POLITECNICO DI TORINO

e-Lite

Representing and visiting graphs

Tecniche di Programmazione – A.A. 2013/2014

Summary

- Graph representation
- The JGraphT library
- Graph visits
- Visits in JGraphT

Graph representation

Representing and visiting graphs

Representing graphs

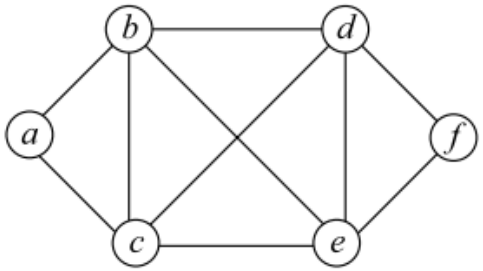
List structures

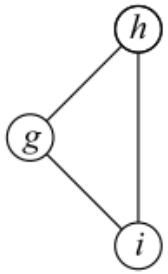
- **Adjacency list**
  - Each vertex has a list of which vertices it is adjacent to.
  - For undirected graphs, information is duplicated
- Incidence list
  - Each vertex has a list of 'edge' objects
  - Edges are represented by a pair (a tuple if directed) of vertices (that the edge connects) and possibly weight and other data.

Matrix structures

- **Adjacency matrix**
  - A = |V| x |V| matrix of Booleans or integers
  - If there is an edge from a vertex v to a vertex v', then the element A[v,v'] is 1, otherwise 0.
- Incidence matrix
  - IM = |V| x |E| matrix of integers
  - IM[v,e] = 1 (incident), 0 (not incident)
  - For directed graphs, may be -1 (out) and +1 (in)

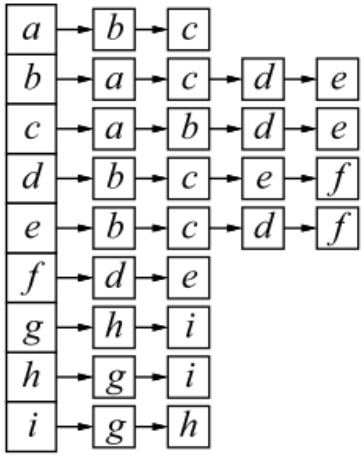
Example





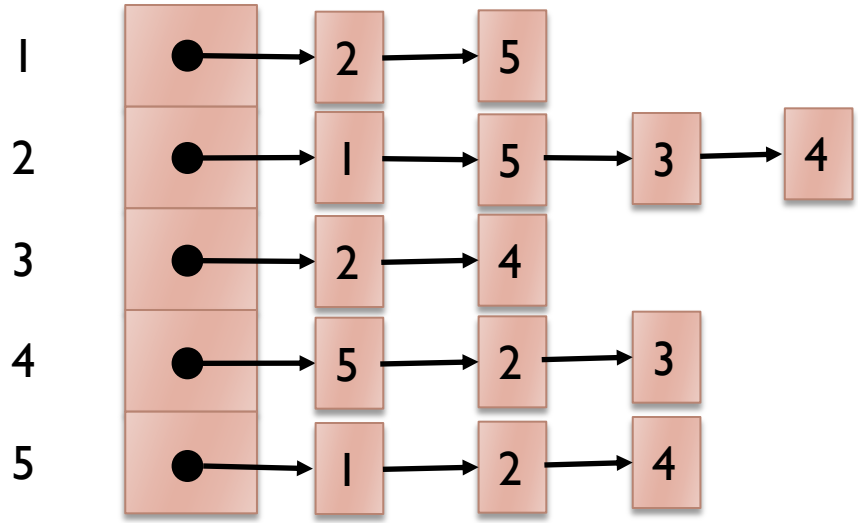
Adjacency matrix

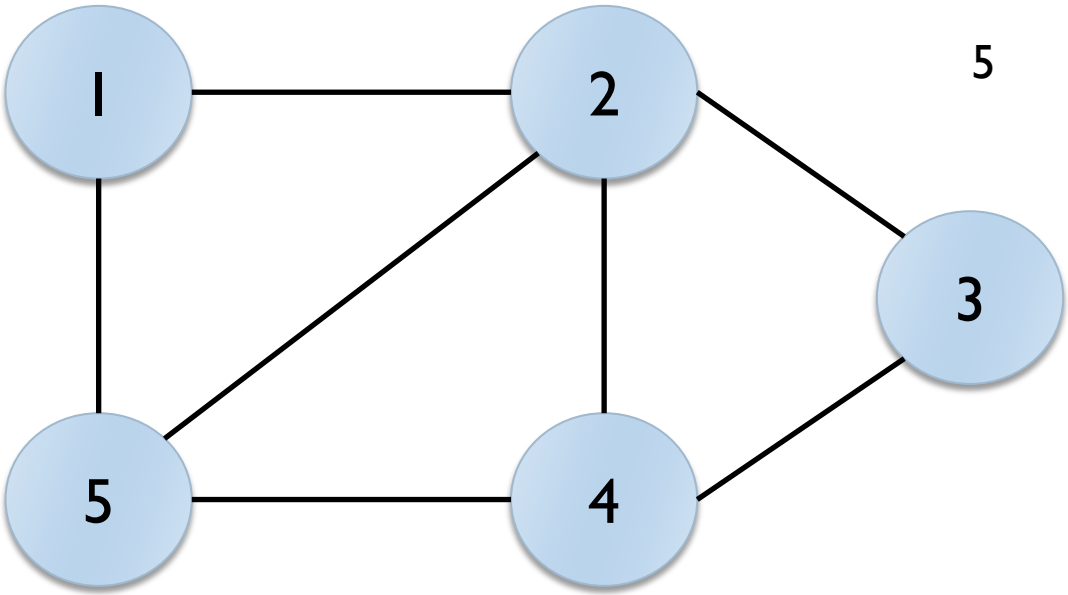
	a	b	c	d	e	f	g	h	i
a	0	1	1	0	0	0	0	0	0
b	1	0	1	1	1	0	0	0	0
c	1	1	0	1	1	0	0	0	0
d	0	1	1	0	1	1	0	0	0
e	0	1	1	1	0	1	0	0	0
f	0	0	0	1	1	0	0	0	0
g	0	0	0	0	0	0	0	1	0
h	0	0	0	0	0	0	1	0	1
i	0	0	0	0	0	0	1	1	0



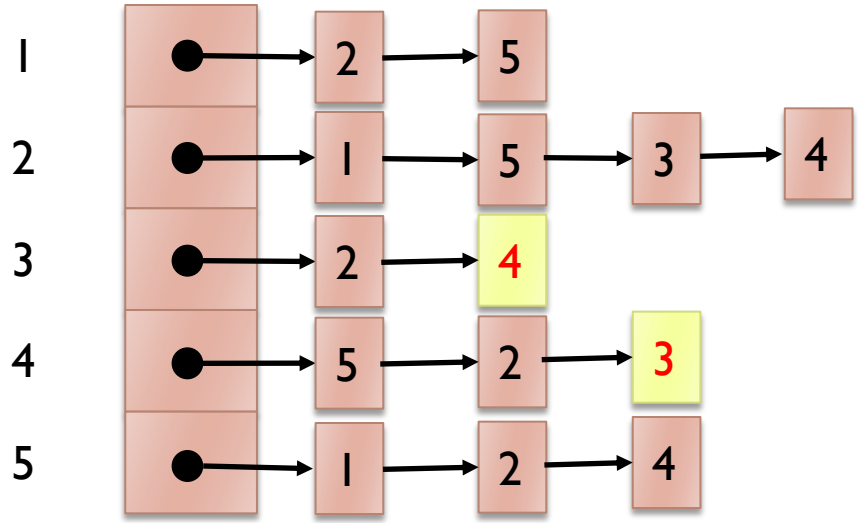
Adjacency list

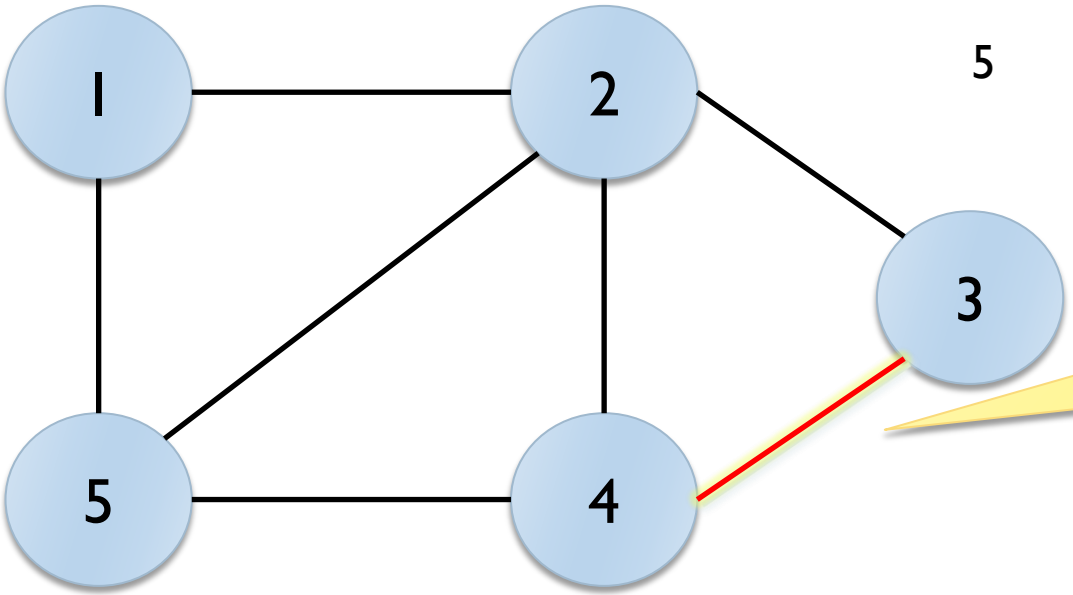
Adjacency list (undirected graph)





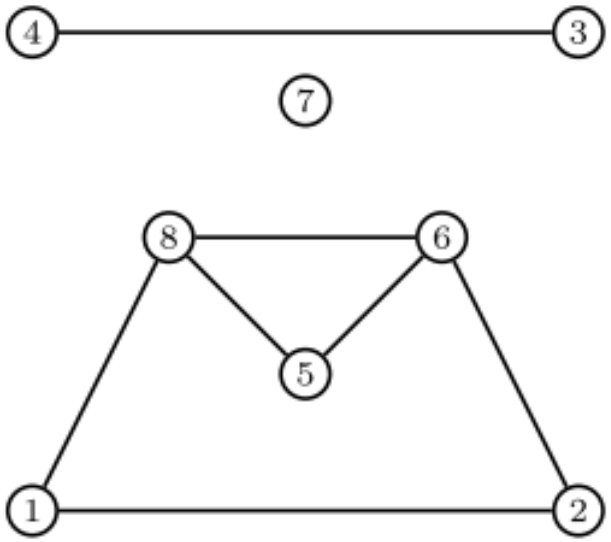
Adjacency list (undirected graph)





Undirected ==>
All edges are
represented twice

Adjacency list (un-connected graph)



$L_1 = [2, 8]$

$L_2 = [1, 6]$

$L_3 = [4]$

$L_4 = [3]$

$L_5 = [6, 8]$

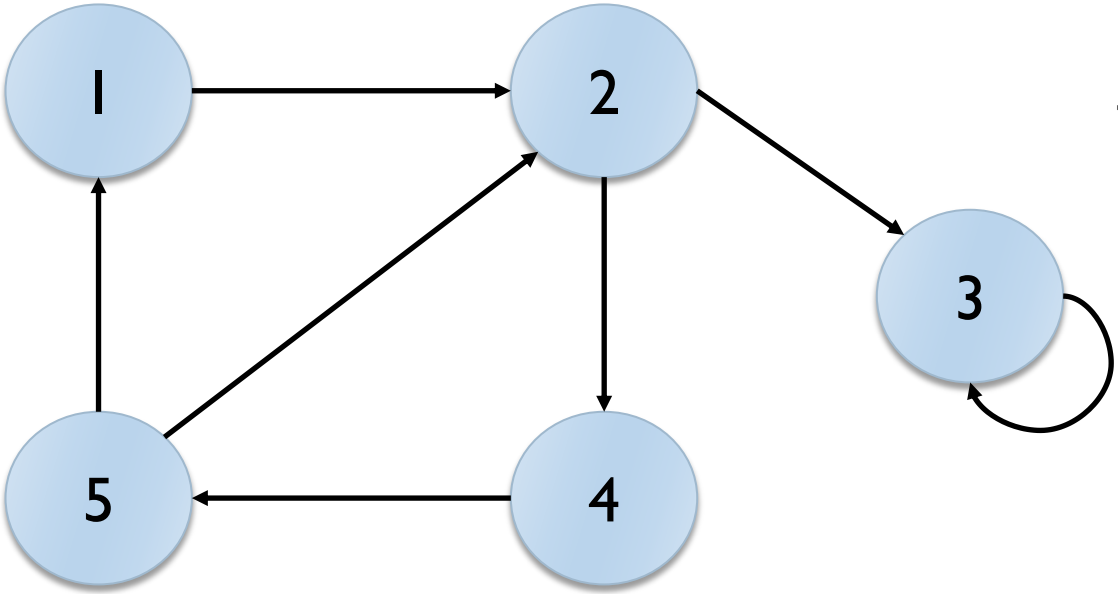
$L_6 = [2, 5, 8]$

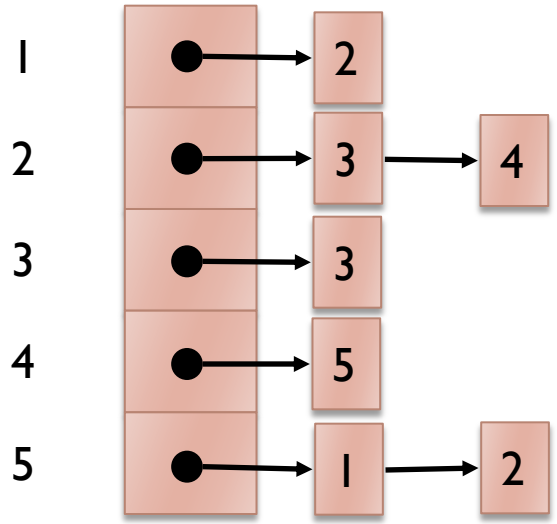
$L_7 = [\,]$

$L_8 = [1, 5, 6]$

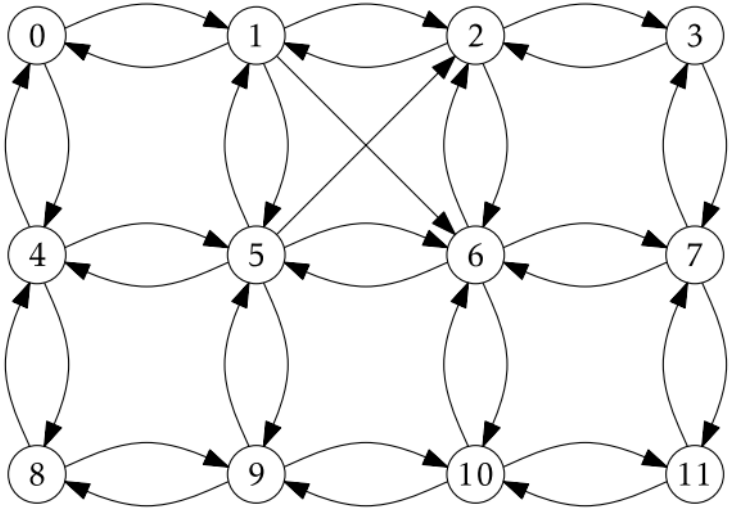
Un-connected graph, same rules

Adjacency list (directed graph)



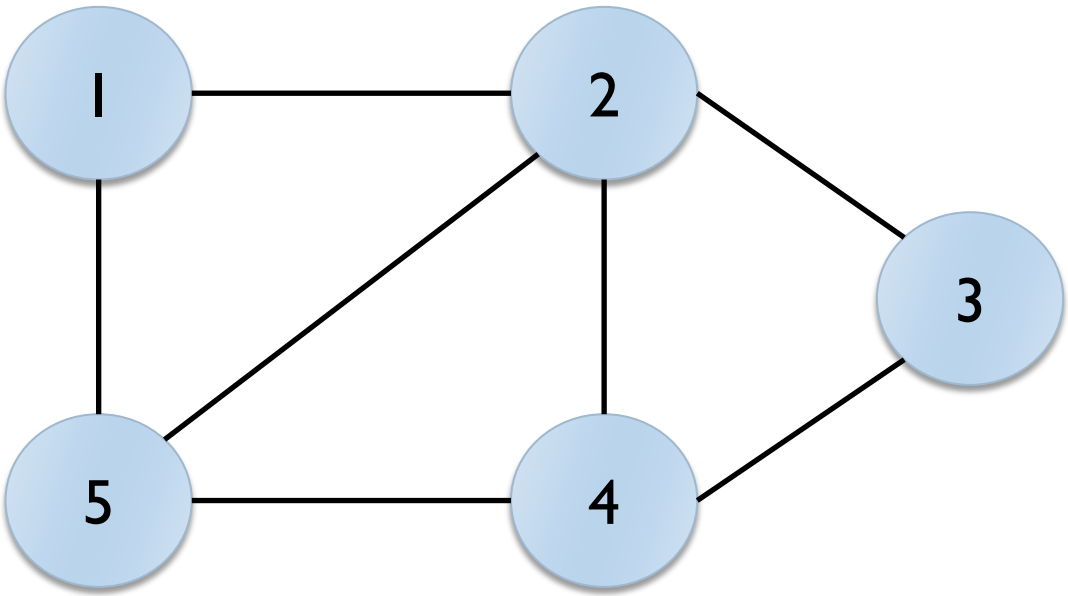


Adjacency list



0	1	2	3	4	5	6	7	8	9	10	11
1	0	1	2	0	1	5	6	4	8	9	10
4	2	3	7	5	2	2	3	9	5	6	7
	6	6		8	6	7	11		10	11	
	5				9	10					
					4						

