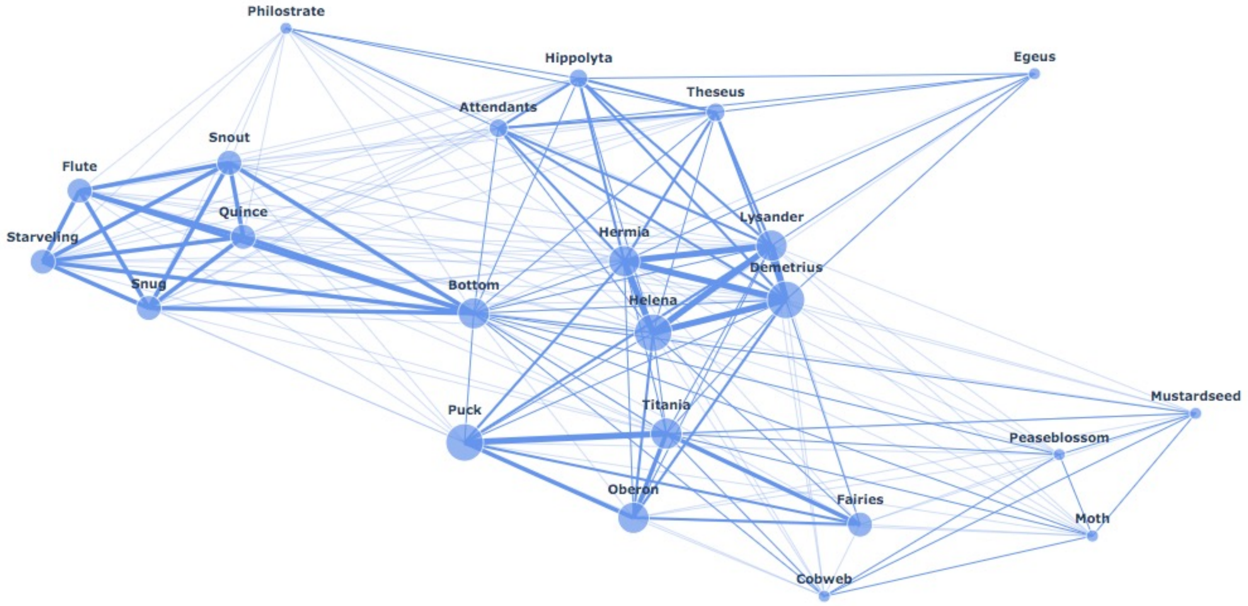
Graph visits

How to explore graphs

Fulvio Corno
Giuseppe Averta
Carlo Masone
Francesca Pistilli



Representing and visiting graphs

GRAPH VISITS

Visit Algorithms

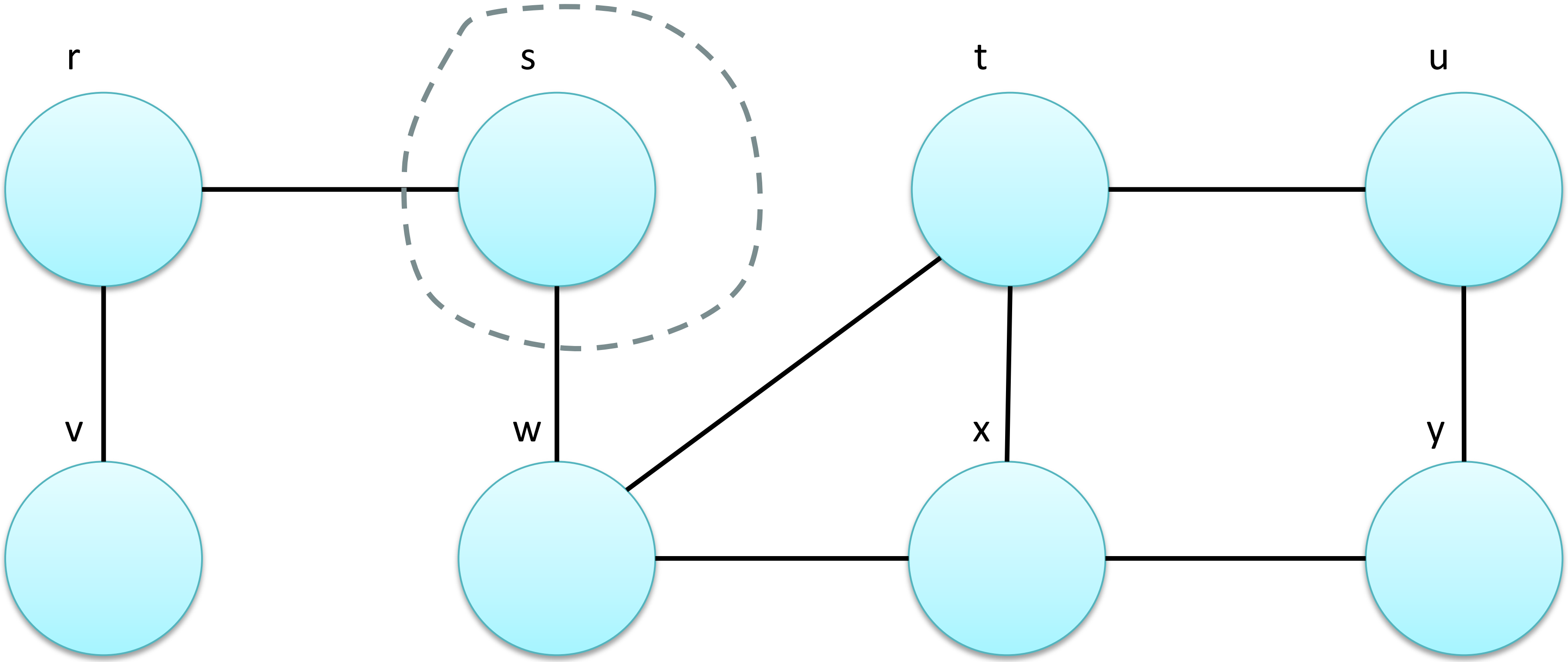
- Visit =
  - Systematic exploration of a graph
  - Starting from a ‘source’ vertex
  - Reaching all reachable vertices
- Main strategies
  - Breadth-first visit (“in ampiezza”)
  - Depth-first visit (“in profondità”)

Breadth-First Visit

- Also called Breadth-first search (BFV or BFS)
- All reachable vertices are visited “by levels”
  - L – level of the visit
  - SL – set of vertices in level L
  - L=0, S0={ vsource }
  - Repeat while SL is not empty:
    - SL+1 = set of all vertices:
      - not visited yet, and
      - adjacent to at least one vertex in SL
    - L=L+1

