Examen de Programación II
(Ingeniería Informática)

Septiembre 2010

1) Lenguaje C (2 puntos)

Escribir el módulo "elimina_substring" (ficheros `elimina_substring.h` y `elimina_substring.c`) que defina una única función (`elimina_substring`) que retorna un string que es igual a otro que se pasa como parámetro al que se le ha eliminado un substring.

Por ejemplo, si se llama a la función `elimina_substring` pasándole el string `"rojo azul verde"` indicando que se quiere eliminar el substring comprendido entre los caracteres 4 y 8 la función retornaría el string `"rojo verde"`.

Una descripción más detallada de la función `elimina_substring` sería la siguiente:

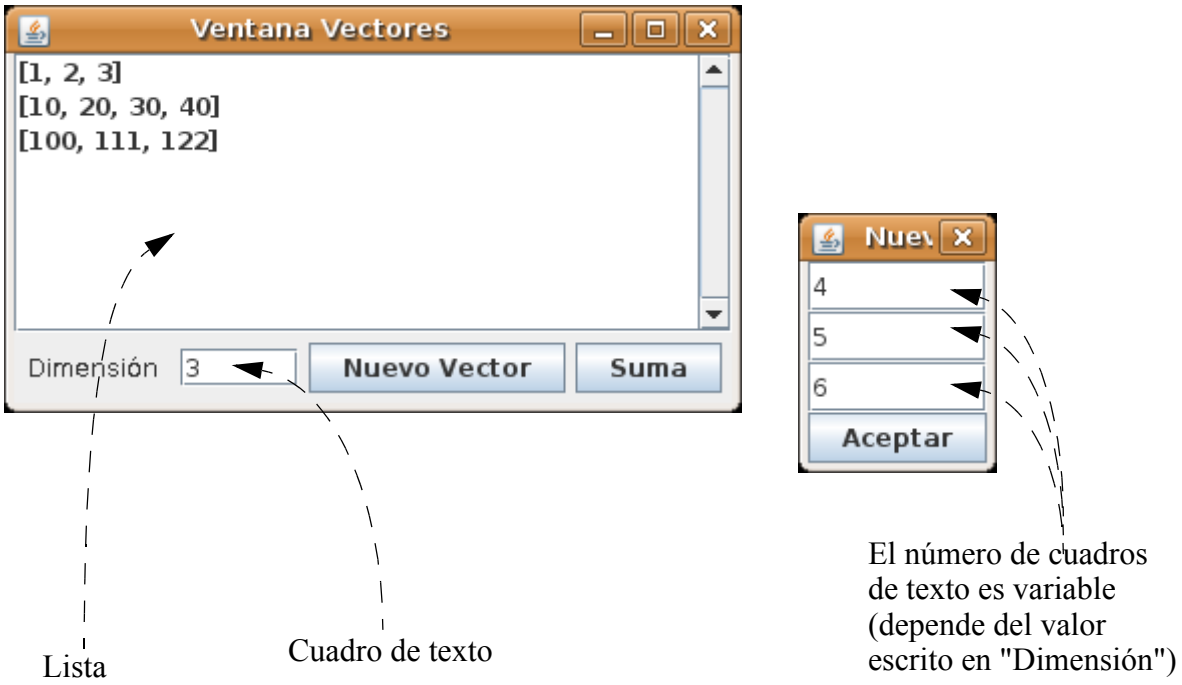
- `orig` (parámetro de entrada): string original (en el ejemplo `"rojo azul verde"`).
- `ini` (parámetro de entrada): posición del carácter de comienzo del substring a eliminar (en el ejemplo 4).
- `fin` (parámetro de entrada): posición del carácter final del substring a eliminar (en el ejemplo 8).
- `correcto` (parámetro de salida): devuelve un 0 si no se puede eliminar el substring, ya que el valor del parámetro `fin` es mayor que la longitud del string `orig`. Devuelve 1 en caso de que sí se pueda eliminar.
- valor de retorno: nuevo string sin el substring eliminado (en el ejemplo `"rojo verde"`). En las circunstancias en que `correcto` devuelve 0, la función retornará `NULL`.

Escribir un programa principal muy sencillo (que no pida datos al usuario) que utilice la función `elimina_substring`.