Adjacency matrix (undirected graph)

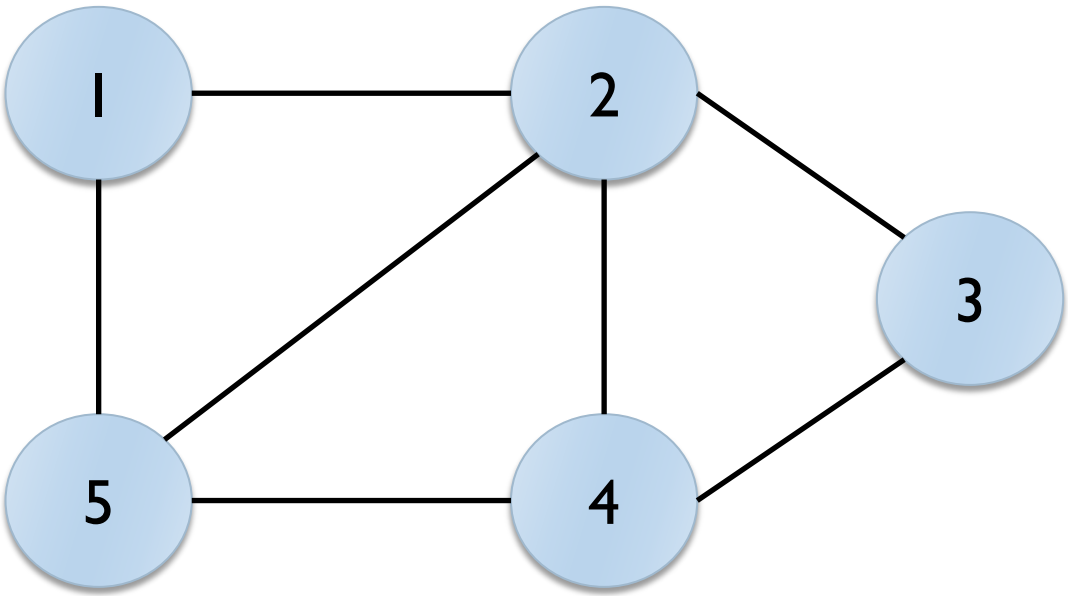


	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Undirected => symmetric matrix

No self-loops: zero diagonal

Adjacency matrix (undirected graph)



	1	2	3	4	5
1		1	0	0	1
2			1	1	1
3				1	0
4					1
5					

Undirected => symmetric matrix

50% of memory can be saved

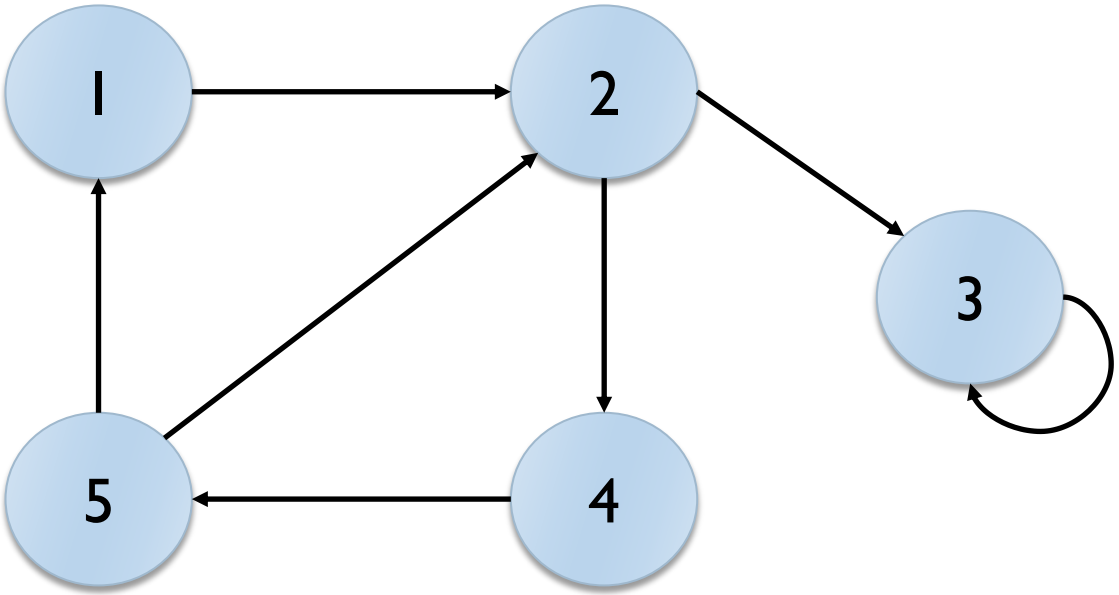
Adjacency matrix (directed graph)

To-vertex
(in-edges)

From-vertex
(out-edges)

	1	2	3	4	5
1	0	1	0	0	0
2	0	0	1	1	0
3	0	0	1	0	0
4	0	0	0	0	1
5	1	1	0	0	0

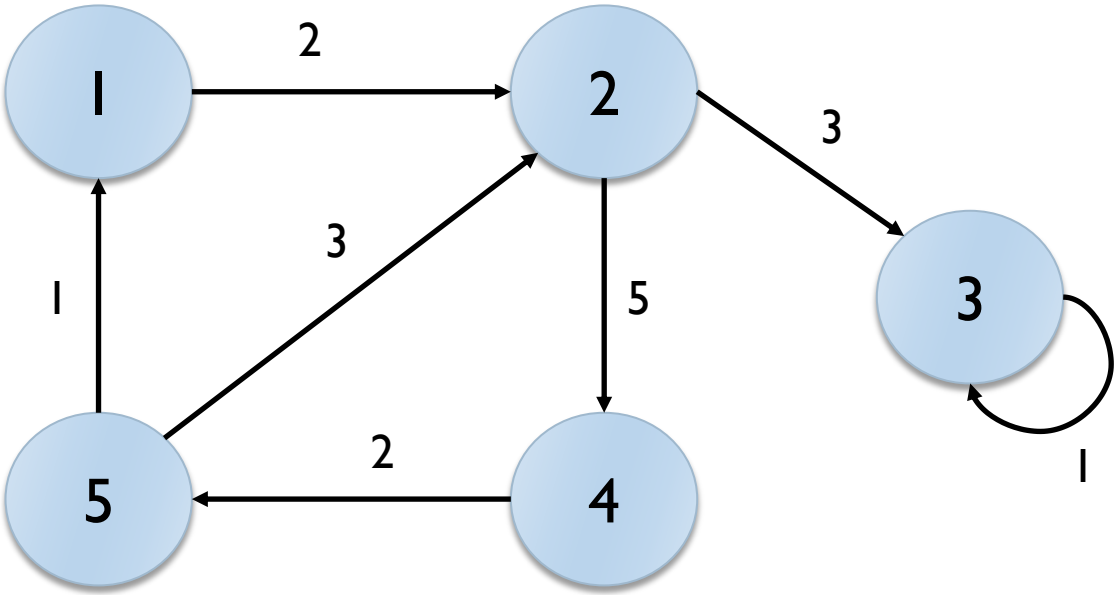
Self-loops in the diagonal



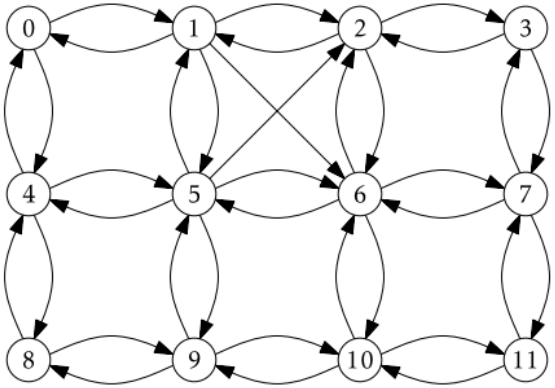
Adjacency matrix (weighted graph)

Values = edge weight

	1	2	3	4	5
1	0	2	0	0	0
2	0	0	3	5	0
3	0	0	1	0	0
4	0	0	0	0	2
5	1	3	0	0	0

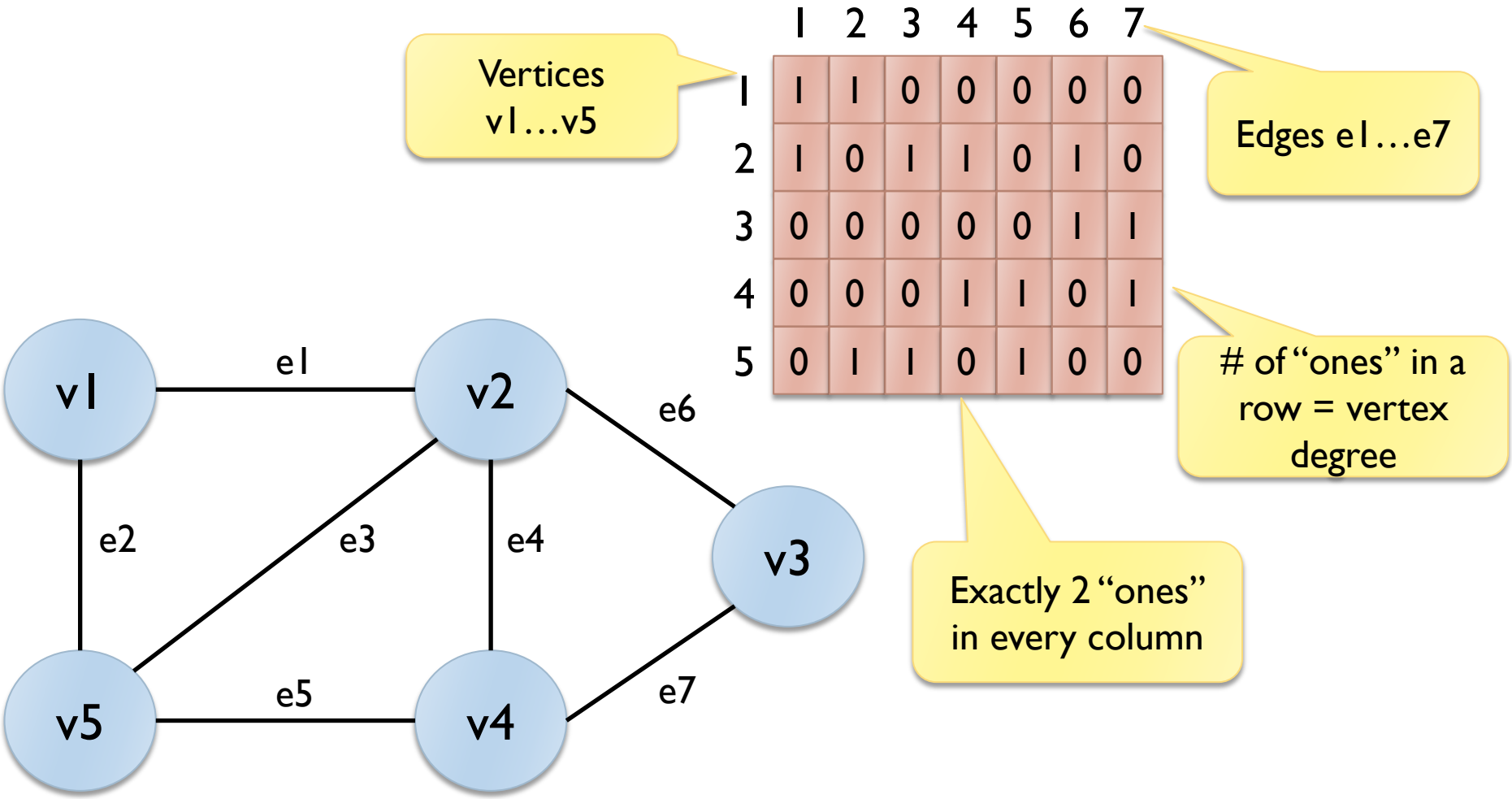


Adjacency matrix

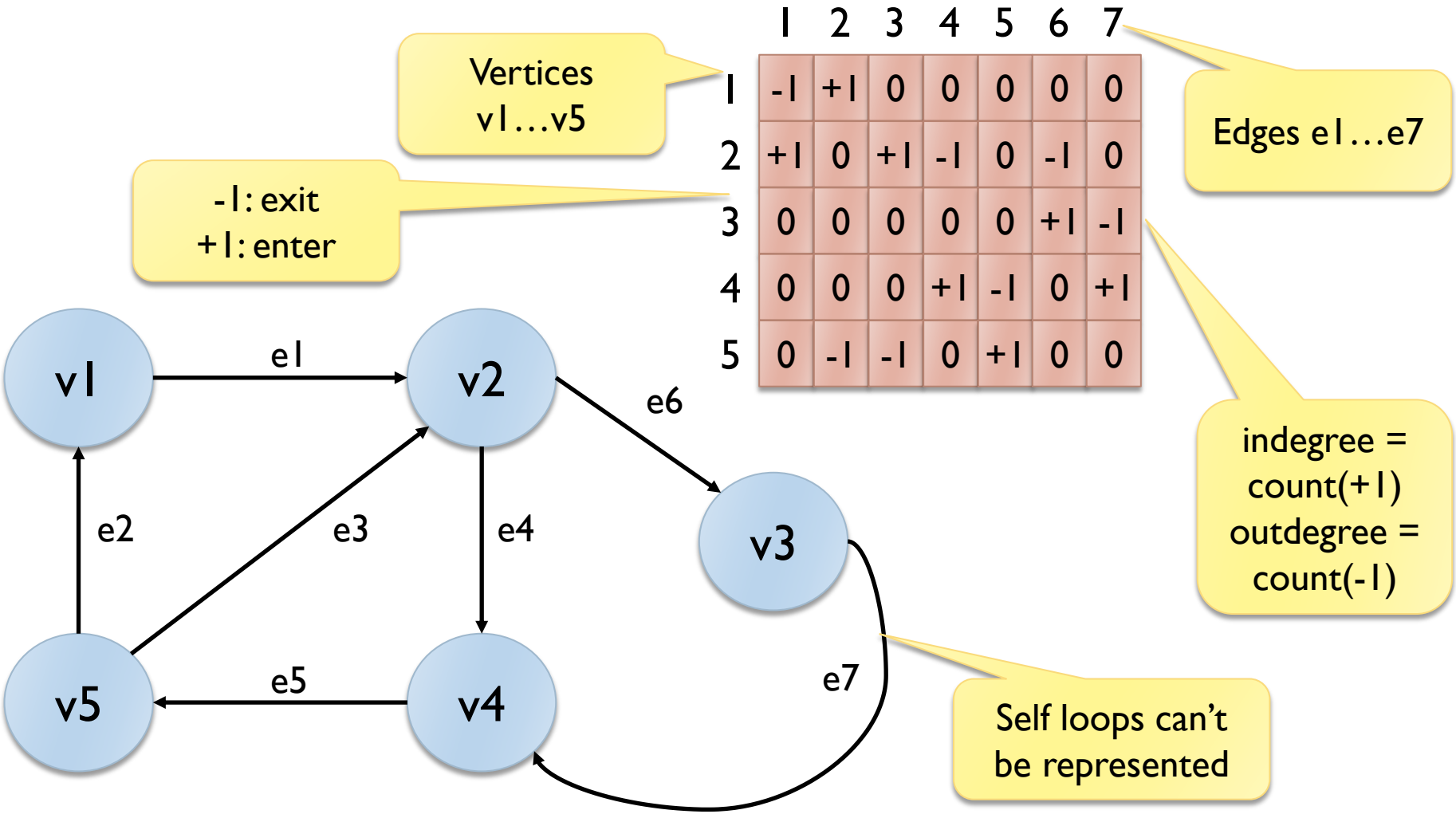


	0	1	2	3	4	5	6	7	8	9	10	11
0	0	1	0	0	1	0	0	0	0	0	0	0
1	1	0	1	0	0	1	1	0	0	0	0	0
2	1	0	0	1	0	0	1	0	0	0	0	0
3	0	0	1	0	0	0	0	1	0	0	0	0
4	1	0	0	0	0	1	0	0	1	0	0	0
5	0	1	1	0	1	0	1	0	0	1	0	0
6	0	0	1	0	0	1	0	1	0	0	1	0
7	0	0	0	1	0	0	1	0	0	0	0	1
8	0	0	0	0	1	0	0	0	0	1	0	0
9	0	0	0	0	0	1	0	0	1	0	1	0
10	0	0	0	0	0	0	1	0	0	1	0	1
11	0	0	0	0	0	0	0	1	0	0	1	0

