Data Structures in Java

Session 16

Instructor: Bert Huang

http://www.cs.columbia.edu/~bert/courses/3134

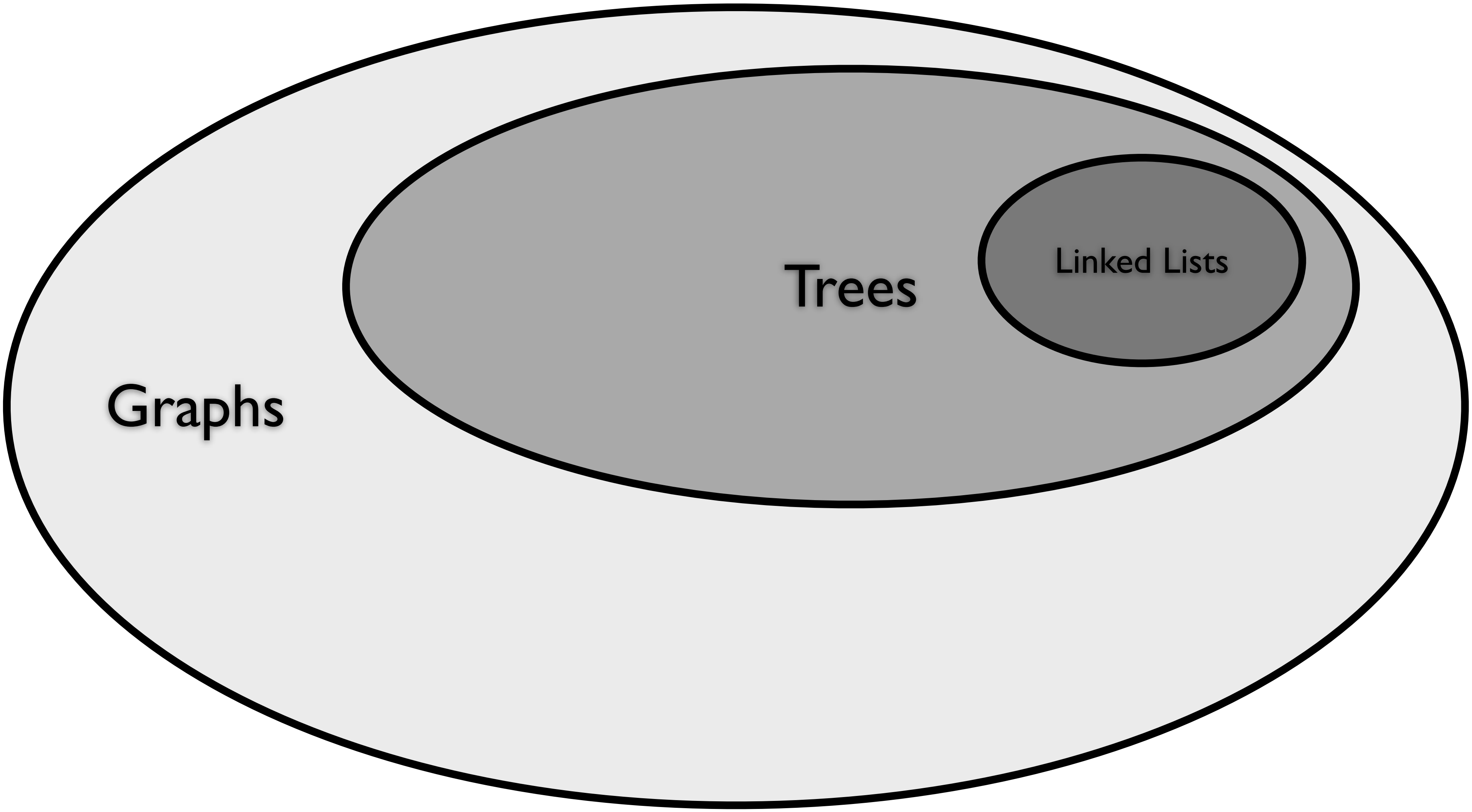
Announcements

- Homework 4 due next class
- Midterm grades posted. Avg: 79/90
- Remaining grades:
  - hw4, hw5, hw6 – 25%
  - Final exam – 30%

