Technische Universität Berlin **SoSe 2023**
Fakultät II, Institut für Mathematik
Sekretariat MA 6–2, Antje Schulz
Prof. Dr. Michael Joswig
Dr. Frank Lutz, Martin Knaack, Marcel Wack

4. Programmieraufgabe Computerorientierte Mathematik II

Abgabe: 26.5.2023 über den Comajudge bis 17 Uhr

Aufgabenstellung

In dieser Aufgabe soll eine Union-Find-Datenstruktur erstellt werden. Es sei $V \subset \mathbb{Z}^2_{\geq 0}$ eine als Liste von Tupeln $(x, y) \in \mathbb{Z}^2_{\geq 0}$ gegebene endliche Grundmenge.

Type `TupleSet`:

Schreiben Sie einen `Type Alias` für `Vector{Tuple{Int,Int}}` mit dem Namen `TupleSet`. Zusätzlich benötigen Sie einene Konstruktor `TupleSet(V:Vector{Tuple{Int,Int}})`. Der ein Objekt vom Typ `TupleSet` als lexikographisch geordneten Vektor erstellt.
Beispielaufruf:

```
>>> S = TupleSet([(0,3),(0,1),(1,3),(1,0)])
4-element Vector{Tuple{Int64, Int64}}:
 (0, 1)
 (0, 3)
 (1, 0)
 (1, 3)
```

Type `Partition`:

Schreiben Sie einen `Type Partition` mit dem Attribut

- `Sets::Vector{TupleSet}` Ein Vektor von `TupleSets`.

Zusätzlich brauchen sie folgende Funktionen

a) `Partition(V::TupleSet)::Partition` Erzeugt eine Partition von V in einelementige Mengen, die in der Liste `Sets` abgelegt werden.

c) `MakeSet(P::Partition, (x,y)::Tuple{Int,Int})` Fügt der Liste `Sets` ein `Set`-Objekt hinzu, das mit dem Tupel `(x,y)` initialisiert wird. Falls das Tupel `(x,y)` bereits in einem `TupleSet` von `Sets` enthalten ist, wird nichts hinzugefügt.

d) `FindSet(P::Partition, (x,y)::Tuple{Int,Int})` Es sei `S` das TupleSet, welches das Tupel `(x,y)` enthält. Dann gibt die Funktion das Repräsentanten-Tupel `S[1]` zurück. `S` soll hierzu lexikographisch geordnet sein. Falls das Tupel `(x,y)` in keinem `TupleSet` von `Sets` enthalten ist, geben sie `-1` zurück. Tipp: Sie können `sort` benutzen.

e) `union!(P::Partition, (x1,y1)::TupleInt,Int, (x2,y2)::TupleInt,Int)`

Es seien `S1` und `S2` Objekte in `Sets` , die `(x1,y1)` bzw. `(x2,y2)` enthalten. Dann entfernt `Union` die beiden Objekte `S1` und `S2` aus `Sets` und fügt statt ihrer ein neues Objekt `S` hinzu, das die lexikographisch geordnete Vereinigung von `S1` und `S2` ist. Den Fall, dass `(x1,y1)` und `(x2,y2)` in keinem Element von `Sets` enthalten ist, müssen sie nicht beachten.

Beispielaufrufe:

```
1>S = TupleSet([(0,3),(0,1),(1,3),(1,0)]);
2>P = Partition(S);

3>union!(P,(1,3),(0,1)).Sets
3-element Vector{Vector{Tuple{Int64, Int64}}}:
 [(0, 3)]
 [(1, 0)]
 [(0, 1), (1, 3)]
4>union!(P,(0,1),(0,3)).Sets
2-element Vector{Vector{Tuple{Int64, Int64}}}:
 [(1, 0)]
 [(0, 1), (0, 3), (1, 3)]
5>FindSet(P,(0,3))
(0, 1)
6>MakeSet(P,(300,1)).Sets
3-element Vector{Vector{Tuple{Int64, Int64}}}:
 [(1, 0)]
 [(0, 1), (0, 3), (1, 3)]
 [(300, 1)]
7>union!(P,(300,1),(0,1)).Sets
2-element Vector{Vector{Tuple{Int64, Int64}}}:
 [(1, 0)]
 [(0, 1), (0, 3), (1, 3), (300, 1)]
```

Info: In Julia können Typen einer Variable zugeordnet werden. Dies nennt man einen `Type Alias`.

```
#Beispiel
const MeineZahl = Union{Int,Nothing}
```