
BUAP

**FACULTAD DE CIENCIAS DE LA COMPUTACIÓN**

**Materia**:

Minería de datos

**Alumno:**

Mendoza Reyes Jose de Jesus

**Matricula:**

202263899

**Fecha de entrega:**

29/Enero/2025

**Tarea:**

Practica 1: Propiedades de conjuntos y operadores

# *Descripción*

En esta práctica se  trabajó con los operadores y propiedades que hay en la teoría de Conjuntos. En clase, repasamos la definición y ejemplos de conjuntos con propiedades que tienen respecto a ciertos operadores. Los operadores vistos fueron la Union (), la Intersección (), la diferencia (-), y el Complemento de un conjunto A ().

Las propiedades vistas en clase fueron:

- Asociatividad de la unión
- Asociatividad de la intersección
- Distributivita de la intersección respecto de la unión
- Distributivita de la unión respecto a la intersección
- Distributivita del complemento en la unión
- Complemento de la unión
- Complemento de la intersección

Para poder probar que estas propiedades son ciertas y cómo funcionan los operandos se hizo la practica 1 en Python, simulando los operandos con ciclos a conjuntos de tamaños definidos en clase (10, 200 mil, 100 mil, 1 millón).  Después de esto se verifica que las operaciones fueron hechas correctamente usando funciones para conjuntos integradas en Python y finalmente demostramos con los mismos conjuntos que las 6 propiedades vistas en clase sean ciertas.

# *Codigo*

El codigo hecho en la práctica es el siguiente:

```python
import random

def generar_numeros(tipo):

    tamanos = {
        '10': 10,
        '200k': 200000,
        '100k': 100000,
        'millon': 1000000
    }

    if tipo in tamanos:
        return random.sample(range(1, 1000001), tamanos[tipo])
    else:
        raise ValueError("Tipo no valido")
```

Primero tenemos una función llamada **generar_numeros** que nos generan de manera aleatoria listas rellenas de números aleatorios en un rango de 1 a 1 millón con un tamaño dado por un diccionario llamado ***tamaños***. La función **.sample** garantiza que los elementos generados no estén repetidos, algo que es necesario si queremos trabajar con conjuntos (ya que en un conjunto no se pueden repetir elementos). Retornamos finalmente la lista y si no se eligió una opción disponible en el diccionario entonces lanzamos un error en tiempo de ejecución debido a que el tipo no es válido.

```python
def imprimir_conjunto(conjunto, limite=10):
    lista_conjunto = list(conjunto)
    if len(lista_conjunto) > limite:
        elementos = ", ".join(map(str, lista_conjunto[:limite]))
        return f"{{{elementos}, ...}}"
    else:
        return f"{{{', '.join(map(str, lista_conjunto))}}}"
```

Tenemos una función llamada **imprimir_conjunto** y nos sirve si es que la impresión del conjunto es muy grande (por ejemplo, si

queremos imprimir los conjuntos salientes de operaciones como Unión, intersección, etc.). Esto nos ayuda para una mejor visualización en los datos a la hora de la experimentación.

```python
def opciones(conjunto_A, conjunto_B, conjunto_C):

    def union(A, B):
        union = set(A)
        for elemento in B:
            if elemento not in union:
                union.add(elemento)
        return union

    def interseccion(A, B):
        interseccion = set()
        for elemento_x in A:
            if elemento_x in B:
                interseccion.add(elemento_x)
        return interseccion

    def diferencia(A, B):
        diferencia = set()
        for elemento in A:
            if elemento not in B:
                diferencia.add(elemento)
        return diferencia
```

En la función **opciones** tenemos todas las 3 implementaciones requeridas en clase de los operandos de unión, intersección y diferencia.

En la función **unión** se implementa el operando unión apartir de dos conjuntos dados $A, B$. Primero, debido a que $A$ $y$ $B$ son listas (por la función **generar_numeros**) se hace un cast de lista a set (conjunto) de la lista $A$. Después iteramos en los elementos de $B$ y verificamos que si dado elemento $x$ en $B$ si $x$ no está en la unión (osease en $A$) entonces agregamos dicho elemento a la unión.