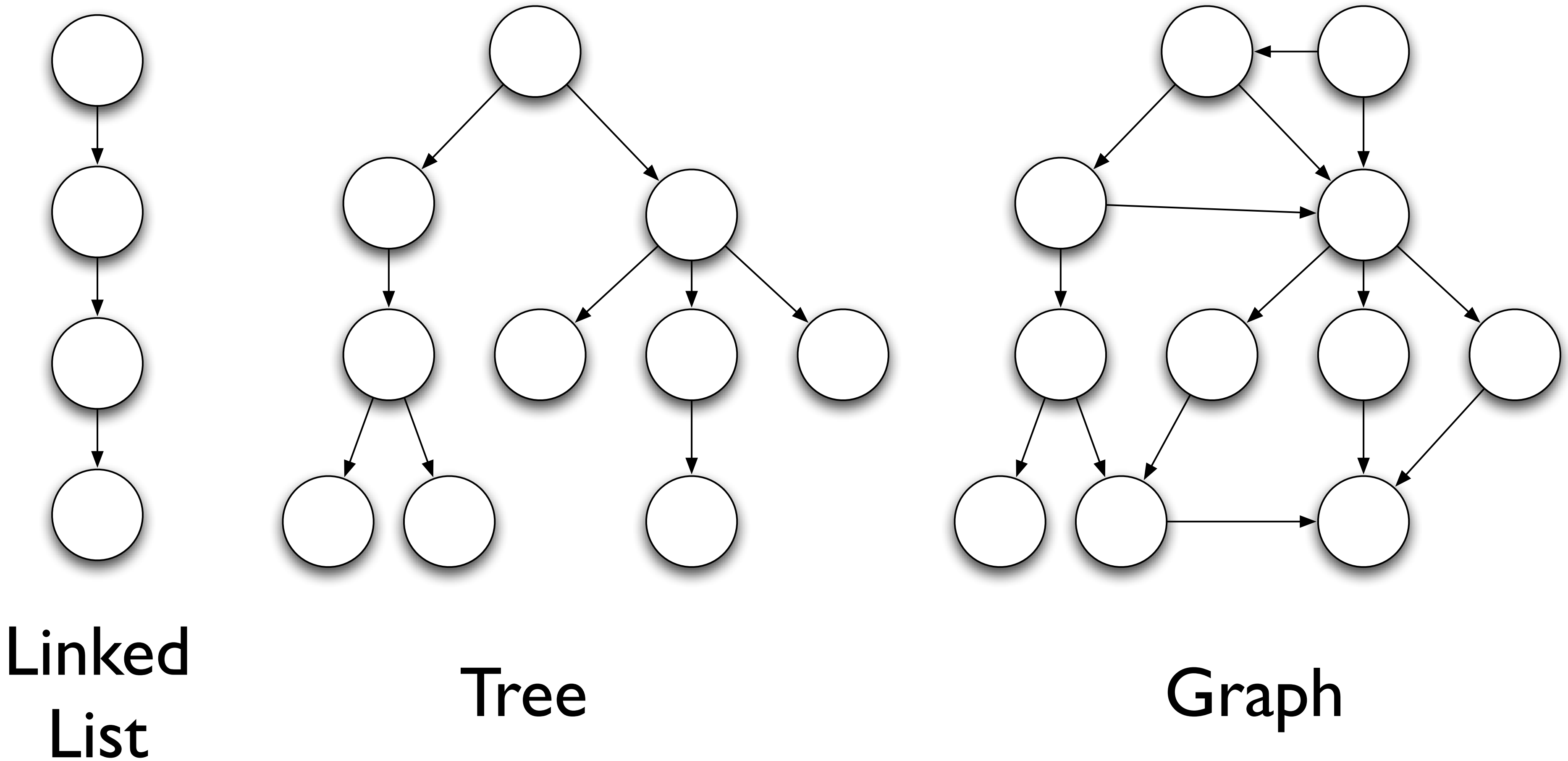
Today's Plan

- Graphs
- Topological Sort
- Shortest Path Algorithms: Dijkstra's

Graphs



Graphs



Graph Terminology

- A **graph** is a set of **nodes** and **edges**
  - nodes aka vertices
  - edges aka arcs, links
- Edges exist between pairs of nodes
  - if nodes x and y share an edge, they are **adjacent**

Graph Terminology

- Edges may have **weights** associated with them
- Edges may be **directed** or **undirected**
- A **path** is a series of adjacent vertices
  - the **length** of a path is the sum of the edge weights along the path (1 if unweighted)
- A **cycle** is a path that starts and ends on a node

Graph Properties

- An undirected graph with no cycles is a tree
- A directed graph with no cycles is a special class called a **directed acyclic graph (DAG)**
- In a **connected** graph, a path exists between every pair of vertices
- A **complete** graph has an edge between every pair of vertices

Graph Applications: A few examples

- Computer networks
- The World Wide Web
- Social networks
- Public transportation
- Probabilistic Inference
- Flow Charts

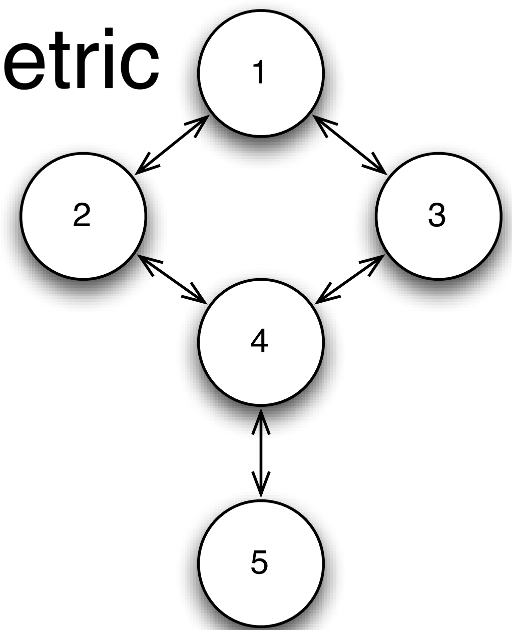
Implementation

- Option 1:
  - Store all nodes in an indexed list
  - Represent edges with **adjacency matrix**
- Option 2:
  - Explicitly store **adjacency lists**

Adjacency Matrices

- 2d-array **A** of boolean variables
- A[i][j] is true when node **i** is adjacent to node **j**
  - If graph is undirected, A is symmetric

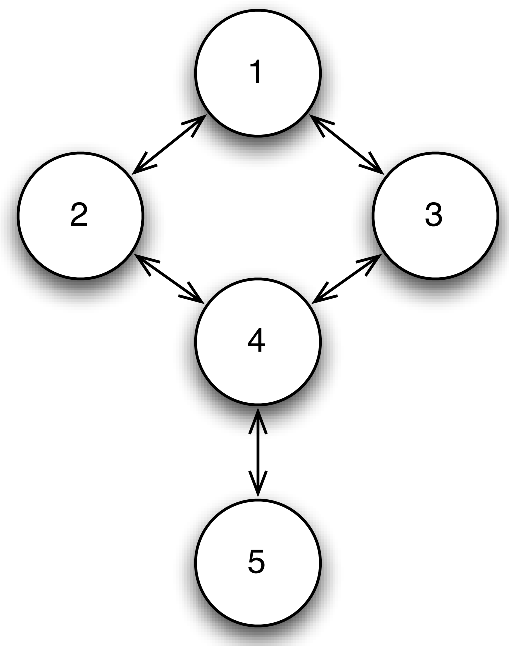
	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	0
3	1	0	0	1	0
4	0	1	1	0	1
5	0	0	0	1	0



