Graphs: Cycles

Tecniche di Programmazione – A.A. 2016/2017

Summary

- Definitions
- Algorithms

Definitions

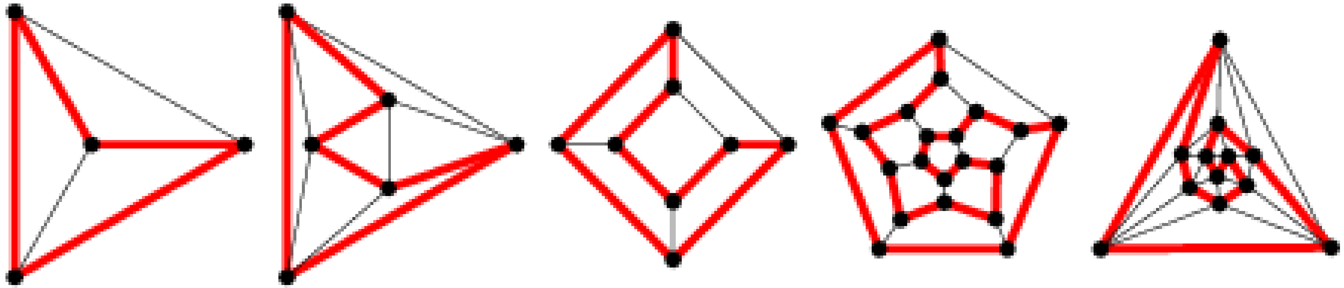
Graphs: Cycles

Cycle

- A **cycle** of a graph, sometimes also called a circuit, is a subset of the edge set of that forms a path such that the first node of the path corresponds to the last.

Hamiltonian cycle

- A cycle that uses each graph vertex of a graph exactly once is called a Hamiltonian cycle.



Hamiltonian path

- A Hamiltonian path, also called a Hamilton path, is a path between two vertices of a graph that visits each vertex exactly once.
  - N.B. does not need to return to the starting point

Eulerian Path and Cycle

- An **Eulerian path**, also called an Euler chain, Euler trail, Euler walk, or "Eulerian" version of any of these variants, is a walk on the graph edges of a graph which **uses each graph edge** in the original graph **exactly once**.
- An **Eulerian cycle**, also called an Eulerian circuit, Euler circuit, Eulerian tour, or Euler tour, is a trail which starts and ends at the **same** graph vertex.

Theorem

- A connected graph has an Eulerian **cycle** if and only if it **all vertices have even degree**.
- A connected graph has an Eulerian **path** if and only if it has **at most two graph vertices of odd degree**.

  - …easy to check!

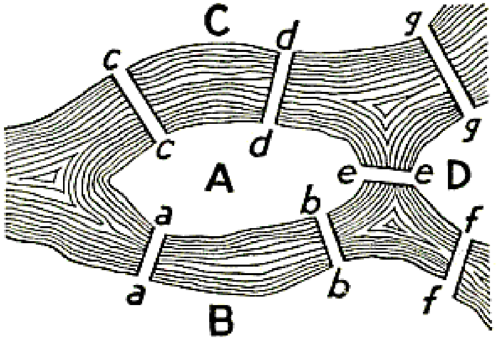
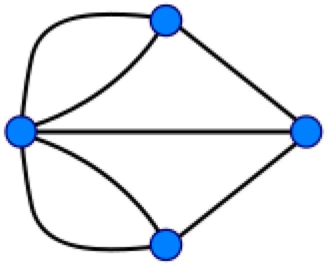


FIGURE 98. *Geographic Map: The Königsberg Bridges.*



Königsberg Bridges

Weighted vs. Unweighted

- Classical versions defined on Unweighted graphs
- Unweighted:
  - Does such a cycle exist?
  - If yes, find at least one
    - Optionally, find all of them
- Weighted
  - Does such a cycle exist?
    - Often, the graph is complete ☺
  - If yes, find at least one
  - If yes, find **the best one** (with **minimum** weight)

Algorithms

Graphs: Cycles

Eulerian cycles: Hierholzer's algorithm (1)

- Choose **any** starting vertex *v*, and **follow a trail** of edges from that vertex until returning to *v*.
  - It is **not** possible to get stuck at any vertex other than *v*, because the even degree of all vertices ensures that, when the trail enters another vertex *w* there must be an unused edge leaving *w*.
  - The tour formed in this way is a **closed** tour, but may **not** cover all the vertices and edges of the initial graph.