En la función **intersección** creamos un conjunto vacío que contendrá los elementos en común de $A$ $y$ $B$ (si es que los hay). Iteramos en los elementos de $A$ y dado $x$ elemento en $A$ si $x$ tambien esta en $B$ entonces agregamos el elemento $x$ a la intersección.

En la función **diferencia** creamos un conjunto vacío que contendrá los elementos de la diferencia de $A$ y B e iteramos en los elementos de $A$. Verificamos que dado elemento $x$ en $A$ si el elemento no está en $B$ entonces lo agregamos a la diferencia.

```
#union
union_result = union(conjunto_A, conjunto_B)
union_operadores = set(conjunto_A) | set(conjunto_B)

#interseccion
interseccion_result = interseccion(conjunto_A, conjunto_B)
interseccion_operadores = set(conjunto_A) & set(conjunto_B)

#diferencia
diferencia_result = diferencia(conjunto_A, conjunto_B)
diferencia_operadores = set(conjunto_A) - set(conjunto_B)

#resultados
print(f"Union: {imprimir_conjunto(union_result)}")
print(f"Union con operadores: {imprimir_conjunto(union_operadores)}")
print(f"Verificacion: {union_result == union_operadores}")

print(f"Interseccion: {imprimir_conjunto(interseccion_result)}")
print(f"Interseccion con operadores: {imprimir_conjunto(interseccion_operadores)}")
print(f"Verificacion: {interseccion_result == interseccion_operadores}")

print(f"Diferencia: {imprimir_conjunto(diferencia_result)}")
print(f"Diferencia con operadores: {imprimir_conjunto(diferencia_operadores)}")
print(f"Verificacion: {diferencia_result == diferencia_operadores}")
```

Llamamos a estas funciones dentro de la misma función **opciones** y las guardamos en variables para su posterior visualización. Tambien hacemos las mismas operaciones pero con funciones (en este caso operandos) implementados en el mismo Python. Imprimimos los resultados y verificamos que tanto las funciones creadas y las funciones de Python sean correctas.

```
# propiedades:
A = set(conjunto_A)
B = set(conjunto_B)
C = set(conjunto_C)

#propiedad 1: asociatividad de la union
union1 = union(A, union(B,C))
print(f"A u (B u C) = {imprimir_conjunto(union1)}")
union2 = union(union(A,B), C)
print(f"(A u B) u C = {imprimir_conjunto(union2)}")
print(f"Validacion de prop 1: {union1 == union2}")

#propiedad 2: asociatividad de la interseccion
inters1 = interseccion(A, interseccion(B,C))
print(f"A & (B & C) = {imprimir_conjunto(inters1)}")
inters2 = interseccion(interseccion(A,B), C)
print(f"(A & B) & C = {imprimir_conjunto(inters2)}")
print(f"Validacion de prop 2: {inters1 == inters2}")

#propiedad 3: distributividad de la interseccion respecto de la union
inters_distri_1 = interseccion(A, union(B, C))
print(f"A & (B u C) = {imprimir_conjunto(inters_distri_1)}")
inters_distri_2 = union(interseccion(A, B), interseccion(A, C))
print(f"(A & B) u (A & C) = {imprimir_conjunto(inters_distri_2)}")
print(f"Validacion de prop 3: {inters_distri_1 == inters_distri_2}")

#propiedad 4: distributividad de la union respecto de la interseccion
union_distri_1 = union(A, interseccion(B, C))
print(f"A u (B & C) = {imprimir_conjunto(union_distri_1)}")
union_distri_2 = interseccion(union(A, B), union(A, C))
print(f"(A u B) & (A u C) = {imprimir_conjunto(union_distri_2)}")
print(f"Validacion de prop 4: {union_distri_1 == union_distri_2}")
```