Adjacency Lists

- Each node stores references to its neighbors

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



Math Notation for Graphs

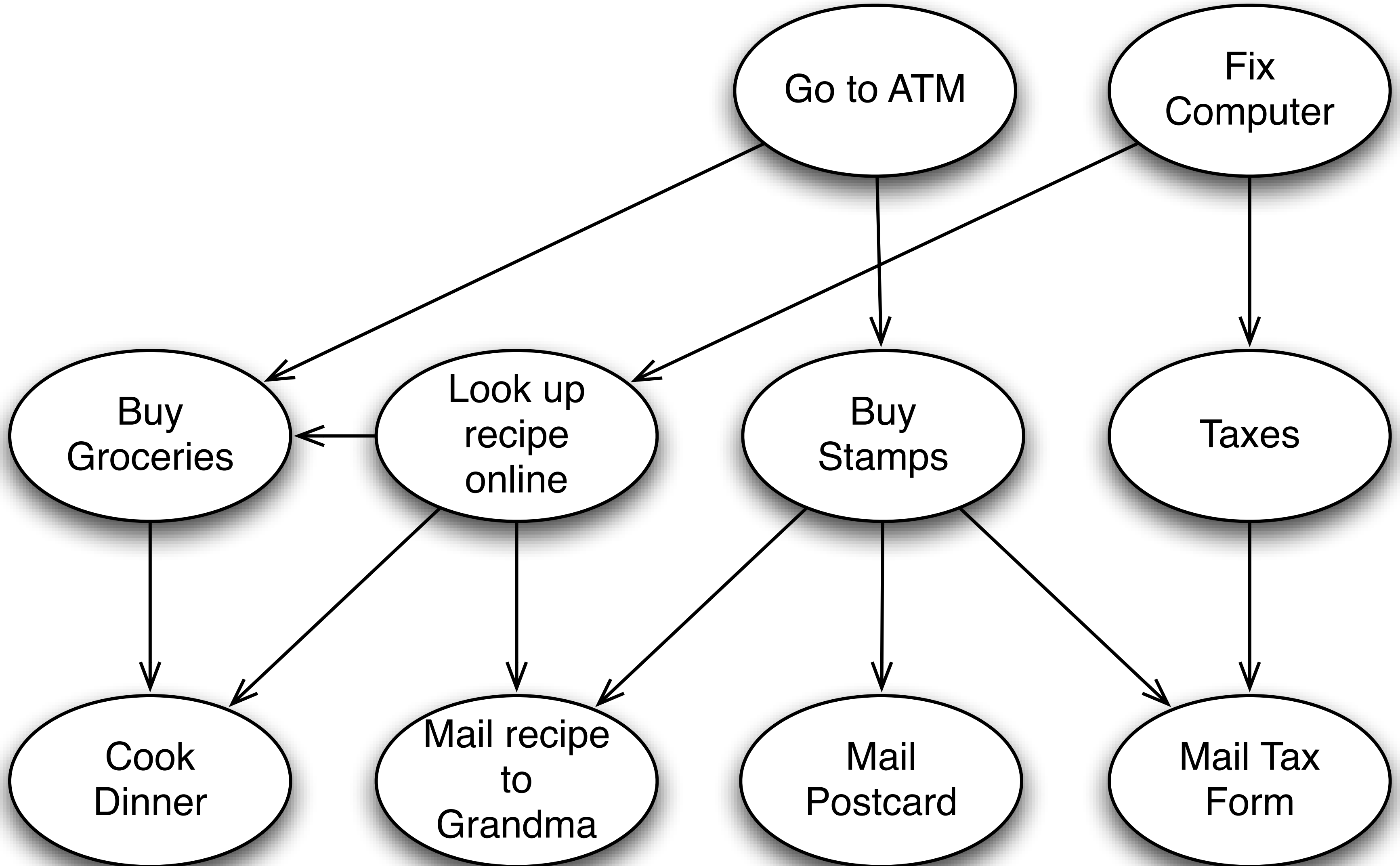
- Set Notation:
  - $v \in V$ (v is in V)
  - $U \cup V$ (union)
  - $U \cap V$ (intersection)
  - $U \subset V$ (U is a subset of V)

- G = {V, E}
- G is the graph
- V is set of vertices
- E is set of edges
- $(v_i, v_j) \in E$
- |V| = N = size of V

Topological Sort

- Problem definition:
  - Given a directed acyclic graph G, order the nodes such that for each edge $(v_i, v_j) \in E$, v_i is before v_j in the ordering.
- e.g., scheduling errands when some tasks depend on other tasks being completed.

Topological Sort Ex.

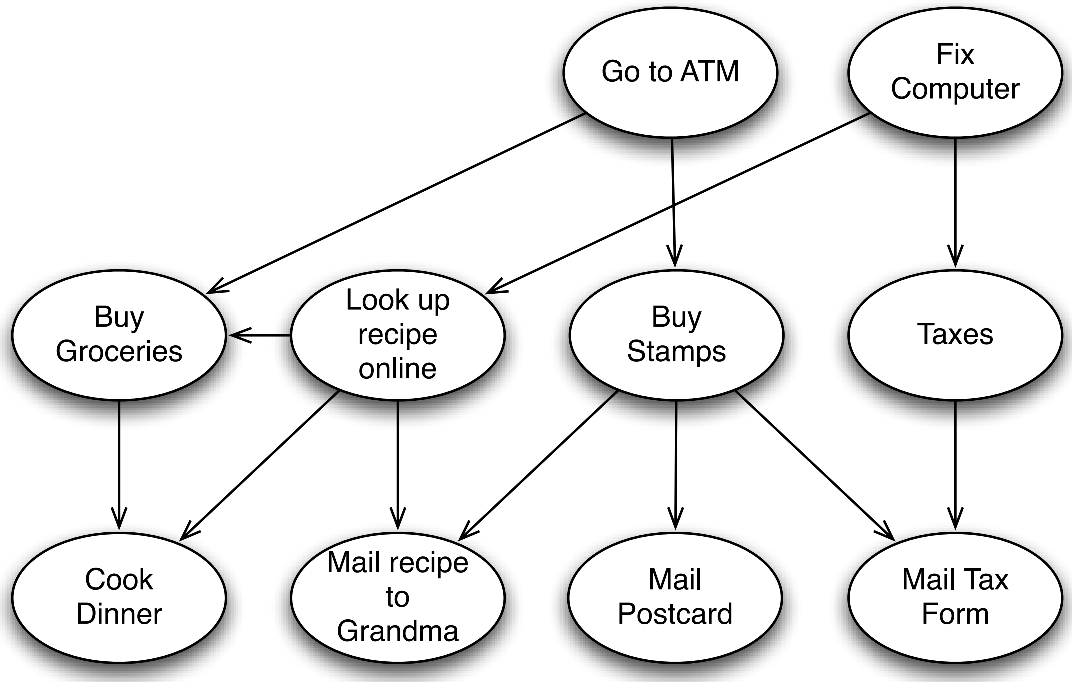


Topological Sort Naïve Algorithm

- **Degree** means # of edges,
  indegree means # of incoming edges
- 1. Compute the **indegree** of all nodes
- 2. Print any node with indegree 0
- 3. Remove the node we just printed. Go to 1.
- Which nodes' indegrees change?