Indicar cual sería el comando que permitiría compilar y enlazar el programa principal desarrollado y crear un ejecutable llamado "`pru_elimina.exe`".

2) Programación dirigida por eventos (3.5 puntos)

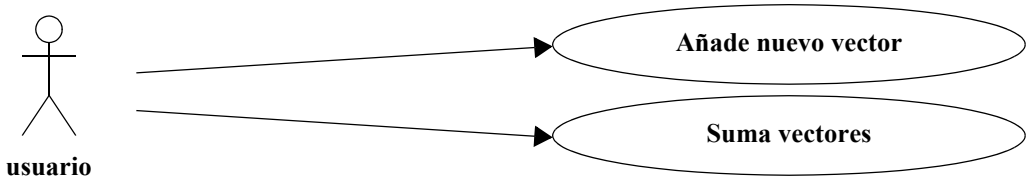
Se pretende realizar una aplicación gráfica formada por la ventana `VentanaVectores` y el diálogo `DialogoNuevoVector` que tienen la apariencia mostrada a continuación:



La ventana `VentanaRectangulos` tiene en su parte superior una lista con un conjunto de vectores de números enteros. En su parte inferior hay un cuadro de texto y dos botones. El cuadro de texto "Dimensión" sirve para indicar la dimensión del nuevo vector a crear. El botón "Nuevo Vector" provoca la aparición del diálogo `DialogoNuevoVector` que permite introducir las componentes de un nuevo vector que se añadirá a la lista. El número de cuadros de texto que tiene ese diálogo depende del valor introducido en el cuadro de texto "Dimensión".

El botón "Suma" permite sumar dos vectores seleccionados en la lista. El vector resultante de la suma se pone en la última posición de la lista.

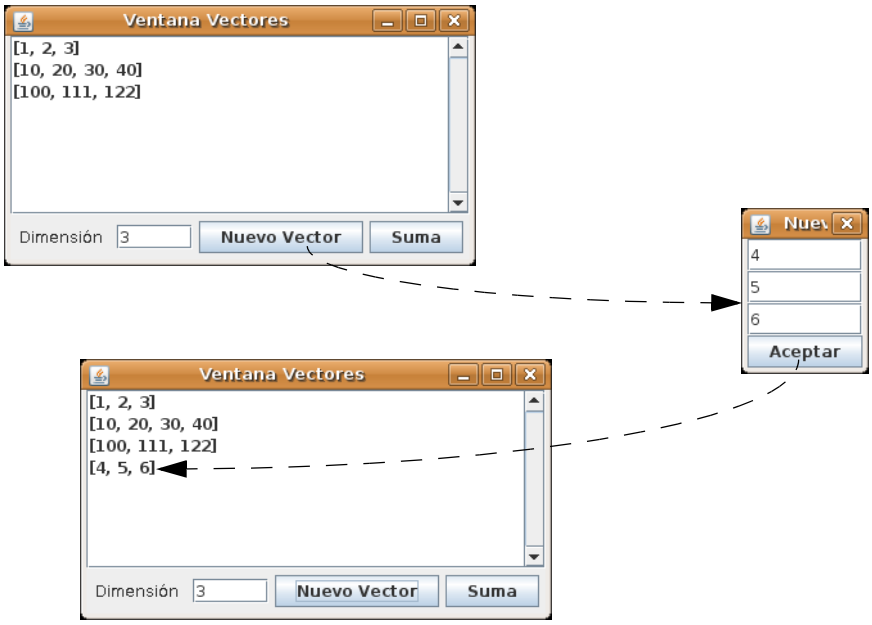
La interacción del usuario con la aplicación es la descrita en los siguientes caso de uso:



Caso de uso "Añade nuevo vector":

