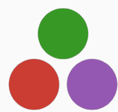
Julia For Data Science | Cheat Sheet
LightGraphs.jl Essentials



Created by: Camilo Chacón Sartori | www.camilochacon.com

September 2021

Julia version: v1.6.2 - Package versions: [LightGraphs=v1.3.5;SimpleWeightedGraphs=v1.1.1;GraphPlot=v0.4.4;Compose=v0.9.2;Cairo=v1.0.5]

What is LightGraphs.jl?

LightGraphs.jl is a library used to graph data manipulation that for practical ends, its functionality is similar to NetworkX in python.

Below how to install it:

```
julia> using Pkg; Pkg.add("LightGraphs")
julia> using LightGraphs
```

Graphs Constructors

To create a undirected graph of three vertex:

```
julia> g = SimpleGraph(3) # create a three vertices undirected graph.

julia> g = SimpleDiGraph(3) # create a three vertices directed graph.
```

Basic Operations On A Graph

```
julia> ne(g) # number of edges.
julia> nv(g) # number of vertices.

julia> degree(g) # graph degree.

julia> has_vertex(g, 1) # checks for whether graph includes vertex.

julia> has_edge(g, 1, 2) # checks for whether graph includes edge.
julia> has_edge(g, (1, 2)) # checks for whether graph includes edge (tuple format).

julia> collect(vertices(g)) # return array of all vertices.
julia> collect(edges(g)) # return array of all edges.

julia> is_directed(g) # checks if graph is directed.
```

Transforming

```
julia> add_edge!(g, 1, 2) # add a edge.
julia> add_edge!(g, (2, 3)) # add a edge using a tuple.

julia> add_vertex!(g) # add one vertex.
julia> add_vertices!(g, 4) # add n vertices.

julia> rem_edge!(g, 1, 2) # remove a edge.
julia> rem_vertex!(g, 1) # remove a vertex.
```

I/O

Save A Graph To Disk

```
julia> savegraph("my_graph.lgz", g)
```

Load A Graph From disk

```
julia> g = loadgraph("mygraph.lgz")
```

Properties

Vertex Properties

To undirected graph:

```
julia> neighbors(g, 1) # return array of neighbors of a vertex.
```

To directed graph:

```
julia> all_neighbors(g, 1) # # return array of all neighbors of a vertex (for directed graph).
julia> inneighbors(g, 1) # return array of in-neighbors
julia> outneighbors(g, 1) # return array of out-neighbors
```

Edges Properties

```
julia> src(collect(edges(g))[1]) # get source vertex of a edge

julia> dst(collect(edges(g))[1]) # get destination vertex of a edge

julia> reverse(collect(edges(g))[1]) # Create a new edge from other edge with source and destination vertices reversed.
```

Distances

```
julia> center(g)
julia> diameter(g)
julia> eccentricity(g)
julia> periphery(g)
julia> radius(g)
```

Parallel

```
julia> import LightGraphs.Parallel

julia> Parallel.dijkstra_shortest_paths(g, 1)

julia> Parallel.center(g)
julia> Parallel.diameter(g)
julia> Parallel.eccentricity(g)
julia> Parallel.periphery(g)
julia> Parallel.radius(g)
```

Weighted Graphs

```
julia> Using Pkg; Pkg.add("SimpleWeightedGraphs")
julia> using SimpleWeightedGraphs

julia> g1 = SimpleWeightedGraph(5)

julia> add_edge!(g1, 1, 2, 0.1)
julia> add_edge!(g1, 2, 3, 0.4)
julia> add_edge!(g1, 2, 4, 0.9)
julia> add_edge!(g1, 4, 5, 0.1)
```

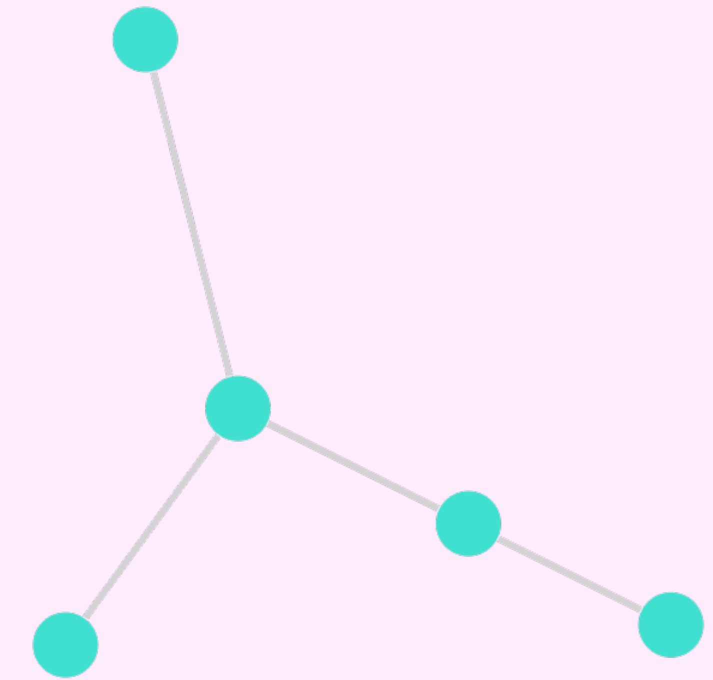
Shortest Path

```
julia> ds = dijkstra_shortest_paths(g1, 1) # return shortest distances between src and all other vertices.
julia> ds.dists

julia> enumerate_paths(dijkstra_shortest_paths(g1, 1), 5) # get the shortest path from vertex 1 to 5 taking into account the weights.
```

Visualization

As a prerequisite we need to have installed GraphPlot.jl:

```
julia> using Pkg; Pkg.add("GraphPlot")

julia> gplot(g1)
```



Save A Graph To Disk

As a prerequisite we need to have installed Compose.jl and Cairo.jl:

```
julia> using Pkg; Pkg.add(["Compose", "Cairo"])

julia> draw(PDF("my_gplot.pdf", 15cm, 15cm), gplot(g1)) # save to png
julia> draw(PNG("my_gplot.png", 15cm, 15cm), gplot(g1)) # save to svg
julia> draw(SVG("my_gplot.svg", 15cm, 15cm), gplot(g1)) # save to svg
```