Databases Exercise Sheet 2

Exercise 1 (Operationen auf Mengen)

Gegeben sind die Mengen

```
Starters    = {salad, soup},
Main        = {pizza, steak},
Dessert     = {fruit, icecream},

Relationen
Father      = {(adam, bert), (bert, carl)}
Car         = {(adam, audi), (bert, bmw), (carl, chevy)}.
```

Berechnen Sie:

```
1.Starters×Main×Dessert
```

solution

```
(salad, pizza, fruit),(salad, pizza, icecream),
(salad, steak, fruit),(salad, steak, icecream),

(soup, pizza, fruit),(soup, pizza, icecream),
(soup, steak, fruit),(soup, steak, icecream),
```

```
2.Father.Father
```

```
{(adam, bert), (bert, carl), (adam, bert), (bert, carl)}
```

```
3.Father.Car
```

```
{(adam, bert), (bert, carl), (adam, audi), (bert, bmw), (carl, chevy)}
```

```
4.Car.Father
```

```
{(adam, audi), (bert, bmw), (carl, chevy), (adam, bert), (bert, carl)}
```

Exercise 2 (Aussagen ̈uber Mengen)

Welche der folgenden Aussagen sind wahr? Begr ̈unden Sie jeweils, warum. Wenn eine Aussage falsch ist, dann zeigen Sie das mit einem Gegenbeispiel. Die MengenA, B, Cseien Teilmengen einer beliebigen endlichen GrundmengeU.

1. Wenn A ∪ B = A ∪ C dann B = C.

solution

True

2. Es gilt A ⊆ B genau dann wenn ̄B ⊆ ̄A.

solution

false

3. A × B = B × A

solution

true

4. |A × B|= |A|·|B|

solution

true

Exercise 3 (Mengen als Listen)

Mengen werden h ̈aufig mit Hilfe von Listen implementiert. Geben Sie in einer Program- miersprache ihrer Wahl oder in Pseudocode eine Funktionboolean equals(list l1,list l2)an, die genau danntrueliefert, wenn die

Mengen der Elemente der beiden Listen gle- ich sind. Gegeben sind die Funktionenint head(list l)undlist tail(list l), die f¨ur nichtleere Listen jeweils das erste Element und die Restliste liefern, sowie die Funktionboolean empty(list l). Geben Sie m¨oglichst einfachen, kurzen und offensichtlich korrekten Code an, die Laufzeit spielt keine Rolle. Definieren Sie sich geeignete Hilfsfunktionen.

solution

```
def remove_duplicates(a_1):
    return list(dict.fromkeys(a_1))

def equals(a_1, a_2):
    a_1 = remove_duplicates(a_1)
    a_2 = remove_duplicates(a_2)

    b_equal = True
    for n in a_1:
        if n not in a_2:
            b_equal = False
            break

    if len(a_1) != len(a_2):
        b_equal = False

    return b_equal

a_1 = [1,3,5,6,5]
a_2 = [5,3,1,6,5]
print(f"{a_1} equals {a_2} : {equals(a_1, a_2)}")

a_1 = [3,3,3,1]
a_2 = [5,3,1,6,5]
print(f"{a_1} equals {a_2} : {equals(a_1, a_2)}")

a_1 = [2,3,4]
a_2 = [3,4,4]
print(f"{a_1} equals {a_2} : {equals(a_1, a_2)}")

a_1 = [4,4,2,2,11]
a_2 = [2,4,2,4,11]
print(f"{a_1} equals {a_2} : {equals(a_1, a_2)}")
```

console output

```
[1, 3, 5, 6, 5] equals [5, 3, 1, 6, 5] : True
[3, 3, 3, 1] equals [5, 3, 1, 6, 5] : False
[2, 3, 4] equals [3, 4, 4] : False
[4, 4, 2, 2, 11] equals [2, 4, 2, 4, 11] : True
```

Exercise 4 (Mengen von Nat¨urlichen Zahlen als Boolean Arrays)

Eine endliche MengeSvon nat¨urlichen Zahlen kann man als Arrayboolean[] srepr¨asen- tieren, wie folgt: so dass gilt:

```
s[n] = true fallsn∈S
s[n] = false fallsn /∈S.
```

Die MengeS={ 1 , 3 , 4 }ist dann durch folgendes Array repr¨asentiert:

```
s[0] = false
s[1] = true
s[2] = false
s[3] = true
s[4] = true.
```