Incidence matrix (undirected graph)



Incidence matrix (directed graph)



Complexity & trade-offs

	Adjacency List	Adjacency Matrix	Incidence Matrix
Space	O(|V| + |E|)	O(|V| x |V|)	O(|V| x |E|)
Space	For sparse graphs $\|E\| << \|V\|^2$	For dense graphs $\|E\| \sim \|V\|^2$	
Check edge	O(1+deg(v))	O(1)	O(|E|) if *v,v'* are known, or O(1) if *e* is known
Find all adjacent	O(1+deg(v))	O(|V|)	O(|V|+|E|)

The JGraphT library

Representing and visiting graphs

JGraphT

- http://jgrapht.org
  - (do not confuse with jgraph.com)
- Free Java graph library that provides graph objects and algorithms
- Easy, type-safe and extensible thanks to <generics>
- Just add **`jgrapht-core-0.9.0.jar`** to your project

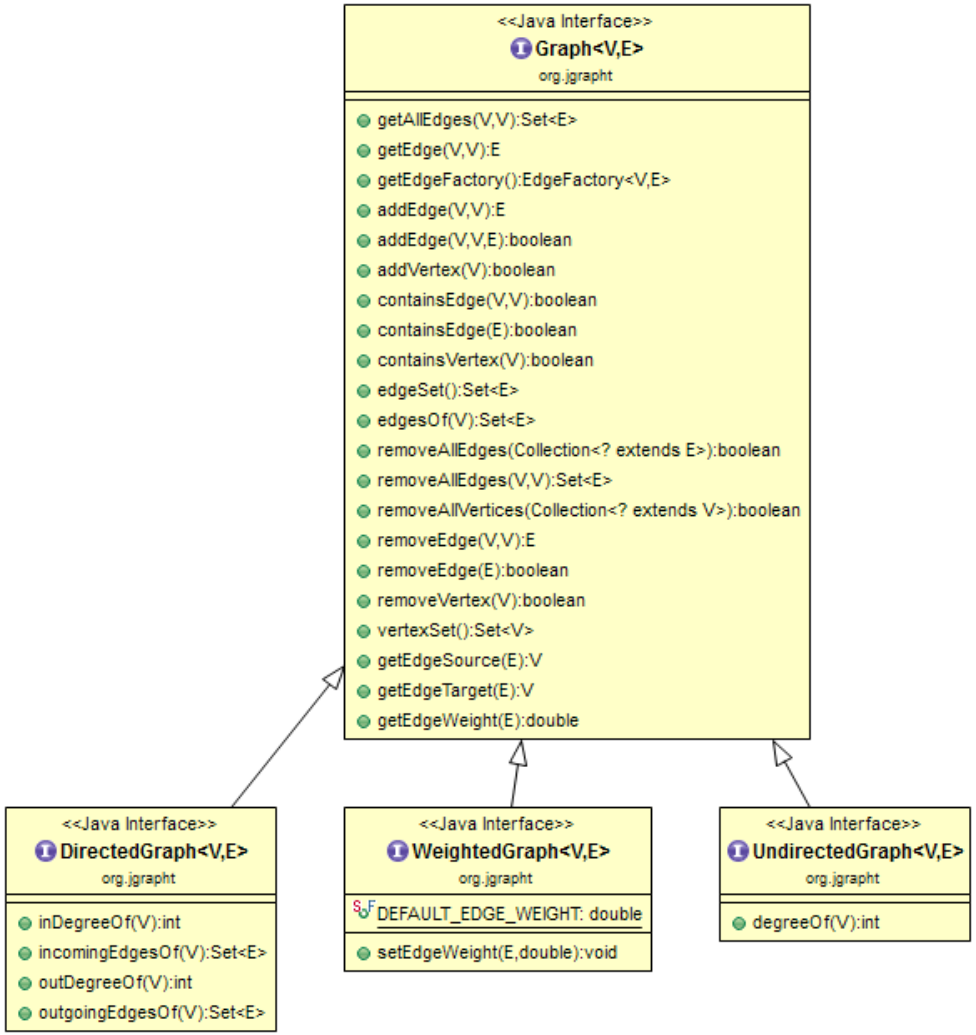
JGraphT structure

Packages	
org.jgrapht	The front-end API's interfaces and classes, including Graph, DirectedGraph and UndirectedGraph.
org.jgrapht.alg	Algorithms provided with JGraphT.
org.jgrapht.alg.util	Utilities used by JGraphT algorithms.
org.jgrapht.demo	Demo programs that help to get started with JGraphT.
org.jgrapht.event	Event classes and listener interfaces, used to provide a change notification mechanism on graph modification events.
org.jgrapht.ext	Extensions and integration means to other products.
org.jgrapht.generate	Generators for graphs of various topologies.
org.jgrapht.graph	Implementations of various graphs.
org.jgrapht.traverse	Graph traversal means.
org.jgrapht.util	Non-graph-specific data structures, algorithms, and utilities used by JGraphT.

Graph objects

- All graphs derive from
  - Interface `Graph<V,E>`
  - V = type of vertices
  - E = type of edges
    - usually `DefaultEdge` or `DefaultWeightedEdge`
- Main interfaces
  - `DirectedGraph<V,E>`
  - `UndirectedGraph<V,E>`
  - `WeightedGraph<V,E>`

JGraphT main interfaces



Graph classes

org.jgrapht

Graph (I)

DirectedGraph (I)

UndirectedGraph (I)

WeightedGraph (I)