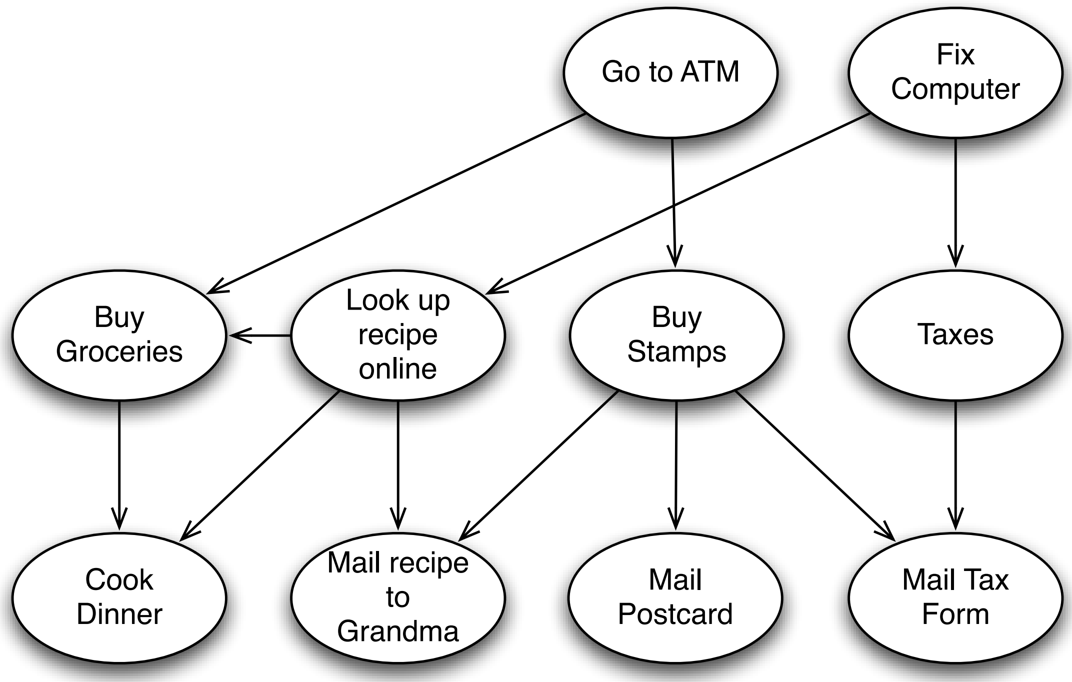
Topological Sort Better Algorithm

- 1. Compute all indegrees
- 2. Put all indegree 0 nodes into a Collection
- 3. Print and remove a node from Collection
- 4. Decrement indegrees of the node's neighbors.
- 5. If any neighbor has indegree 0, place in Collection. Go to 3.



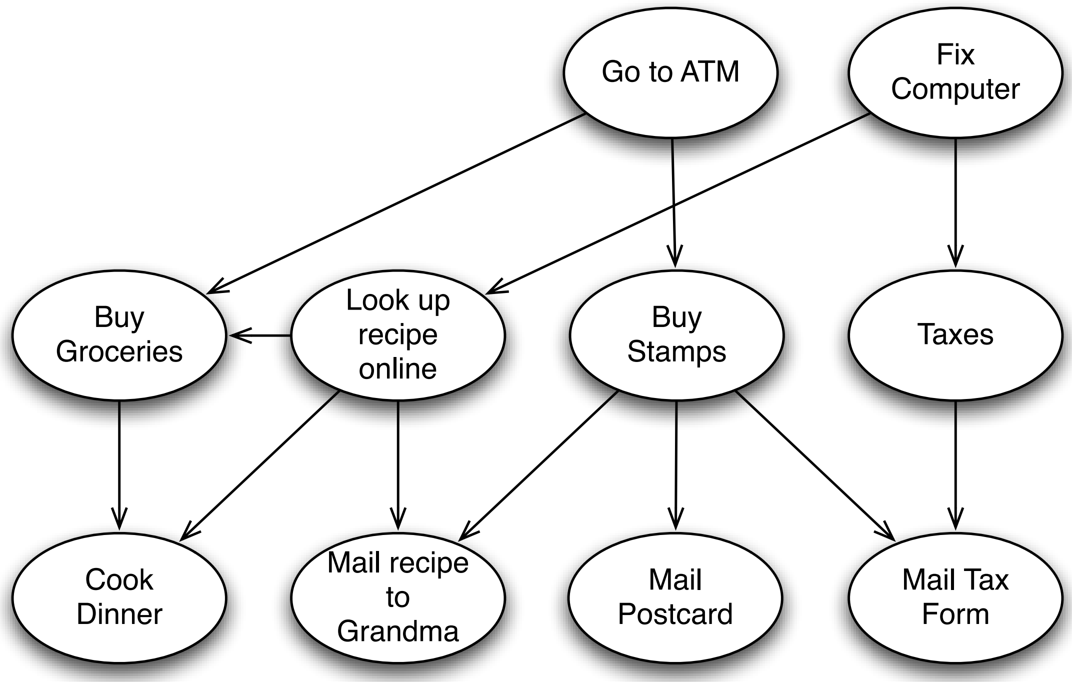
ATM	comp	grocer-ies	recipe	stamps	taxes	cook	grand-ma	post-card	mail taxes
0	0	2	1	1	1	2	2	1	2

