo. Cada fecha, a su vez, es representada por una tupla `(año, mes, día)`. A continuación se muestra una posible lista `vuelos`.

   ```
   vuelos = [('AF10', (2014, 1, 2)), ('AF11', (2014, 1, 2)), ('AA12', (2014, 1, 3)),
             ('DE13', (2014, 5, 1)), ('DE14', (2014, 5, 1)), ('LA14', (2014, 7, 1))]
   ```

   Por otra parte, cada vuelo puede tener varios destinos. Esta información se guarda en una lista llamada `destinos`, en la que cada entrada es una tupla que contiene el número de vuelo y una lista de destinos. Cada destino, a su vez, se almacena en una tupla `(ciudad, pais)`. A continuación se muestra una posible lista `destinos`.

   ```
   destinos = [
     ('AF10', [('Lima', 'Peru'), ('San Jose', 'Costa Rica'), ('Los Angeles', 'USA')]),
     ('AF11', [('San Jose', 'Costa Rica'), ('C. de Panama', 'Panama')]),
     ('AA12', [('Sao Paulo', 'Brasil'), ('San Jose', 'Costa Rica')]),
     ('DE13', [('Lima','Peru'), ('San Jose', 'Costa Rica')]),
     ('DE14', [('San Jose', 'Costa Rica'), ('Buenos Aires', 'Argentina')]),
     ('LA14', [('San Jose', 'Costa Rica')])]
   ```

   En esta pregunta deberás escribir dos funciones.

   a) `vuelos_destino(vuelos, destinos, destino, fecha)`. Esta función debe retornar una lista con los vuelos que salen hacia `destino` en la fecha `fecha`.
   Por ejemplo, `vuelos_destino(vuelos, destinos, ('San Jose', 'Costa Rica'), (2014, 5, 1))` debe retornar `['DE13', 'DE14']`.

   b) `destinos_repetidos(destinos)`. Esta función debe retornar una lista de tuplas con los destinos que figuran en todos los vuelos.
   Por ejemplo, `destinos_repetidos(destinos)` debe retornar `[('San Jose', 'Costa Rica')]`
   ```