Minimalist Graph

Setup

Objective: Implementation of a generic graph representation and traversal, applied to three example graph implementations/domains.

Implementation language is Scala (2 or 3). The code should not exceed 250 LoC. You can use any libraries you like (the Scala standard library should be fully sufficient, though). When using external libraries, please provide a sbt project, otherwise plain source code is fine. If you happen to write any unit test cases, please add them, but tests are not required.

You should not spend more than two, max three, hours on this. Don't worry if you don't complete everything or if you feel your code may not be perfect – nobody expects completed production code. The aim is to have some basis for a free-form discussion about language/library features, implementation choices/alternatives, algorithmic variants, etc.

Graph Representation

Given: A minimalist generic type representing a directed graph over vertices of an arbitrary type `T`. (Feel free to modify this definition if it better suits your implementation ideas for the tasks below.)

```
trait Graph[T] {
  def edges(v: T): List[T]
}
```

Instances

Rock, Paper, Scissors Instance

Implement an `RPSMove` type representing the moves of the game of Rock, Paper, Scissors. Then, implement a `Graph[RPSMove]` instance - there is an edge between vertices A and B if A beats B in the game. Edges for `Rock` should be `[ Scissors ]`.

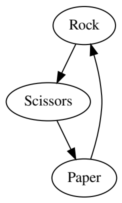

Figure 1: Rock, Paper, Scissors Graph

Map Instance

Provide a graph implementation for arbitrary types `T` backed by an adjacency map `Map[T, List[T]]`. Given a `Graph[Int]` backed by `Map(1 -> List(2, 3), 2 -> List(1))`, edges for `1` should be `[ 2, 3 ]`.

Path Instance

Provide a `Graph[Path]` implementation for `java.nio.file.Path`. A `Path` instance's edges are its child files if it is a directory, no children otherwise. In the following scenario…

```
foo
├── bar
│   ├── b.scala
│   └── qux.txt
├── baz.txt
└── f.scala
```

…edges of `foo` should be `[ foo/bar, foo/baz.txt, foo/f.scala ]`.

> Hint: Use `scala.jdk.StreamConverters` to convert Java `Stream` to a Scala collection.

Traversal

Plain DFS

Write a generic depth-first traversal for `Graph[T]` that takes a starting vertex of type `T` as its input. You are free to choose any way/return type you like for passing the discovered vertices to the caller. Don't check for cycles, assume an acyclic graph for now.

File Search

Use the `Path` implementation and the DFS together with some appropriate filtering to list all `**/*.scala` files inside an arbitrary directory. For the above file system layout, the output should be `[ foo/bar/b.scala, foo/f.scala ]`.

Optional: Safe Traversal

This part is not required. Take it if you're still within the three hours and have nothing better to do.

Safe DFS

Write a depth-first traversal that checks for cycles and never visits a vertex twice.

Map Traversal

Exercise the `Map` implementation and the safe DFS using this instance:

```
Map(
  1 -> List(2, 5, 8),
  2 -> List(3),
  3 -> List(4),
  4 -> List(2),
  5 -> List(6),
  6 -> List(3, 7, 8)
)
```

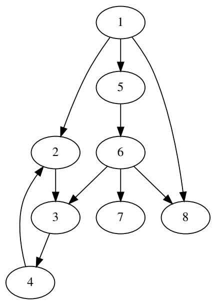

Figure 2: Example Map Graph

For starting vertex `1`, the result should be `1,2,3,4,5,6,7,8`.