Queue
comp
ATM



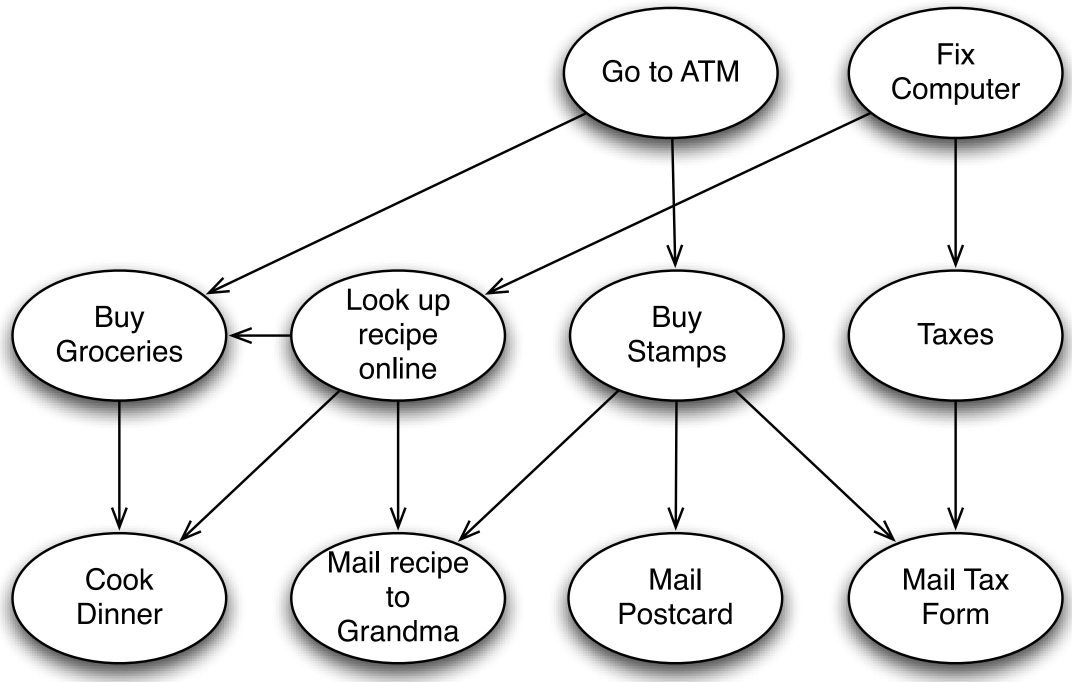
ATM	comp	grocer-ies	recipe	stamps	taxes	cook	grand-ma	post-card	mail taxes
-	-	2	1	1	1	2	2	1	2

Queue
comp



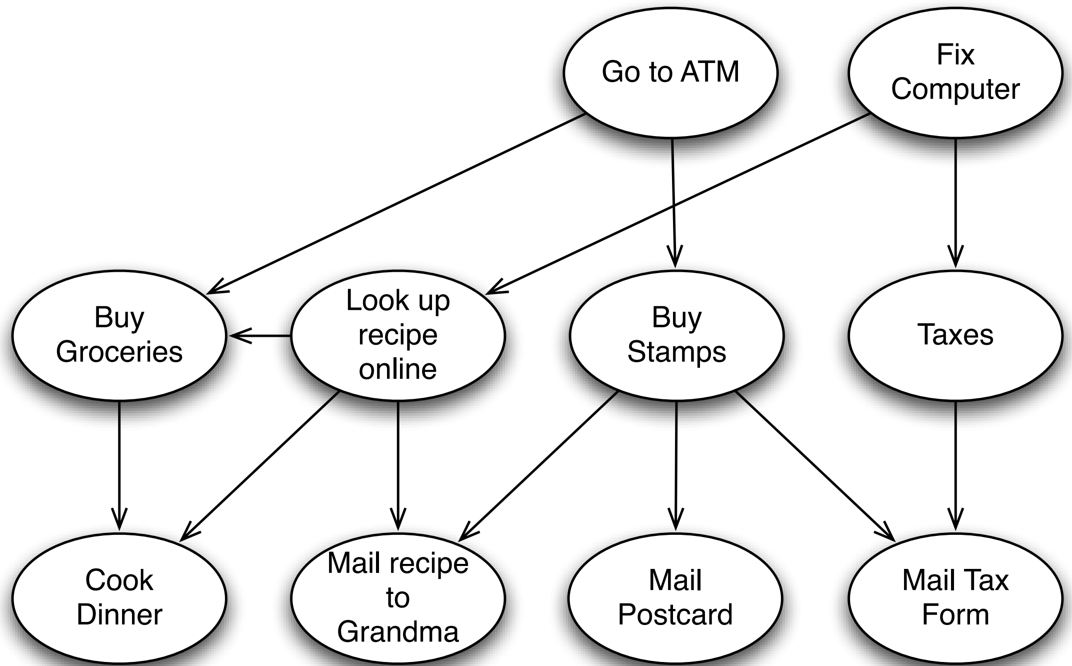
ATM	comp	grocer-ies	recipe	stamps	taxes	cook	grand-ma	post-card	mail taxes
0	-	~~2~~ 1	1	~~1~~ 0	1	2	2	1	2

Queue
stamps
comp



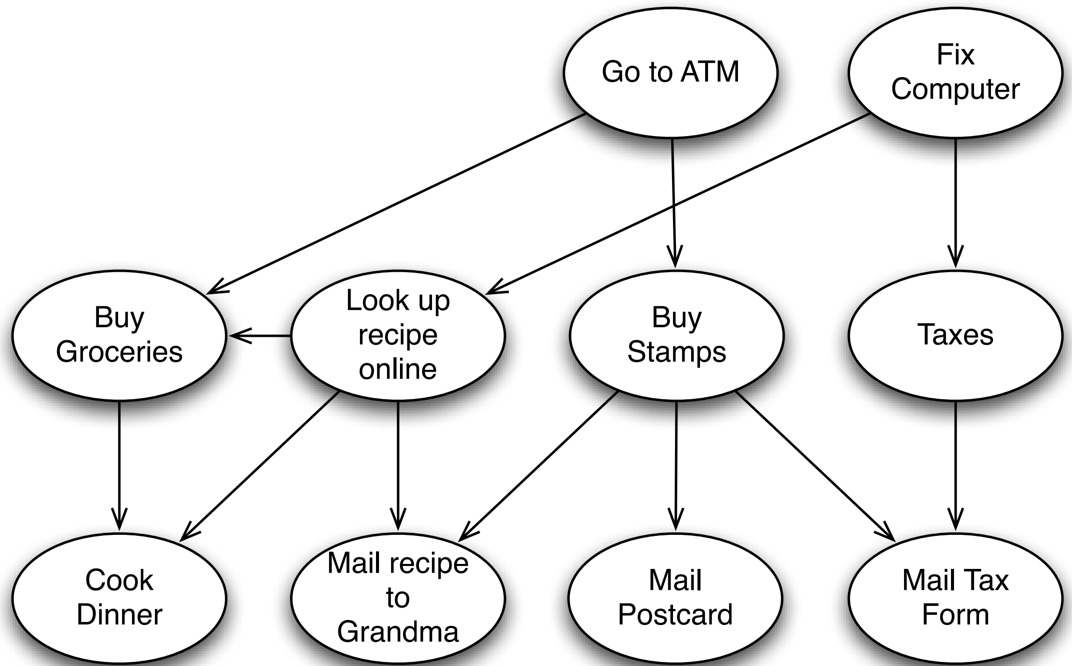
ATM	comp	grocer-ies	recipe	stamps	taxes	cook	grand-ma	post-card	mail taxes
0	-	~~2~~ 1	1	-	1	2	2	1	2