org.jgrapht.graphs

DefaultDirectedGraph

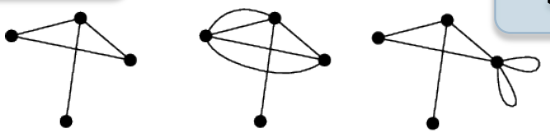
DefaultDirectedWeightedGraph

SimpleGraph

SimpleWeightedGraph

SimpleDirectedGraph

SimpleDirectedWeightedGraph



simple graph *multigraph* *pseudograph*

DirectedMultigraph

Multigraph

DirectedWeightedMultigraph

WeightedMultigraph

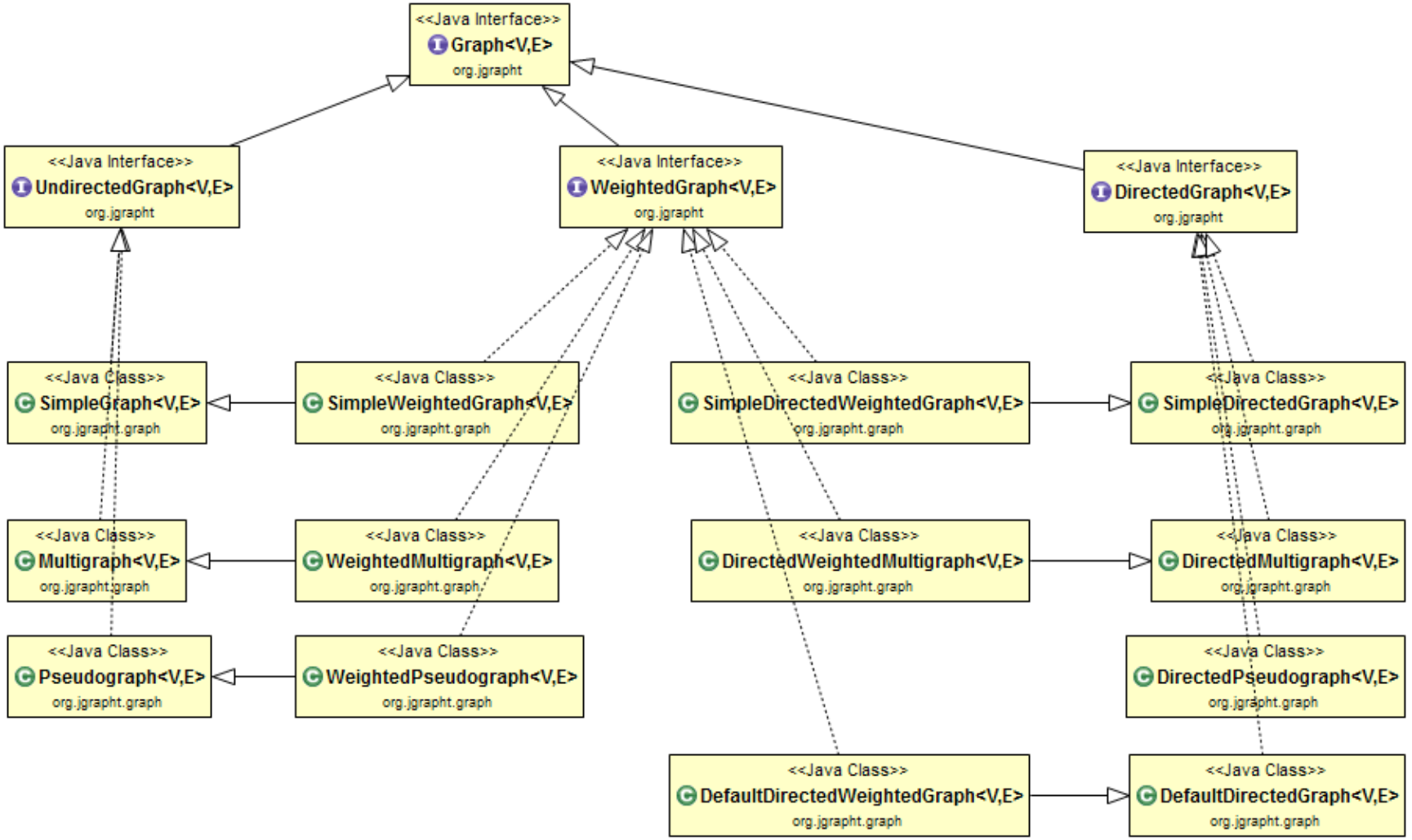
DirectedPseudograph

DirectedWeightedPseudograph

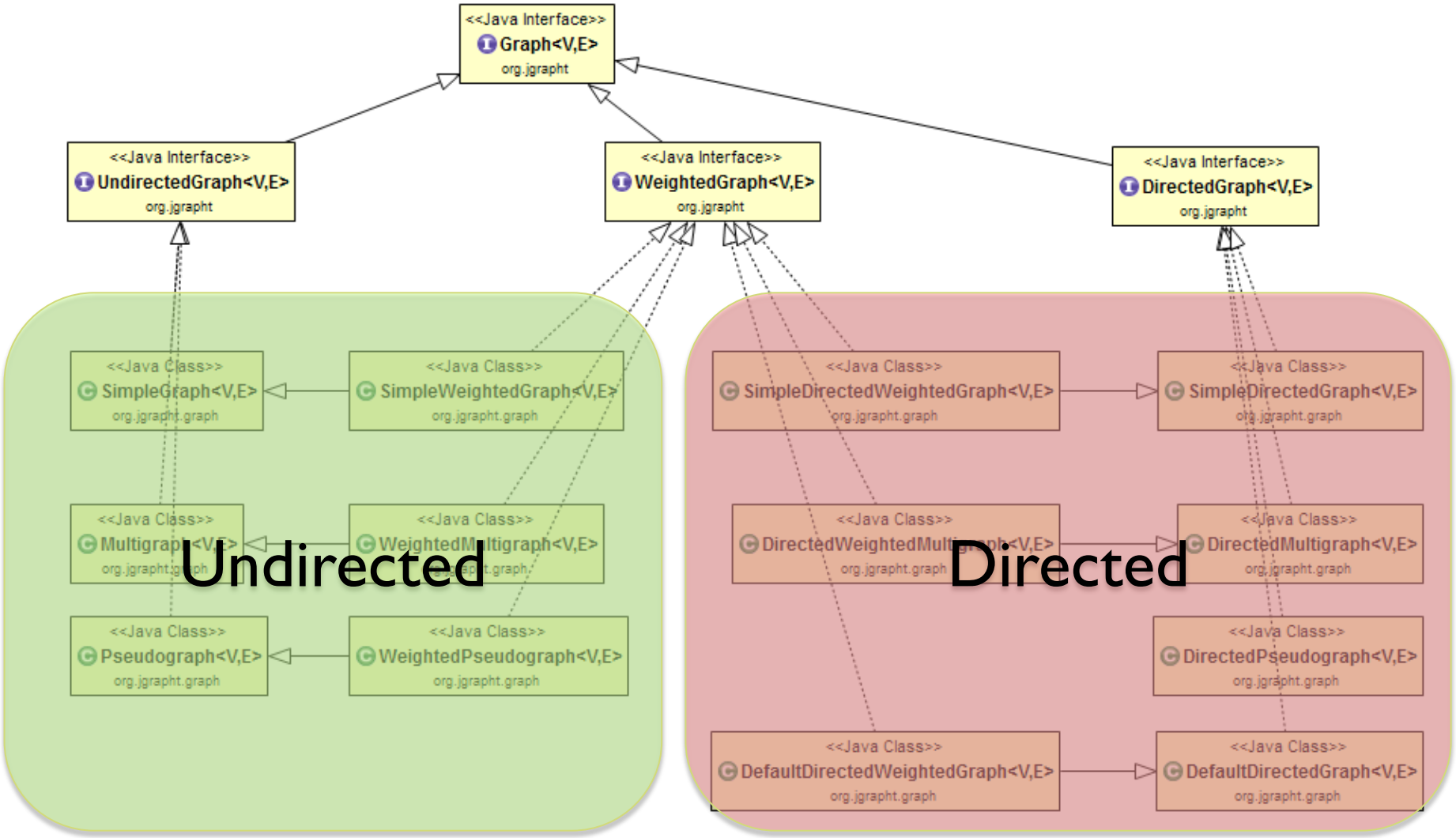
Pseudograph

WeightedPseudograph

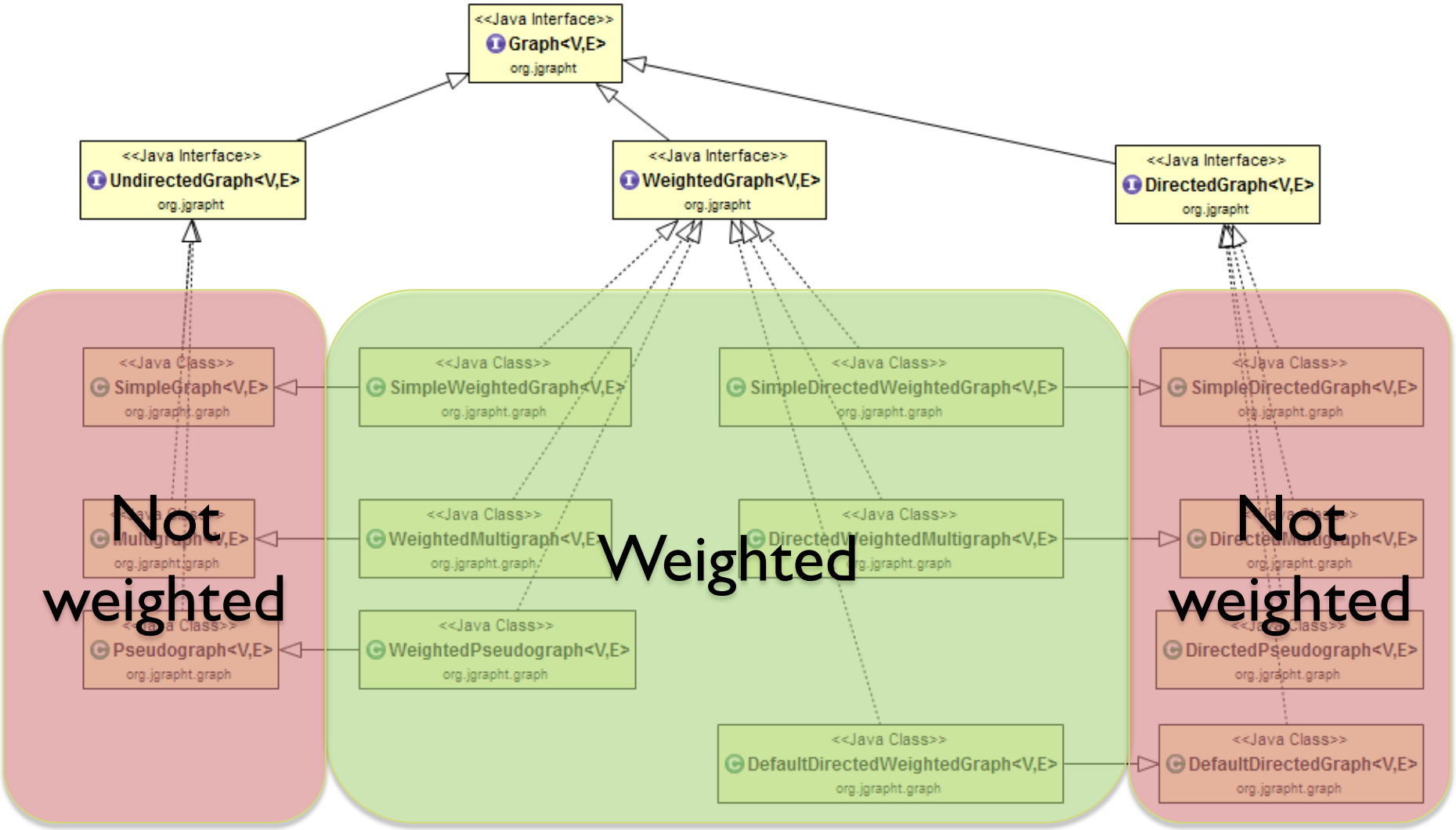
Graph classes



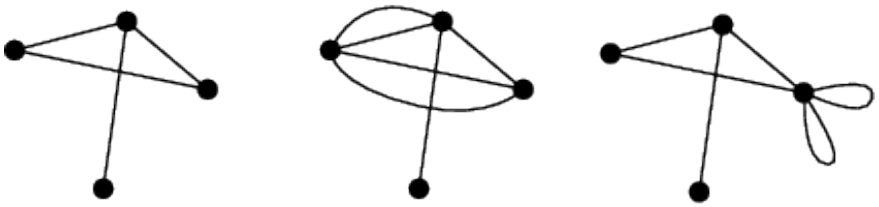
Graph classes

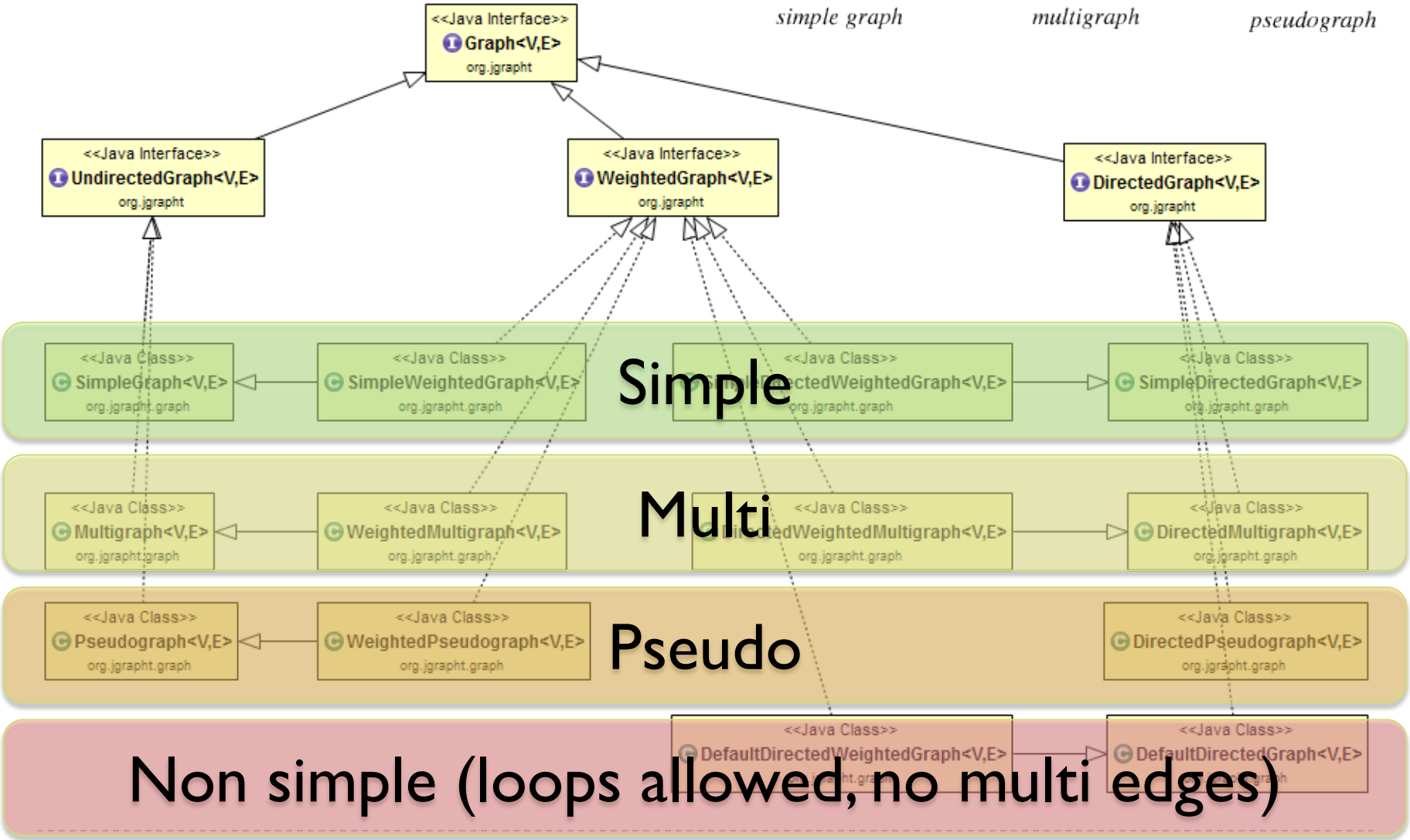


Graph classes



Graph classes



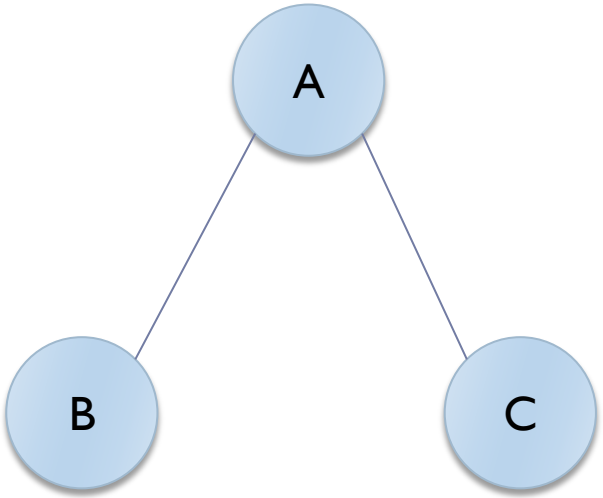


Creating graphs

- Construct your desired type of graph
- Add vertices
  - boolean **addVertex**(V v)
- Add edges
  - E **addEdge**(V sourceVertex,V targetVertex)
  - boolean **addEdge**(V sourceVertex,V targetVertex, E e)
  - void **setEdgeWeight**(E e, double weight)
- Print graph (for debugging)
  - toString()
- Warning: E and V should correctly implement .equals() and .hashCode()

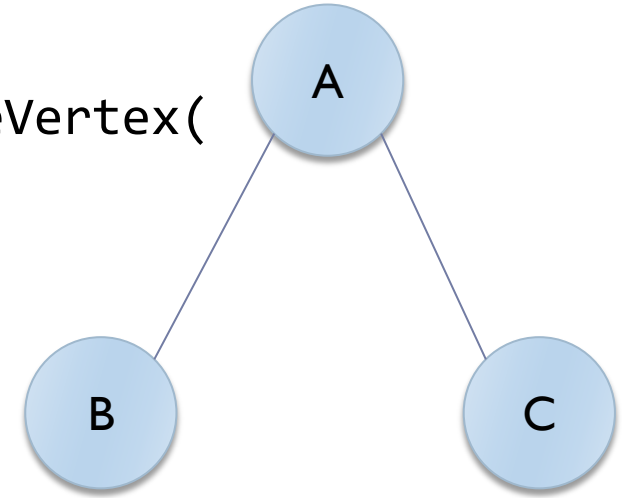
Example

```
UndirectedGraph<String, DefaultEdge> graph = new
SimpleGraph<>(DefaultEdge.class) ;

graph.addVertex("A") ;
graph.addVertex("B") ;
graph.addVertex("C") ;

graph.addEdge("A", "B") ;
graph.addEdge("A", "C") ;
```

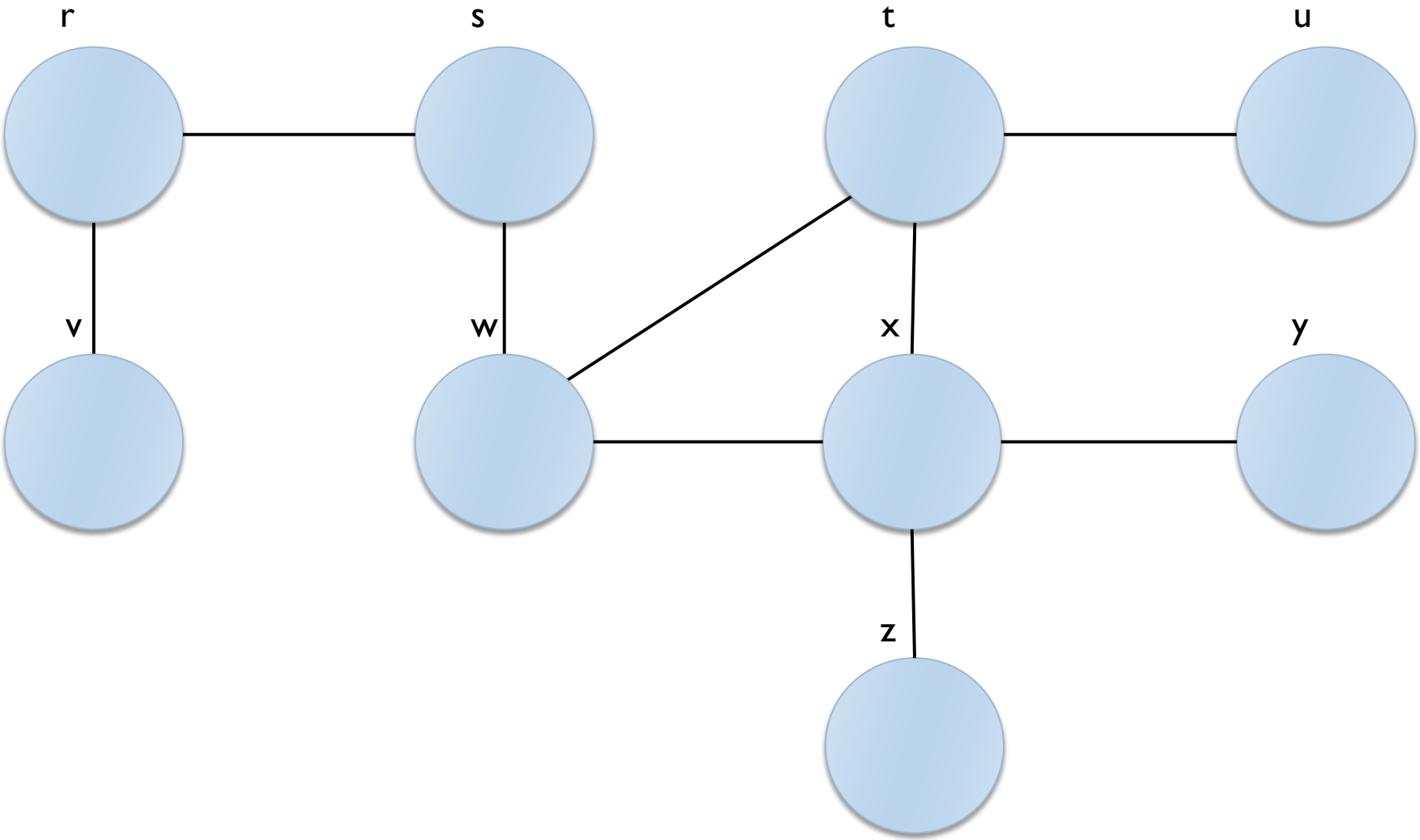


Example

```
for( String s: graph.vertexSet() ) {
        System.out.println("Vertex "+s) ;
        for( DefaultEdge e: graph.edgesOf(s) ) {
                System.out.println("Degree: “
                        +graph.degreeOf(s)) ;
                System.out.println(
                        Graphs.getOppositeVertex(
                        graph, e, s)) ;
        }
}
```



Example



For testing…

Package org.jgrapht.generate

Generators for graphs of various topologies.

See:

Description

Interface Summary	
GraphGenerator<V,E,T>	GraphGenerator defines an interface for generating new graph structures.
RandomGraphGenerator.EdgeTopologyFactory<VV,EE>	This class is used to generate the edge topology for a graph.

Class Summary	
CompleteBipartiteGraphGenerator<V,E>	Generates a complete bipartite graph of any size.
CompleteGraphGenerator<V,E>	Generates a complete graph of any size.
EmptyGraphGenerator<V,E>	Generates an empty graph of any size.
GridGraphGenerator<V,E>	Generates a bidirectional grid graph of any size.
HyperCubeGraphGenerator<V,E>	Generates a hyper cube graph of any size.
LinearGraphGenerator<V,E>	Generates a linear graph of any size.
RandomGraphGenerator<V,E>	This Generator creates a random-topology graph of a specified number of vertexes and edges.
RingGraphGenerator<V,E>	Generates a ring graph of any size.
ScaleFreeGraphGenerator<V,E>	Generates directed or undirected scale-free network of any size.
StarGraphGenerator<V,E>	Generates a star graph of any size.
WheelGraphGenerator<V,E>	Generates a wheel graph of any size.

Example

Presentazione del corso
Guida dello studente

Percorso Generalista
Orientamento "Information technology engineering" - Shanghai

Percorso Generalista

Top

1° anno

Periodo	Codice	Lingua	Insegnamento	Crediti	Docente	Note	Vincoli
1	16ACFOA		Analisi matematica I	10	A. Tabacco E. Serra F. Ceragioli ...		
1	15AHMOA		Chimica	8	B. Onida S. Ronchetti E. Angelini ...		
1	07LKIOA		Lingua inglese I livello	3			
2			Crediti liberi del 1 anno	6			
2	17AXOOA		Fisica I	10	M. Agnello A. Montorsi A. Gamba ...		
2	17BCGOA		Geometria	10	M. Ferrarotti C. Massaza J. Cordovez Manriquez ...		
2	12BHDOA		Informatica	8	P. Laface A. Acquaviva L. Sterpone ...		

2° anno

Periodo	Codice	Lingua	Insegnamento	Crediti	Docente	Note	Vincoli
1	02MNOOA		Algoritmi e programmazione	10	P. Camurati		Si
1	23ACIOA		Analisi matematica II	8	L. Scuderi S. Rolando		Si
1	01AULOA		Elettrotecnica	10	F. Corinto		Si
1	03AXPOA		Fisica II	6	M. Pretti		Si
2	12AGAOA		Calcolatori elettronici	8	M. Sonza Reorda		Si
2	05BQXOA		Metodi matematici per l'ingegneria	10	D. Bazzanella V. Recupero		Si
2	02NVAOA		Sistemi e tecnologie elettroniche	10	F. Bonani		Si

3° anno

Periodo	Codice	Lingua	Insegnamento	Crediti	Docente	Note	Vincoli
1	03MOAOA		Elettronica applicata e misure	10	D. Del Corso		Si
1	12CDUOA		Reti di calcolatori	8	G. Marchetto		Si
1	05CJCOA		Sistemi operativi	6	S. Quer		Si
1	01MOOOA		Teoria ed elaborazione dei segnali	10	G. Bosco		Si
1,2	26IBNOA		Prova finale	1			
1,2	11CWHOA		Tirocinio	12	C. Passerone		Si
1,2	02CWHOA		Tirocinio	10	C. Passerone	(1)(2)	Si
2	04AFQOA		Basi di dati	6	S. Chiusano		Si
2	18AKSOA		Controlli automatici	10	M. Taragna		Si
2			Crediti liberi del 3 anno	6			
2	05CBIOA		Programmazione a oggetti	6	G. Bruno		Si

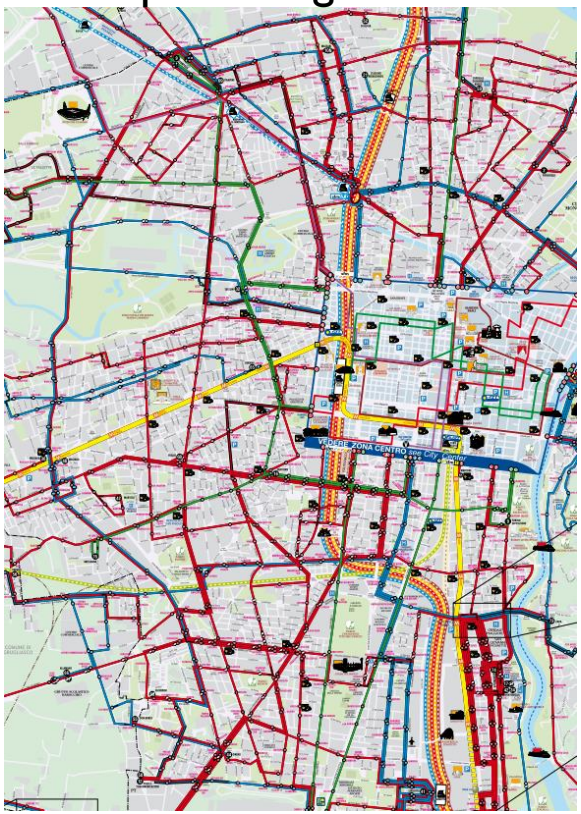
Crediti liberi del 1 anno

Periodo	Codice	Lingua	Insegnamento	Crediti	Docente	Note	Vincoli
2	01DDVOA		Automotive evolution	6	S. Genta	(4)	Si
2	01OHOOA		Chimica sperimentale per l'ingegneria	6	V. Pennazzi	(4)	Si
2	01OQCOA		Etica	6	M. Ghisleni	(4)	Si

https://didattica.polito.it/pls/portal30/gap.a_mds.espandi2?p_aacc=2013&p_sdu=37&p_cds=3&p_header=&p_lang=IT

