Introducción a la computación

Trabajo Práctico

Fecha de entrega: Viernes 17 de Junio de 2016.

1. Ejercicio 1

Se tiene un archivo de texto con un par de números (enteros o de punto flotante) por línea, que representan puntos en el plano 2D. Los números pueden ser positivos o negativos. Un ejemplo de entrada válida es la siguiente:

```
1.3 4.4
-1.0 5.3
0.0 2.3
-0.5 5.5
5 5.5
7.5 10.8
```

Se necesita, a partir de un archivo en este formato, obtener el par de puntos diferentes cuya distancia euclídea sea mínima. La distancia euclídea entre los pares de puntos (x_0, y_0) y (x_1, y_1) se define de la siguiente manera:

$$\text{distancia}((x_0, y_0), (x_1, y_1)) \equiv \|(x_1 - x_0, y_1 - y_0)\| = \sqrt{(x_1 - x_0)^2 + (y_1 - y_0)^2}$$

Se pide:

1. Implemente una función `listaDePuntos(fn)` que tome como entrada un archivo, cuyo nombre está indicado por `fn`, con una lista de puntos en el formato especificado y devuelva una lista de *tuplas* con los valores de cada punto del archivo.

   Ayuda 1: una tupla con valores `x` e `y` se define en Python como `(x, y)`.

2. Implemente una función `distanciaMinima(a)` que tome como entrada una lista como la definida en el punto anterior y devuelva cuál es el par cuya distancia es mínima. En el caso del ejemplo, el par de distancia mínima es el $(-1{,}0; 5{,}3), (-0{,}5; 5{,}5)$, cuya distancia es $0{,}5385$. La estrategia a utilizar es la de *fuerza bruta*, es decir calcular las distancias entre todos los pares de puntos (resulta en un orden de complejidad de $O(n^2)$ en la cantidad de puntos).

3. El algoritmo anterior puede ser mejorado en complejidad a $O(n \log^2 n)$ si se utiliza la técnica de *divide & conquer*. Diseñe e implemente una función `distanciaMinimaDyC(a, algoritmo)` siguiendo esta técnica, teniendo en cuenta el esquema descripto a continuación:

a) Preprocesamiento: la lista de puntos debe encontrarse ordenada en su coordenada x. Implemente *merge sort*, *bubble sort*, *upsort* y *quick sort*, y decida qué algoritmo utilizar en función del valor de la cadena de caracteres indicada por el parámetro `algoritmo`.

Los valores posibles son los siguientes:

- `merge`: utiliza *merge sort*.
- `bubble`: utiliza *bubble sort*.
- `up`: utiliza *upsort*.
- `quick`: utiliza *quick sort*.

En caso de que el algoritmo especificado no corresponda a ninguno de los anteriores, debe fallar con una excepción de tipo `NameError`.

b) Dividir los puntos en dos mitades cortando sobre el eje x.

c) Buscar recursivamente las distancias menores en ambas mitades, y quedarse con la menor de ambas.

d) Combinar dicho resultado con las distancias entre pares de puntos que se encuentren cada uno en una mitad diferente.

Ayuda 2: puede utilizar la menor de las distancias encontradas en los pasos recursivos como cota para determinar cuánto debe alejarse hacia cada lado en la coordenada x de la recta que corta ambas mitades, ya que la lista de puntos se encuentra ordenada en x[1].

4. Genere un gráfico en el cual se analice el tiempo de ejecución variando el tamaño del conjunto de puntos de entrada y midiendo los dos algoritmos implementados, incluyendo una curva por cada algoritmo de ordenamiento en el caso del algoritmo de *divide & conquer*. Saque conclusiones con respecto a la performance del algoritmo de *divide & conquer* con los distintos algoritmos de ordenamiento utilizados.

   Ayuda 3: Para tomar tiempos se puede usar `time.time()` (del módulo `time`) y para generar puntos de entrada se puede usar `random.random()` (del módulo `random`) que genera números aleatorios entre 0 y 1 (ajustar al rango de valores que necesite por medio de operaciones aritméticas).

2. Ejercicio 2

problema subListaAsc($[a : \mathbb{Z}], n : \mathbb{Z}) = res : \mathbb{B}$

Especifique, implemente y demuestre correctitud para el problema de indicar si una lista de enteros tiene una subsecuencia ascendente (monótona creciente) de longitud mayor o igual a n. El algoritmo propuesto deberá terminar el ciclo al encontrar la primera subsecuencia que cumpla lo pedido.

