Technische Universität Berlin **SoSe 2022**
Fakultät II, Institut für Mathematik
Sekretariat MA 5–2, Dorothea Kiefer-Hoeft
Prof. Dr. Martin Skutella
Dr. Frank Lutz, Ekin Ergen, Martin Knaack, Manuel Radons

4. Programmieraufgabe Computerorientierte Mathematik II

Abgabe: 27.5.2022 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.

Klasse `Set`

Schreiben Sie eine Klasse `Set` mit dem Attribut

- `_elements` Liste V von Tupeln $(x, y) \in \mathbb{Z}^2_{\geq 0}$

und den Methoden

a) `__init__(self, V)` Initialisiert das Attribut `_elements` mit der übergebenen Liste V.

b) `Elements(self)` Gibt Inhalt von `_elements` als lexikographisch geordnete Liste zurück.

Beispielaufruf:

```
>>> S = Set([(0,3),(0,1),(1,3),(1,0)])
>>> print(S._elements)
[(0, 3), (0, 1), (1, 3), (1, 0)]
>>> S.Elements()
[(0, 1), (0, 3), (1, 0), (1, 3)]
```

Klasse `Partition`

Schreiben Sie eine Klasse `Partition` mit dem Attribut

- `Sets` (Liste von `Set`-Objekten)

und den Methoden

a) `__init__(self, V)` Erzeugt eine Partition von V in einelementige Mengen, die als `Set`-Objekte in der Liste `Sets` abgelegt werden. Falls ein Tupel doppelt in der Liste `V` vorkommt, werfen Sie die `Exception` 'invalid operation'.

b) `__str__(self)` String-Repräsentation des `Partition`-Objekts. Gibt eine Liste von Listen als String wieder. Die Teillisten sind die `_element`-Listen der `Set`-Objekte in `Sets`.
Anmerkung: Die Methode wird nicht vom Comajudge abgefragt, sollte aber vorgestellt werden können. Sie dient insbesondere Ihrer Selbstkontrolle. Dies heißt auch, dass die `Set`-Objekte in keiner besonderen Reihenfolge ausgedruckt werden müssen.

c) `MakeSet(self, (x,y))` Fügt der Liste `Sets` ein `Set`-Objekt hinzu, das mit dem Tupel `(x,y)` initialisiert wird. Falls das Tupel `(x,y)` bereits in einem `Set` von `Sets` enthalten ist, werfen Sie die `Exception` 'invalid operation'.

d) `FindSet(self, (x,y))` Es sei `S` das Set, welches das Tupel `(x,y)` enthält. Dann gibt die Methode das Repräsentanten-Tupel `S._elements[0]` zurück. Falls das Tupel `(x,y)` in keinem `Set` von `Sets` enthalten ist, werfen Sie die `Exception` 'invalid operation'.

e) `Union(self, (x1,y1), (x2,y2))` Es seien `S1` und `S2` `Set`-Objekte in `Sets` so dass die Repräsentanten `S1._elements[0]` und `S2._elements[0]` gleich `(x1,y1)` bzw. `(x2,y2)` sind. 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. Insbesondere heißt dies, dass `S._elements[0]`, der Repräsentant des neuen Objekts `S`, das lexikographisch kleinere der Tupel `(x1,y1)` und `(x2,y2)` ist. Falls es keine `Set`-Objekte mit Repräsentanten `(x1,y1)` oder `(x2,y2)` gibt, dann werfen Sie die `Exception` 'invalid operation'. Existieren die Mengen, so sind sie aufgrund der Konstruktion der Klasse disjunkt. Sie müssen diesen Spezialfall also nicht abfangen.

Beispielaufrufe:

```
>>> S = Partition([(0,3),(0,1),(1,3),(1,0)])
>>> S.Union((1,3),(0,1))
>>> S.Union((0,3),(0,1))
>>> print(S)
[[(1, 0)], [(0, 1), (0, 3), (1, 3)]]
>>> S.FindSet((1,3))
(0, 1)
>>> S.MakeSet((300,1))
>>> S.Union((300,1),(0,1))
>>> print(S)
[[(1, 0)], [(0, 1), (0, 3), (1, 3), (300, 1)]]
>>> S.FindSet((300,1))
(0, 1)
>>> S.MakeSet((0,0))
>>> S.Union((0,0),(0,1))
>>> S.FindSet((300,1))
(0, 0)
>>> print(S)
[[(1, 0)], [(0, 0), (0, 1), (0, 3), (1, 3), (300, 1)]]
```