Example: Turin public transportation

http://www.gtt.to.it/



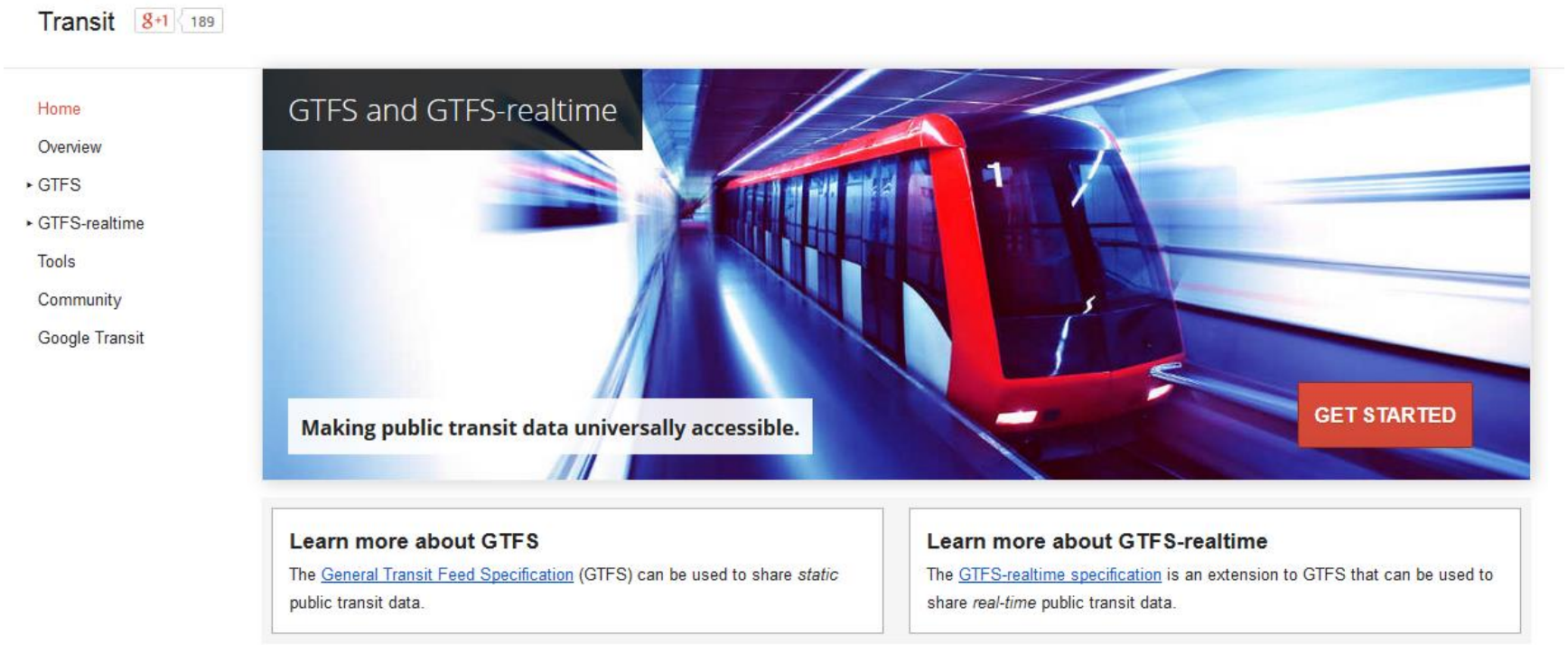
GRUPPO TORINESE TRASPORTI

http://www.sfmtorino.it/

REGIONE PIEMONTE
sfm
servizio ferroviario metropolitano

Google's GTFS standard

https://developers.google.com/transit/



GTFS Specification

Filename	Required	Defines
agency.txt	Required	One or more transit agencies that provide the data in this feed.
stops.txt	Required	Individual locations where vehicles pick up or drop off passengers.
routes.txt	Required	Transit routes. A route is a group of trips that are displayed to riders as a single service.
trips.txt	Required	Trips for each route. A trip is a sequence of two or more stops that occurs at specific time.
stop_times.txt	Required	Times that a vehicle arrives at and departs from individual stops for each trip.
calendar.txt	Required	Dates for service IDs using a weekly schedule. Specify when service starts and ends, as well as days of the week where service is available.
calendar_dates.txt	Optional	Exceptions for the service IDs defined in the calendar.txt file. If calendar_dates.txt includes ALL dates of service, this file may be specified instead of calendar.txt.
fare_attributes.txt	Optional	Fare information for a transit organization's routes.
fare_rules.txt	Optional	Rules for applying fare information for a transit organization's routes.
shapes.txt	Optional	Rules for drawing lines on a map to represent a transit organization's routes.
frequencies.txt	Optional	Headway (time between trips) for routes with variable frequency of service.
transfers.txt	Optional	Rules for making connections at transfer points between routes.
feed_info.txt	Optional	Additional information about the feed itself, including publisher, version, and expiration information.

https://developers.google.com/transit/gtfs/reference

Where to find data?

GRUPPO TORINESE TRASPORTI

http://opendata.5t.torino.it/gtfs/torino_it.zip

REGIONE PIEMONTE sfm servizio ferroviario metropolitano

http://opendata.5t.torino.it/gtfs/sfm_torino_it.zip

Need more?

http://www.gtfs-data-exchange.com/

Querying graph structure

- Navigate structure
  - java.util.Set<V> **vertexSet**()
  - boolean **containsVertex**(V v)
  - boolean **containsEdge**(V sourceVertex,V targetVertex)
  - java.util.Set<E> **edgesOf**(V vertex)
  - java.util.Set<E> **getAllEdges**(V sourceVertex,V targetVertex)
- Query Edges
  - V **getEdgeSource**(E e)
  - V **getEdgeTarget**(E e)
  - double **getEdgeWeight**(E e)

Utility functions

- Static class **org.jgrapht.Graphs**
- Easier creation
  - public static <V,E> E **addEdge**(Graph<V,E> g,V sourceVertex, V targetVertex, double weight)
  - public static <V,E> E **addEdgeWithVertices**(Graph<V,E> g, V sourceVertex,V targetVertex)
- Easier navigation
  - public static <V,E> java.util.List<V> **neighborListOf**(Graph<V,E> g, V vertex)
  - public static String **getOppositeVertex**(Graph<String, DefaultEdge> g, DefaultEdge e, String v)
  - public static <V,E> java.util.List<V> **predecessorListOf**(DirectedGraph<V,E> g,V vertex)
  - public static <V,E> java.util.List<V> **successorListOf**(DirectedGraph<V,E> g,V vertex)

Graph visits

Representing and visiting graphs

Visit Algorithms

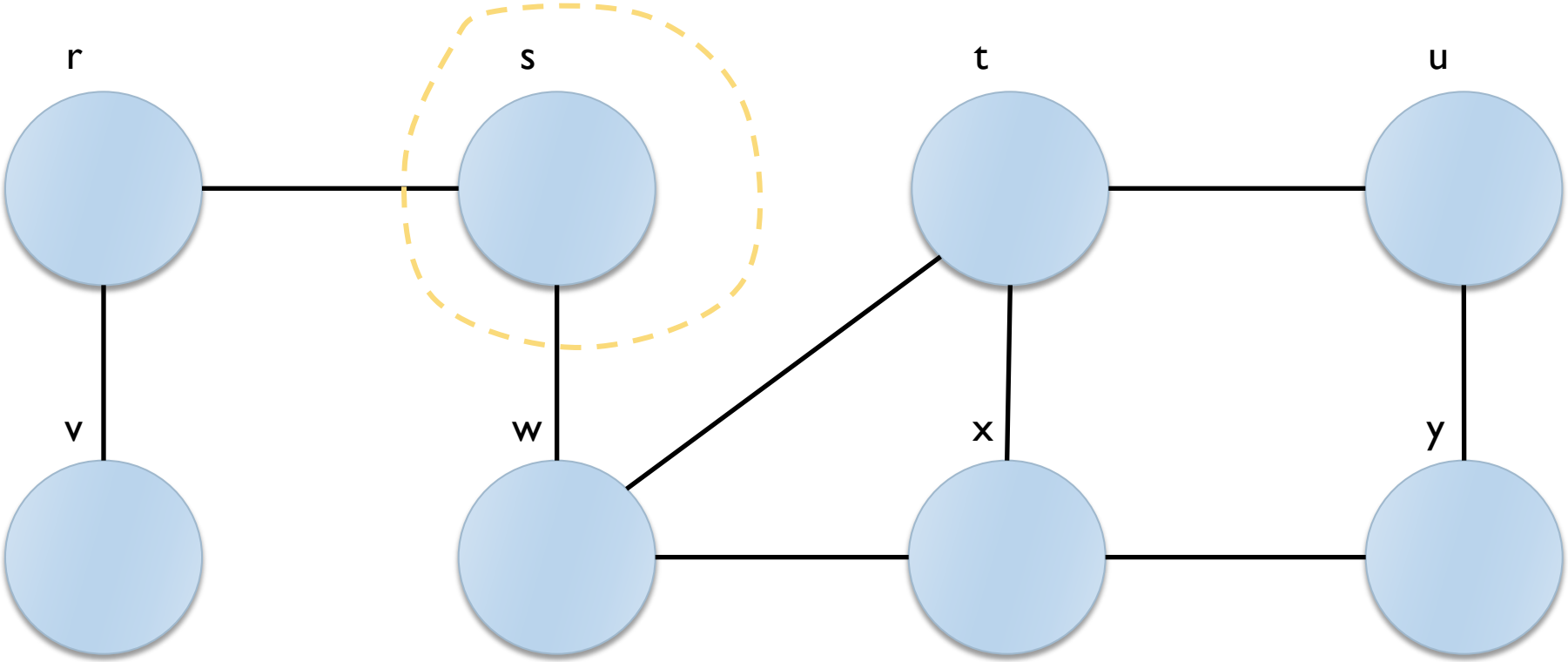
- Visit =
  - Systematic exploration of a graph
  - Starting from a 'source' vertex
  - Reaching all reachable vertices
- Main strategies
  - Breadth-first visit ("in ampiezza")
  - Depth-first visit ("in profondità")

Breadth-First Visit

- Also called Breadth-first search (BFV or BFS)
- All reachable vertices are visited "by levels"
  - L – level of the visit
  - S_L – set of vertices in level L
  - L=0, S_0={ v_{source} }
  - Repeat while S_L is not empty:
    - S_{L+1} = set of all vertices:
      - not visited yet, and
      - adjacent to at least one vertex in S_L
    - L=L+1