Queue
stamps



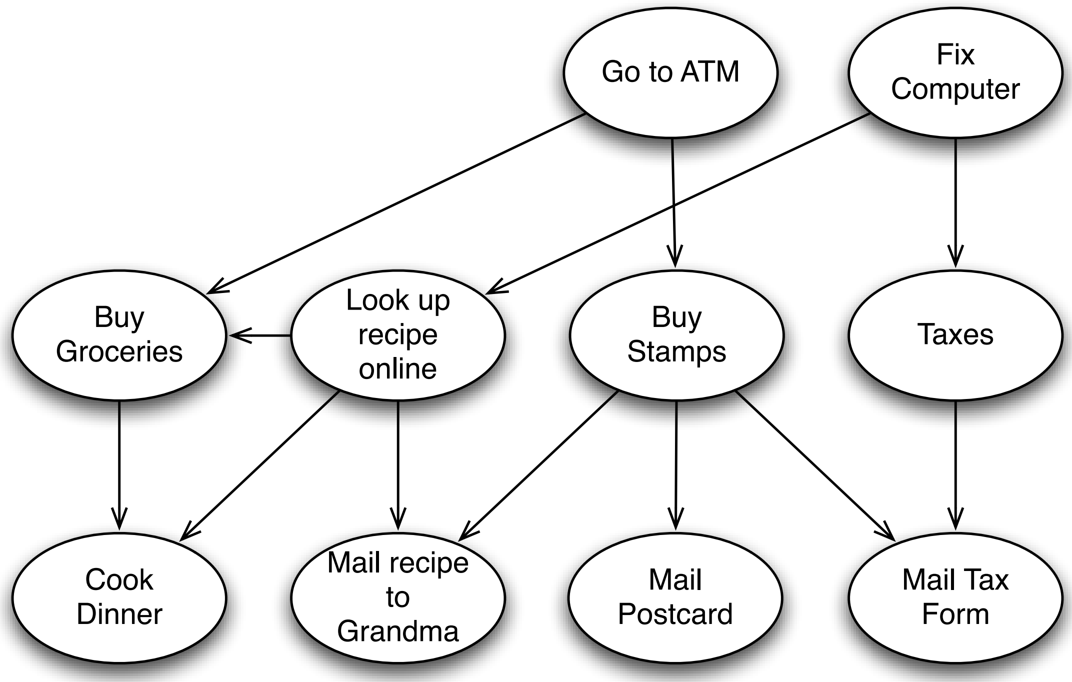
ATM	comp	grocer-ies	recipe	stamps	taxes	cook	grand-ma	post-card	mail taxes
0	1	~~2~~ 1	~~1~~ 0	-	~~1~~ 0	2	2	1	2

Queue
recipe
taxes
stamps



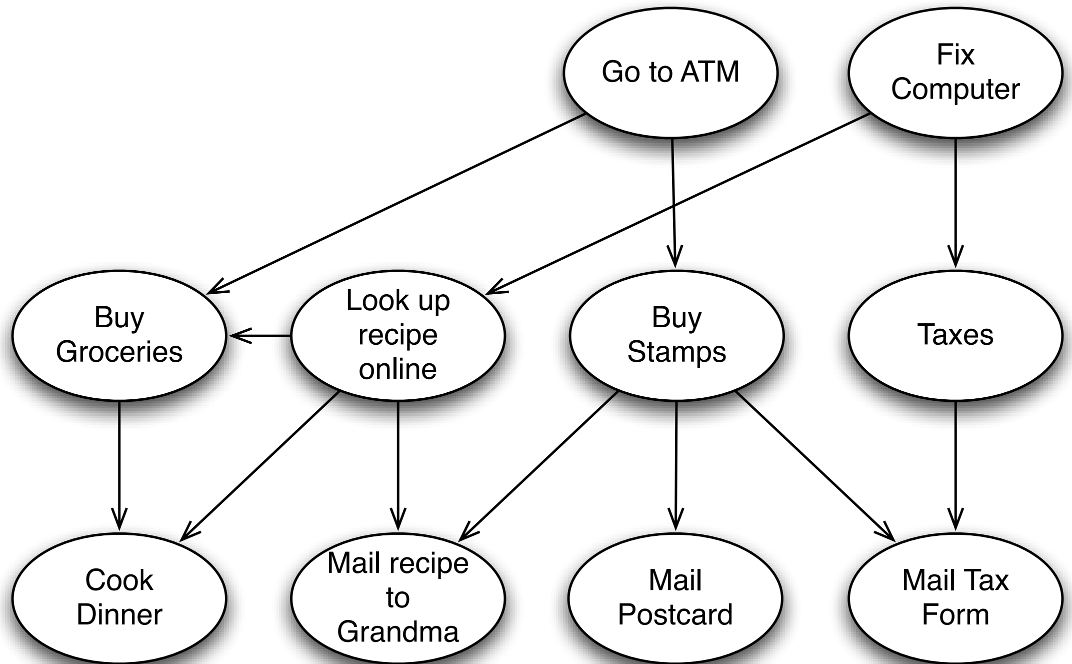
ATM	comp	grocer-ies	recipe	stamps	taxes	cook	grand-ma	post-card	mail taxes
0	1	~~2~~ 1	-	-	-	2	2	1	2