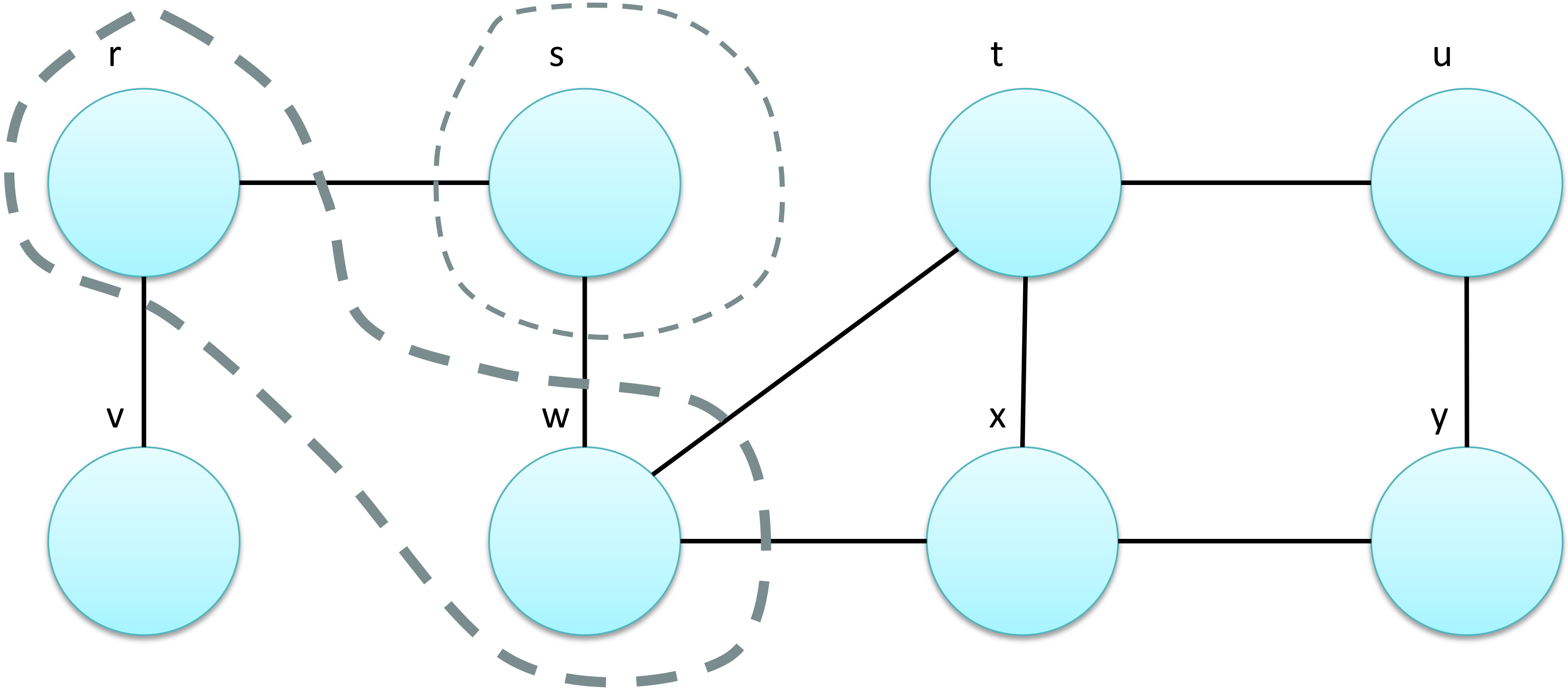
Example

Source = s
L = 0
S_0 = {s}



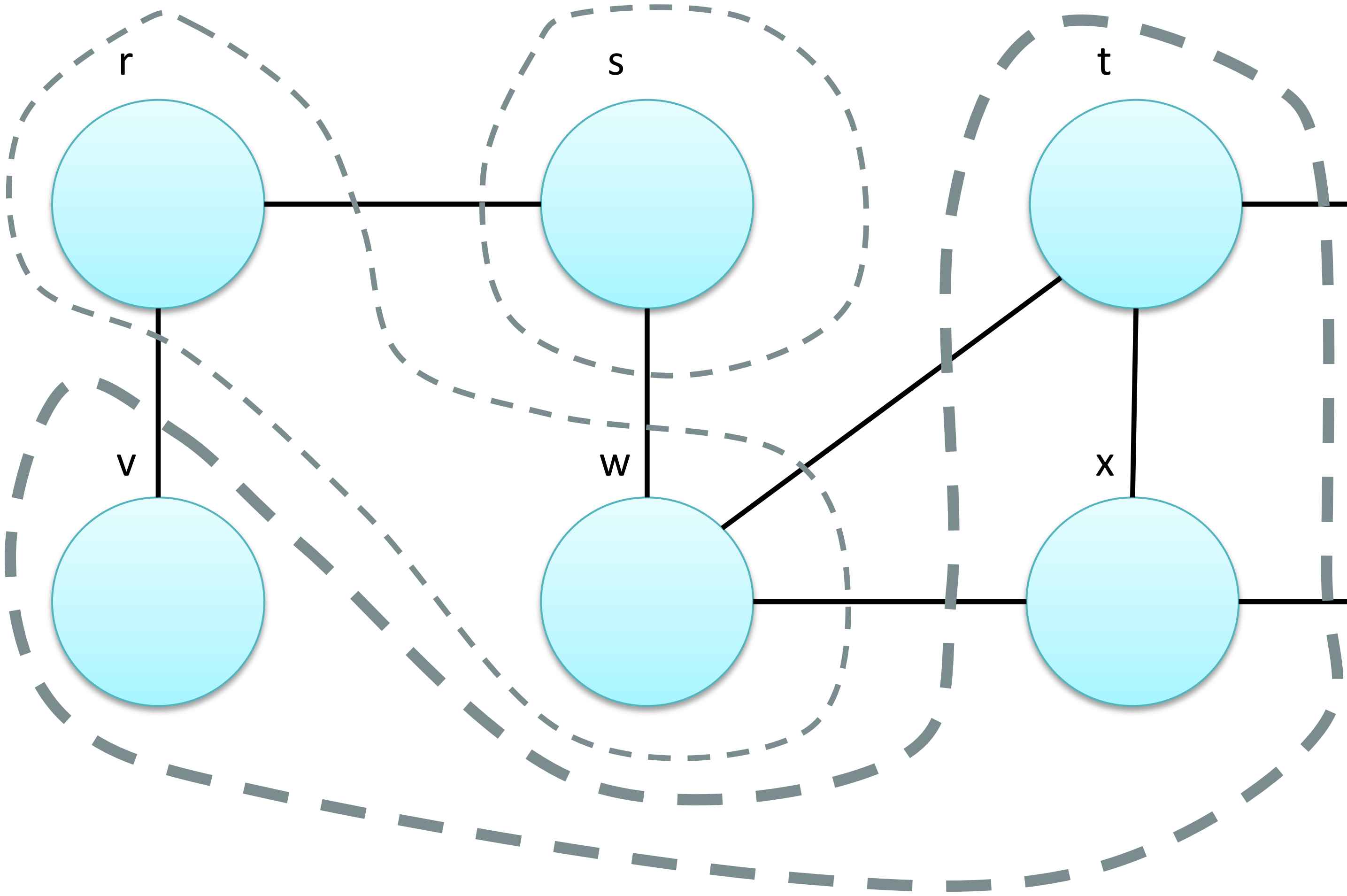
Example

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



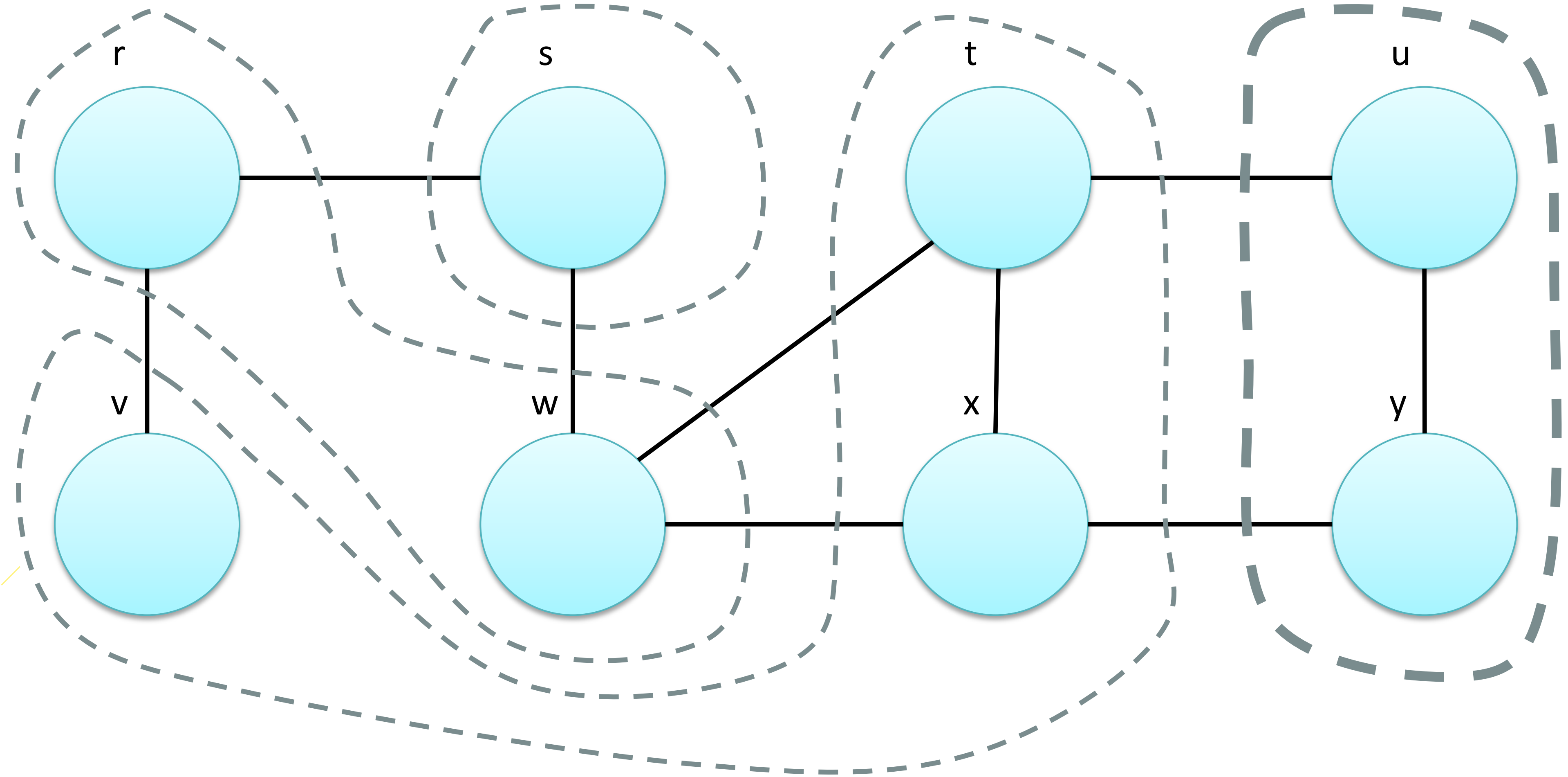
Example

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



Example

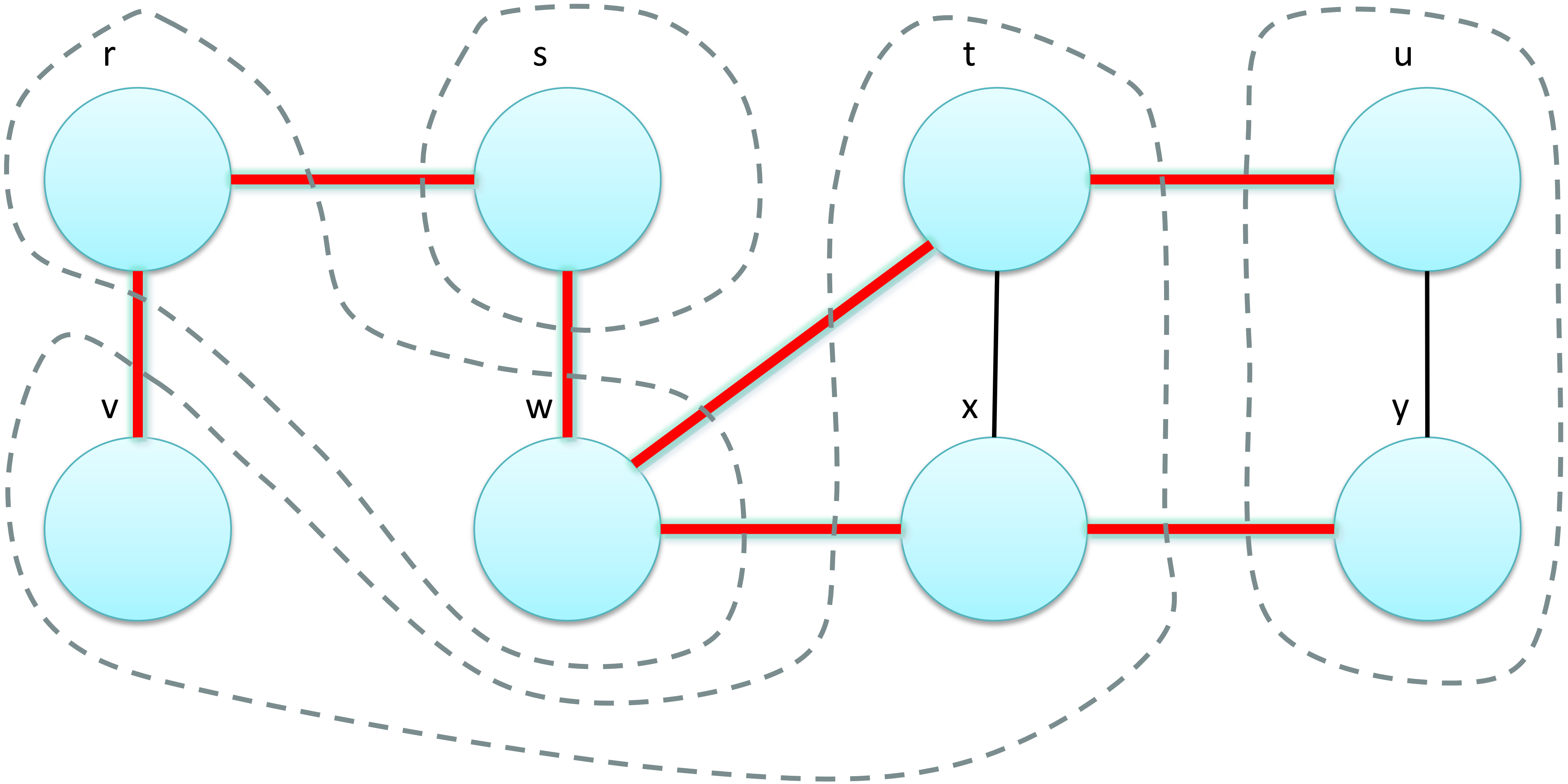
L = 3