Eulerian cycles: Hierholzer's algorithm (2)

- As long as there exists a vertex *v* that belongs to the current tour but that has adjacent edges not part of the tour, **start another trail** from *v*, following **unused** edges until returning to *v*, **and join** the tour formed in this way to the previous tour.

Finding Eulerian circuits

Hierholzer's Algorithm

Given: an Eulerian graph G

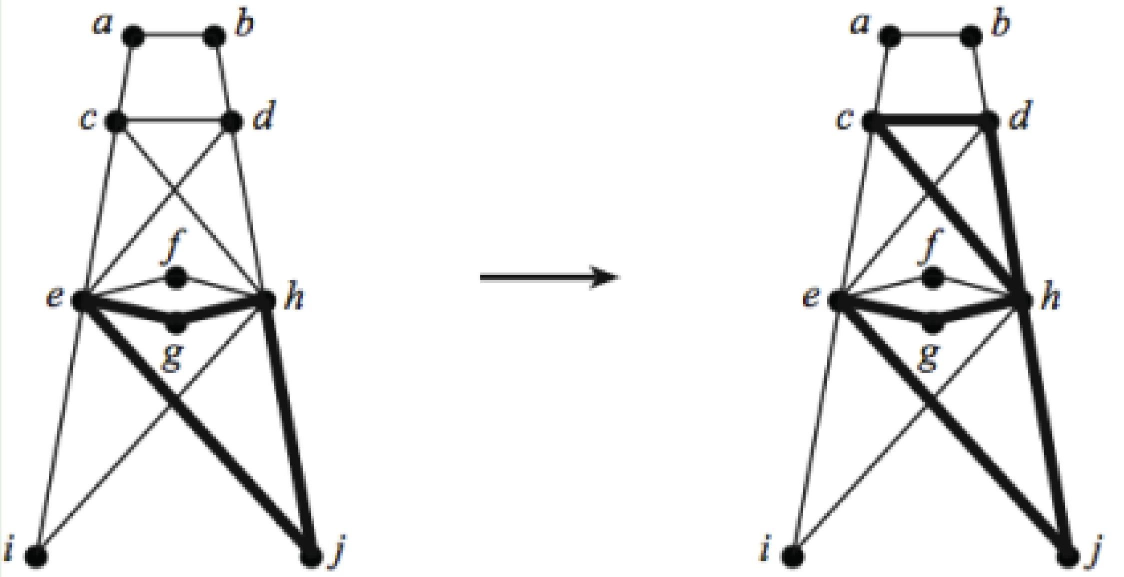
Find an Eulerian circuit of G.

1. Identify a circuit in G and call it R_1. Mark the edges of R_1. Let $i = 1$.
2. If R_i contains all edges of G, then stop (since R_i is an Eulerian circuit).
3. If R_i does not contain all edges of G, then let v_i be a node on R_i that is incident with an unmarked edge, e_i.
4. Build a circuit, Q_i, starting at node v_i and using edge e_i. Mark the edges of Q_i.
5. Create a new circuit, R_{i+1}, by patching the circuit Q_i into R_i at v_i.
6. Increment i by 1, and go to step (2).