Despues de las validaciones de operaciones imprimimos las validaciones de las 6 propiedades de los conjuntos en pantalla. Para esto llamamos a nuestras funciones creadas respetando asociatividad y distribución. Después de esto verificamos que la igualdad de las propiedades se cumpla.

```python
#suponemos que el universo es la union de los 3 conjuntos
universo = A | B | C
complemento_A = universo - A
complemento_B = universo - B
complemento_C = universo - C

#propiedad 5
complemento_union_1 = universo - union(A, B) #tambien se puede ver de esta forma (A u B)^c
print(f"(A u B)^c = {imprimir_conjunto(complemento_union_1)}")
complemento_union_2 = interseccion(complemento_A, complemento_B)
print(f"A^c & B^c = {imprimir_conjunto(complemento_union_2)}")
print(f"Validacion de prop 5: {complemento_union_1 == complemento_union_2}")

#propiedad 6
complemento_interseccion_1 = universo - interseccion(A, B)
print(f"(A & B)^c = {imprimir_conjunto(complemento_interseccion_1)}")
complemento_interseccion_2 = union(complemento_A, complemento_B)
print(f"A^c u B^c = {imprimir_conjunto(complemento_interseccion_2)}")
print(f"Validacion de prop 6: {complemento_interseccion_1 == complemento_interseccion_2}")
```

En el caso de las propiedades del complemento suponemos que nuestro universo $U$ es igual a la unión de los conjuntos $A, B$ $y$ $C$. Después de esto, hacemos las operaciones de conjuntos sacando los complementos de A, B y C y verificando que las igualdades se cumplan.

```python
def menu():
    conjunto_a = []
    conjunto_b = []
    conjunto_c = []
    opt = 0

    while opt != 1:

        # tam conjunto A
        print("Dame el tamaño de la lista A (10, 200k, 100k, millon): ")
        conjunto_a = generar_numeros(input())

        # tam conjunto B
        print("Dame el de la lista B: ")
        conjunto_b = generar_numeros(input())

        print("Dame el de la lista C: ")
        conjunto_c = generar_numeros(input())

        # resultado
        print(f"Conjunto A = {imprimir_conjunto(conjunto_a)}")
        print(f"Conjunto B = {imprimir_conjunto(conjunto_b)}")
        print(f"Conjunto C = {imprimir_conjunto(conjunto_c)}")

        print("Resultados: ")
        opciones(conjunto_a, conjunto_b, conjunto_c)

        print("Quieres salir? Escribe 1 si es asi")
        opt =  int(input())
menu()
```

Finalmente tenemos la función principal **menu** que llama a todas las demas y que funciona para rellenar los conjuntos $A, B\ y\ C$. Si es que se quiere salir del programa se le pide al usuario escribir 1 u otra cosa si quiere continuar con otros ejemplos.

# *Experimentos/Resultados*

Para poder saber si nuestras implementaciones fueron las correctas se experimentó con 3 casos distintos y se dieron los siguientes resultados:

*Experimento 1, tamaño 10 en los 3 conjuntos:*

```
Dame el tamaño de la lista A (10, 200k, 100k, millon):
10
Dame el de la lista B:
10
Dame el de la lista C:
10
Conjunto A = {279243, 951677, 17361, 532523, 674876, 385358, 29893, 758971, 332002, 510559}
Conjunto B = {809019, 387530, 51881, 594300, 73534, 575388, 992033, 452136, 864783, 282481}
Conjunto C = {54358, 955133, 702242, 430711, 865919, 650199, 773826, 433149, 483602, 296028}
Resultados:
Union: {864783, 575388, 992033, 452136, 51881, 532523, 809019, 758971, 674876, 73534, ...}
Union con operadores: {29893, 387530, 279243, 385358, 864783, 17361, 575388, 510559, 992033, 332002, ...}
Verificacion: True
Interseccion: {}
Interseccion con operadores: {}
Verificacion: True
Diferencia: {332002, 29893, 279243, 532523, 385358, 17361, 758971, 674876, 951677, 510559}
Diferencia con operadores: {332002, 29893, 532523, 279243, 385358, 17361, 758971, 674876, 951677, 510559}
Verificacion: True
A u (B u C) = {955133, 864783, 483602, 575388, 992033, 702242, 452136, 51881, 532523, 809019, ...}
(A u B) u C = {773826, 29893, 955133, 387530, 279243, 758971, 385358, 864783, 17361, 433149, ...}
Validacion de prop 1: True
A & (B & C) = {}
(A & B) & C = {}
Validacion de prop 2: True
A & (B u C) = {}
(A & B) u (A & C) = {}
Validacion de prop 3: True
A u (B & C) = {332002, 29893, 279243, 532523, 385358, 17361, 758971, 674876, 951677, 510559}
(A u B) & (A u C) = {332002, 29893, 532523, 279243, 385358, 17361, 758971, 674876, 951677, 510559}
Validacion de prop 4: True
(A u B)^c = {773826, 702242, 430711, 955133, 483602, 54358, 650199, 296028, 433149, 865919}
A^c & B^c = {773826, 702242, 430711, 955133, 483602, 54358, 650199, 296028, 433149, 865919}
Validacion de prop 5: True
(A & B)^c = {773826, 29893, 955133, 387530, 279243, 385358, 864783, 17361, 433149, 483602, ...}
A^c u B^c = {773826, 29893, 955133, 387530, 279243, 758971, 674876, 864783, 385358, 17361, ...}
Validacion de prop 6: True
Quieres salir? Escribe 1 si es asi
1
PS C:\Users\USER>
```

Le damos como entrada al programa que los conjuntos $A, B\ y\ C$ son todos de tamaño 10. Se imprimen los conjuntos y se imprimen las operaciones de unión, intersección y diferencia tanto las creadas por nosotros como las de Python. Nótese que a pesar de que la verificación da **true** en ciertos casos los conjuntos tienen diferente

orden como en el caso de la Unión. Esto ocurre debido a la implementación tanto hecha por nosotros y la de Python, y debido a que se está trabajando con conjuntos **su orden no importa**, por lo cual **ambos conjuntos se consideran iguales**.

*Experimento 2, tamaño 10 en los 2 conjuntos, tamaño 100 mil en el último:*

```
Dame el tamaño de la lista A (10, 200k, 100k, millon):
10
Dame el de la lista B:
10
Dame el de la lista C:
100k
Conjunto A = {259091, 721673, 49199, 315419, 619664, 596288, 204021, 691695, 593529, 478960}
Conjunto B = {339554, 390086, 544869, 299777, 841083, 802828, 297573, 244717, 470453, 632324}
Conjunto C = {61528, 704435, 27186, 500989, 58518, 269346, 299467, 420533, 826027, 758659, ...}
Resultados:
Union: {299777, 632324, 721673, 802828, 619664, 259091, 315419, 49199, 470453, 596288, ...}
Union con operadores: {596288, 299777, 632324, 390086, 721673, 802828, 619664, 259091, 315419, 339554, ...}
Verificacion: True
Interseccion: {}
Interseccion con operadores: {}
Verificacion: True
Diferencia: {596288, 721673, 49199, 619664, 691695, 478960, 259091, 204021, 593529, 315419}
Diferencia con operadores: {596288, 721673, 691695, 619664, 49199, 478960, 259091, 204021, 593529, 315419}
Verificacion: True
A u (B u C) = {786432, 262144, 262145, 262147, 11, 524302, 14, 786448, 17, 524307, ...}
(A u B) u C = {786432, 262144, 262145, 262147, 11, 524302, 14, 786448, 17, 524307, ...}
Validacion de prop 1: True
A & (B & C) = {}
(A & B) & C = {}
Validacion de prop 2: True
A & (B u C) = {721673, 315419}
(A & B) u (A & C) = {721673, 315419}
Validacion de prop 3: True
A u (B & C) = {596288, 299777, 721673, 49199, 619664, 691695, 478960, 259091, 204021, 593529, ...}
(A u B) & (A u C) = {596288, 299777, 721673, 49199, 619664, 691695, 478960, 259091, 204021, 593529, ...}
Validacion de prop 4: True
(A u B)^c = {786432, 262144, 262145, 262147, 11, 524302, 14, 786448, 17, 524307, ...}
A^c & B^c = {786432, 262144, 262145, 262147, 11, 524302, 14, 786448, 17, 524307, ...}
Validacion de prop 5: True
(A & B)^c = {786432, 262144, 262145, 262147, 11, 524302, 14, 786448, 17, 524307, ...}
A^c u B^c = {786432, 262144, 262145, 262147, 11, 524302, 14, 786448, 17, 524307, ...}
Validacion de prop 6: True
Quieres salir? Escribe 1 si es asi
1
PS C:\Users\USER>
```