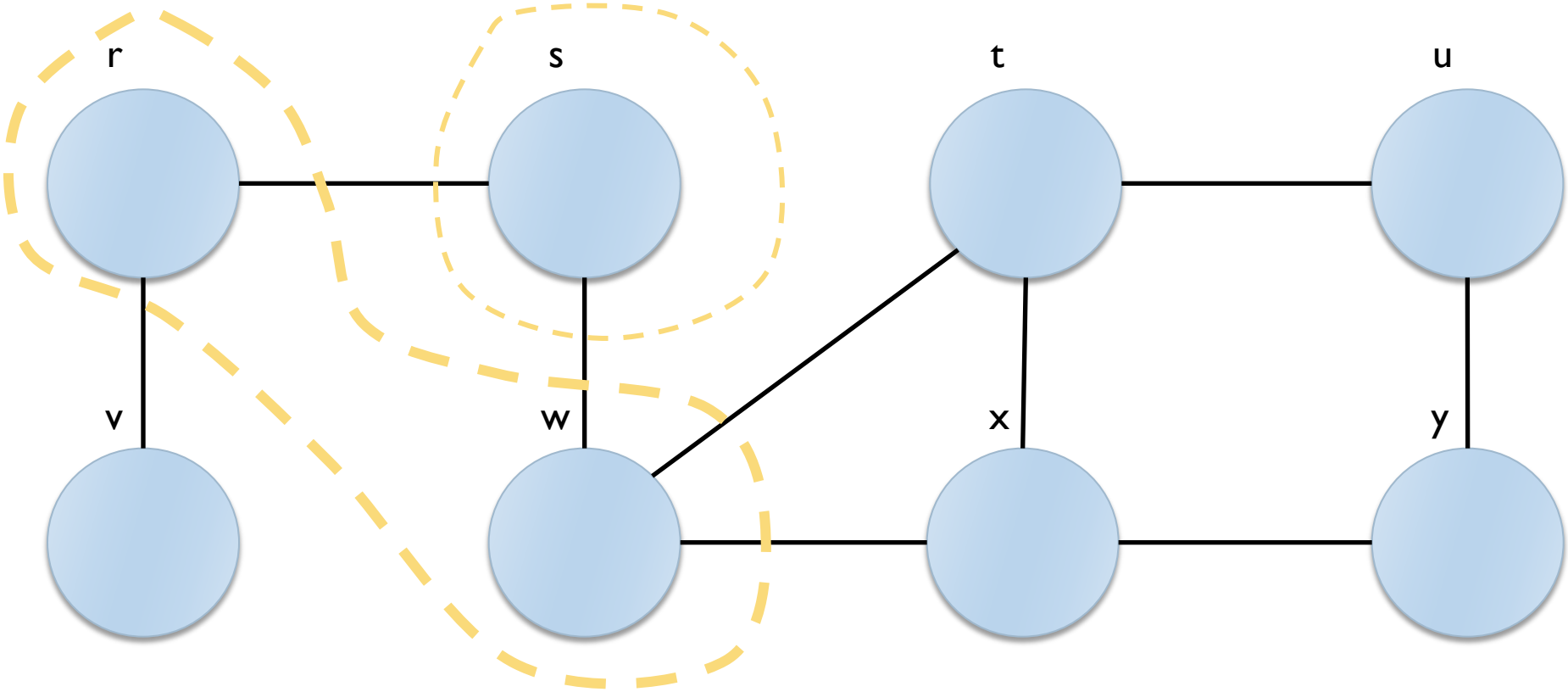
Example

Source = s
L = 0
S_0 = {s}



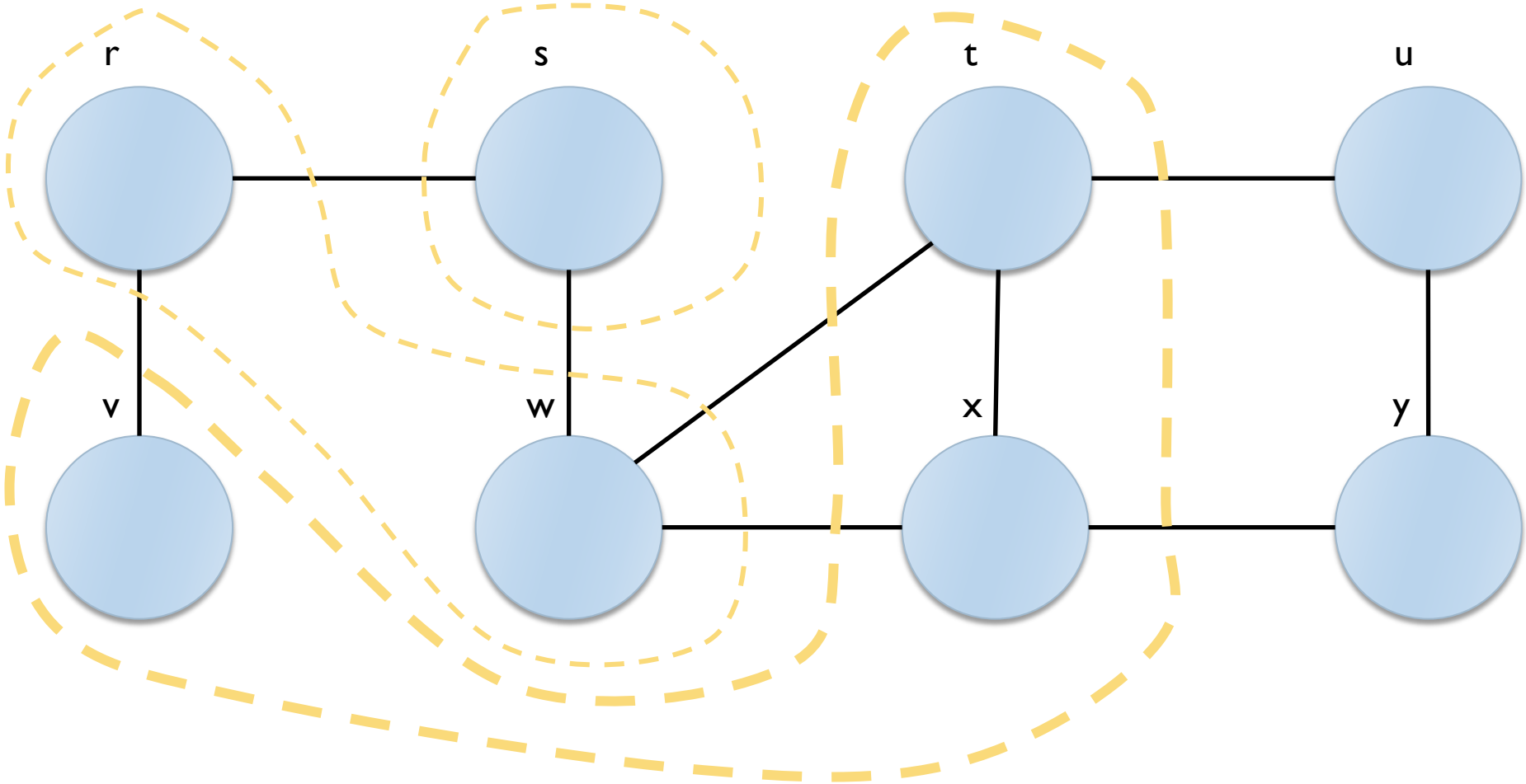
Example

$L = 1$
$S_0 = \{s\}$
$S_1 = \{r, w\}$



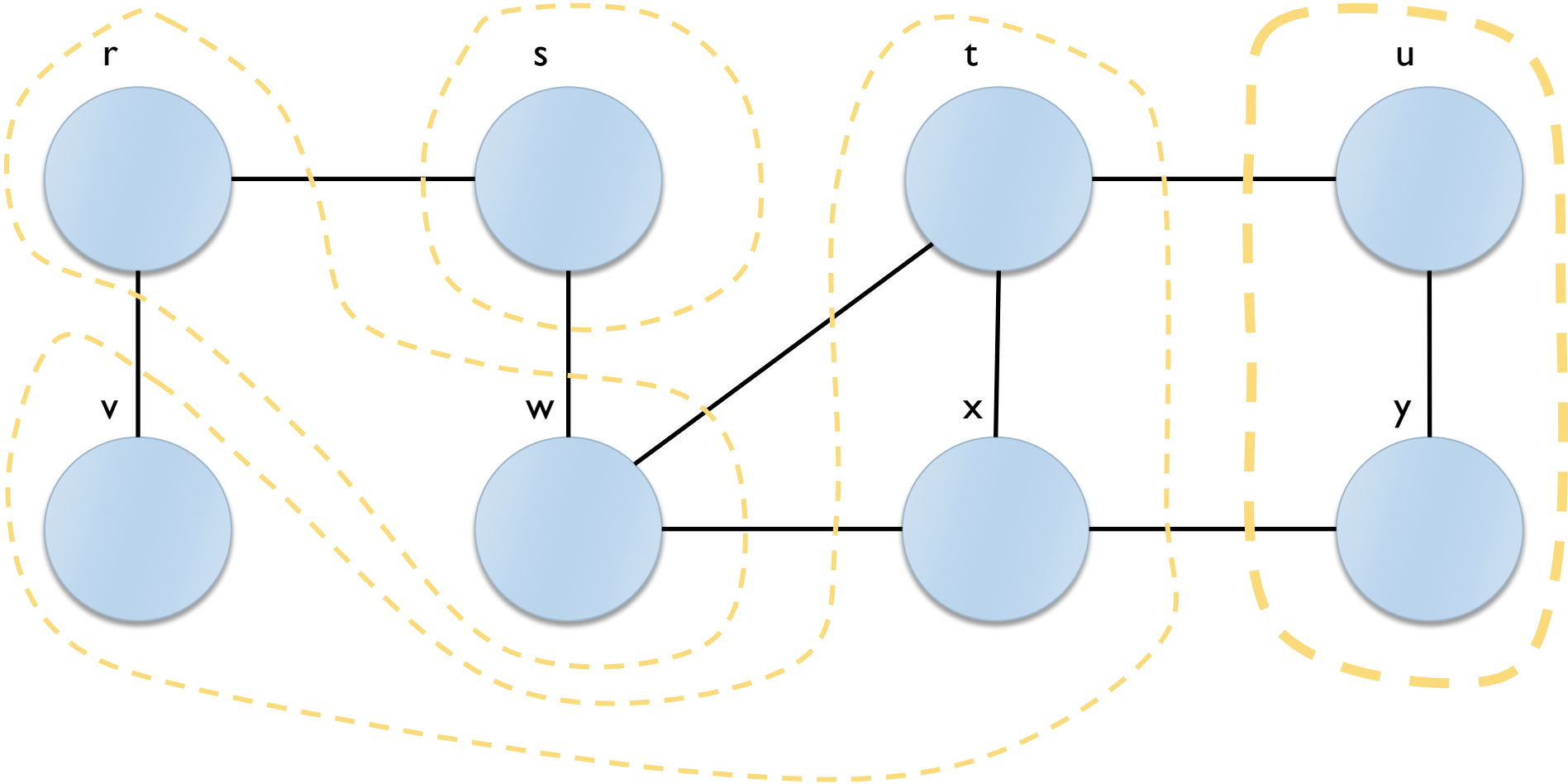
Example

$L = 2$
$S_1 = \{r, w\}$
$S_2 = \{v, t, x\}$



Example

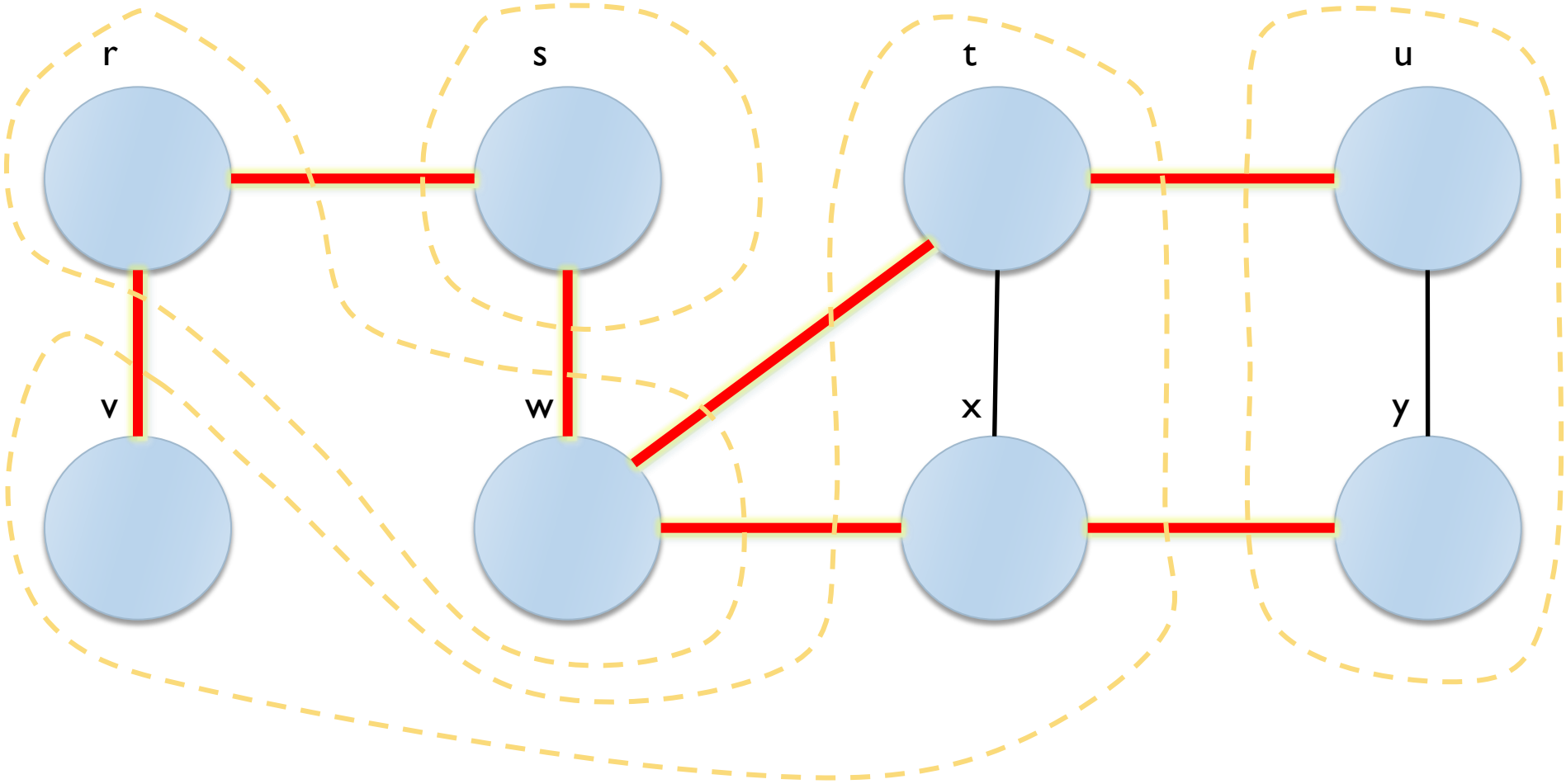
$L = 3$
$S_2 = \{v, t, x\}$
$S_3 = \{u, y\}$



BFS Tree

- The result of a BFV identifies a "visit tree" in the graph:
  - The tree root is the source vertex
  - Tree nodes are all graph vertices
    - (in the same connected component of the source)
  - Tree are a subset of graph edges
    - Those edges that have been used to "discover" new vertices.

BFS Tree



Minimum (shortest) paths

- Shortest path: the minumum number of edges on any path between two vertices
- The BFS procedure computes all minimum paths for all vertices, starting from the source vertex
- NB: unweighted graph : path length = number of edges

Depth First Visit

- Also called Depth-first search (DFV or DFS)
- Opposite approach to BFS
- At every step, visit one (yet unvisited) vertex, adjacent to the last visited one
- If no such vertex exist, go back one step to the previously visited vertex
- Lends itself to recursive implementation
  - Similar to tree visit procedures

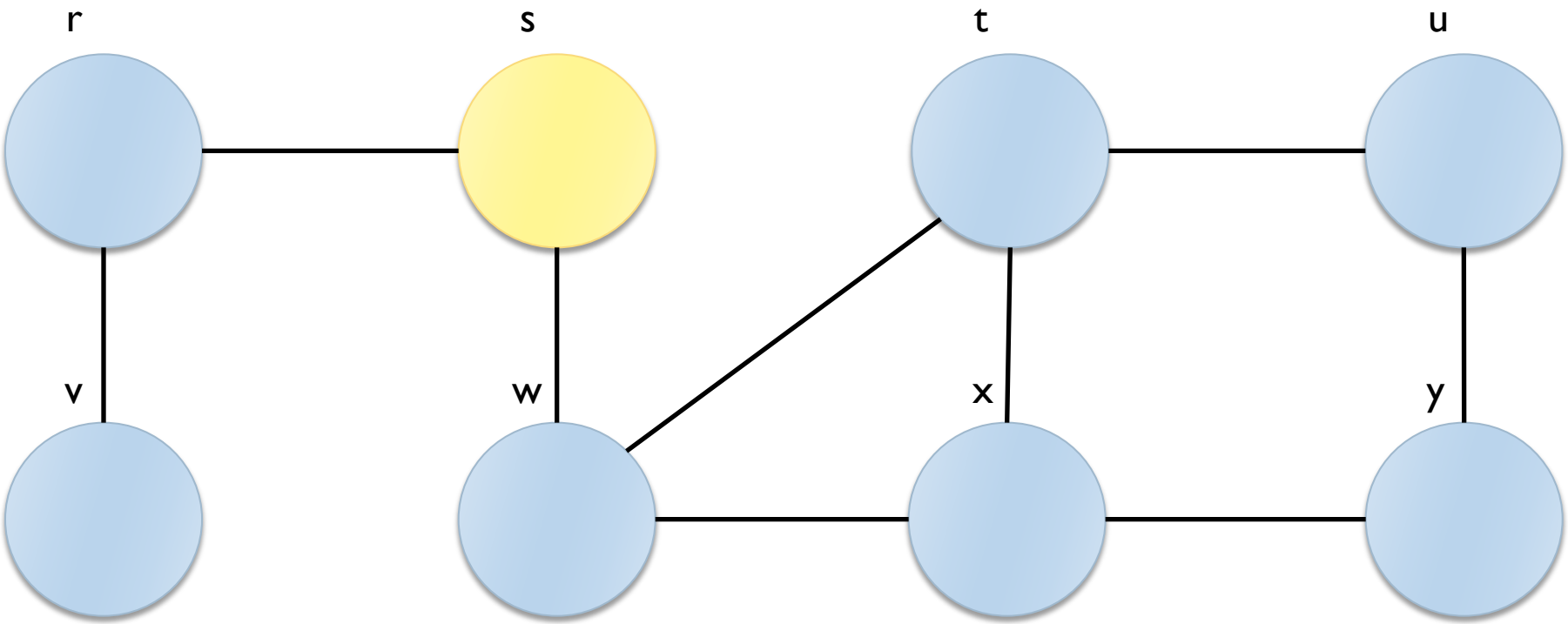
DFS Algorithm

- DFS(Vertex v)
  - For all (w : adjacent_to(v))
    - If(not visited (w))
      - Visit (w)
      - DFS(w)

- Start with: DFS(source)

