Ejercicio – Sets

February 18, 2024

1 Ejercicio – Sets

Presenta: Juliho Castillo Colmenares

https://github.com/julihocc/ebac-python-backend

1.1 Instrucciones

Crea un Jupyter Notebook para el ejercicio y modifica este programa para crear un generador de combinaciones. En el ejemplo se generan las combinaciones de 3 caracteres. El objetivo es que incluyas la letra D y generes las combinaciones posibles con 4 caractéres

[ ]:
```
from collections import defaultdict

numbers = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
letters = ['A', 'B', 'C']
```

[ ]:
```
# Calculo de la cantidad teórica de combinaciones con todos los caracteres␣
 ↪posibles

from math import comb

n = len(numbers) + len(letters)
k = 3
comb(n, k)
```

[ ]:
```
286
```

[ ]:
```
unique_combinations = defaultdict(set)

for number in numbers:
    for letter in letters:
        unique_combinations['code base'].add("{}{}{}".
 ↪format(number,number,number))
        unique_combinations['code base'].add("{}{}{}".
 ↪format(number,number,letter))
        unique_combinations['code base'].add("{}{}{}".
 ↪format(number,letter,number))
```

```
        unique_combinations['code base'].add("{}{}{}".
  ↪format(number,letter,letter))
        unique_combinations['code base'].add("{}{}{}".
  ↪format(letter,number,number))
        unique_combinations['code base'].add("{}{}{}".
  ↪format(letter,letter,number))
        unique_combinations['code base'].add("{}{}{}".
  ↪format(letter,number,letter))
        unique_combinations['code base'].add("{}{}{}".
  ↪format(letter,letter,letter))
print(len(unique_combinations['code base']))
```

193

Sin embargo, en la solución base no se calculan todas las combinaciones posibles.

1.1.1 Solución

Propondremos una solución que sea comparable con la proporcionada en el código base. Posteriormente añadiremos la letra 'D', y se generarán las combinaciones de 4 caracteres.

[ ]:
```python
from itertools import combinations


def generate_combinations(arr: list[str], r:int) -> list[list[str]]:
    result = []

    def backtrack(start, comb):
        # When the combination is complete
        if len(comb) == r:
            result.append(comb.copy())
            return

        # Add each element into the combination and recurse
        for i in range(start, len(arr)):
            # Add the current element to the combination
            comb.append(arr[i])
            # Recurse with the next element
            backtrack(i + 1, comb)
            # Backtrack, remove the last element added
            comb.pop()

    backtrack(0, [])
    return result

def generate_words(characters:list[str], length:int):
    words = generate_combinations(characters, length)
    return set([''.join(combo) for combo in words])
```

```
# Example usage
items = ['A', 'B', 'C']  # Define your set of items
r = 2  # Number of items to choose

# Generate and print all combinations
all_combinations = generate_words(items, r)
for combo in all_combinations:
    print(combo)
```

```
AB
BC
AC
```

[ ]:
```
#  Generamos las combinaciones con los caracteres dados en el enunciado

numbers = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
letters = ['A', 'B', 'C']
characters = numbers + letters

unique_combinations['modified code'] = generate_words(characters, 3 )

print(len(unique_combinations['modified code']))
```

```
286
```

[ ]:
```
unique_combinations['code base'] == unique_combinations['modified code']
```

[ ]:
```
False
```

[ ]:
```
len(unique_combinations['code base'] - unique_combinations['modified code'])
```

[ ]:
```
193
```

[ ]:
```
len(unique_combinations['modified code'] - unique_combinations['code base'])
```

[ ]:
```
286
```

Como se puede obsevar, existen combinaciones en la solución propuesta que no existen en la solución proporcionado en las instrucciones. Pero todas las soluciones proporcionadas sí están en la solución propuesta.

[ ]:
```
# Verificar si una combinación específica está en un conjunto o en el otro
example =  (unique_combinations['modified code'] - unique_combinations['code␣
 ↪base']).pop()
print(example)
print(example in unique_combinations['code base'])
print(example in unique_combinations['modified code'])
```

```
03C
False
True
```

[ ]:
```python
# Cálculo teórico de la cantidad de combinaciones posibles

numbers = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
letters = ['A', 'B', 'C', 'D']

n = len(numbers) + len(letters)
k = 3

comb(n, k)
```

[ ]:
```
364
```

[ ]:
```python
#solución

numbers = ['0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
letters = ['A', 'B', 'C', 'D']
characters = numbers + letters

unique_combinations['solution'] = generate_combinations(characters, 3)

print(len(unique_combinations['solution']))
```

```
364
```