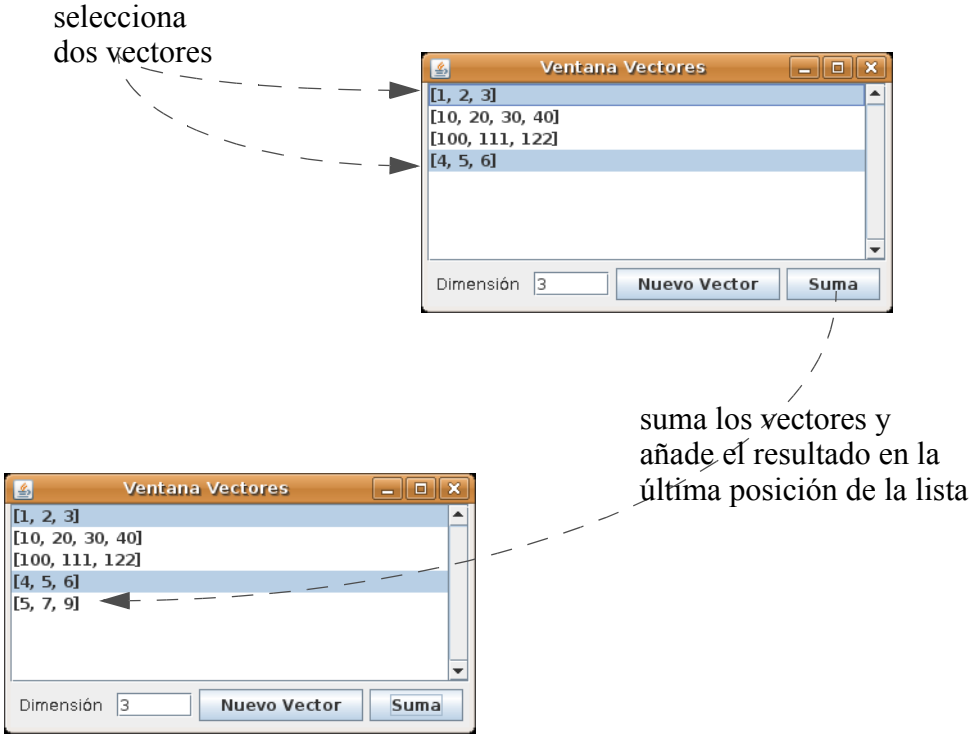
1. El usuario rellena el campo de texto "Dimensión" y pulsa el botón "Nuevo Vector".
2. La aplicación muestra el diálogo `DialogoNuevoVector` para introducir las componentes del vector (el diálogo tendrá tantos cuadros de texto como indique el valor existente en el cuadro de texto "Dimensión").
3. El usuario introduce las componentes del nuevo vector y pulsa aceptar o cierra el diálogo
   3.a. Si el usuario cierra el diálogo finaliza el caso de uso.

3.b. En el caso pulsar aceptar, la aplicación muestra el nuevo vector en la lista de la `VentanaVectores`.



Caso de uso "Suma vectores":

1. El usuario selecciona dos vectores de la lista y pulsa el botón "Suma".
2. Si el usuario ha seleccionado dos vectores de la misma dimensión la aplicación calcula el vector suma y le añade a la última posición de la lista
   2.a. En el caso de que el usuario haya seleccionado un número de vectores distinto de dos o que los vectores seleccionados no sean de la misma dimensión finaliza el caso de uso sin mostrar ningún mensaje de error.



La clase `VentanaVectores` se encuentra parcialmente implementada:

```
public class VentanaVectores extends JFrame {

  // botones
  private JButton bNuevoVector = new JButton("Nuevo Vector");
  private JButton bSuma = new JButton("Suma");

  // campo de texto
  private JTextField tfDimension = new JTextField("3", 5);

  // diálogo
  private DialogoNuevoVector diagNuevoVector =
     new DialogoNuevoVector(this);

  // lista
  private DefaultListModel lstVectoresModel =
     new DefaultListModel();
  private JList lstVectores = new JList(lstVectoresModel);

  // constructor
  public VentanaVectores() {
    super("Ventana Vectores");
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    lstVectores.setSelectionMode(
              ListSelectionModel.MULTIPLE_INTERVAL_SELECTION);
    JScrollPane spList = new JScrollPane(
          lstVectores,
          ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS,
          ScrollPaneConstants.
                    HORIZONTAL_SCROLLBAR_AS_NEEDED);
    add(spList, BorderLayout.CENTER);
    JPanel pBotones = new JPanel(new FlowLayout());
    pBotones.add(new Label("Dimensión"));
    pBotones.add(tfDimension);
    pBotones.add(bNuevoVector);
    pBotones.add(bSuma);
    add(spList, BorderLayout.CENTER);
    add(pBotones, BorderLayout.SOUTH);

    bNuevoVector.addActionListener(new ManejadorNuevoVector());
    bSuma.addActionListener(new ManejadorSuma());

    pack();
    setVisible(true);
  }

  // programa principal: crea la ventana
  public static void main(String[] args) {
    new VentanaVectores();
  }

  // clases manejadoras                    Hacer por el alumno
  ...
}
```

