Algorithm graph theory: a practical approach

Master de Bioinformática (UV)

Author: Alejandro Alemán Ramos

Bioinformatic at the Computational Genomics Institute (CIPF)

October 17th, 2013

Overview

iGraph

- Library for complex network research
- R & python implementations
- Core implementation: C
- http://igraph.sourceforge.net
- Python API: http://igraph.sourceforge.net/doc/python/index.html

iGraph

- Library for complex network research
- R & python implementations
- Core implementation: C
- http://igraph.sourceforge.net
- Python API: http://igraph.sourceforge.net/doc/python/index.html

iGraph

- Library for complex network research
- R & python implementations
- Core implementation: C
- http://igraph.sourceforge.net
- Python API: http://igraph.sourceforge.net/doc/python/index.html

iGraph

- Library for complex network research
- R & python implementations
- Core implementation: C
- http://igraph.sourceforge.net
- Python API: http://igraph.sourceforge.net/doc/python/index.html

iGraph

- Library for complex network research
- R & python implementations
- Core implementation: C
- http://igraph.sourceforge.net
- Python API: http://igraph.sourceforge.net/doc/python/index.html

Installation

- Debian:
  - sudo apt-get install python-igraph
- Open Suse
  - sudo yum install python-igraph
- easy_install or pip
  - sudo easy_install python-igraph
  - sudo pip install python-igraph

Installation

- Debian:
  - sudo apt-get install python-igraph
- Open Suse
  - sudo yum install python-igraph
- easy_install or pip
  - sudo easy_install python-igraph
  - sudo pip install python-igraph

Installation

- Debian:
  - sudo apt-get install python-igraph
- Open Suse
  - sudo yum install python-igraph
- easy_install or pip
  - sudo easy_install python-igraph
  - sudo pip install python-igraph

Installation

- Debian:
  - sudo apt-get install python-igraph
- Open Suse
  - sudo yum install python-igraph
- easy_install or pip
  - sudo easy_install python-igraph
  - sudo pip install python-igraph

Installation

- Debian:
  - sudo apt-get install python-igraph
- Open Suse
  - sudo yum install python-igraph
- easy_install or pip
  - sudo easy_install python-igraph
  - sudo pip install python-igraph

Installation

- Debian:
  - sudo apt-get install python-igraph
- Open Suse
  - sudo yum install python-igraph
- easy_install or pip
  - sudo easy_install python-igraph
  - sudo pip install python-igraph

Installation

- Debian:
  - sudo apt-get install python-igraph
- Open Suse
  - sudo yum install python-igraph
- easy_install or pip
  - sudo easy_install python-igraph
  - sudo pip install python-igraph

Basic Concepts

- Graph
- Node
- Edge
- Weight
- DAG: Directed Acyclic Graph

Basic Concepts

- Graph
- Node
- Edge
- Weight
- DAG: Directed Acyclic Graph

Basic Concepts

- Graph
- Node
- Edge
- Weight
- DAG: Directed Acyclic Graph

Basic Concepts

- Graph
- Node
- Edge
- Weight
- DAG: Directed Acyclic Graph

Basic Concepts

- Graph
- Node
- Edge
- Weight
- DAG: Directed Acyclic Graph

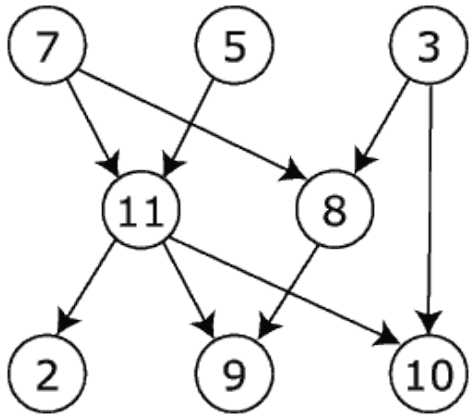

Figura: DAG

Breadth-first search

The algorithm uses a queue data structure to store intermediate results as it traverses the graph, as follows:

1. Create empty array "path" and a queue
2. Enqueue the root node
3. Dequeue a node and examine it
   - Add node to path
   - Enqueue any successors (the direct child nodes) that have not yet been discovered.
4. If the queue is empty, every node on the graph has been examined.
5. If the queue is not empty, repeat from Step 3.

