Puntos de articulación. Árboles abarcadores mínimos.

Diseño y análisis de algoritmos 2019-2020

Práctica 3

ÍNDICE

Fecha límite de entrega: jueves 18 de diciembre de 2019, 23:59 horas

AVISO IMPORTANTE: PAUTAS A SEGUIR EN LA ENTREGA

Leer con cuidado y ajustarse a las mismas. De no seguirse se aplicará una penalización.

1. **Ajustarse escrupulosamente a las instrucciones de entrega al final del texto de esta práctica.**
2. **Ajustarse escrupulosamente al procedimiento de comprobación del código indicado al al final del texto de esta práctica.**
3. Todas las gráficas se generarán con `matplotlib` **con fondo blanco.**
4. El nombre de los estudiantes y el número de pareja se identificará claramente en la memoria.
5. Todas las funciones desarrolladas en esta práctica deberan incorporar su correspondiente docstring, así como incoporar unos controles `assert` adecuados.
6. El código se deberá ajustar razonablemente al estándar Python PEP 8. Utilizar por ejemplo para ello un formateador de código como Autopep8, Black o similar.

I. CIFRADO MERKLE–HELLMAN

I-A. Generar sucesiones supercrecientes

Vamos a generar sucesiones supercrecientes ordenadas aleatorias con un número `n\_terms` de términos y

- con un valor inicial aleatorio;
- con un valor s_i que supere a $\sum_0^{i-1} s_j$ en otro entero aleatorio.

Escribir una función

```
gen_super_crec(n_terms)
```

que devuelva una tal sucesión. Tanto esta función como todas las siguientes deben trabajar con enteros Python 3 de longitudes esencialmente arbitrarias.

I-B. Módulo, multiplicador e inverso, y sucesión pública

Para el cálculo del módulo, multiplicador e inverso, y una sucesión pública,

1. Escribir una función
   `multiplier(mod, mult_ini)`
   que devuelva un entero primo relativo con el módulo `mod` y sea superior al entero `mult_ini` (para evitar multiplicadores demasiado pequeños). Para ello generar números aleatorios en el rango en cuestión hasta encontrar uno primo relativo con `mod`. Comprobad esto mediante una función `mcd(x, y)` que implemente el algoritmo de Euclides de cálculo del máximo común divisor entre `x, y`.
2. Escribir una función
   `inverse(p, mod)`
   que devuelva el inverso de `p` módulo `mod`.
3. Escribir una función
   `mod_mult_inv(l_sc)`
   que calcule un módulo adecuado para la sucesión supercreciente `l_sc`, esto es, un entero aleatorio mayor que la suma de sus términos, y devuelva el multiplicador, su inverso y el módulo en este orden.
4. Escribir una función
   `gen_sucesion_publica(l_sc, p, mod)`
   que devuelva la solución publica asociada a la la sucesión supercreciente `l_sc`, al multiplicador `p` y al módulo `mod`.
5. Escribir una función
   `l_publica_2_l_super_crec(l_pub, q, mod)`
   que devuelva la solución privada asociada a la sucesión pública `l_pub`, al inverso `q` y al módulo `mod`.

I-C. Cifrado de cadenas binarias y su descifrado

Queremos cifrar mediante el cifrado Merkle–Hellman (o de la mochila) una cadena aleatoria de 0 y 1 usando la sucesión pública generada con los métodos anteriores. Para ello,

1. Escribir una función
   `gen_random_bit_list(n_bits)`
   que devuelva una lista de `n_bits` aleatorios.
2. Escribir una función
   `mh_encrypt(l_bits, l_pub, mod)`
   que devuelva una lista con el cifrado de cada bloque de la lista binaria `l_bits` mediante la lista pública `l_pub` y el módulo `mod`. Si es necesario, añadir ceros al final de `s` para que su longitud sea un múltiplo de la de `l_pub`.

A continuación desarrollamos el procedimiento de descifrado. Para ello

1. Escribir una función
   `mh_block_decrypt(c, l_sc, inv, mod)`
   que descifre un mensaje cifrado mediante un entero `c` mediante la sucesión supercreciente `l_sc` y los enteros inverso y módulo `inv, mod`.
2. Escribir una función
   `mh_decrypt(l_cifra, l_sc, inv, mod)`
   que devuelva una cadena de bits conteniendo el descifrado de los sucesivos enteros en la lista `l_cifra` mediante la sucesión supercreciente `l_sc` y los enteros inverso y módulo `inv, mod`

I-D. Cuestiones sobre el cifrado Merkle-Hellman

Contestar razonadamente a las siguientes cuestiones.

1. Dado el posible tamaño de los términos de la sucesión supercreciente, es necesario trabajar con enteros de tamaño adecuado. Averiguar el tamaño máximo de un entero en Python.
2. Un elemento importante en el algoritmo Merkle–Hellman es la longitud de las sucesiones empleadas, lo que a su vez influye en el valor máximo de sucesión supercreciente y el módulo. Si dicha sucesión tiene N términos, estimar los valores mínimos del último término de uns sucesión supercreciente de N términos y del módulo. Sugerencia: considerar el ejemplo de la sucesión $s_n = 2^n, n = 0, 1, 2, \ldots$
3. A la vista de las dos cuestiones previas, discutir cuál puede ser la longitud máxima razonable de la sucesión supercreciente.
4. Un enfoque trivial para la función `inverse(p, mod)` es probar con enteros de manera iterada hasta encontrar un `q` tal que `p * q % mod == 1`. Sin embargo, esto es muy costoso computacionalmente y se puede mejorar mediante una variante del algoritmo de Euclides. Describir aquí dicha variante y estimar su coste computacional.