Finding Eulerian circuits

Hierholzer's Algorithm

Example



R_1: *e*, *g*, *h*, *j*, *e*

Q_1: *h*, *d*, *c*, *h*

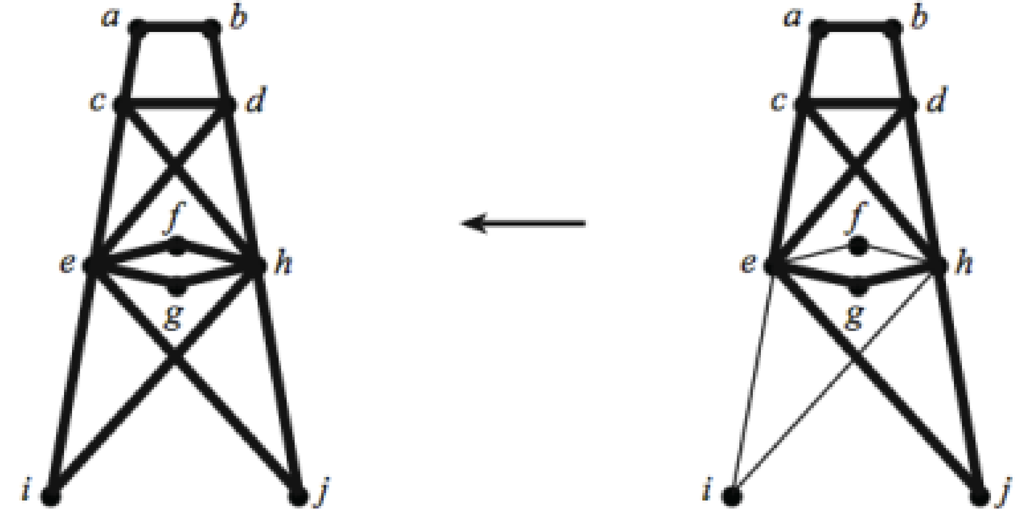
R_2: *e*, *g*, ***h*, *d*, *c*, *h***, *j*, *e*

Q_2: *d*, *b*, *a*, *c*, *e*, *d*

Finding Eulerian circuits

Hierholzer's Algorithm

Example (continued)