Queue
recipe
taxes



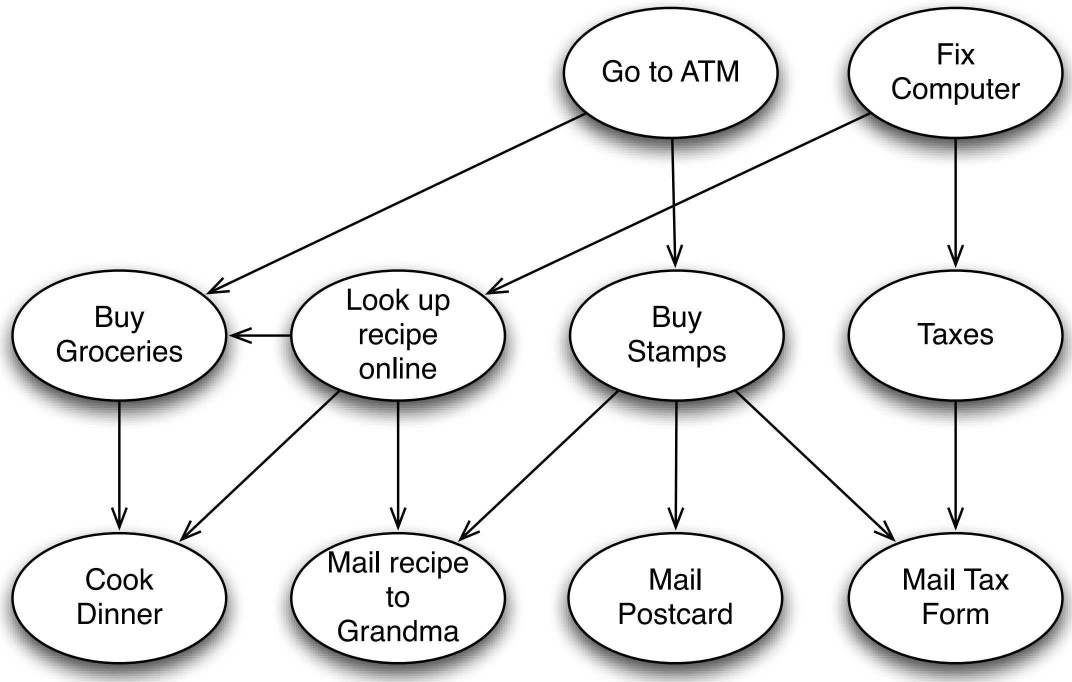
ATM	comp	grocer-ies	recipe	stamps	taxes	cook	grand-ma	post-card	mail taxes
0	**1**	~~2~~ 1	-	**2**	-	2	~~2~~ 1	~~1~~ 0	~~2~~ 1

Queue
postcard
recipe
taxes



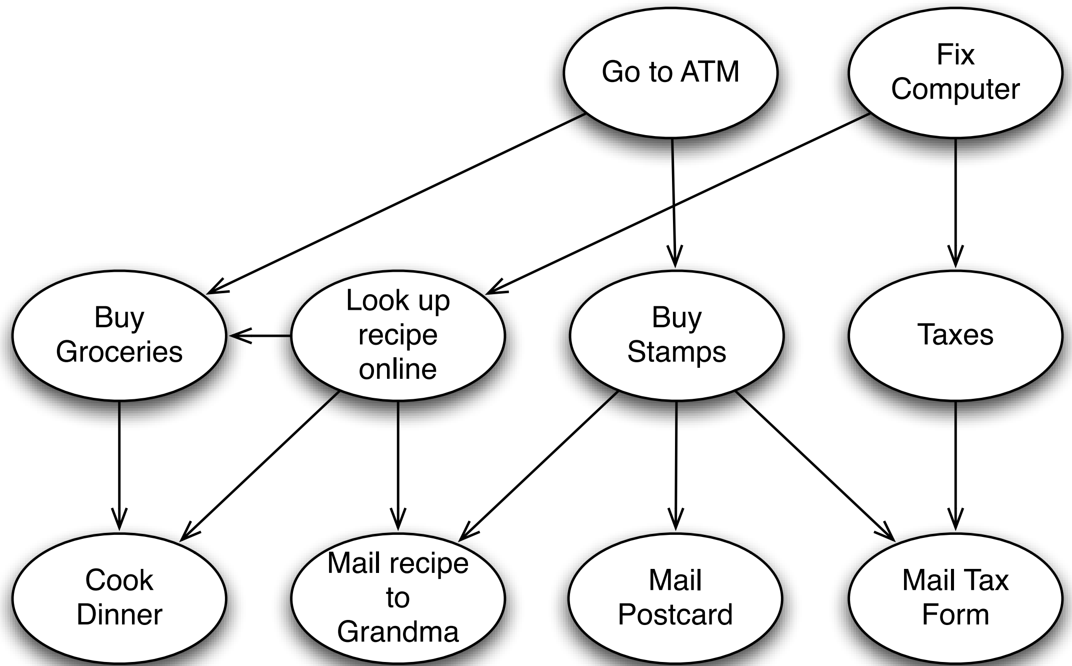
ATM	comp	grocer-ies	recipe	stamps	taxes	cook	grand-ma	post-card	mail taxes
0	1	~~2~~ 1	-	2	-	2	~~2~~ 1	-	~~2~~ 1

Queue
postcard
recipe



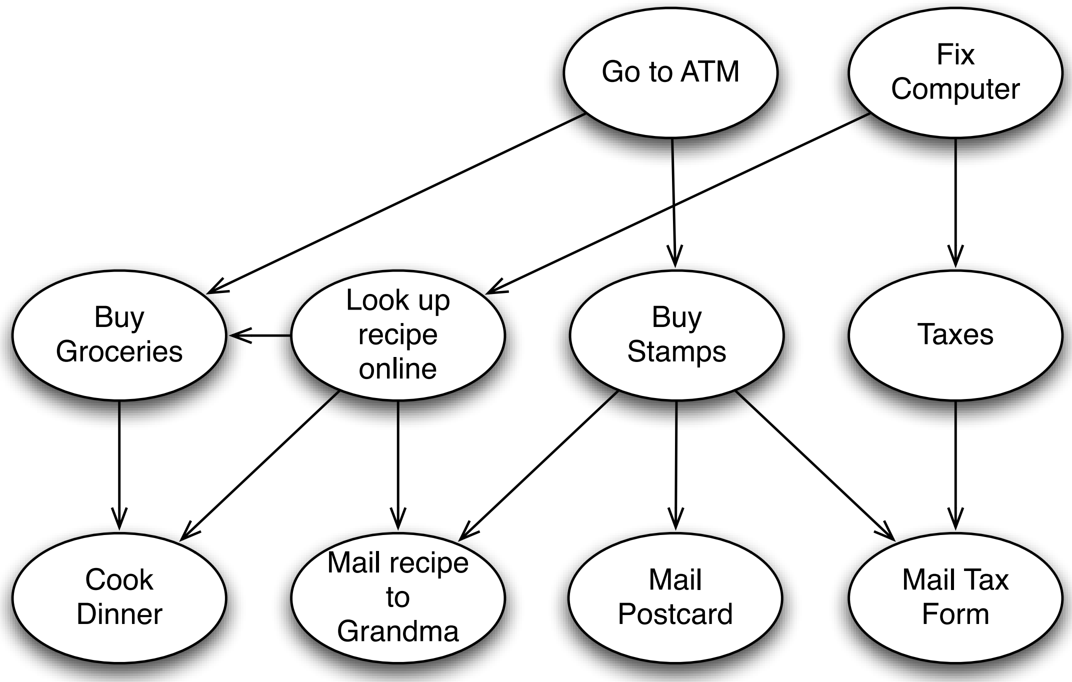
ATM	comp	grocer-ies	recipe	stamps	taxes	cook	grand-ma	post-card	mail taxes
0	**1**	~~2~~ 1	-	**2**	**3**	2	~~2~~ 1	-	~~2~~ ~~1~~ 0