Se pide escribir el código de la aplicación completa salvo las partes que ya están implementadas. Puede ser necesario crear nuevas clases, métodos, atributos, etc. Las partes marcadas como "Hacer por el alumno" se indican a modo de orientación y recomendación, dependiendo de la implementación elegida por el alumno las partes a modificar podrían ser diferentes.

(Indicar claramente a qué parte de cada clase parcialmente implementada corresponde el código escrito).

Según la manera de resolver el problema podría ser necesario utilizar alguno de los métodos existentes en la clase `JDialog` que se describen a continuación:

- `remove`: elimina un componente del `JDialog`. Ejemplo de uso (desde un método del `JDialog`):
  ```
  remove(unCampoDeTexto);
  ```
- `dispose`: cierra y elimina un `JDialog`. Ejemplo de uso (desde un método del `JDialog`):
  ```
  dispose();
  ```

3) Esquemas algorítmicos (4.5 puntos)

3.a) (3.5 puntos) Resolver mediante un algoritmo basado en el esquema de "Vuelta Atrás" la formulación del problema del viajante que se expone a continuación.

Se desea encontrar el itinerario más corto que, comenzando por la primera ciudad, recorra un conjunto de ciudades pasando por cada una de ellas una única vez. Las distancias entre las ciudades se encuentran almacenadas en una matriz simétrica en la que el valor de la casilla `(i,j)` corresponde a la distancia entre las localidades `i` y `j`.

Ejemplo sencillo:

matriz de distancias

```
⎡  0.0 10.0  2.1  1.0 ⎤
⎢ 10.0  0.0  2.0  4.0 ⎥
⎢  2.1  2.0  0.0  8.5 ⎥
⎣  1.0  4.0  8.5  0.0 ⎦
```

En este caso el itinerario más corto que recorre las cuatro ciudades empezando por la 0 y sin repetir ninguna es:

```
mejor solución: 0 -> 3 -> 1 -> 2
                (distancia recorrida: 1.0 + 4.0 + 2.0 = 7.0)
```

Cualquier otro itinerario sería más largo, por ejemplo:

```
otra solución peor: 0 -> 2 -> 1 -> 3
                    (distancia recorrida: 2.1 + 2.0 + 4.0 = 8.1)
```

3.b) (0.5 puntos) Optimizar el algoritmo de forma que no se sigan explorando aquellas ramas en las que la distancia recorrida ya supere la distancia total de la mejor solución encontrada hasta el momento.

3.c) (0.5 puntos) Calcular la eficiencia del algoritmo desarrollado.

Para implementar el algoritmo se dispone de la clase `Itinerario` ya implementada, cuya interfaz es la mostrada en la página siguiente:

Itinerario(double[][] distancias, int ciudadOrigen)

Crea un itinerario que incluye sólo la ciudad de origen.

Parameters:
distancias - matriz de distancias entre ciudades
ciudadOrigen - ciudad comienzo del itinerario
última localidad visitada

boolean recorridoCompleto()

Comprueba si el itinerario es completo (contiene todas las ciudades)

Returns:
verdadero si el itinerario es completo y falso en caso contrario

void incluyeCiudad(int ciudad)

Incluye la ciudad al final del itinerario.

Parameters:
ciudad - ciudad a incluir

void eliminaUltimaCiudad()

Elimina la última ciudad incluida en el itinerario

boolean ciudadYaIncluida(int ciudad)

Permite conocer si una ciudad ya ha sido incluida en el itinerario

Parameters:
ciudad - ciudad a comprobar si está incluida
Returns:
verdadero si la ciudad ya está incluida en el itinerario y falso en caso contrario

Itinerario copia()

Realiza una copia idéntica al objeto actual

Returns:
copia del objeto actual

double distanciaRecorrida()

Retorna la distancia recorrida en el itinerario

Returns:
distancia recorrida en el itinerario