Geben Sie m¨oglichst einfache Algorithmen in Pseudocode an f¨ur folgende Funktionen:

```
1.boolean subset(boolean[] s, boolean[] t), die genau danntrueliefert, wennS
eine Teilmenge vonTist
2.boolean[] union(boolean[] s, boolean[] t), die die (Repr ¨asentation der) Mengen-
vereinigung vonSundTliefert.
```

solution

```
def union(a_booleans_s, a_booleans_t):
    a_booleans_bigger = a_booleans_t
    a_booleans_smaller = a_booleans_s
    if len(a_booleans_s) > len(a_booleans_t):
        a_booleans_bigger = a_booleans_s
        a_booleans_smaller = a_booleans_t

    for index, b in enumerate(a_booleans_bigger):
        if b == False:
            if len(a_booleans_smaller)-1 >= index:
                a_booleans_bigger[index] = a_booleans_smaller[index]

    return a_booleans_bigger


def subset(a_booleans_s, a_booleans_t):
    # return true if a_booleans_s is a subset of a_booleans_t
```

```
    b_is_subset = True
    b_first_value = False
    for index, b in enumerate(a_booleans_s):
        if(b):
            b_first_value = True
        if(b_first_value):
            if(a_booleans_t[index] != b):
                b_is_subset = False
                break

    return b_is_subset


a_b_1 = [False, False, False, False, True, False, False, True] # {4,7}
a_b_2 = [False, True, False, True, True, False, False, True] # {1,3,4,7}
print(f"{a_b_1} subset {a_b_2} :\n {subset(a_b_1, a_b_2)}")



a_b_1 = [False, False, True, True, False, True] # {2, 3, 5}
a_b_2 = [False, True, True, False, False, False, True] # {1,2,6}
a_unioned = union(a_b_1, a_b_2) # [False, True, True, True, False, True, True] ->
{1,2,3,5,6}

print(f"{a_b_1} union {a_b_2} :\n {union(a_b_1, a_b_2)}")
```

console output

```
[False, False, False, False, True, False, False, True] subset [False, True, False,
True, True, False, False, True] :
 True
[False, False, True, True, False, True] union [False, True, True, True, False,
True, True] :
 [False, True, True, True, False, True, True]
```

Exercise 5 (Relationen als Boolean Arrays)

Eine endliche bin¨are RelationRauf den nat¨urlichen Zahlen sei als ein zweidimensionales Array boolean[][] rrepr¨asentiert, so dass gilt:

```
r[m][n] = true falls (m, n)∈R
r[m][n] = false falls (m, n)∈/R.
```

Geben Sie einen m¨oglichst einfachen Algorithmus in Pseudocode an f¨ur die Funktion boolean[][] compose(boolean[][] r, boolean[][] t), die die (Repr¨asentation der) Komposition der RelationenRundTliefert.

solution

```
from re import L


def compose(a_2d_booleans_r, a_2d_booleans_t):
    #a_2d_booleans_r after a_2d_booleans_t
    o_mapping_1 = {}
    for index_y, a_2d_booleans_t_x in enumerate(a_2d_booleans_t):
        for index_x, b in enumerate(a_2d_booleans_t_x):
            if(b):
                o_mapping_1[index_y] = index_x

    o_mapping_2 = {}
    for index_y, a_2d_booleans_r_x in enumerate(a_2d_booleans_r):
        for index_x, b in enumerate(a_2d_booleans_r_x):
            if(b):
                o_mapping_2[index_y] = index_x


    o_mapping_final = {}

    for key in o_mapping_1:
        val = o_mapping_1[key]

        o_mapping_final[key] = o_mapping_2[int(val)]

    return o_mapping_final



a_2d_booleans_r = [
    [False, True, True, False],
    [True, True, False, False,],
    [False, False, False, True],
    [False, True, True, True]
]
a_2d_booleans_t = [
    [False, True, False, True],
    [True, True, False, False,],
    [False, True, False, False],
    [False, True, False, False]
]
o_mapping_final = compose(a_2d_booleans_r, a_2d_booleans_t)
print(
"""
a_2d_booleans_r = [
    [False, True, True, False],
    [True, True, False, False,],
    [False, False, False, True],
    [False, True, True, True]
]
```

```
composed with
a_2d_booleans_t = [
    [False, True, False, True],
    [True, True, False, False,],
    [False, True, False, False],
    [False, True, False, False]
]
= """
)
print(
    o_mapping_final
)
```

console output

```
a_2d_booleans_r = [
    [False, True, True, False],
    [True, True, False, False,],
    [False, False, False, True],
    [False, True, True, True]
]
composed with
a_2d_booleans_t = [
    [False, True, False, True],
    [True, True, False, False,],
    [False, True, False, False],
    [False, True, False, False]
]
=
{0: 3, 1: 1, 2: 1, 3: 1}
```

Exercise 6 (Anzahl Relationen zwischen Mengen)

Gegeben seien zwei MengenAundBmit|A|=mund|B|=n. Wieviele Relationen zwischen AundBgibt es? Betrachten Sie Beispiele. Begr¨unden Sie Ihre Antwort.

solution

da jedes element aus menge A mit jedem element aus menge B kombiniert wird, gibt es |A| * |B| anzahl elemente, es wie ein bild/2d matrix mit width und height anzahl pixel ist dann width * height. Kommt eine dritte dimension dazu, wird es x * y * z