S_2 = {v, t, x}
S_3 = {u, y}



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

ed in questo momento si sta parlando di archi che non possiedono alcun peso

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

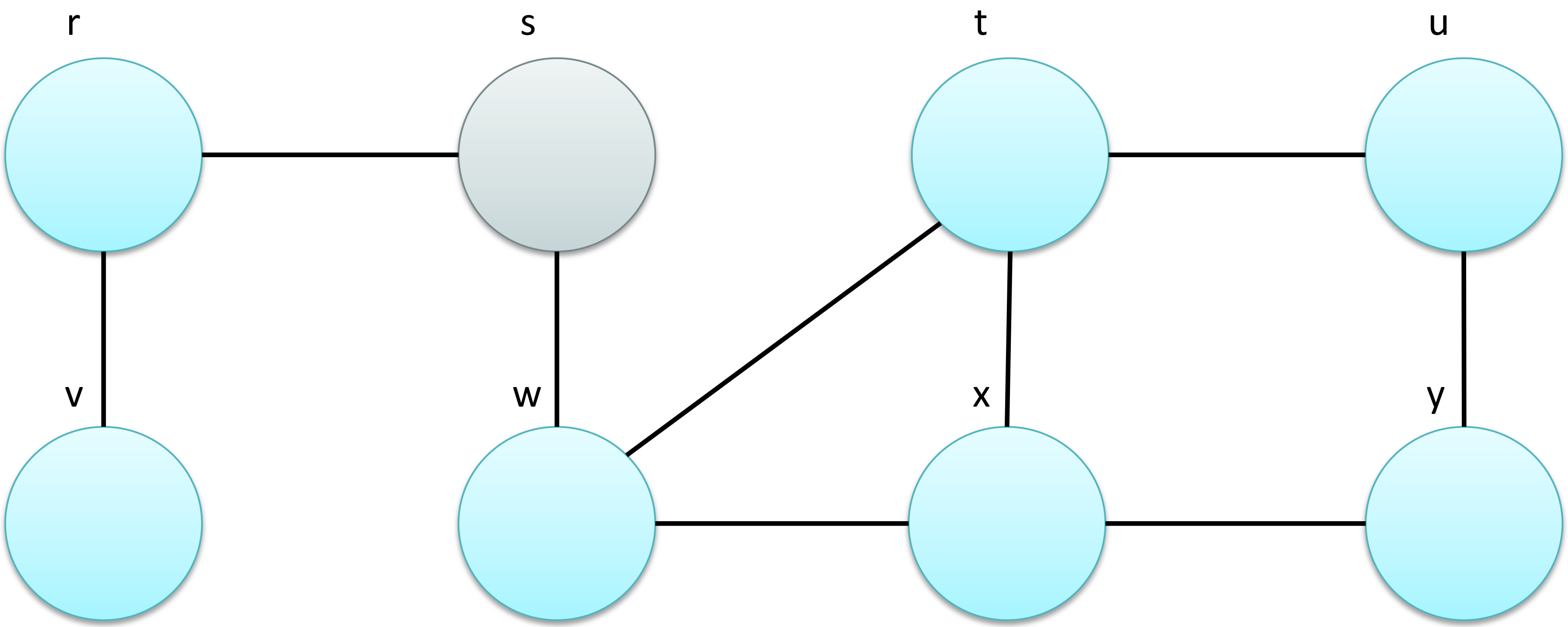
DFS Algorithm

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

- Start with: DFS(source)