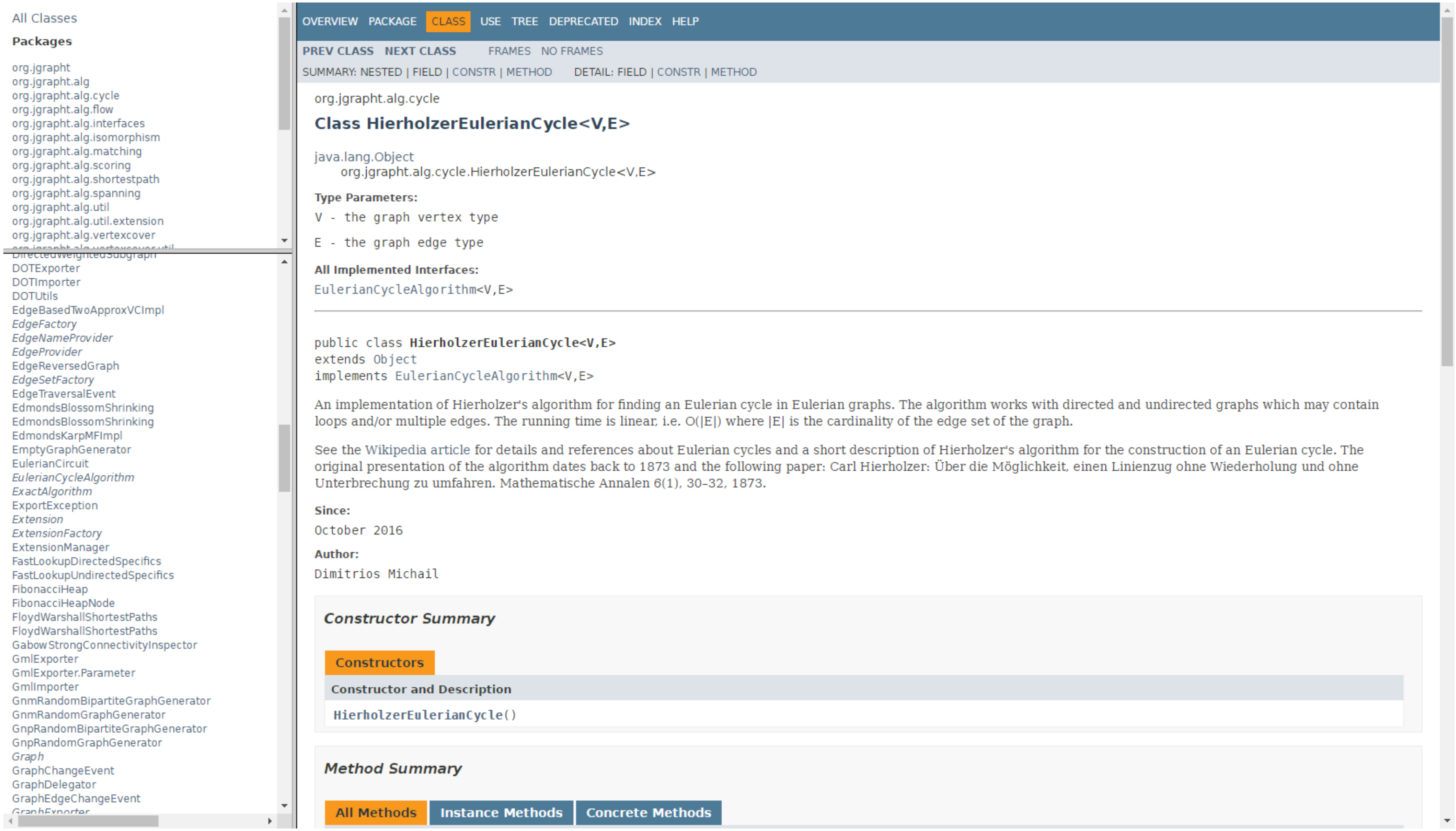


R_4: e, g, **h, f, e, i, h**, d, b, a, c, e, d, c, h, j, e

R_3: e, g, h, **d, b, a, c, e, d**, c, h, j, e

Q_3: h, f, e, i, h

Eulerian Circuits in JGraphT



org.jgrapht
org.jgrapht.alg
org.jgrapht.alg.cycle
org.jgrapht.alg.flow
org.jgrapht.alg.interfaces
org.jgrapht.alg.isomorphism
org.jgrapht.alg.matching
org.jgrapht.alg.scoring
org.jgrapht.alg.shortestpath
org.jgrapht.alg.spanning
org.jgrapht.alg.util
org.jgrapht.alg.util.extension
org.jgrapht.alg.vertexcover

DOTExporter
DOTImporter
DOTUtils
EdgeBasedTwoApproxVCImpl
EdgeFactory
EdgeNameProvider
EdgeProvider
EdgeReversedGraph
EdgeSetFactory
EdgeTraversalEvent
EdmondsBlossomShrinking
EdmondsBlossomShrinking
EdmondsKarpMFImpl
EmptyGraphGenerator
EulerianCircuit
EulerianCycleAlgorithm
ExactAlgorithm
ExportException
Extension
ExtensionFactory
ExtensionManager
FastLookupDirectedSpecifics
FastLookupUndirectedSpecifics
FibonacciHeap
FibonacciHeapNode
FloydWarshallShortestPaths
FloydWarshallShortestPaths
GabowStrongConnectivityInspector
GmlExporter
GmlExporter.Parameter
GmlImporter
GnmRandomBipartiteGraphGenerator
GnmRandomGraphGenerator
GnpRandomBipartiteGraphGenerator
GnpRandomGraphGenerator
Graph
GraphChangeEvent
GraphDelegator
GraphEdgeChangeEvent

org.jgrapht.alg.cycle

Class HierholzerEulerianCycle<V,E>

java.lang.Object
    org.jgrapht.alg.cycle.HierholzerEulerianCycle<V,E>

Type Parameters:

`V - the graph vertex type`

`E - the graph edge type`

All Implemented Interfaces:

`EulerianCycleAlgorithm<V,E>`

```
public class HierholzerEulerianCycle<V,E>
extends Object
implements EulerianCycleAlgorithm<V,E>
```

An implementation of Hierholzer's algorithm for finding an Eulerian cycle in Eulerian graphs. The algorithm works with directed and undirected graphs which may contain loops and/or multiple edges. The running time is linear, i.e. O(|E|) where |E| is the cardinality of the edge set of the graph.

See the Wikipedia article for details and references about Eulerian cycles and a short description of Hierholzer's algorithm for the construction of an Eulerian cycle. The original presentation of the algorithm dates back to 1873 and the following paper: Carl Hierholzer: Über die Möglichkeit, einen Linienzug ohne Wiederholung und ohne Unterbrechung zu umfahren. Mathematische Annalen 6(1), 30–32, 1873.