II. Algoritmos de programación dinámica

II-A. *Dando cambio*

Nuestro primer enfoque al problema de dar cambio se ha centrado en obtener el mínimo número de monedas necesarias. Sin embargo, lo realmente importante es saber cómo dar cambio, esto es, el tipo y número de monedas a devolver.

Para abordar esta cuestión

- Escribir una función
  `min_coin_number(c, l_coins)`
  que devuelva el mínimo número de monedas de `l_coin` necesarias para dar cambio de una cantidad `c`.
- Escribir una función
  `optimal_change(c, l_coins)`
  que devuelva un dict con las monedas de cada tipo en `l_coin` necesarias para dar cambio de una cantidad `c`.

II-B. *Encontrando subsecuencias*

Hemos visto en clase un algoritmo de programación dinámica que proporciona la subsecuencia común más larga entre dos cadenas. Pero tanto o más importante es obtener una tal cadena común. Para ello

- Escribir una función
  `max_length_common_subsequence(str_1, str_2)`
  que reciba dos cadenas `str_1, str_2` y devuelva la matriz de longitudes de subsecuencias máximas comunes parciales.
- Escribir una función
  `find_max_common_subsequence(str_1, str_2)`
  que reciba dos cadenas `str_1, str_2` y, aprovechando por ejemplo la función anterior, devuelva una posible subsecuencia común de longitud máxima.

II-C. *Arboles binarios de búsqueda óptimos*

Nuestro algoritmo para calcular la estructura de un árbol binario de búsqueda óptimo también puede calcular las raíces de los diferentes subárboles intermedios. A su vez, esto puede servir para obtener el orden óptimo de inserción de las claves en el árbol de búsqueda final.

Para obtenerlo

- Escribir una función
  `optimal_order(l_probs)`
  que recibe una lista de probabilidades asociada a claves `0, 1, ...` y devuelve una matriz con los costes óptimos de búsqueda en subárboles y otra con sus correspondientes raíces.
- Escribir una función
  `list_opt_ordering_search_tree(m_roots, l, r)`
  que recibe la matriz de raíces devuelta por la función anterior y unos índices `l, r` izquierdos y derechos, y devuelve una lista con el orden de inserción de las claves `l, l+1, ..., r` en el correspondiente árbol binario de búsqueda óptimo.

II-D. *Cuestiones sobre los algoritmos anteriores*

Contestar razonadamente a las siguientes cuestiones.

1. Estimar en detalle el coste computacional del algoritmo usado en la función `optimal_order(l_probs)`.
2. El problema de encontrar la máxima subsecuencia común (no consecutiva) a veces se confunde con el de encontrar la máxima subcadena (consecutiva) común. Ver por ejemplo la entrada Longest common substring problem en Wikipedia. Describir un algoritmo de programación dinámica para encontrar dicha subcadena común máxima y aplicarlo "a mano" a las cadenas de los primeros apellidos de los miembros de tu pareja de prácticas.

III. Material a entregar y corrección

III-A. *Material a entregar*

Crear una carpeta de nombre `p2NN` donde `NN` indica el número de pareja e incorporar a la misma **únicamente** los siguientes archivos:

1. Archivo del módulo Python `prog_din_NN.py` que contenga las funciones desarrolladas en la práctica.
   Dicho archivo contendrá **únicamente** los `import` imprescindibles para la comprobación de la práctica (por ejemplo, NO DEBE importar `matplotlib`.
   La corrección de la práctica se efectuará en un shell Linux como Ubuntu o similares, por lo que se debe asegurar que `grafosNN.py` se pueda usar sin errores en dicho entorno.

Los nombres y parámetros de las funciones definidas en ellos deben ajustarse EXACTAMENTE a los usados en este documento.

2. Archivo `prog_din_NN.html` con el resultado de aplicar al módulo Python la herramienta `pydoc3`.
3. Archivo `memoP3NN.pdf` con una breve memoria que **únicamente** contenga las respuestas a las cuestiones en formato pdf. El archivo debe incluir el número de pareja y los nombres de sus miembros.

Comprimir dicha carpeta en un archivo `.zip` o `.7z` de nombre `p2NN` **No añadir a la carpeta ninguna subestructura de subdirectorios.**

No se corregirá la práctica hasta que la entrega siga esta estructura y se podrán penalizar reenvíos debidos a no tenerse esto en cuenta.

III-B. Corrección

La corrección de la práctica se va a efectuar en función de los siguientes elementos:

- Ejecución de un script o notebook que reciba unos datos (parámetros, grafos concretos) para comprobación de la corrección del código en los módulos Python. Los mismos se situarán en Moodle antes de la entrega de práctica.
  Es muy importante que los nombres de funciones y argumentos, así como los valores devueltos por las distintas funciones que componen la práctica se ajusten a lo indicado en los distintos apartados anteriores de este guión.
  No se corregirá la práctica mientras estos scripts no se ejecuten correctamente, penalizándose segundas entregas debidas a esta causa.
- Revisión de la documentación del código contenida en los archivos html generado mediante `pydoc` con los docstrings y otros elementos de los módulos a entregar.
  docstrings y asserts deben cuidarse particularmente.
- Revisión de una pequeña selección de las funciones Python contenidas en los módulos.
- Revisión de la memoria de resultados con las respuestas a las cuestiones anteriores.