[1] Puede pensar que las soluciones posibles se categorizan en tres grupos: 1) los pares de puntos que se encuentran en el lado izquierdo, 2) los pares de puntos que se encuentran en el lado derecho y 3) los pares de puntos tales que van de un punto del lado izquierdo a un punto del lado derecho. El paso *combine* involucra encontrar potenciales soluciones que se encuentran en la categoría 3. La restricción de distancia se da por una propiedad matemática de las distancias euclídeas, ya que es imposible encontrar un par de puntos en esa categoría que se encuentre más cerca que los pares de puntos en las categorías 1 y 2 si alguno de los dos puntos que constituyen el par se encuentra más alejado de la mitad que la menor de las distancias mínimas encontradas recursivamente.

Debe implementar la función `sufijoSubAscN`(a, i, n), que devuelve `True` si la lista `a[i:]` es una subsecuencia ascendente (monótona creciente) de longitud mayor o igual a n. Debe escribir la especificación de esta función, pero no es necesario demostrar su correctitud.

Ayuda 4: utilice la función `sufijoSubAscN` para implementar `subListaAsc`.

3. Ejercicio 3

El juego rompecabezas deslizante consiste en un tablero rectangular de tamaño $N \times M$ (N filas por M columnas) de piezas cuadradas formando una grilla, donde uno de los lugares se encuentra libre[2]. Los únicos movimientos que se pueden hacer son llevar cualquiera de las piezas adyacentes al lugar vacío, intercambiando su lugar con este último. Cada pieza puede ser un número, una letra o un fragmento de una imagen. El objetivo de este juego es reordenar las piezas hasta que se obtenga la secuencia ordenada o la imagen original.

Se tiene una implementación parcial que modela este juego y su resolución computacional, con un programa que permite visualizarla, pero falta el código que permite que todo funcione. Su objetivo será encargarse de esta tarea.

Para que todo vuelva a funcionar, se pide completar los siguientes métodos de la clase `Rompecabezas` (en el archivo `slidingpuzzle.py`) para que cumplan con la funcionalidad requerida en cada caso:

1. `cargar(self, fn)`: abre, lee, procesa el archivo de texto cuyo nombre se indica en la variable `fn`, cargando el rompecabezas allí descripto. El archivo debe respetar el formato indicado en el apéndice B. Debe fallar con una excepción de tipo `IOError` si no se puede acceder al archivo o si el formato del mismo no es el correcto.

2. `alto(self)`: devuelve la cantidad de filas del rompecabezas.

3. `ancho(self)`: devuelve la cantidad de columnas del rompecabezas.

4. `get(self, i, j)`: devuelve una *string* con el valor almacenado en la fila `i`, columna `j`.

5. `__str__(self)`: devuelve una cadena de caracteres con una representación textual del rompecabezas. La misma debe corresponder al formato indicado en el apéndice B.

6. `resuelto(self)`: devuelve `True` si el rompecabezas está resuelto, es decir, si la secuencia almacenada se encuentra ordenada crecientemente, de izquierda a derecha y de arriba a abajo, o `False` en otro caso.

7. `mover(self, direccion)`: intercambia el espacio vacío con la pieza que se encuentra en la dirección especificada por `direccion`, en caso de que este movimiento sea posible. En caso de no serlo, deja el tablero intacto. Devuelve `True` en caso de éxito, `False` si no es posible el movimiento. Las direcciones se especifican de la siguiente manera:

   - Arriba: 0

[2] `https://es.wikipedia.org/wiki/Rompecabezas_deslizantes`

- Izquierda: 1
- Abajo: 2
- Derecha: 3

Por conveniencia, las variables `UP`, `LEFT`, `DOWN` y `RIGHT` se encuentran ya definidas con los valores correspondientes dentro del módulo `slidingpuzzle`.

8. `guardar(self, fn)`: guarda la representación textual del rompecabezas en el archivo cuyo nombre se encuentra indicado por `fn`.

9. `resolver(self, n)`: resuelve el rompecabezas utilizando la técnica de *backtracking*, probando todas las secuencias de movimientos posibles de hasta `n` movimientos.

   Devuelve `True` si logra resolver el rompecabezas en la cantidad de pasos indicados, `False` si esto no es posible.

   Importante: Para ver actualizaciones en pantalla durante el *backtracking*, cada vez que cambie la posición de una pieza es necesario llamar al método `self.update()`, ya provisto por la cátedra.