Since:

`October 2016`

Author:

`Dimitrios Michail`

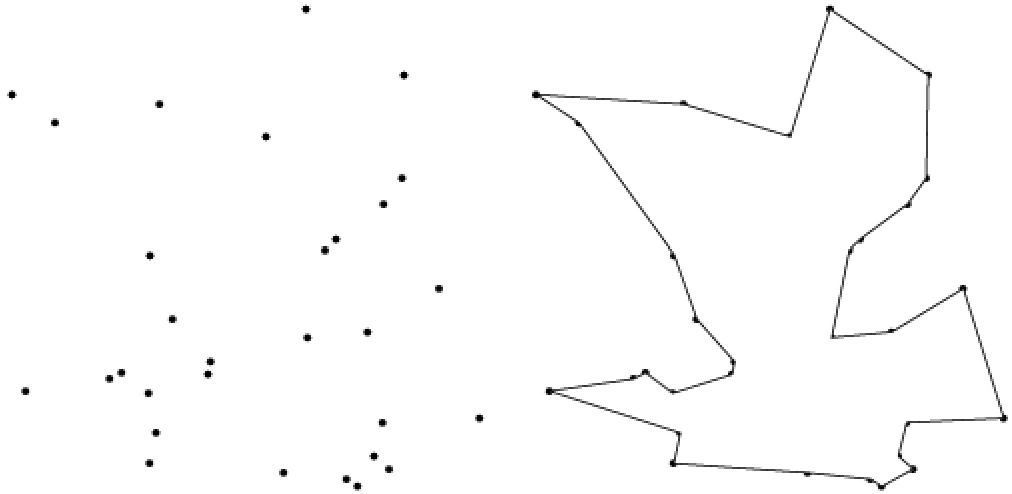
Constructor Summary

Constructor and Description
`HierholzerEulerianCycle()`

Method Summary

Hamiltonian Cycles

- There are theorems to identify **whether** a graph is Hamiltonian (i.e., whether it contains at least one Hamiltonian Cycle)
- **Finding** such a cycle has **no** known efficient solution, in the general case
- Example: the **Traveling Salesman Problem** (TSP)



The Traveling Salesman Problem (TSP)

Weighted or unweighted

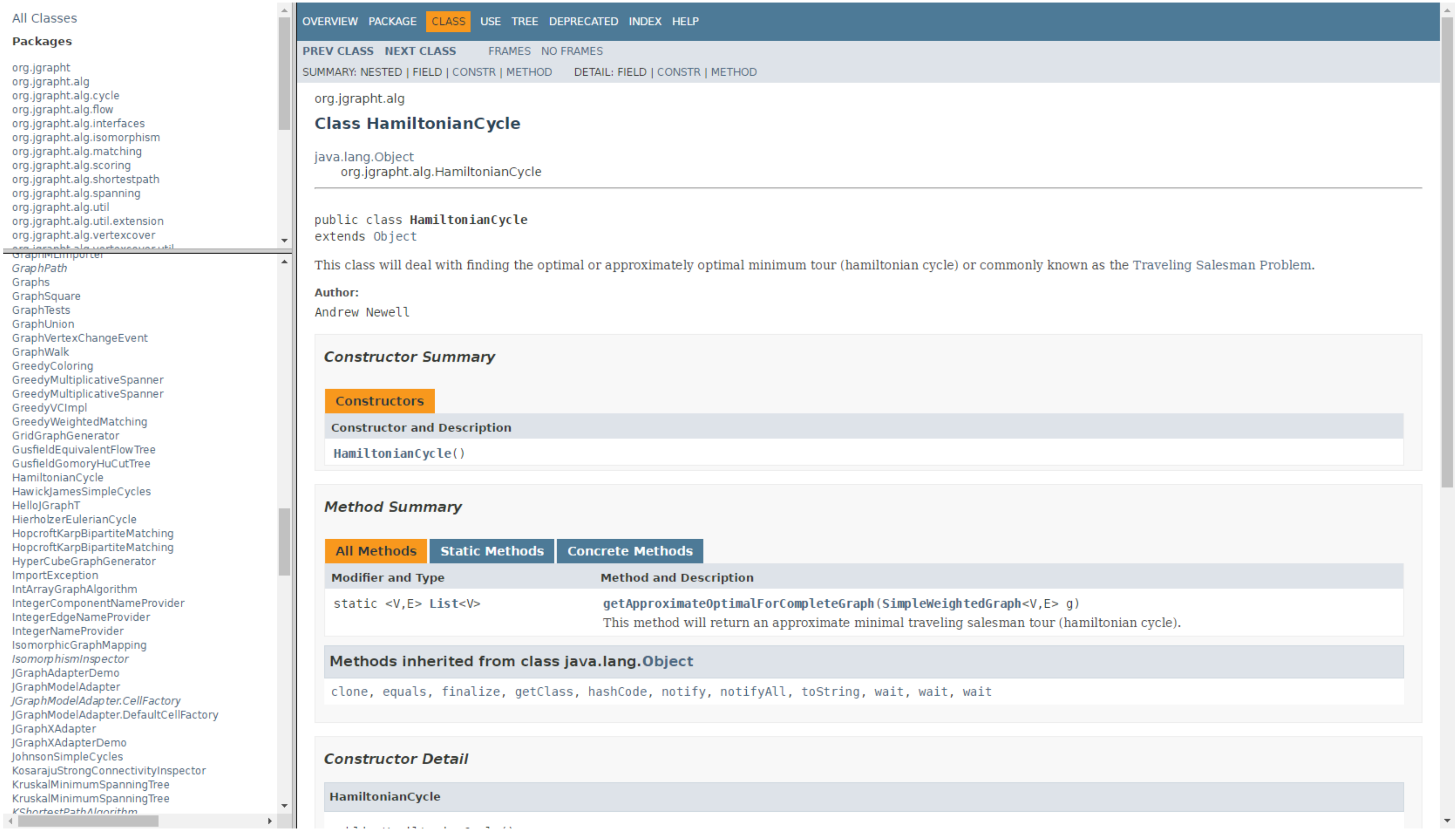
Given a collection of cities connected by roads