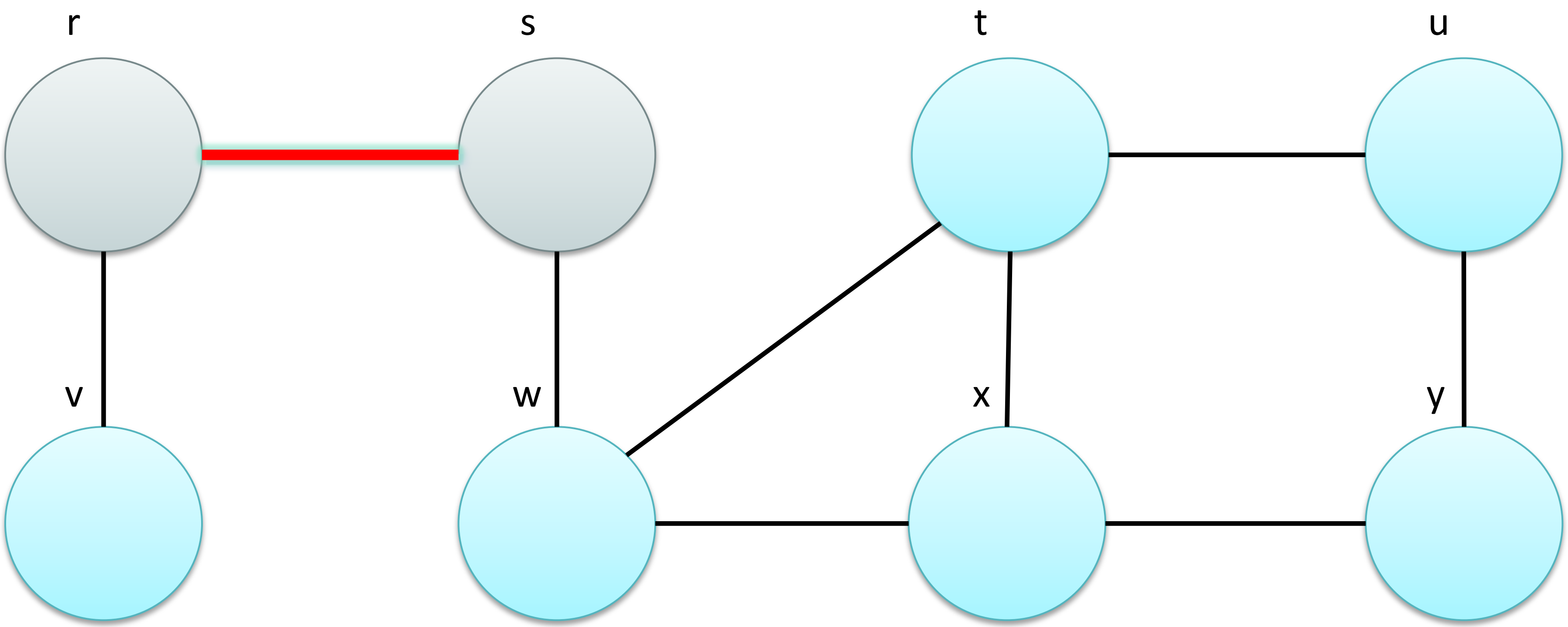
Example

Source = s



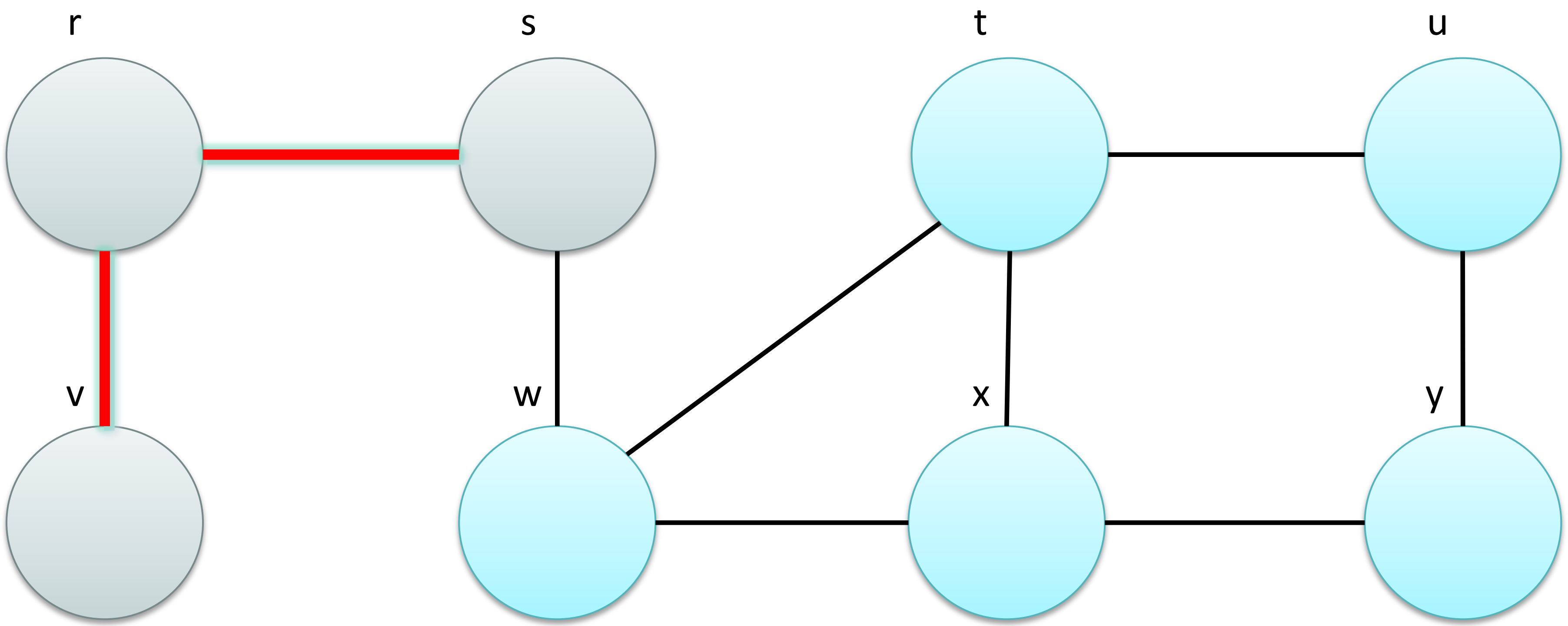
Example

Source = s
Visit r



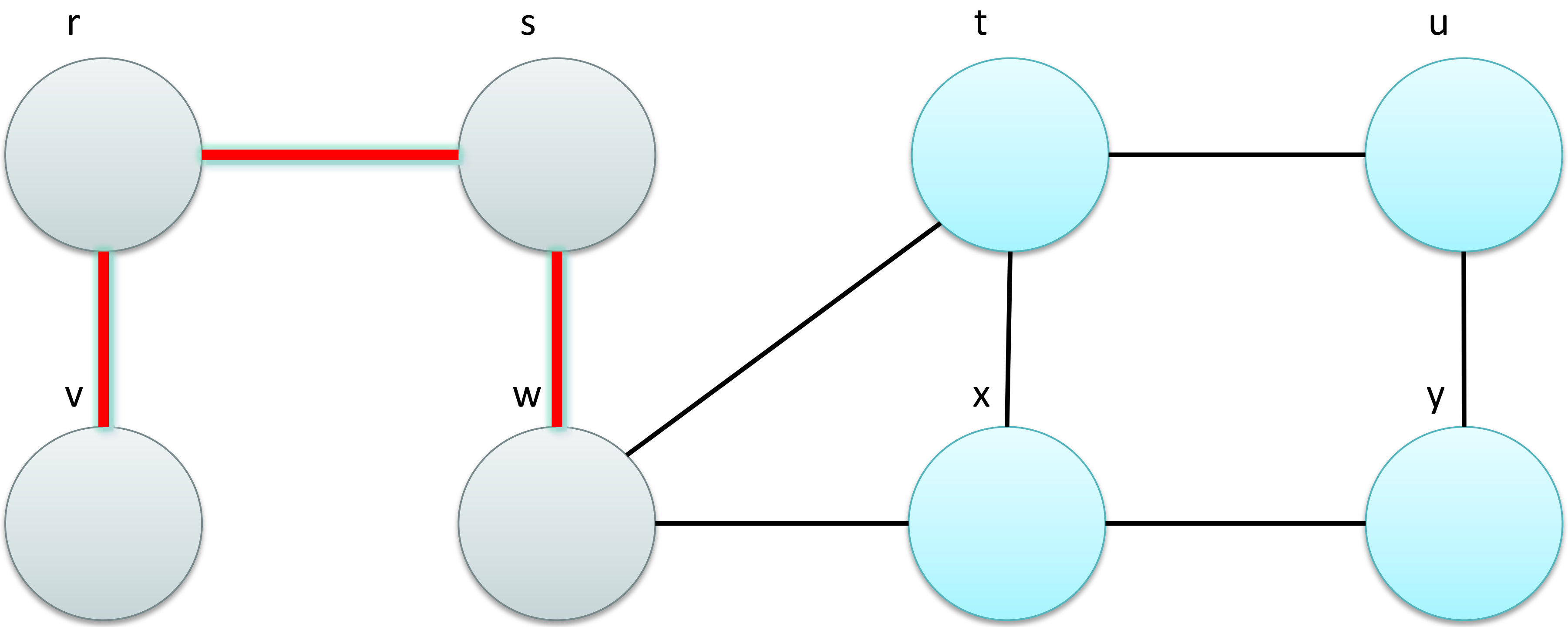
Example

Source = s
Visit r
Visit v



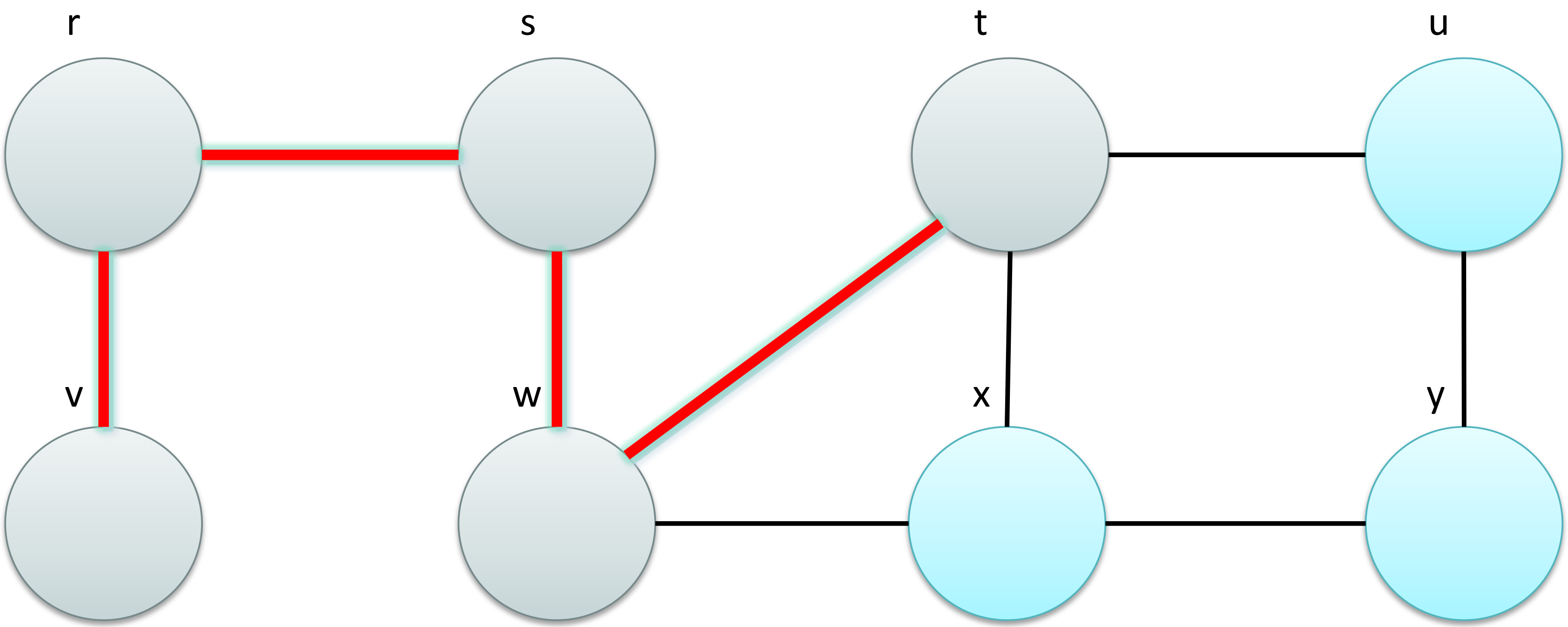
Example

Source = s
Back to r
Back to s
Visit w



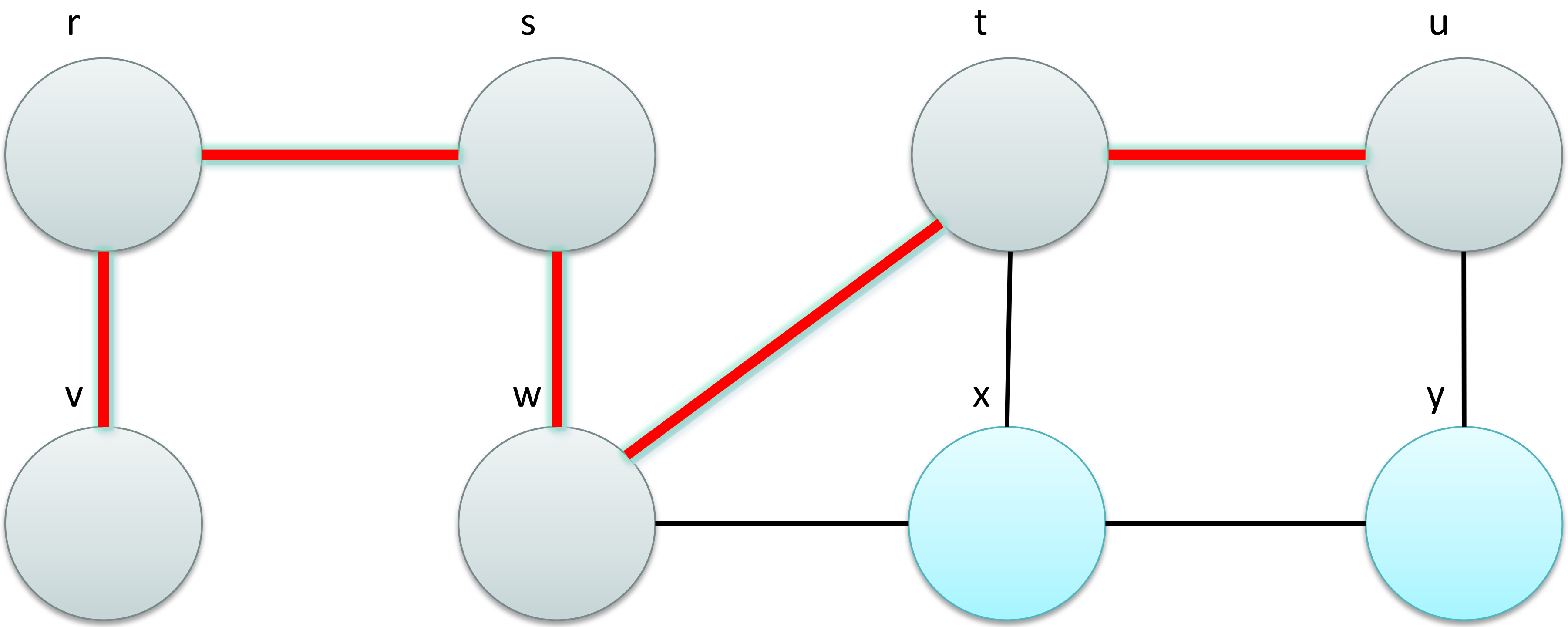
Example

Source = s
Visit w
Visit t



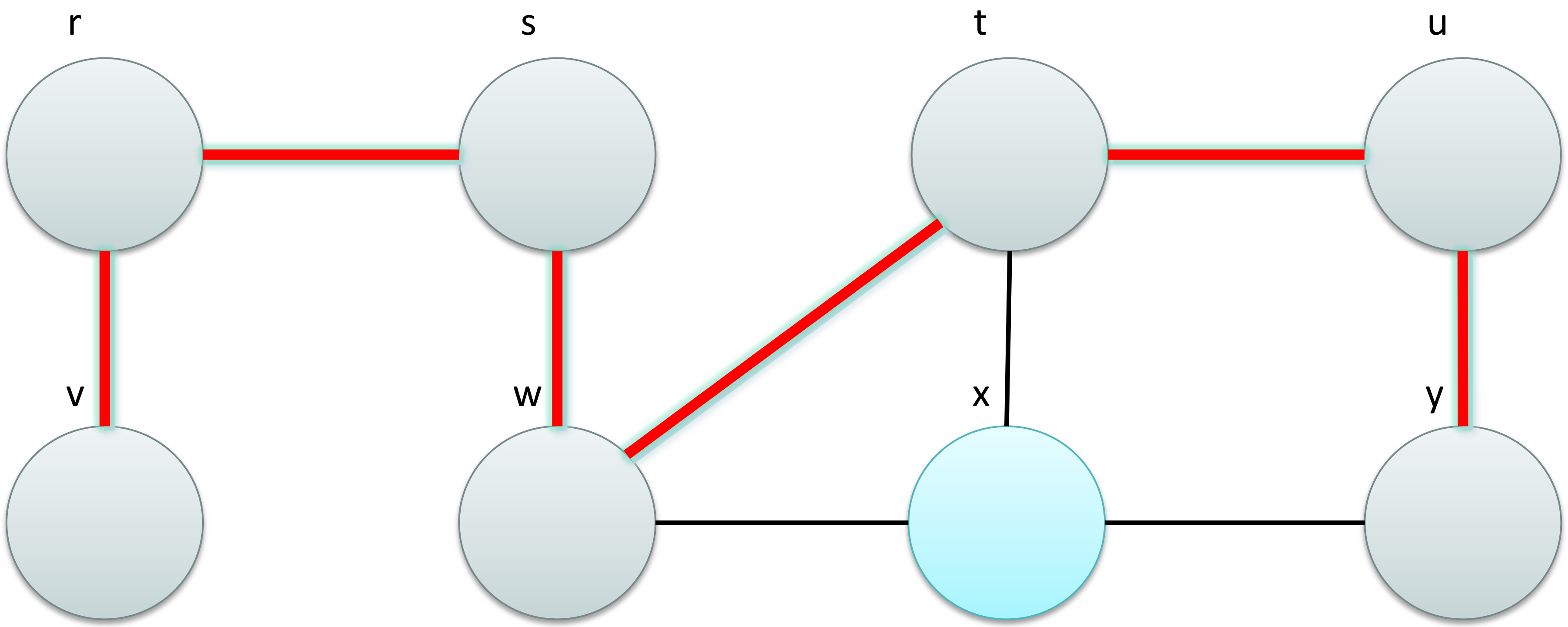
Example

Source = s
Visit w
Visit t
Visit u



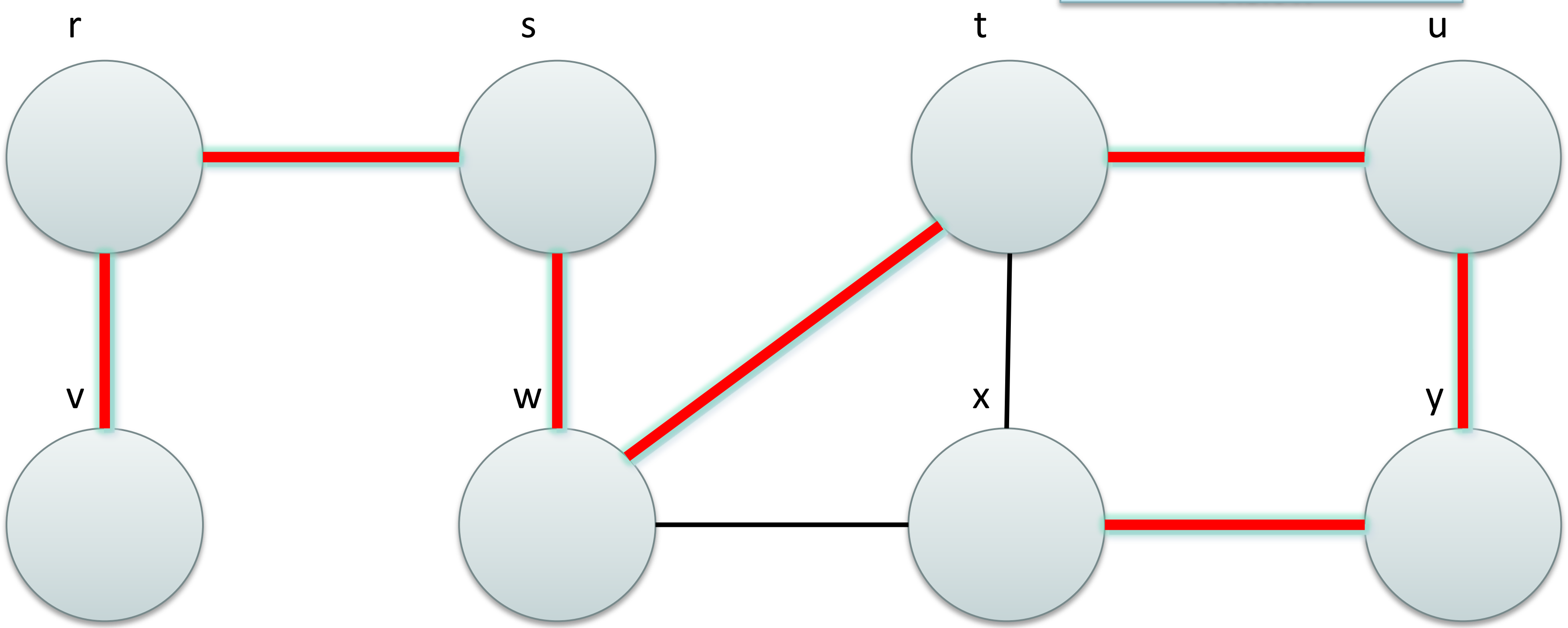
Example

Source = s
Visit w
Visit t
Visit u
Visit y

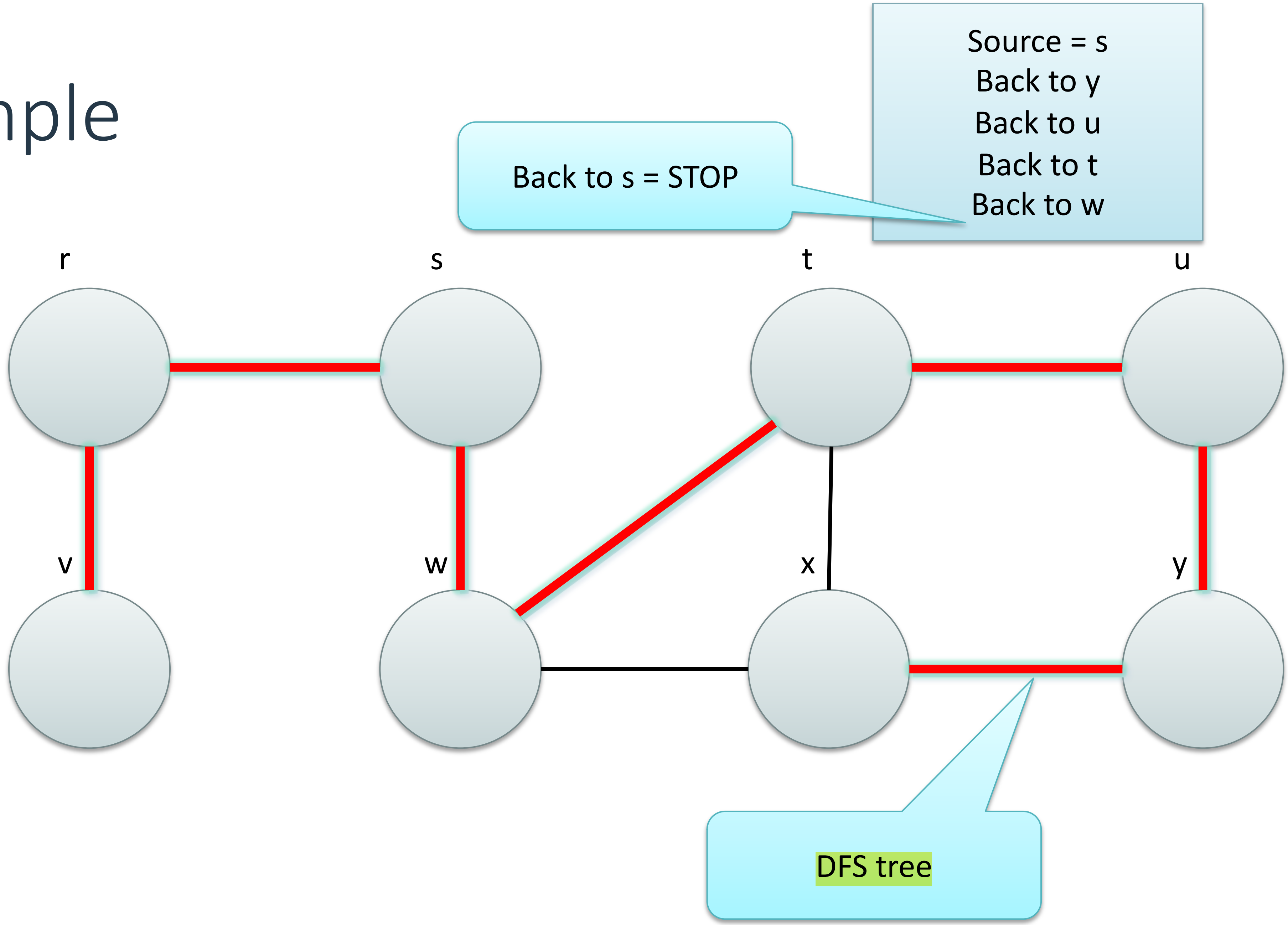


Example

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

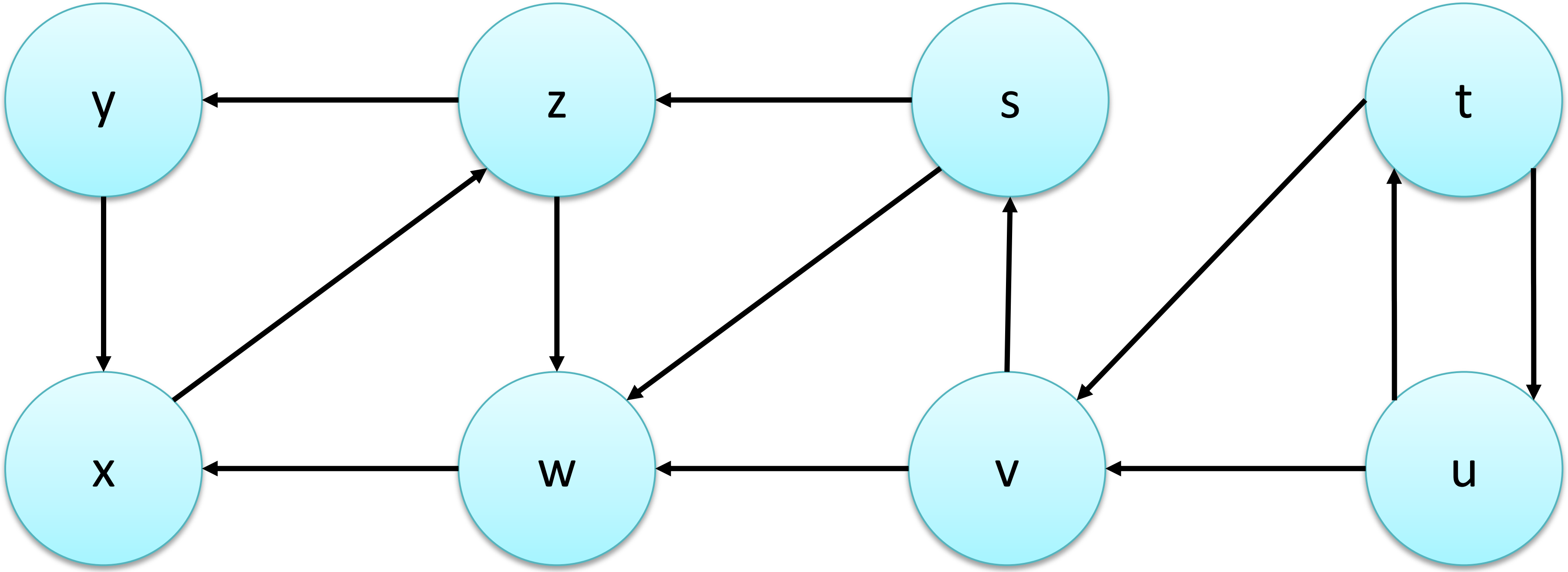