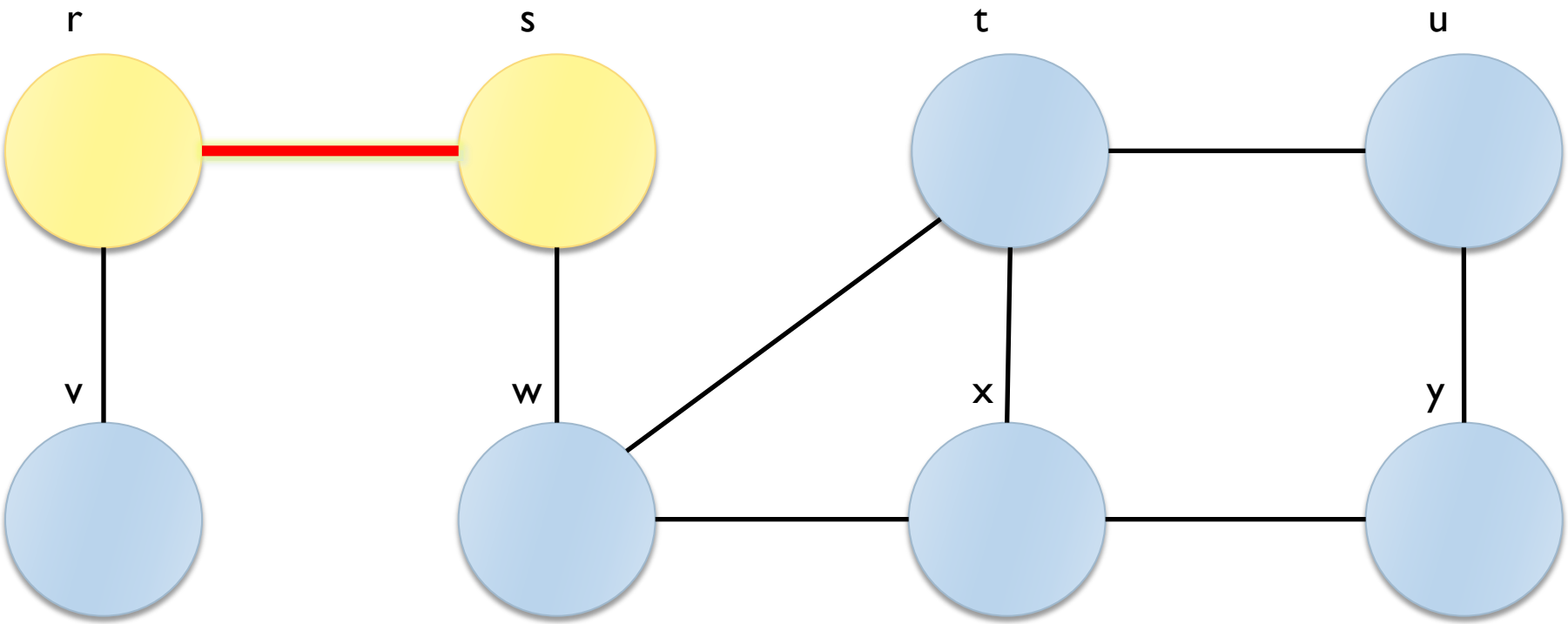
Example

Source = s



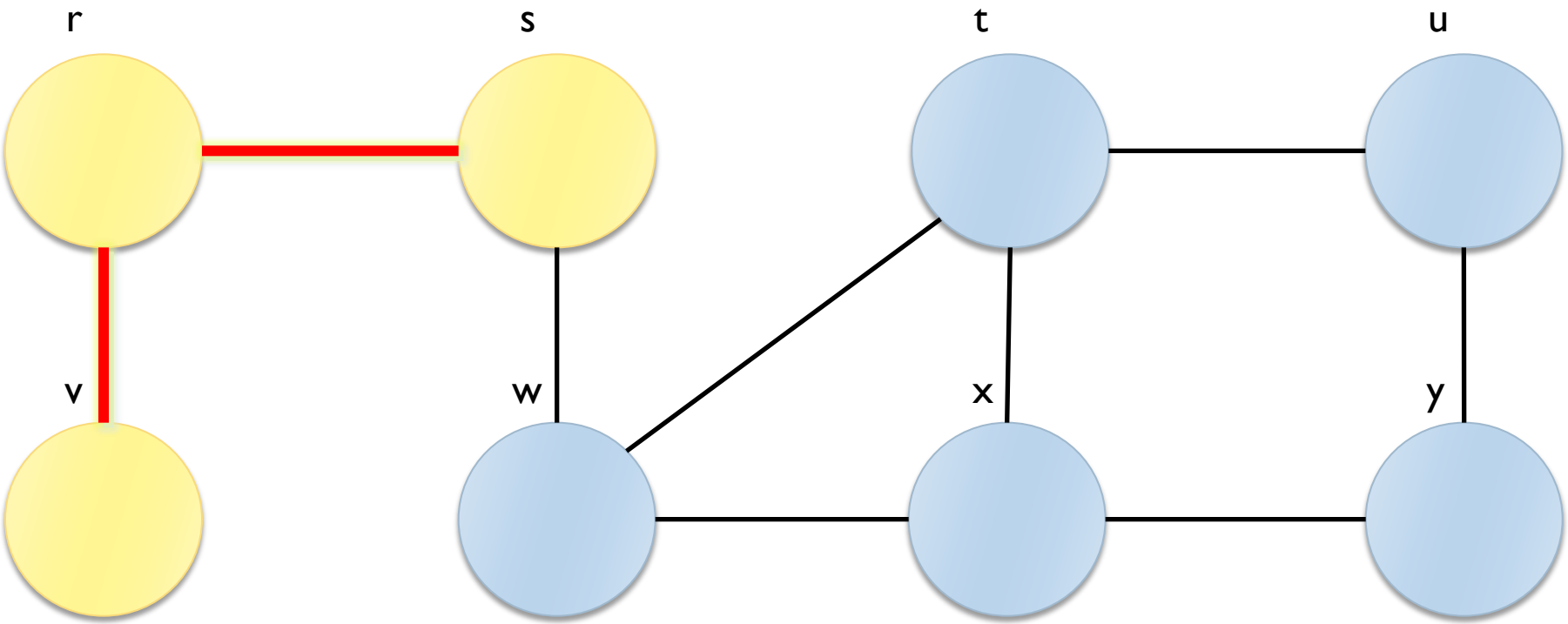
Example

Source = s
Visit r



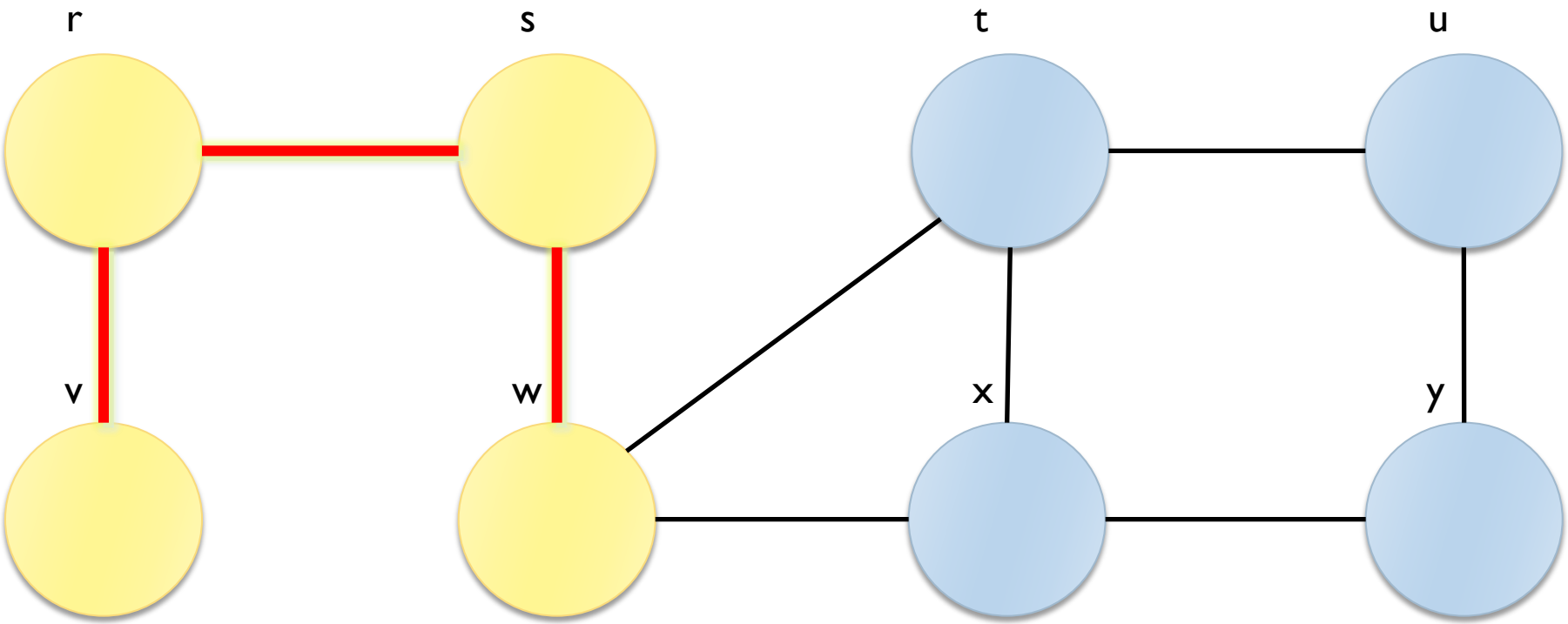
Example

Source = s
Visit r
Visit v



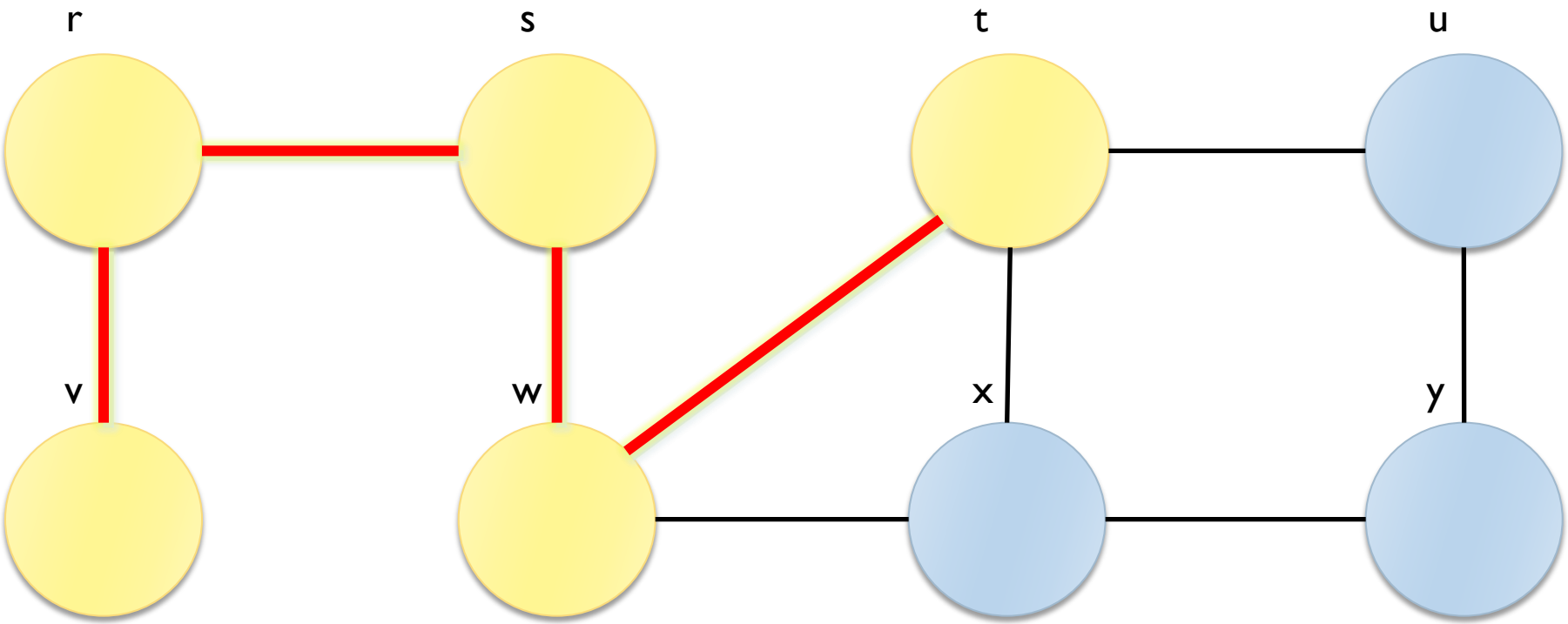
Example

Source = s
Back to r
Back to s
Visit w



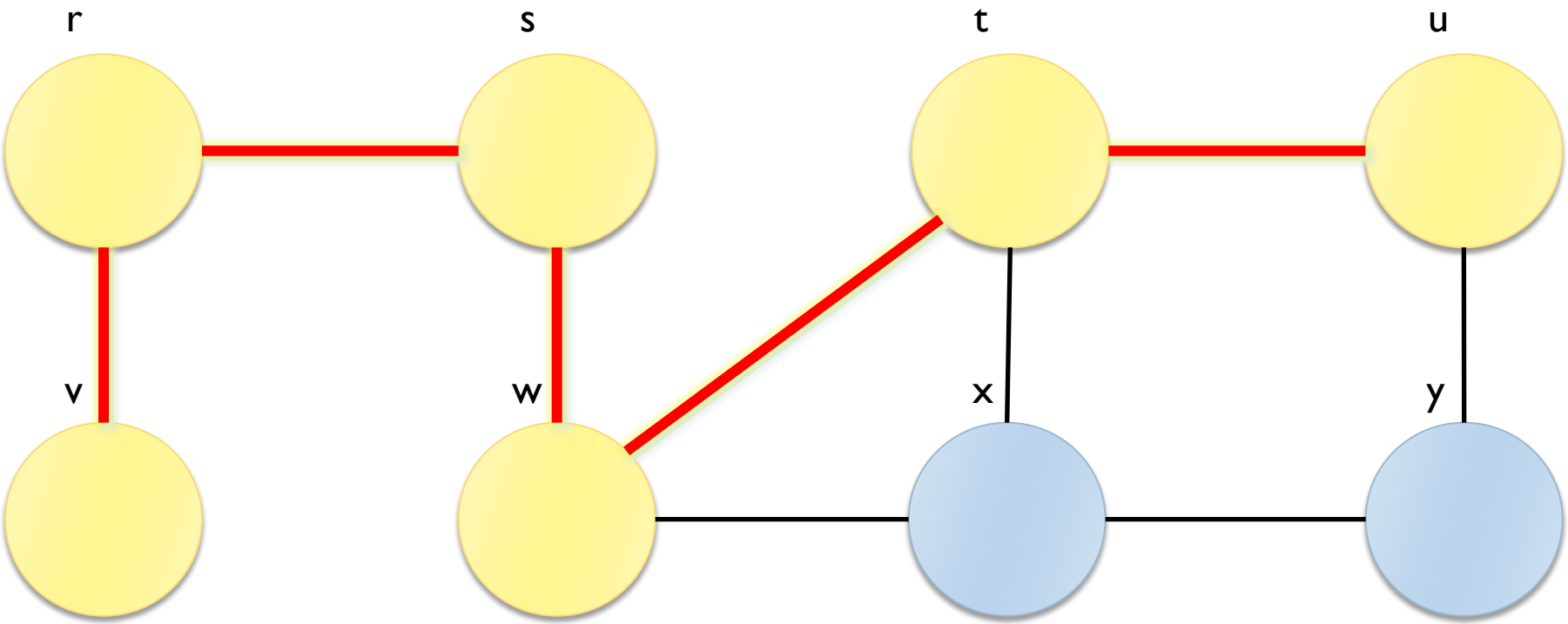
Example

Source = s
Visit w
Visit t



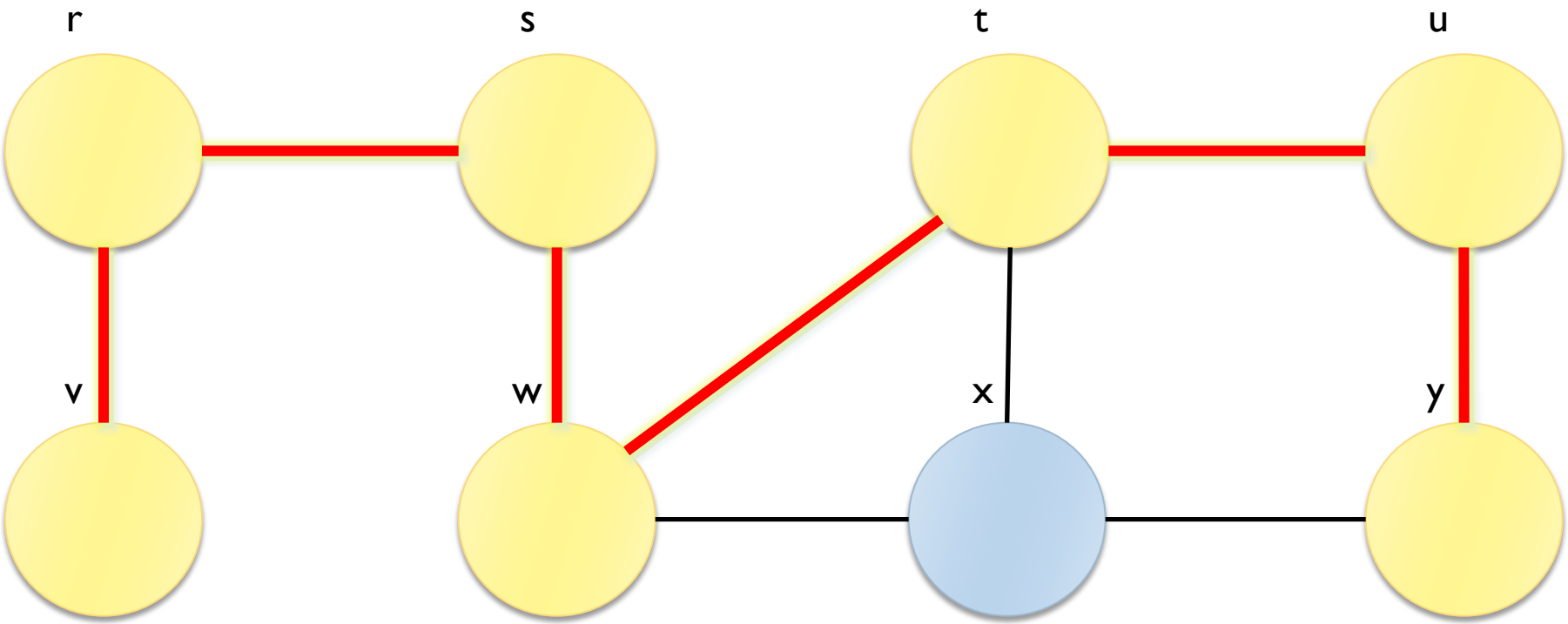
Example

Source = s
Visit w
Visit t
Visit u



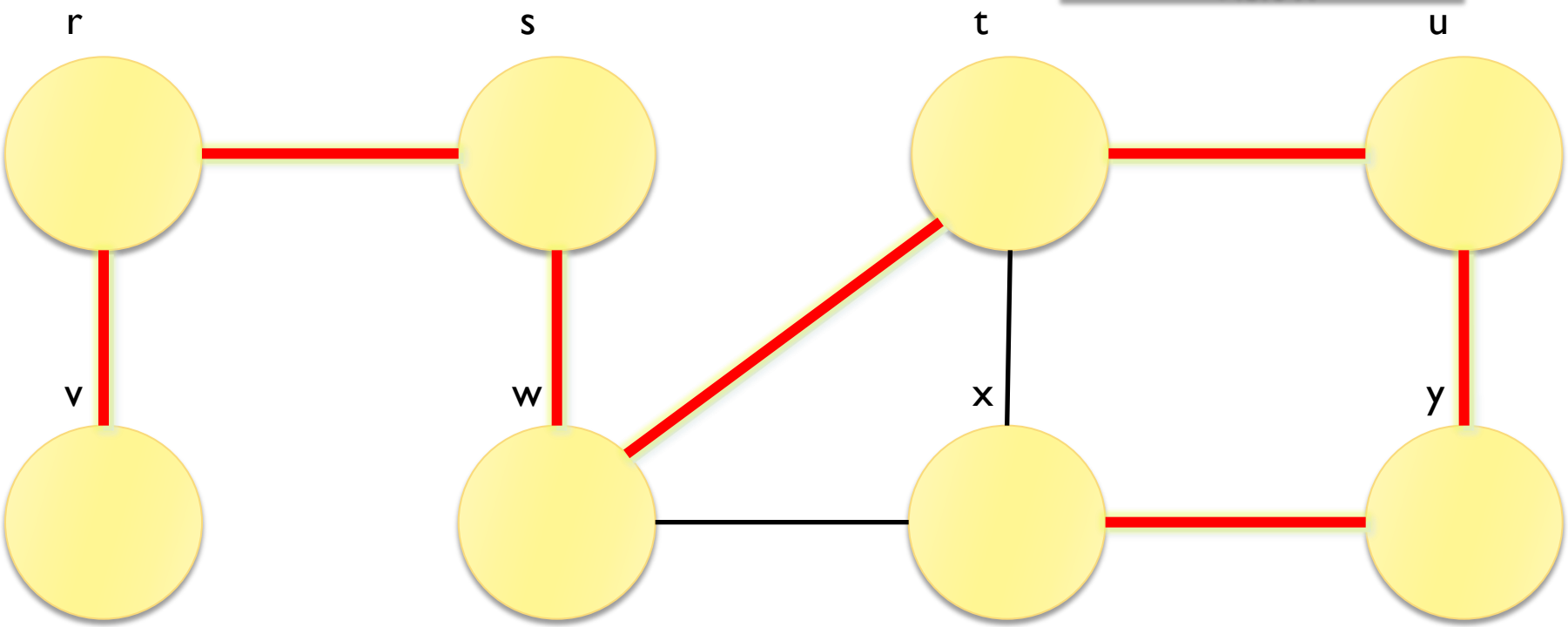
Example

Source = s
Visit w
Visit t
Visit u
Visit y

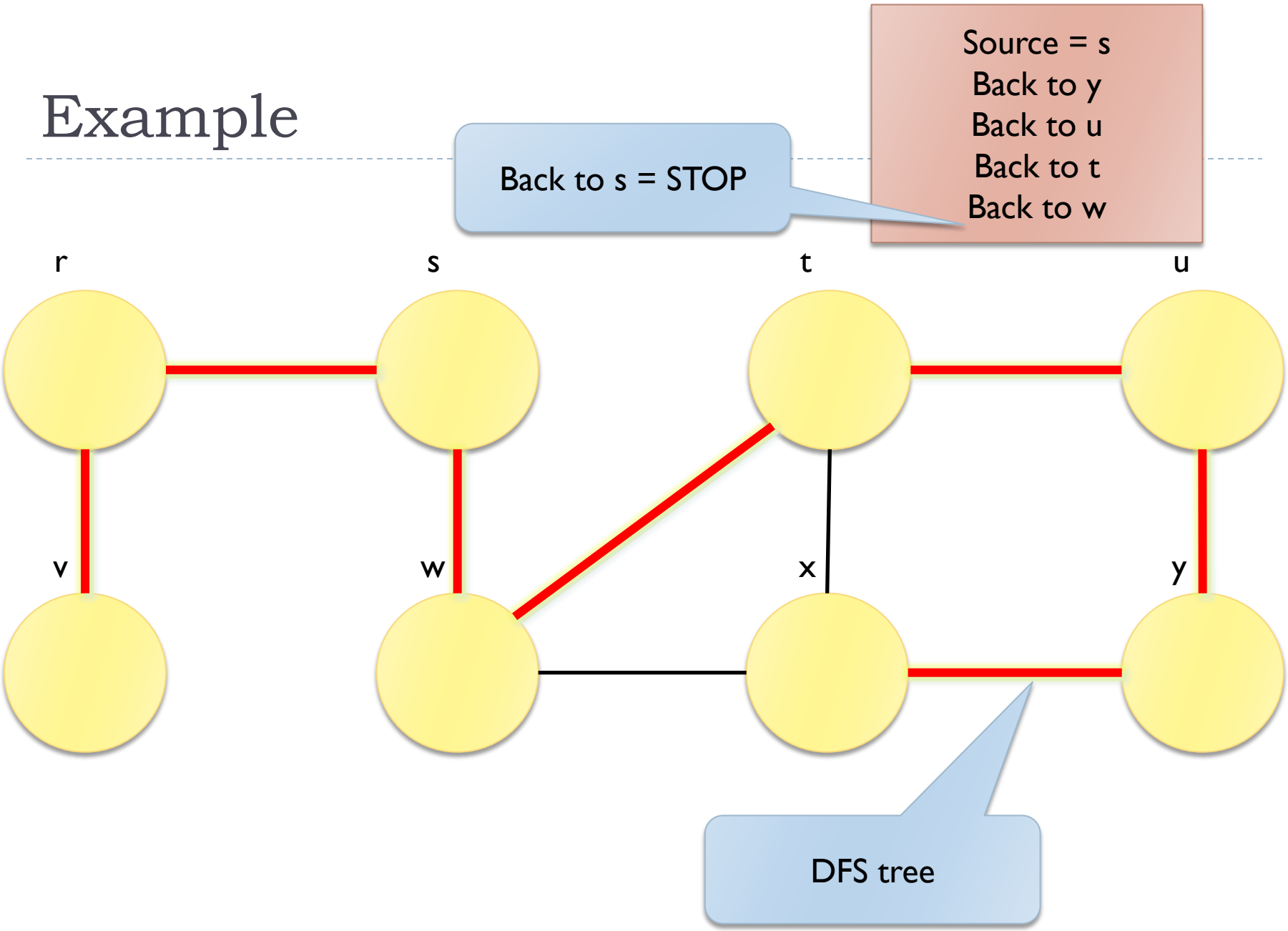


Example

Source = s
Visit w
Visit t
Visit u
Visit y
Visit x



Example

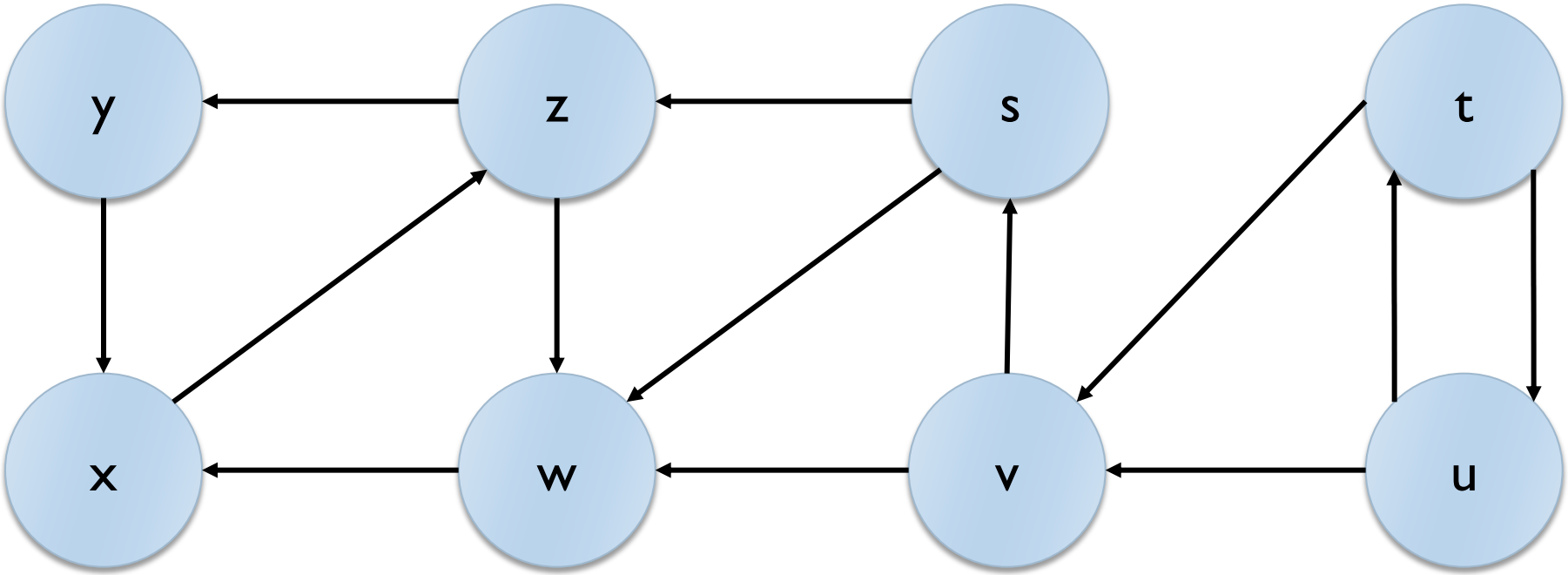

Source = s
Back to y
Back to u
Back to t
Back to w
Back to s = STOP
r
s
t
u
v
w
x
y
DFS tree

Edge classification

- In an directed graph, after a DFS visit, all edges fall in one of these 4 categories:
  - T: **Tree** edges (belonging to the DFS tree)
  - B: **Back** edges (not in T, and connect a vertex to one of its ancestors)
  - F: **Forward** edges (not in T and B, and connect a vertex to one of its descendants)
  - C: **Cross** edges (all remaining edges)

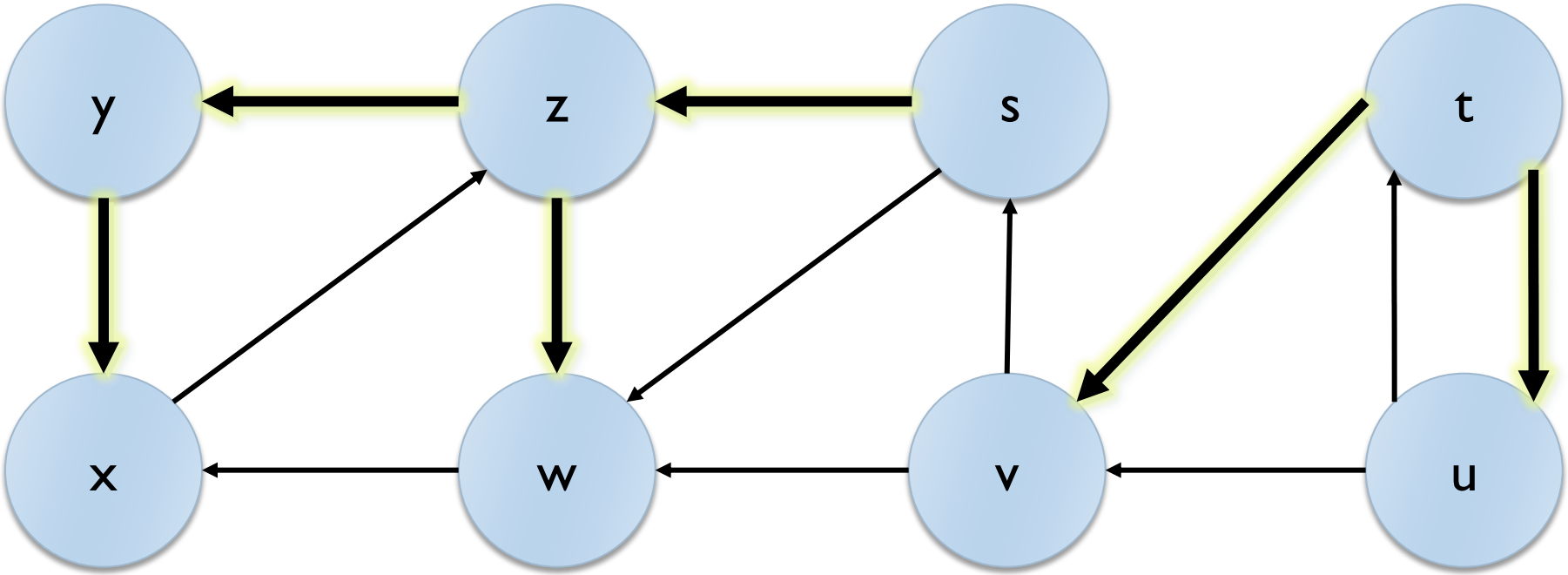
Example

Directed graph



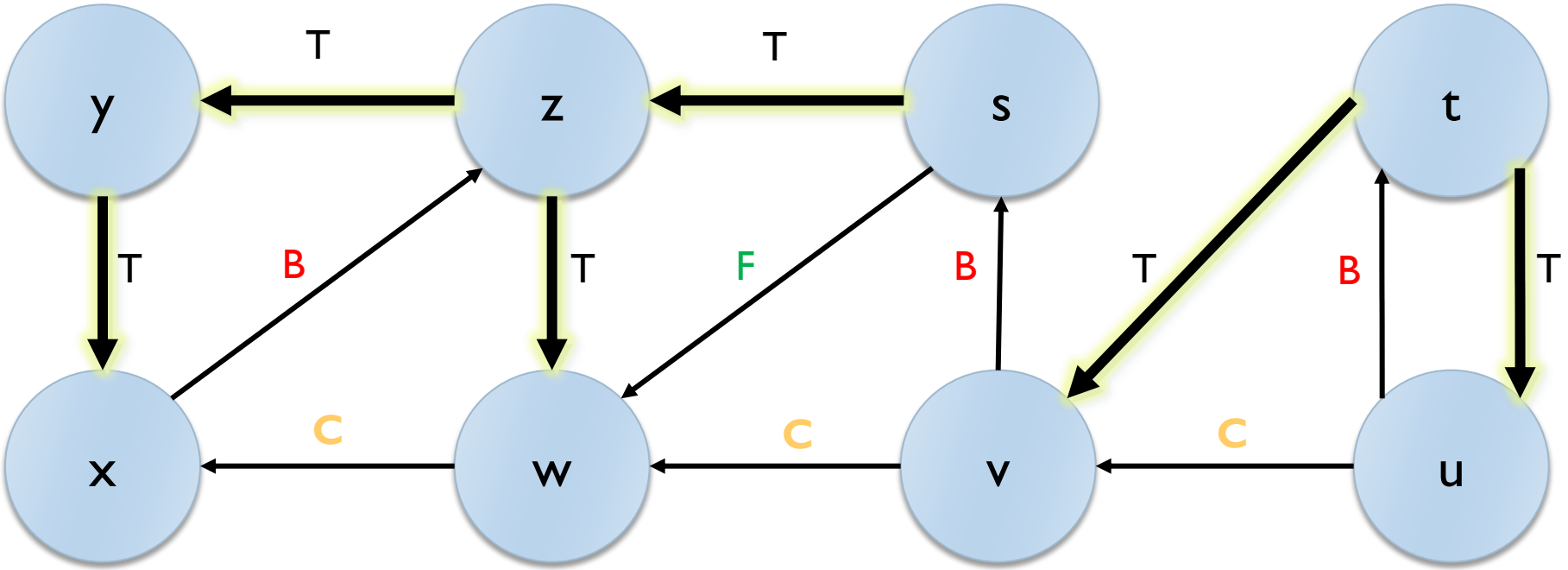
Example

DFS visit
(sources: s, t)



Example

Edge classification



Cycles

- Theorem:
- A directed graph is acyclic if and only if a depth-first visit does not produce any B edge

Complexity

- Visits have linear complexity in the graph size
  - BFS : $O(V+E)$
  - DFS : $\Theta(V+E)$
- N.B. for dense graphs, $E = O(V^2)$

Visits in JGraphT

Representing and visiting graphs

JGraphT and visits

- Visits are called "traversals"
- Implemented through **iterator** classes
- Package **org.jgrapht.traverse**

Graph traversal classes

Package org.jgrapht.traverse

Graph traversal means.

See:

Description

