Lucas Schmidt Ferreira de Araujo

Report 02

Exercice I

In [1]:
```
mutable struct Graph
    vertices::Dict{Any, Vector{Any}}
end

function Graph()
    return Graph(Dict{Any, Vector{Any}}())
end

function addVertix!(graph::Graph , element::Any)
    graph.vertices[element] = []
end

function addVerticesFromList!(graph::Graph, arr::Array{T}) where T
    for q in arr
        addVertix!(graph, q)
    end
end

function addEdge!(graph::Graph, from::T, to::T) where T
    if !haskey(graph.vertices, from) || !haskey(graph.vertices, to)
        error("Both vertices must exist in the graph.")
    else
        push!(graph.vertices[from], to)
        push!(graph.vertices[to], from)
    end
end

function addEdgesFromList!(graph::Graph, arr::Vector{Vector{T}}) where T
    for q in arr
        addEdge!(graph,q[1],q[2])
    end
end

function getVertices(graph::Graph)
    return collect(keys(graph.vertices))
end

function getEdges(graph::Graph)
    edges = []
    for q in keys(graph.vertices)
        for j in graph.vertices[q]
            if !([j,q] in edges)
                push!(edges,[q,j])
            end
        end
    end
    return edges
```

```
end

function getNeighbors(graph::Graph , element::T) where T
    return graph.vertices[element]
end

function isin(graph::Graph , element::T) where T
    return haskey(graph.vertices,element)
end

function saveGraph(graph::Graph, filename::String)
    file = open(filename, "w")
    println(file, "graph {")
    edges = getEdges(graph)

    for q in edges
        println(file,"$(q[1]) -- $(q[2]);")
    end

    println(file, "}")
    close(file)
end

function f!(graph::Graph, element::T,paths::Dict,searched) where T
    nb = getNeighbors(G,element)
    for q in nb
        if paths[q] == 0
            paths[q] = paths[element] + 1
        end
    end
    return filter(x -> !(x in searched),nb)
end

function getShortestPathsLengths(graph::Graph, element::T) where T
    vertices = getVertices(G)
    paths = Dict(zip(vertices,zeros(length(vertices))))
    searched = [element]
    j = 1
    nb = getNeighbors(G,searched[j])
    while(length(searched) != length(vertices) && j <= length(searched))
        nb = f!(G,searched[j],paths,searched)
        searched = vcat(searched,nb)
        j+=1
    end
    return delete!(paths,element)
end
```

```
getShortestPathsLengths (generic function with 1 method)
```

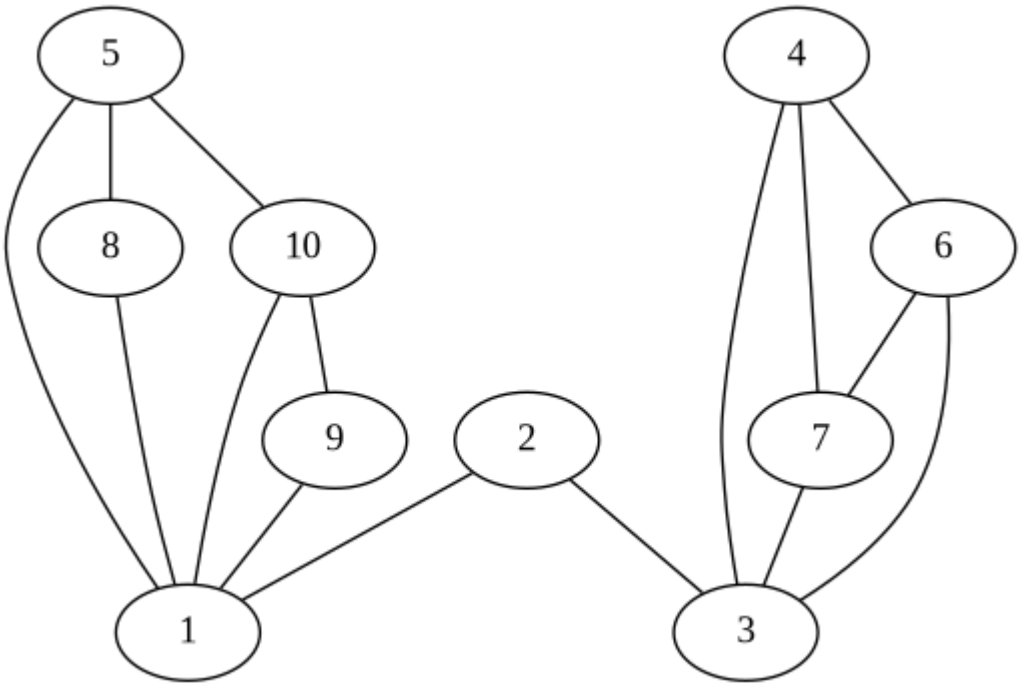
- Alice - 1, Bob - 2 , Gail - 3 , Irene - 4 , Carl - 5
- Harry - 6 , Jen - 7 , David - 8 , Ernest - 9 , Frank - 10

In [2]:
```
G = Graph()
addVertix!(G,1)
addVerticesFromList!(G,[2,3,4,5,6,7,8,9,10])
addEdge!(G,1,2)
addEdge!(G,2,3)
addEdge!(G,4,3)
addEdge!(G,5,1)
```

```
addEdge!(G,3,6)
addEdge!(G,4,7)
addEdge!(G,1,8)
addEdge!(G,6,7)
addEdge!(G,9,10)
addEdge!(G,1,9)
addEdge!(G,7,3)
addEdge!(G,8,5)
addEdge!(G,1,10)
addEdge!(G,6,4)
addEdge!(G,5,10)

saveGraph(G,"teste.dot")
```

The graphviz software can be installed as described in the website. Running the command $dot test.dot -Tpng -o teste.png, we have the visualization



In [3]:
```
isin(G,2)
```

```
true
```

In [4]:
```
isin(G,20)
```

```
false
```

In [5]:
```
getVertices(G)
```

```
10-element Vector{Any}:
  5
  4
  6
  7
  2
 10
  9
  8
  3
  1
```

In [6]:
```
getNeighbors(G,2)
```

```
2-element Vector{Any}:
 1
 3
```

In [7]:
```
getEdges(G)
```

```
15-element Vector{Any}:
 [5, 1]
 [5, 8]
 [5, 10]
 [4, 3]
 [4, 7]
 [4, 6]
 [6, 3]
 [6, 7]
 [7, 3]
 [2, 1]
 [2, 3]
 [10, 9]
 [10, 1]
 [9, 1]
 [8, 1]
```

In [8]:
```
getShortestPathsLengths(G,7)
```

```
Dict{Any, Float64} with 9 entries:
  5  => 4.0
  4  => 1.0
  6  => 1.0
  2  => 2.0
  10 => 4.0
  9  => 4.0
  8  => 4.0
  3  => 1.0
  1  => 3.0
```

In [9]:
```
getShortestPathsLengths(G,1)
```

```
Dict{Any, Float64} with 9 entries:
  5  => 1.0
  4  => 3.0
  6  => 3.0
  7  => 3.0
  2  => 1.0
  10 => 1.0
  9  => 1.0
  8  => 1.0
  3  => 2.0
```

In [10]:
```
getShortestPathsLengths(G,8)
```

```
Dict{Any, Float64} with 9 entries:
  5  => 1.0
  4  => 4.0
  6  => 4.0
  7  => 4.0
  2  => 2.0
  10 => 2.0
  9  => 2.0
  3  => 3.0
  1  => 1.0
```