Example


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

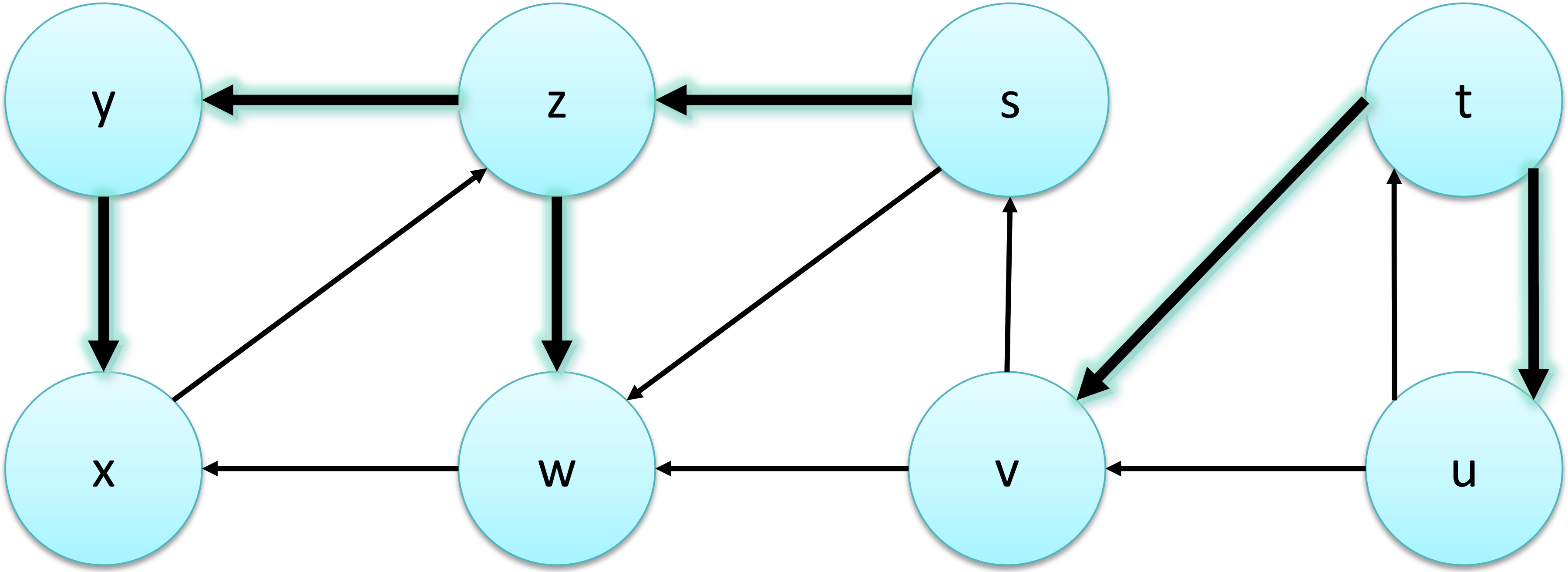
Example

Directed graph



Example

DFS visit
(sources: s, t)



Complexity

- Visits have linear complexity in the graph size
  - BFS : $O(V+E)$
  - DFS : $\Theta(V+E)$

la complessità degli algoritmi è pari circa alla somma del numero di nodi e di archi

- N.B. for dense graphs, E = O(V2)

Resources

- Maths Encyclopedia: http://mathworld.wolfram.com/
- Basic Graph Theory with Applications to Economics http://www.isid.ac.in/~dmishra/mpdoc/lecgraph.pdf
- Application of Graph Theory in real world
- http://prezi.com/tseh1wvpves-/application-of-graph-theory-in-real-world/