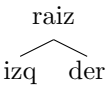
Algunas notas adicionales que pueden ser útiles:

- Se recomienda programar los métodos pedidos en el orden en el que aparecen en el enunciado, de forma de poder ir probando de manera parcial la funcionalidad implementada.
- Una vez finalizada la implementación de todos los métodos anteriores a `resolver(self, n)`, se puede utilizar la funcionalidad de generación de nuevos rompecabezas. Se recomienda probar esta funcionalidad antes de proseguir con la implementación del backtracking.
- Para hacer pruebas se puede utilizar el modo interactivo del intérprete sobre el código de la clase Rompecabezas (ejecutando en la terminal el comando `python3 -i slidingpuzzle.py`).
- Como guía para elegir la forma de almacenamiento de los datos dentro del TAD, pensar cómo hacer lo más sencillo posible devolver lo pedido en cada método.
- Se supone que ustedes deben elegir las variables para almacenar el estado del laberinto, que deberán ser marcadas como *privadas*, de manera de no utilizarse fuera de la clase.
- Pueden agregar métodos privados, pero no modificar la interfaz pública de la clase.

4. Ejercicio 4

Definir una clase en `Python` que implemente el TAD `Arbol` (un árbol binario, que tiene una raíz, una rama derecha y una izquierda). Debe tener los siguientes métodos:

1. `__init__(self, raiz=None, izquierda=None, derecha=None)`: devuelve un árbol con `raiz` como raíz del árbol, y `izquierda` y `derecha` como subárboles.

   Si `raiz is None`, se considera que el árbol es vacío, en cuyo caso `izquierda` y `derecha` deben ser `None`; de otro modo, se debe levantar una excepción `TypeError`.

   Si `raiz` no es `None` y `izquierda` o `derecha` son `None`, se los debe reemplazar por un árbol vacío.

2. `vacio(self)`: indica si el árbol no posee ningún elemento.

3. `raiz(self)`: devuelve el elemento correspondiente a la raíz del árbol.

4. `izquierda(self)`: devuelve el subárbol correspondiente a la rama izquierda.

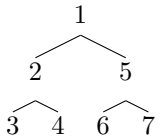
5. `derecha(self)`: devuelve el subárbol correspondiente a la rama derecha.

6. `__eq__(self, otro_arbol)`: indica si `otro_arbol` es igual a `self`. Decimos que dos árboles son iguales si tienen igual raíz, y sus subárboles a izquierda y derecha son iguales

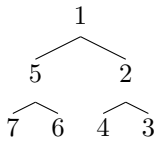
7. `find(self, a)`: indica si el elemento `a` está en el árbol.

8. `espejo(self)`: devuelve un nuevo árbol "espejado", que resulta de intercambiar los subárboles espejados de izquierda a derecha y viceversa.

   Por ejemplo

   se convierte en

9. `preorder(self)`: retorna una lista de enteros que resulta de colocar la raíz, seguido del preorder de las ramas izquierda y derecha.

10. `posorder(self)`: retorna una lista de enteros que resulta de colocar los posorder de las ramas derecha e izquierda y luego la raíz.

11. `inorder(self)`: retorna una lista de enteros que resulta ser el recorrido de los elementos del parámetro implícito de izquierda a derecha como si se lo aplastara (la raíz queda entre las ramas izquierda y derecha).

El código provisto en el ZIP `src_ej4.zip` contiene el constructor y los tests para las funciones 2 y 3.

Se debe escribir dos o más casos de test (siguiendo la estrategia de *caja negra*) para cada una de las funciones de la 4 a la 11. También se debe implementar un caso de prueba sobre el constructor para verificar que se levante una excepción en el caso de un árbol mal formado.

Los tests escritos tienen, en lo posible, que capturar la mayor cantidad de escenarios posibles, entre ellos aquellos casos *borde*: por ejemplo, si el árbol es vacío. Utilizar nombres descriptivos para los tests, evitando nombres del estilo `test_1, test_2, etc`. Finalmente, se debe lograr que pasen no sólo los nuevos tests sino también aquellos provistos por la cátedra.

La implementación de los casos de tests deberá ser realizada utilizando la herramienta `unittest`[3].

[3]Se explicará en clase

A. Cómo utilizar el visualizador para el Ej. 3

El programa provisto en el archivo `src_ej3.zip` nos permite cargar, mostrar en pantalla, resolver y generar nuevos rompecabezas, así como también visualizar el mecanismo de backtracking utilizado en la resolución. Para ejecutarla, descomprimir el ZIP, y en una terminal posicionada adecuadamente, ejecutar el siguiente comando: `./tppuzzle.py <argumentos>`[4].

