Modul
Datenstrukturen, Algorithmen und Programmierung 1 (DAP 1)

Praktikumsblatt 7

Dr. Stefan Dissmann
Fakultät für Informatik

Klasse `IntIntPairs`

```
public class IntIntPairs
{
    private boolean[] valids;
    private int[] keys;
    private int[] values;
    private int size;
    private int cap;

    public IntIntPairs( int c ) { ... }

    public boolean uncomplete() { return size < cap; }

    public void put( int k, int v )    { ... }

    public void combine( IntIntPairs other ) {
        for ( int i = 0; i < other.cap ; i++ ) {
            if ( other.valids[i] ) {
                put( other.keys[i], other.values[i] );
            }
        }
    }

    public String toString() { ... }

    public void show() { ... }
```

Klasse `IntIntPairs` (Fortsetzung)

```
public int accumulate( IntFunction func ) {
    int acc = 0;
    for ( int i = 0; i < cap ; i++ ) {
        if ( valids[i] ) {
            acc += func.apply( keys[i], values[i] );
        }
    }
    return acc;
}


public void manipulate( IntFunction keyFunc, IntFunction valueFunc ) {
    for ( int i = 0; i < cap ; i++ ) {
        if ( valids[i] ) {
            keys[i] = keyFunc.apply( keys[i], values[i] );
            values[i] = valueFunc.apply( keys[i], values[i] );
        }
    }
}
```

Klasse `IntIntPairs` (Fortsetzung)

```
public void remove( BoolFunction validFunc ) {
    for ( int i = 0; i < cap ; i++ ) {
        if ( valids[i] && validFunc.apply( keys[i], values[i] ) ) {
            valids[i] = false;
            size--;
        }
    }
}


public IntIntPairs extract( BoolFunction exFunc ) {
    IntIntPairs result = new IntIntPairs( size );
    for ( int i = 0; i < cap ; i++ ) {
        if ( valids[i] && exFunc.apply( keys[i], values[i] ) ) {
            result.put( keys[i], values[i] );
        }
    }
    return result;
}
```

Lösungen

- Ermittle für alle gültigen Einträge des Wertes 0 in `keys` die Summe der zugehörigen Werte in `values`.
  Die Lösung dieser Aufgabe findest Du als Beispiel in der Testumgebung.

```
testPairs.accumulate( (k,v) -> { if (k==0) { return v; } else { return 0; } } )
```

- Bestimme die Anzahl der gültigen geraden Werte in `keys`.

```
testPairs.accumulate( (k,v) -> { if (k%2==0) { return 1; } else { return 0; } }  )
```

- Lösche alle gültigen Paare, deren Wert in `keys` gleich dem Wert 6 ist.

```
testPairs.remove( (k,v) -> k==6 )
```

Lösungen

(Fortsetzung)

- Erhöhe jeden gültigen Wert in `keys` um den Wert 10.

```
testPairs.manipulate( (k,v) -> k+10, (k,v) -> v )
```

- Erstelle ein `IntIntPairs`-Objekt, das Kopien aller gültigen Paare enthält, deren Wert in `values` ganzzahlig durch 3 teilbar ist.

```
testPairs.extract( (k,v) -> v%3==0 )
```

- Bestimme die Anzahl der gültigen Werte in `values`, die größer als 10 sind.

```
testPairs.accumulate( (k,v) -> { if (v>10) { return 1; } else { return 0; } }  )
```

- Erhöhe die Werte in `keys` um den Wert 5, deren zugehöriger Wert in `values` größer als 3 ist.

```
testPairs.manipulate((k,v)->{ if(v>3) {return k+5;} else {return k;}}, (k,v)->v )
```

Lösungen (Fortsetzung)

- Lösche alle gültigen Paare, die einen negativen Wert in `values` besitzen.

```
testPairs.remove( (k,v) -> v<0 )
```

- Verdopple jeden gültigen Wert in `values`.

```
testPairs.manipulate( (k,v) -> k, (k,v) -> 2*v )
```

- Erstelle eine Kopie des `IntIntPairs`-Objekts, die alle gültigen Paare enthält.

```
testPairs.extract( (k,v) -> true )
```

- Lösche alle gültigen Paare, deren beide Werte identisch sind.

```
testPairs.remove( (k,v) -> k==v )
```

Lösungen (Fortsetzung)

- Bilde die Summe aller gültigen Werte in `values`.

```
testPairs.accumulate( (k,v) ->  v )
```

- Erstelle ein `IntIntPairs`-Objekt, das Kopien aller gültigen Paare enthält, deren Wert in `values` größer als der Wert 5 ist.

```
testPairs.extract( (k,v) -> v>5 )
```

Lösungen (Fortsetzung)

- Implementiere eine Methode `sumUp(IntIntPairs pairs)`, die für die gültigen Paare in `pairs`, deren Wert in `keys` gleich 0 ist, die zugehörigen Werte aus `values` aufsummiert.
  Die Lösung dieser Aufgabe finden Sie als Beispiel in der Testumgebung.

```
public static int sumUp(IntIntPairs pairs)
{
   return pairs.accumulate( (k,v) -> { if (k==0) { return v; } else { return 0; } } );
}
```

Lösungen (Fortsetzung)

- Implementiere eine Methode `addNToValue(IntIntPairs pairs, int n)`, die für die gültigen Paare in `pairs` die Werte in `values` um den Wert `n` erhöht.

```
public static void addNToValue( IntIntPairs pairs, int n )
{
    pairs.manipulate( (k,v) -> k, (k,v) -> v+n );
}
```

Lösungen (Fortsetzung)

- Implementiere eine Methode `uniqueKey(IntIntPairs pairs, int n)`, die `true` zurückgibt, wenn in allen gültigen Paaren in `pairs` der Wert von `n` in `keys` genau einmal auftritt.

```
public static boolean uniqueKey( IntIntPairs pairs, int n )
{
    return
    pairs.accumulate( (k,v) -> { if (k==n) { return 1; } else { return 0; } } ) == 1;
}
```

Lösungen (Fortsetzung)

- Implementiere eine Methode `doubleGreaterN(IntIntPairs pairs, int n)`, die für jedes gültige Paar `(k,v)` in `pairs`, dessen Wert `k` in `keys` größer als der Wert `n` ist, zu `pairs` ein weiteres Paar `(k,2*v)` hinzufügt, sofern in `pairs` noch Einträge möglich sind.

```
public static void doubleGreaterN( IntIntPairs pairs, int n )
{
    IntIntPairs greater = pairs.extract( (k,v) -> k>n );
    greater.manipulate( (k,v) -> k, (k,v) -> 2*v );
    pairs.combine( greater );
}
```

Lösungen (Fortsetzung)

- Implementiere eine Methode
  `IntIntPairs concat(IntIntPairs first, IntIntPairs second)`, die ein neues `IntIntPairs`-Objekt erzeugt, das alle gültigen Paare von `first` und `second` enthält. `first` und `second` sollen nicht geändert werden.

```
public static IntIntPairs concat(IntIntPairs first, IntIntPairs second)
{
    IntIntPairs all =
                new IntIntPairs( first.accumulate( (k,v) -> 1 )
                + second.accumulate( (k,v) -> 1 ) );
    all.combine( first );
    all.combine( second );
    return all;
}
```