FACULTAD DE INFORMÁTICA
GRADO EN CIENCIA E INGENIERÍA DE DATOS

Diseño y Análisis de Algoritmos
Práctica 4: Programación dinámica

25 de noviembre de 2024

Dados dos strings (o listas de enteros) A y B, se considera que un tercer string C es una mezcla válida de ambos si cumple los siguientes requisitos:

- C contiene todos los caracteres que están en A y B, con el mismo número de repeticiones de cada caracter (se distingue entre mayúsculas y minúsculas).
- El orden en el que aparecen los caracteres en A y B se preserva en C.

Por ejemplo, para las palabras A = *Hello* y B = *World*, las siguientes combinaciones **son válidas:** *HelloWorld, WorldHello, HWorellldo, WorHellold, HWeolrllod.* Por contra, las siguientes combinaciones **no son válidas:** *dlroWolleH, oHelloWrld, HelloWorlds, HeloWorld, HelloWooorld.*

Algorithm 1

```
function isMixtureDP (A[1..n], B[1..m], C[1..s]):
    if n + m ≠ s:
        return False;
    #tabla de booleanos t[1..n+1, 1..m+1] a False
    t[1, 1] := True;
    for i := 1 to n+1:
        for j := 1 to m+1:
            t[i,j] := t[max(1,i-1),j] or t[i,max(1,j-1)];
            if t[i,j] and (i<n+1 or j<m+1):
                k := i+j-1;
                t[i,j] := False;
                if i <n+1:
                    t[i,j] := t[i,j] or A[i] = C[k];
                if j <m+1:
                    t[i,j] := t[i,j] or B[j] = C[k];
    return t[n+1, m+1];
```

Algorithm 2

```
function isMixtureCX (A[1..n], B[1..m], C[1..s]):
    if n + m ≠ s:
        return False;
    Known := {(1,1)}; # conjunto
    Trial := [(1,1)]; # pila
    while |Trial| > 0:
        (i,j) := last element in Trial;
        remove (i,j) from Trial;
        k := i+j-1;
        if k >s:
            return True;
        if i<=n and A[i] = C[k] and
                (i+1,j) ∉ Known:
            Trial := Trial + [(i+1,j)];
            Known := Known ∪ {(i+1,j};
        if j<=m and B[j] = C[k] and
                (i,j+1) ∉ Known:
            Trial := Trial + [(i,j+1)];
            Known := Known ∪ {(i,j+1};
    return False;
```

Se pide:

1. Implementar en Python ambos algoritmos incluidos en este documento, modificándolas para que, dada una combinación correcta, devuelva, al menos, una asignación óptima, indicando a qué string, A o B, pertenecería cada uno de los caracteres de C.
2. Validar el correcto funcionamiento de todas las implementaciones. Al inicio del documento se exponen unos casos básicos. Considere más ejemplos en caso de que lo crea necesario.
3. Calcular empíricamente la complejidad computacional de cada algoritmo **en el caso de que la palabra mezclada sea válida**. Se considerará que el tamaño de A y B es el mismo. Se probarán 3 escenarios distintos:
   - Con un vocabulario de dos únicos elementos (ej: '0' y '1').
   - Con todos los caracteres del código ascii (van del 0 al 255).
   - Con un vocabulario de igual tamaño que la longitud de A y B.

   Para **generar las palabras aleatoriamente y mezclarlas**, puede utilizar el código de la figuras 1 y 2, respectivamente. Como referencia, puede tomar los valores [20, 40, 80, 160, 320, 640, 1280, 2560] para generar las palabras. Igualmente se realizará una comprobación empírica utilizando una cota subestimada y otra sobrestimada.

```python
def create_word(n, alphabet=(0, 1)):
    a = np.random.randint(low=0, high=len(alphabet), size=(n,))
    word = np.array(alphabet)[a].tolist()
    return word
```

Figura 1: Código de ejemplo para crear palabras aleatoriamente. Devuelve una lista de números.

```python
def mix_words(a, b, valid=True):
    new_word = []
    a_array = np.array(a)
    b_array = np.array(b)
    if not valid:
        np.random.shuffle(a_array)
        np.random.shuffle(b_array)
    a_index = 0
    b_index = 0
    while len(new_word) < len(a) + len(b):
        p = np.random.randint(2)
        if (p and a_index < len(a_array)) or b_index >= len(b_array):
            new_word.append(a_array[a_index])
            a_index += 1
        else:
            new_word.append(b_array[b_index])
            b_index += 1
    return new_word
```

Figura 2: Código de ejemplo para crear palabras fusionadas aleatoriamente.

- La fecha de entrega límite es el día **6 de diciembre de 2024** a las 23:50 horas.
- Será necesario depositar en la página de la asignatura en el CampusVirtual, el fichero Python que contiene el código fuente de la práctica y el informe con el estudio de complejidad.
- Revise detenidamente el documento PlantillaPracticas para saber qué se va a evaluar.
- Es suficiente con que uno de los integrantes del equipo de práctica deposite el contenido en el CampusVirtual. El nombre de **todos** los integrantes debe figurar en el encabezado de ambos documentos a entregar.
- Es obligatorio realizar una defensa de la práctica en la clase de prácticas posterior a la entrega.
- Todos los aspectos relacionados con dispensa académica, dedicación al estudio, permanencia y fraude académico se regirán de acuerdo con la normativa académica vigente de la UDC. Si las pruebas o actividades de evaluación se llevan a cabo en grupos, todos los miembros del grupo serán responsables solidariamente por el trabajo realizado y entregado, así como de sus posibles consecuencias.