Esto inicializará el programa, mostrando una ventana con el rompecabezas indicado, e iniciará la resolución o generación según corresponda por cómo se haya invocado el programa. Si se cierra la ventana o se hace click en cualquier parte del rompecabezas, el programa termina.

Al ejecutar el programa sin argumentos, imprime en la terminal la siguiente ayuda, que indica cómo se lo debe invocar en el modo resolución y en el modo generación según corresponda:

```
Uso:
  ./tppuzzle.py --generate archivotxt imagen ancho alto pasos [velocidad]
o bien:
  ./tppuzzle.py archivotxt imagen [velocidad]

Si es especifica --generate, el programa generará una nueva instancia de
rompecabezas de manera aleatoria. En otro caso, resolverá el rompecabezas
indicado por archivotxt.

* imagen: indica el nombre de una imagen a rearmar en el rompecabezas.
* velocidad: opcional, indica cuántos pasos se realizan por segundo
(default: 500).
* ancho: solo para --generate indica el ancho en bloques del rompecabezas.
* alto: solo para --generate, indica el alto en bloques del rompecabezas.
* pasos: solo para --generate, indica la cantidad de pasos aleatorios
utilizados en la generación del nuevo rompecabezas.
```

[4]El programa está implementado utilizando la biblioteca **pygame**. Esta biblioteca solamente funciona en Python 2.7 en las computadoras de los laboratorios del DC, pero el código de `slidingpuzzle.py` está adaptado para puedan programar en Python 3 sin problemas. Si utilizan su computadora personal, deben instalarla para que el programa funcione (ver `http://www.pygame.org/download.shtml`).

B. Formato del rompecabezas

Un rompecabezas se representa por medio de un archivo de texto en formato TSV[5], que es idéntico al formato CSV con la excepción de que el separador entre elementos es un tab (el caracter `'\t'`) en lugar de una coma.

La cantidad de filas del rompecabezas está dada por la cantidad de líneas del archivo, y un rompecabezas bien formado debe tener el mismo número de elementos por fila. El lugar en blanco se representa por un espacio (el caracter `' '`), y debe estar presente en el archivo.

Cada una de las celdas se representa utilizando un valor numérico que comienza a partir del 1. El rompecabezas resuelto se representa con una secuencia de números consecutivos ordenados según el criterio implementado en el método `resuelto(self)`.

Junto al código fuente del programa `tppuzzle.py` van encontrar varios ejemplos en este formato, de extensión `*.txt`, listos para probar con su implementación:

- `puzzle1.txt`: rompecabezas de 6×6, desordenado. ¿Qué pasa con la cantidad de pasos necesarios para resolver este rompecabezas?
- `puzzle2.txt`: rompecabezas de 4×4, desordenado.
- `puzzle3.txt`: rompecabezas de 4×4, casi ordenado.
- `puzzle4.txt`: rompecabezas de 4×4, ya ordenado.

[5] Tab-separated Values, `https://en.wikipedia.org/wiki/Tab-separated_values`

Condiciones de entrega:

- Entregar una versión impresa que contenga los archivos fuentes correspondientes a cada ejercicio y un informe (en formato PDF). No incluir en la impresión el código ya provisto.
- El informe debe incluir:
  - **Ej. 1**: decisiones de diseño y cómo correr las pruebas necesarias para generar el gráfico incluido en el informe, además del análisis del mismo.
  - **Ej. 2**: la demostración completa.
  - **Ej. 3**: decisiones de diseño (atributos que hayan definido en la clase `Rompecabezas`, cómo se almacenan en memoria los datos que cargan, estrategia del backtracking, etc.)
  - **Ej. 4**: decisiones de diseño, descripción de los casos de test, cómo ejecutarlos.
  - El log del repositorio donde se encuentra el TP.
  - Cualquier otra aclaración que consideren pertinente.
- Los archivos fuentes deberán tener comentarios.
- La entrega electrónica se realiza vía Bitbucket.
- **Nota**: se puede tomar coloquio individual a criterio de los docentes sobre cualquier tema del TP.
- Una vez terminado, deben enviar un mail de aviso a la dirección de correo electrónico de los docentes de la materia: **icb-doc@dc.uba.ar**. Deberán poner como *subject*:
  [Grupo ⟨NN⟩: ⟨Nombre grupo⟩] TP - ⟨Apellido1 LU1⟩ - ⟨Apellido2 LU2⟩. Por ejemplo, un grupo podría ser:
  [Grupo 99: "Pajarones"] TP - Gómez 334/89 - González 671/16
- El cuerpo del mail de aviso debe indicar el comando que tiene que ejecutar el docente para clonar el repositorio donde se encuentra el TP.