Interface Summary	
GraphIterator<V,E>	A graph iterator.

Class Summary	
AbstractGraphIterator<V,E>	An empty implementation of a graph iterator to minimize the effort required to implement graph iterators.
BreadthFirstIterator<V,E>	A breadth-first iterator for a directed and an undirected graph.
ClosestFirstIterator<V,E>	A closest-first iterator for a directed or undirected graph.
CrossComponentIterator<V,E,D>	Provides a cross-connected-component traversal functionality for iterator subclasses.
DepthFirstIterator<V,E>	A depth-first iterator for a directed and an undirected graph.
TopologicalOrderIterator<V,E>	Implements topological order traversal for a directed acyclic graph.

Graph iterators

- Usual hasNext() and next() methods
- May register event listeners to traversal steps
  - void **addTraversalListener**(TraversalListener<V,E> l)
- TraversalListeners may react to:
  - Edge traversed
  - Vertex traversed
  - Vertex finished
  - Connected component started
  - Connected component finished

Types of traversal iterators

- BreadthFirstIterator
- DepthFirstIterator
- ClosestFirstIterator
  - The metric for *closest* here is the path length from a start vertex. Graph.getEdgeWeight(Edge) is summed to calculate path length. Optionally, path length may be bounded by a finite radius.
- TopologicalOrderIterator
  - A topological sort is a permutation *p* of the vertices of a graph such that an edge {*i*,*j*} implies that *i* appears before *j* in *p*. Only directed acyclic graphs can be topologically sorted.

Resources

- Open Data Structures (in Java), Pat Morin, http://opendatastructures.org/
- Algorithms Course Materials, Jeff Erickson, http://www.cs.uiuc.edu/~jeffe/teaching/algorithms/
- Graphbook - A book on algorithmic graph theory, David Joyner, Minh Van Nguyen, and David Phillips, https://code.google.com/p/graphbook/

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/