BIJECTION TYPE FOR JULIA

Fundamentals

This is documentation for a `Bijection` datatype in Julia. A `Bijection` behaves like an extension of a `Dict` but we ensure that the mapping from keys to values is one-to-one and we provide an efficient way to map backwards from values to keys.

To get this module, do this (one time):

```
Pkg.clone("https://github.com/scheinerman/Bijections.jl.git")
```

Then `using Bijections` each session to load this module.

To create a `Bijection` we do this:

```
julia> b = Bijection()
```

in which the domain and range can be `Any`. Alternatively, to specify the types of the domain and range, use something like this:

```
julia> b = Bijection{Int, String}()
```

To add pairs to `b`, we can use the usual square bracket notation:

```
julia> b[5] = "hello"
julia> b[0] = "what?"
```

We cannot now define `b[0]` to have another value, nor can we define `b[1]` to be an existing range element:

```
julia> b[5] = "bye"
ERROR: One of x or y already in this Bijection
 in setindex! at .....

julia> b[1] = "hello"
ERROR: One of x or y already in this Bijection
 in setindex! at .....
```

To get the value associated with a given key, we use the usual square brackets:

```
julia> print(b[0])
what?
```

Because distinct keys map to distinct values, we can invert from values to keys:

```
julia> inverse(b,"hello")
5
```

If we want to change the mapping of `5` to `hello` we need to first delete the pair and redefine `b[5]` like this:

```
julia> delete!(b,5)
[(0,"what?")]

julia> b[5]="bye"
"bye"
```

User Methods

These are the methods that users should use. Other methods in the file support the implementation and need not be (should not be) used.

- `Bijection`: This is the constructor used like this:

  `b = Bijection{S,T}()`

  where `S` and `T` are types. One can also use `b = Bijection()` which is equivalent to `b = Bijection{Any,Any}()`.

  There is one other form: `b = Bijection(x,y)` where `x` and `y` are any two objects. This sets up `b` in which the domain elements have the same type as `x` and whose range elements have the same type as `y`. And this initializes `b` with the pair `(x,y)`.
- `setindex!`: This is used to add a key-value pair to a bijection using the syntax `b[x] = y`. If `x` is already in the domain or `y` is already in the range, then an error is raised.
- `getindex`: This is used to query the value associated with a given key using the syntax `b[x]`. If `x` is not in the domain, an error is raised.
- `inverse`: This is used for the inverse mapping from values to keys. Syntax is `inverse(b,y)` to get the key `x` such that `b[x]==y`. If `y` is not in the range, then an error is raised.
- `delete!`: This is used to remove a key-value pair from a bijection. If `x` is in the domain, use `delete!(b,x)`.
- `length`: Give the number of elements in the bijection. Syntax is `length(b)`.
- `isempty`: Invoking `isempty(b)` returns `true` is `b` has no elements; otherwise it returns `false`.
- `collect`: Returns the pairs in a bijection as an `Array` of tuples.
- `domain`: Return, as an `Array`, the elements of the domain of a bijection (the keys).
- `range`: Return, as an `Array`, the elements of the range of a bijection (the values).