Find the shortest route that visits each city exactly once.

About TSP

- Most notorious NP-complete problem.
- Typically, it is solved with a backtracking algorithm:
  - The best tour found to date is saved.
  - The search backtracks unless the partial solution is cheaper than the cost of the best tour.

Hamiltonian Cycles in JGraphT



org.jgrapht
org.jgrapht.alg
org.jgrapht.alg.cycle
org.jgrapht.alg.flow
org.jgrapht.alg.interfaces
org.jgrapht.alg.isomorphism
org.jgrapht.alg.matching
org.jgrapht.alg.scoring
org.jgrapht.alg.shortestpath
org.jgrapht.alg.spanning
org.jgrapht.alg.util
org.jgrapht.alg.util.extension
org.jgrapht.alg.vertexcover

GraphPath
Graphs
GraphSquare
GraphTests
GraphUnion
GraphVertexChangeEvent
GraphWalk
GreedyColoring
GreedyMultiplicativeSpanner
GreedyMultiplicativeSpanner
GreedyVCImpl
GreedyWeightedMatching
GridGraphGenerator
GusfieldEquivalentFlowTree
GusfieldGomoryHuCutTree
HamiltonianCycle
HawickJamesSimpleCycles
HelloJGraphT
HierholzerEulerianCycle
HopcroftKarpBipartiteMatching
HopcroftKarpBipartiteMatching
HyperCubeGraphGenerator
ImportException
IntArrayGraphAlgorithm
IntegerComponentNameProvider
IntegerEdgeNameProvider
IntegerNameProvider
IsomorphicGraphMapping
IsomorphismInspector
JGraphAdapterDemo
JGraphModelAdapter
JGraphModelAdapter.CellFactory
JGraphModelAdapter.DefaultCellFactory
JGraphXAdapter
JGraphXAdapterDemo
JohnsonSimpleCycles
KosarajuStrongConnectivityInspector
KruskalMinimumSpanningTree
KruskalMinimumSpanningTree

org.jgrapht.alg

Class HamiltonianCycle

java.lang.Object
    org.jgrapht.alg.HamiltonianCycle

```
public class HamiltonianCycle
extends Object
```

This class will deal with finding the optimal or approximately optimal minimum tour (hamiltonian cycle) or commonly known as the Traveling Salesman Problem.

Author:

Andrew Newell

Constructor Summary

Constructors

Constructor and Description
`HamiltonianCycle()`

Method Summary

All Methods **Static Methods** **Concrete Methods**

Modifier and Type	Method and Description
`static <V,E> List<V>`	`getApproximateOptimalForCompleteGraph(SimpleWeightedGraph<V,E> g)` This method will return an approximate minimal traveling salesman tour (hamiltonian cycle).