Representing and visiting graphs

VISITS IN NETWORKX

Traversal

- Visits are called “traversals” --> è un package
- NetworkX already provides implementations for BFV and DFV, together with other visits strategies

Graph traversal methods

Traversal

Depth First Search

Basic algorithms for depth-first searching the nodes of a graph.

`dfs_edges`(G[, source, depth_limit, ...])	Iterate over edges in a depth-first-search (DFS).
`dfs_tree`(G[, source, depth_limit, ...])	Returns oriented tree constructed from a depth-first-search from source.
`dfs_predecessors`(G[, source, depth_limit, ...])	Returns dictionary of predecessors in depth-first-search from source.
`dfs_successors`(G[, source, depth_limit, ...])	Returns dictionary of successors in depth-first-search from source.
`dfs_preorder_nodes`(G[, source, depth_limit, ...])	Generate nodes in a depth-first-search pre-ordering starting at source.
`dfs_postorder_nodes`(G[, source, ...])	Generate nodes in a depth-first-search post-ordering starting at source.
`dfs_labeled_edges`(G[, source, depth_limit, ...])	Iterate over edges in a depth-first-search (DFS) labeled by type.

Breadth First Search

Basic algorithms for breadth-first searching the nodes of a graph.

`bfs_edges`(G, source[, reverse, depth_limit, ...])	Iterate over edges in a breadth-first-search starting at source.
`bfs_layers`(G, sources)	Returns an iterator of all the layers in breadth-first search traversal.
`bfs_tree`(G, source[, reverse, depth_limit, ...])	Returns an oriented tree constructed from of a breadth-first-search starting at source.
`bfs_predecessors`(G, source[, depth_limit, ...])	Returns an iterator of predecessors in breadth-first-search from source.
`bfs_successors`(G, source[, depth_limit, ...])	Returns an iterator of successors in breadth-first-search from source.
`descendants_at_distance`(G, source, distance)	Returns all nodes at a fixed `distance` from `source` in `G`.
`generic_bfs_edges`(G, source[, neighbors, ...])	Iterate over edges in a breadth-first search.

https://networkx.org/documentation/stable/reference/algorithms/traversal.html

Example

License