

4. Programmieraufgabe Computerorientierte Mathematik II

Abgabe: 21.5.2021 über den Comajudge bis 17 Uhr

Aufgabenstellung

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

Klasse Set

Schreiben Sie eine Klasse `Set` mit dem Attribut

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

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.

Beispielaufwurf:

```
1>>> S = Set([(0,3),(0,1),(1,3),(1,0)])
2>>> print(S._elements)
3 [(0, 3), (0, 1), (1, 3), (1, 0)]
4>>> S.Elements()
5 [(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:

```

1>>> S = Partition([(0,3),(0,1),(1,3),(1,0)])
2>>> S.Union((1,3),(0,1))
3>>> S.Union((0,3),(0,1))
4>>> print(S)
5 [[(1, 0)], [(0, 1), (0, 3), (1, 3)]]
6>>> S.FindSet((1,3))
7 (0, 1)
8>>> S.MakeSet((300,1))
9>>> S.Union((300,1),(0,1))
10>>> print(S)
11 [[(1, 0)], [(0, 1), (0, 3), (1, 3), (300, 1)]]
12>>> S.FindSet((300,1))
13 (0, 1)
14>>> S.MakeSet((0,0))
15>>> S.Union((0,0),(0,1))
16>>> S.FindSet((300,1))
17 (0, 0)
18>>> print(S)
19 [[(1, 0)], [(0, 0), (0, 1), (0, 3), (1, 3), (300, 1)]]

```

Sie können diese Aufgabe bis einschließlich Montag, den 8.6.2020, bei Ihrer Tutorin oder Ihrem Tutor vorstellen.