Methods inherited from class java.lang.Object

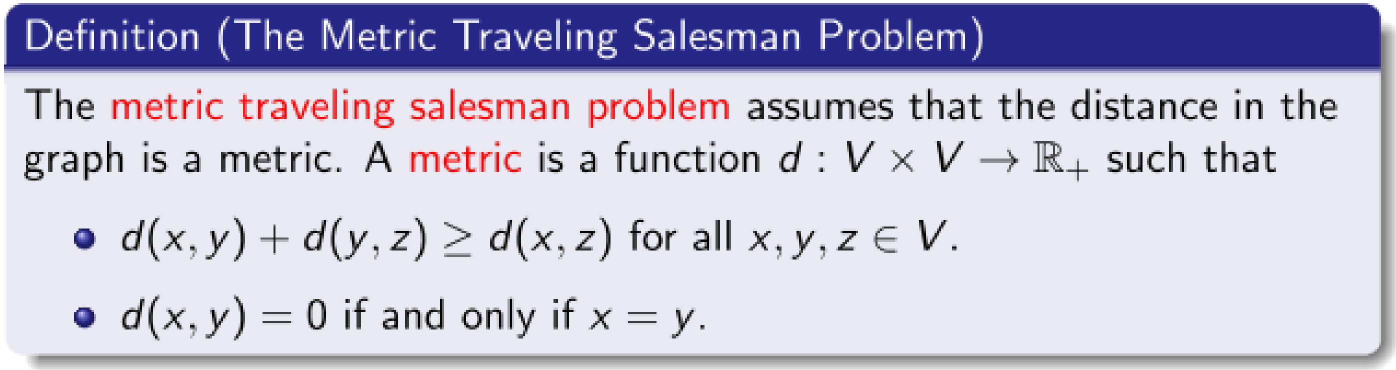
`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Constructor Detail

HamiltonianCycle

Limitations…

- No exact solution:
  - **`getApproximateOptimalForCompleteGraph`** `(SimpleWeightedGraph<V,E> g)`
- But…
  - g must be a **complete** graph
  - g must satisfy the "triangle inequality": d(x,y)+d(y,z)<d(x,z)
  - The cycle length is less than or equal to double the total weight of the optimal hamiltonian cycle



Definition (The Metric Traveling Salesman Problem)

The metric traveling salesman problem assumes that the distance in the graph is a metric. A metric is a function $d : V \times V \to \mathbb{R}_+$ such that

- $d(x,y) + d(y,z) \geq d(x,z)$ for all $x, y, z \in V$.
- $d(x,y) = 0$ if and only if $x = y$.

The Metric Traveling Salesman Problem

An approximation algorithm

ASSUMPTION: G is a metric graph.

1. Compute a minimum weight spanning tree T for G.
2. Perform a depth-first traversal of T starting from any node, and order the nodes of G as they were discovered in this traversal.

$\Rightarrow$ a tour that is at most twice the optimal tour in G.

Resources

- http://mathworld.wolfram.com/
- http://en.wikipedia.org/wiki/Euler_cycle
- Mircea MARIN, Graph Theory and Combinatorics, Lectures 9 and 10, http://web.info.uvt.ro/~mmarin/

Licenza d'uso

- Queste diapositive sono distribuite con licenza Creative Commons "Attribuzione - Non commerciale - Condividi allo stesso modo (CC BY-NC-SA)"
- Sei libero:
  - di riprodurre, distribuire, comunicare al pubblico, esporre in pubblico, rappresentare, eseguire e recitare quest'opera
  - di modificare quest'opera
- Alle seguenti condizioni:
  - Attribuzione — Devi attribuire la paternità dell'opera agli autori origina e in modo tale da non suggerire che essi avallino te o il modo in cui tu usi l'opera.
  - Non commerciale — Non puoi usare quest'opera per fini commerciali.
  - Condividi allo stesso modo — Se alteri o trasformi quest'opera, o se la usi per crearne un'altra, puoi distribuire l'opera risultante solo con una licenza identica o equivalente a questa.
- http://creativecommons.org/licenses/by-nc-sa/3.0/