Breadth-first search

The algorithm uses a queue data structure to store intermediate results as it traverses the graph, as follows:

1. Create empty array "path" and a queue
2. Enqueue the root node
3. Dequeue a node and examine it
   - Add node to path
   - Enqueue any successors (the direct child nodes) that have not yet been discovered.
4. If the queue is empty, every node on the graph has been examined.
5. If the queue is not empty, repeat from Step 3.

Breadth-first search

The algorithm uses a queue data structure to store intermediate results as it traverses the graph, as follows:

1. Create empty array "path" and a queue
2. Enqueue the root node
3. Dequeue a node and examine it
   - Add node to path
   - Enqueue any successors (the direct child nodes) that have not yet been discovered.
4. If the queue is empty, every node on the graph has been examined.
5. If the queue is not empty, repeat from Step 3.

Breadth-first search

The algorithm uses a queue data structure to store intermediate results as it traverses the graph, as follows:

1. Create empty array "path" and a queue
2. Enqueue the root node
3. Dequeue a node and examine it
   - Add node to path
   - Enqueue any successors (the direct child nodes) that have not yet been discovered.
4. If the queue is empty, every node on the graph has been examined.
5. If the queue is not empty, repeat from Step 3.

Breadth-first search

The algorithm uses a queue data structure to store intermediate results as it traverses the graph, as follows:

1. Create empty array "path" and a queue
2. Enqueue the root node
3. Dequeue a node and examine it
   - Add node to path
   - Enqueue any successors (the direct child nodes) that have not yet been discovered.
4. If the queue is empty, every node on the graph has been examined.
5. If the queue is not empty, repeat from Step 3.

Breadth-first search

The algorithm uses a queue data structure to store intermediate results as it traverses the graph, as follows:

1. Create empty array "path" and a queue
2. Enqueue the root node
3. Dequeue a node and examine it
   - Add node to path
   - Enqueue any successors (the direct child nodes) that have not yet been discovered.
4. If the queue is empty, every node on the graph has been examined.
5. If the queue is not empty, repeat from Step 3.

Breadth-first search

The algorithm uses a queue data structure to store intermediate results as it traverses the graph, as follows:

1. Create empty array "path" and a queue
2. Enqueue the root node
3. Dequeue a node and examine it
   - Add node to path
   - Enqueue any successors (the direct child nodes) that have not yet been discovered.
4. If the queue is empty, every node on the graph has been examined.
5. If the queue is not empty, repeat from Step 3.