Debido a que el último conjunto es muy largo solo imprimimos sus primeros números. Todas las operaciones fueron exitosas y la igualdad resultó en true en los operandos y en las propiedades.

*Experimento 3, tamaño 10k, 100k y 200k:*

```
Dame el tamaño de la lista A (10, 200k, 100k, millon):
10
Dame el de la lista B:
100k
Dame el de la lista C:
200k
Conjunto A = {4472, 742888, 431309, 312795, 214090, 678744, 160309, 736581, 21503, 842978}
Conjunto B = {250013, 300918, 149670, 53307, 842672, 34756, 925372, 798592, 118188, 20082, ...}
Conjunto C = {334646, 639829, 111334, 217055, 803134, 371922, 290697, 804285, 586463, 77950, ...}
Resultados:
Union: {524290, 2, 524292, 4, 12, 786446, 14, 17, 262164, 786459, ...}
Union con operadores: {524290, 2, 524292, 4, 12, 786446, 14, 17, 262164, 786459, ...}
Verificacion: True
Interseccion: {21503}
Interseccion con operadores: {21503}
Verificacion: True
Diferencia: {842978, 736581, 742888, 214090, 431309, 678744, 160309, 4472, 312795}
Diferencia con operadores: {842978, 4472, 736581, 742888, 214090, 431309, 160309, 678744, 312795}
Verificacion: True
A u (B u C) = {524290, 2, 524292, 4, 5, 524297, 524298, 12, 524300, 14, ...}
(A u B) u C = {524290, 2, 524292, 4, 5, 524297, 524298, 12, 524300, 14, ...}
Validacion de prop 1: True
A & (B & C) = {21503}
(A & B) & C = {21503}
Validacion de prop 2: True
A & (B u C) = {4472, 21503}
(A & B) u (A & C) = {4472, 21503}
Validacion de prop 3: True
A u (B & C) = {4, 655374, 655375, 393244, 262179, 655444, 524384, 786533, 102, 105, ...}
(A u B) & (A u C) = {4, 655374, 655375, 393244, 262179, 655444, 524384, 786533, 102, 105, ...}
Validacion de prop 4: True
(A u B)^c = {5, 524297, 524298, 524300, 13, 16, 524304, 524305, 524306, 20, ...}
A^c & B^c = {5, 524297, 524298, 524300, 13, 16, 524304, 524305, 524306, 20, ...}
Validacion de prop 5: True
(A & B)^c = {2, 4, 5, 12, 13, 14, 16, 17, 20, 37, ...}
A^c u B^c = {2, 4, 5, 12, 13, 14, 16, 17, 20, 37, ...}
Validacion de prop 6: True
Quieres salir? Escribe 1 si es asi
1
```

Todas las operaciones resultaron exitosas tambien en este experimento. Sin embargo, nótese que **debido al espacio muestral** (1 a 1 millón) **generar 2 conjuntos aleatorios que compartan un elemento en común es muy dificil** y este es el único ejemplo en el que fue posible encontrar esos 2 conjuntos.

# *Conclusión*

La teoría de conjuntos, sus operandos y sus propiedades son parte fundamental de la matemática y por ende de gran parte de la computación. Implementar estos operandos nos hace ver cómo funcionan y como es que estas propiedades siempre serán verdaderas a pesar de los conjuntos dados. Con esto aprendimos a identificar completamente los operandos más conocidos de la teoría de conjuntos y lo que conllevan.