UNIVERSIDAD TORCUATO DI TELLA

Métodos Analíticos y Programación
Parcial - 10 de mayo de 2024

Nombre y apellido:
Legajo:

El examen se aprueba con 5 puntos (sobre 10). Justifique todas sus respuestas. En caso de ser necesario, el examen podrá ser complementado con un coloquio a criterio del docente.

---

1. **(4.00 puntos)** El juego del 'mago goma' es un gran juego para jugar entre dos personas mientras se está de viaje, en la playa o mientras se espera que corra el TP de MAP.

   Consiste en lo siguiente: una persona empieza con una palabra, por ejemplo 'gota' y la otra persona contesta con una palabra cuya primera sílaba sea la última sílaba de la palabra anterior. En este caso 'ta'. Podría responder por ejemplo 'taco'. Observar que no podría responder 'talco' porque, por más que empieza con 'ta', su primera sílaba es 'tal'.

   Esto se repite de manera veloz hasta que una persona se equivoca o no puede seguir.

   Vamos a armar un código que pueda, a gran velocidad, identificar si una secuencia es válida o no.

   a) **(1.00 puntos)** Lo primero que haremos es una función que chequee si dos palabras son válidas para aparecer consecutivamente en el juego. Para eso vamos a definir la función

   ```
   def son_mago_goma(p1, p2, silabas):
   ```

   que toma los palabras `p1` y `p2` en formato de *string* y el diccionario `silabas` y retorna `True` si son compatibles y `False` si no.
   El diccionario `silabas` tiene como claves todas las palabras del idioma español y como significados una tupla con las sílabas de la palabra.

   ```
   silabas = {"mago": ("ma", "go"),
              "goma": ("go", "ma"),
              "persiana": ("per", "sia", "na"),
              ...}
   ```

   Podemos asumir que todas las palabras que recibimos están en este diccionario.

   b) **(1.50 puntos)** Implementar la función

   ```
   def posiciones_invalidas(l, silabas):
   ```

   que dada una lista `l` con una secuencia de palabras y el mismo diccionario antes nombrado, retorna una lista con las posiciones donde alguien dijo una palabra inválida.
   A modo de ejemplo, para la lista

   ```
   ["mago", "goma", "carne", "neto"]
   ```

   debería retornar `[2]`. Para la lista

   ```
   ["hola", "lave", "veces", "cesped"]
   ```

   debería retornar `[]` y para

```
["queso", "sordo", "donde", "delta"]
```

debería retornar `[1, 2, 3]`. Observar que la posición 0 nunca puede estar mal ya que ¡es la primera palabra!

c) **(1.00 puntos)** Por último, falta una regla más, y es que un jugador no puede repetir inmediatamente la palabra que acaba de decir. Por ejemplo, si yo digo 'mago' y mi contrincante dice 'goma', no puedo volver a decir 'mago'.

Para eso queremos implementar la función

```
def posiciones_repeticiones(l):
```

Que recibe una lista de palabras y retorna una lista de las posiciones donde un jugador repitió su última palabra. Observar que aunque la lista

```
["mago", "goma", "mago", "goma"]
```

va a retornar `[2,3]`, la lista

```
["ropero", "ropero", "ropa", "paro", "ropero"]
```

va a retornar `[]`.

d) **(0.50 puntos)** Por último queremos implementar la función

```
def es_valida(l, silabas):
```

que retorna `True` si cumple las dos reglas anteriores y `False` si no cumple alguna.

2. **(3.00 puntos)** Dados dos diccionarios `d1`, `d2` cuyos tipos de claves y significados coincide, buscamos calcular un nuevo diccionario a partir de `d1` y `d2`. A este diccionario `dm` lo llamaremos *diccionario minimal*, y cumple las siguientes reglas:

   - las claves definidas en `dm` deben pertenecer tanto a `d1` como `d2`; y
   - si `k` es una clave válida para `dm`, entonces su significado debe ser el de menor valor entre su significado en `d1` y `d2`.

   Consideremos como ejemplo los siguientes diccionarios, donde las claves son de tipo `string` y los significados de tipo `int`:

```
d1 = {'uno' : 1, 'dos' : 3, 'tres' : 3}
d2 = {'cuatro': 4, 'uno' : 1, 'dos' : 2}
d3 = {'cuatro': 4, 'cinco' : 2}
```

   El diccionario minimal `dm` al considerar `d1` y `d2` es

```
dm = {'uno' : 1, 'dos' : 2}
```

   Notar que tanto `'uno'` como `'dos'` están definidos en ambos diccionarios, y que `dm['dos']` es el mínimo entre `d1['dos']` y `d2['dos']`. Adicionalmente, notar que el diccionario minimal entre `d1` y `d3` es el diccionario vacío. Se pide implementar la función

```
def diccionario_minimal(d1, d2):
```

   que dados dos diccionarios cualesquiera `d1` y `d2`, retorna un diccionario con los lineamientos explicados anteriormente. Los parámetros `d1` y `d2` **no deben modificarse**. La totalidad del puntaje se asignará a aquellas resoluciones que modularicen el código convenientemente.
   Ayuda: dado que las claves del diccionario minimal están en ambos, es posible calcular la intersección recorriendo las claves de uno y validar pertenencia en el otro.

3. **(3.00 puntos)** Dada una lista de números `l`, un número `elem` y dos valores `a` y `b` que representan un rango de posiciones de la lista, mediante la función `mas_cercano_entre` buscamos obtener el valor del elemento de `l` que se encuentre en el rango de posiciones `[a,b]` más cercano a `elem`. Para ello, se utiliza la función `abs`, que retorna el *valor absoluto* (o módulo) de un número. A modo de ejemplo:

```
>>> abs(4)
4
>>> abs(-2)
2
>>> abs(0)
0
```

Para ilustrar el comportamiento esperando de la función:

```
# retorna 5. Busca el elemento mas cercano a 6 en la sublista [3,4,5]
mas_cercano_entre([2,3,4,5,4,6], 6, 1, 3)
# retorna 2. Busca el elemento mas cercano a -2 en la sublista [2,3,4]
mas_cercano_entre([2,3,4,5,4,6], -2, 0, 2)
```

Recibimos un archivo con la siguiente implementación, sin documentar. Asumimos que `a` y `b` son posiciones válidas y `a` $\leq$ `b`.

```
def mas_cercano_entre(l, elem, a, b):

    # iniciamos con el calculo del primer elemento
    ret = l[a]
    min_dist = abs(elem - l[a])

    # miramos del segundo en adelante
    for i in range(a+1,b):
        if min_dist > abs(elem - i):
            ret = l[i]
            min_dist = abs(ret - l[i])

    return ret
```

El código propuesto tiene al menos dos errores. **Sin utilizar los casos anteriores**, se pide:

a) **(1.50 puntos)** Indicar cuáles son los errores identificados. Proponer casos de test, indicando el valor correcto y el valor retornado por la función, donde:

- Un caso de test con una lista de al menos 3 elementos y la implementación dada retorne la respuesta correcta.
- Un caso de test específico para cada uno de los errores, donde la implementación dada retorne una respuesta incorrecta.

b) **(1.50 puntos)** Corregir los errores en la implementación del punto anterior.