Breadth-first search

Example

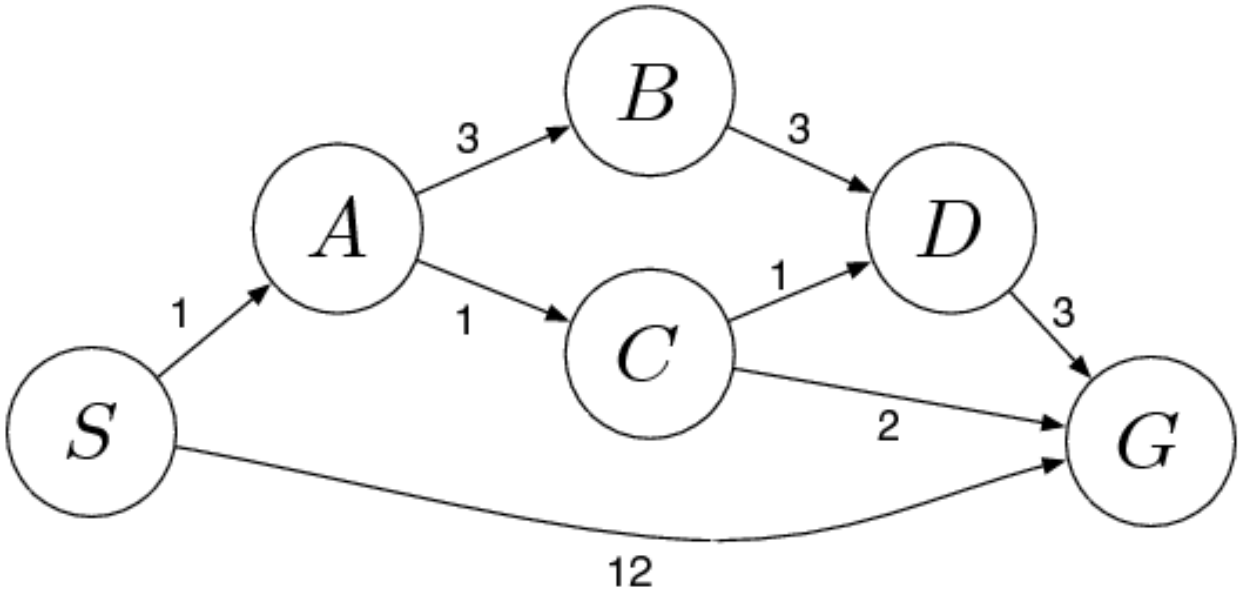

Figura: BFS example

Depth-first search

The algorithm uses a stack data structure to store intermediate results as it traverses the graph, as follows:

1. Create empty array "path" and a stack
2. Add the root node to the stack
3. Take a node and examine it
   - Add node to path
   - Add any successors (the direct child nodes) that have not yet been discovered to the stack.
4. If the stack is empty, every node on the graph has been examined.
5. If the stack is not empty, repeat from Step 3.

Depth-first search

The algorithm uses a stack data structure to store intermediate results as it traverses the graph, as follows:

1. Create empty array "path" and a stack
2. Add the root node to the stack
3. Take a node and examine it
   - Add node to path
   - Add any successors (the direct child nodes) that have not yet been discovered to the stack.
4. If the stack is empty, every node on the graph has been examined.
5. If the stack is not empty, repeat from Step 3.

Depth-first search

The algorithm uses a stack data structure to store intermediate results as it traverses the graph, as follows:

1. Create empty array "path" and a stack
2. Add the root node to the stack
3. Take a node and examine it
   - Add node to path
   - Add any successors (the direct child nodes) that have not yet been discovered to the stack.
4. If the stack is empty, every node on the graph has been examined.
5. If the stack is not empty, repeat from Step 3.

Depth-first search

The algorithm uses a stack data structure to store intermediate results as it traverses the graph, as follows:

1. Create empty array "path" and a stack
2. Add the root node to the stack
3. Take a node and examine it
   - Add node to path
   - Add any successors (the direct child nodes) that have not yet been discovered to the stack.
4. If the stack is empty, every node on the graph has been examined.
5. If the stack is not empty, repeat from Step 3.

Depth-first search

The algorithm uses a stack data structure to store intermediate results as it traverses the graph, as follows:

1. Create empty array "path" and a stack
2. Add the root node to the stack
3. Take a node and examine it
   - Add node to path
   - Add any successors (the direct child nodes) that have not yet been discovered to the stack.
4. If the stack is empty, every node on the graph has been examined.
5. If the stack is not empty, repeat from Step 3.

Depth-first search

The algorithm uses a stack data structure to store intermediate results as it traverses the graph, as follows:

1. Create empty array "path" and a stack
2. Add the root node to the stack
3. Take a node and examine it
   - Add node to path
   - Add any successors (the direct child nodes) that have not yet been discovered to the stack.
4. If the stack is empty, every node on the graph has been examined.
5. If the stack is not empty, repeat from Step 3.

Depth-first search

The algorithm uses a stack data structure to store intermediate results as it traverses the graph, as follows:

1. Create empty array "path" and a stack
2. Add the root node to the stack
3. Take a node and examine it
   - Add node to path
   - Add any successors (the direct child nodes) that have not yet been discovered to the stack.
4. If the stack is empty, every node on the graph has been examined.
5. If the stack is not empty, repeat from Step 3.

Depth-first search

Example

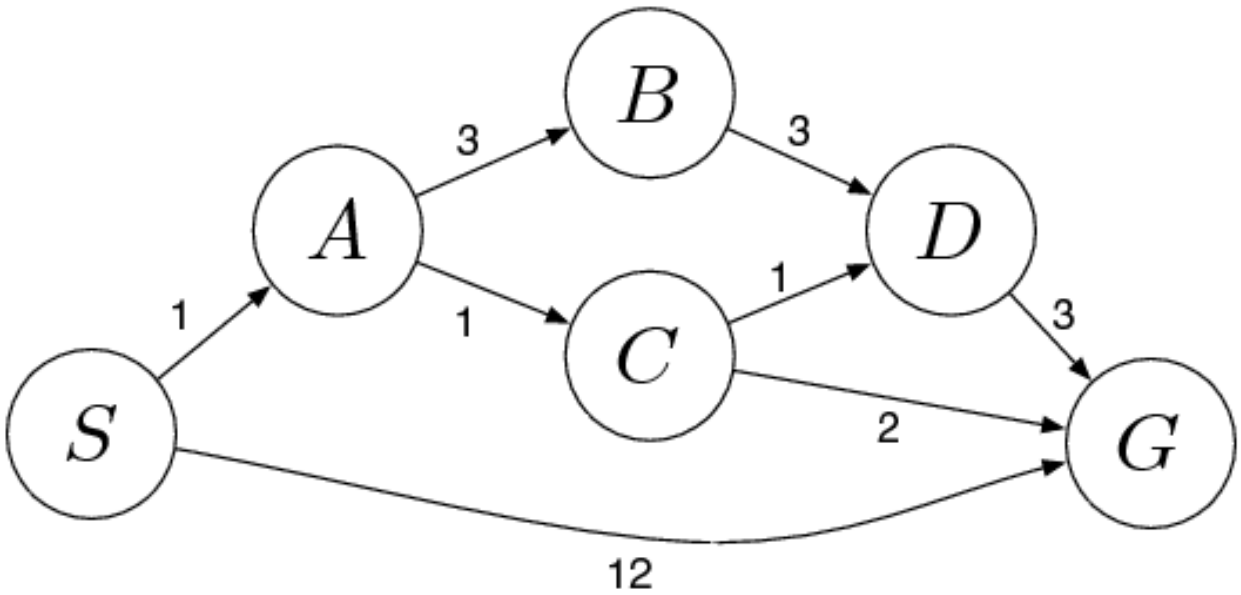

Figura: DFS example

Is there any path?

A path in a graph is a sequence of vertices such that from each of its vertices there is an edge to the next vertex in the sequence

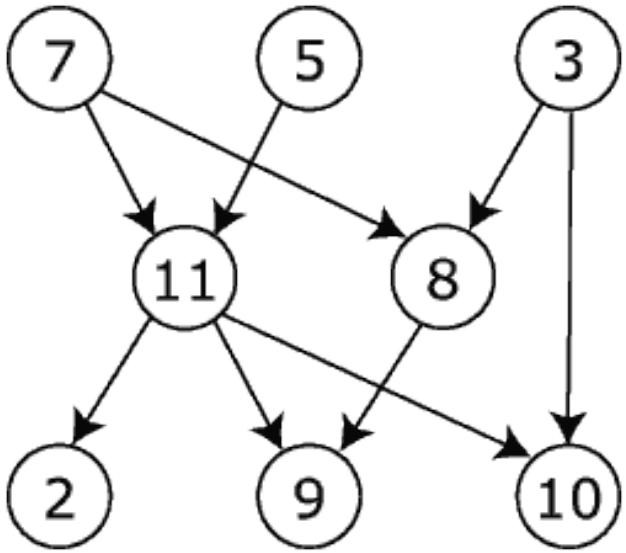

Figura: Is there any path? Example

Dijkstra

Dijkstra algorithm is a graph search algorithm that solves the single-source shortest path problem for a graph with non-negative edge path costs, producing a shortest path tree.

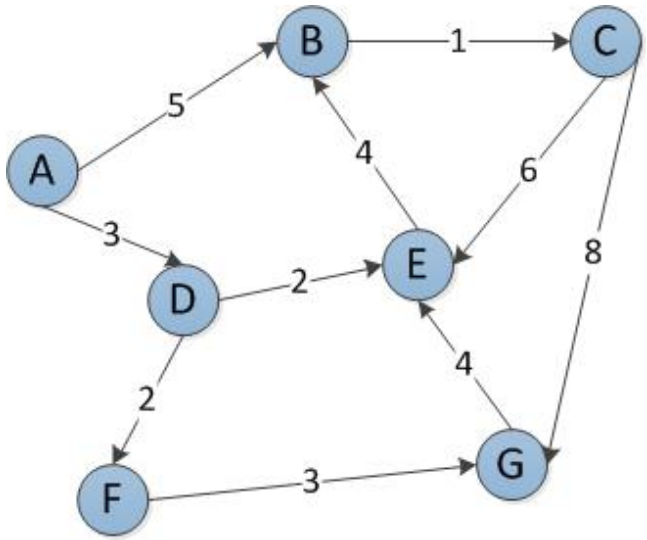

Figura: Dijkstra

Questions

Questions?

Thank you