Queue
mail taxes
postcard
recipe



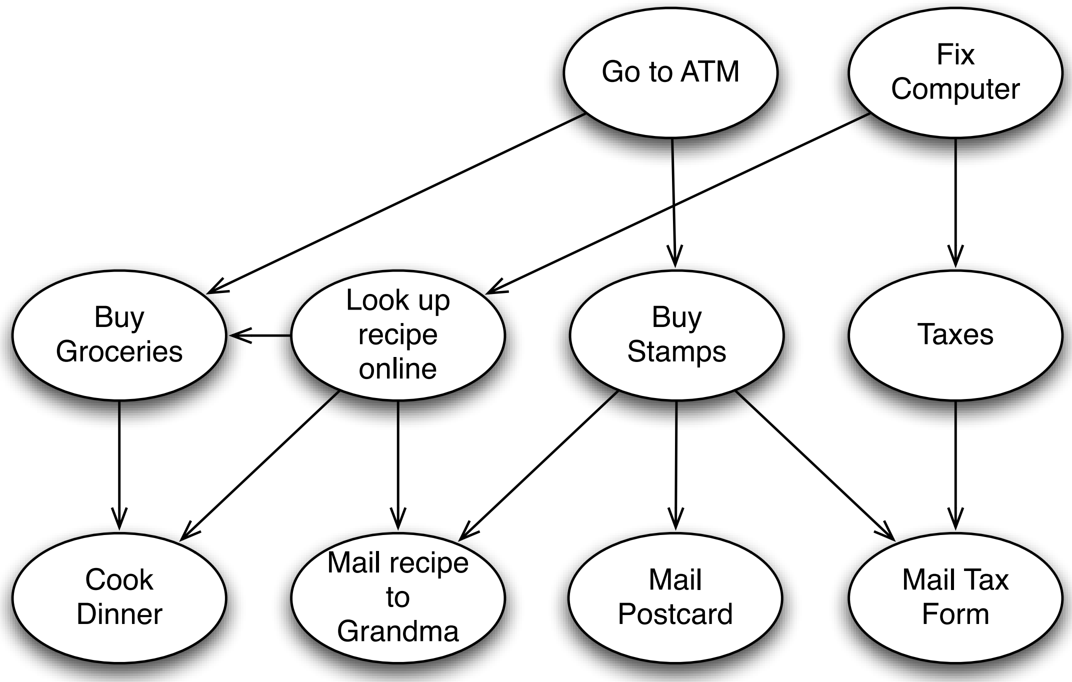
ATM	comp	grocer-ies	recipe	stamps	taxes	cook	grand-ma	post-card	mail taxes
0	**1**	~~2~~ 1	-	**2**	**3**	2	~~2~~ 1	-	-

Queue
mail taxes
postcard



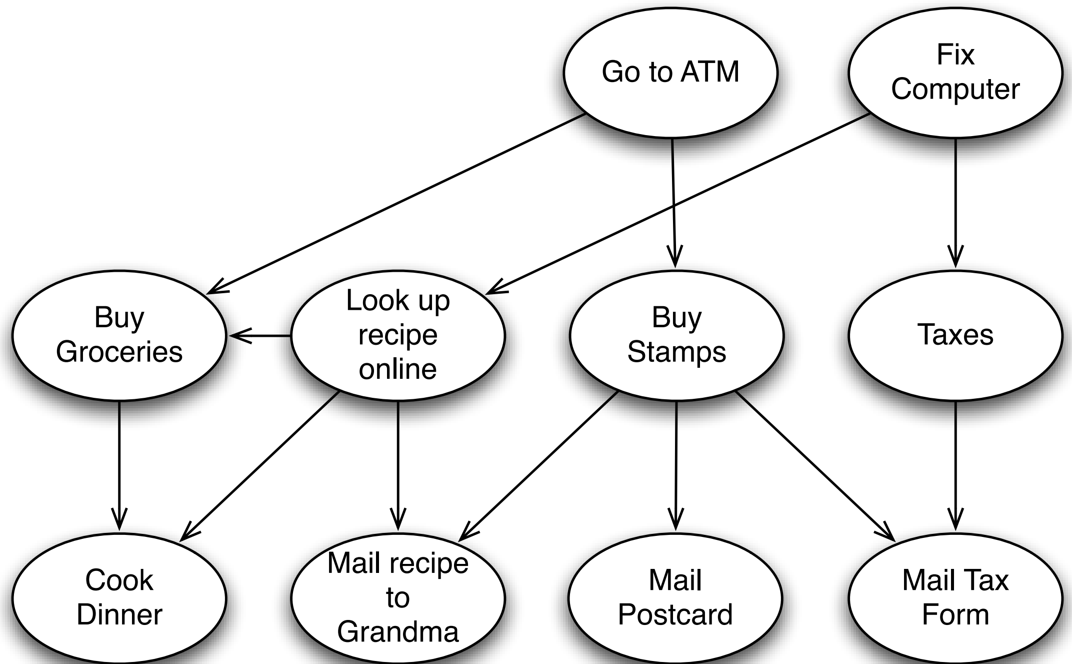
ATM	comp	grocer-ies	recipe	stamps	taxes	cook	grand-ma	post-card	mail taxes
0	**1**	~~2~~ ~~1~~ 0	**4**	**2**	**3**	~~2~~ 1	~~2~~ ~~1~~ 0	-	-

Queue
groceries
grandma
mail taxes
postcard



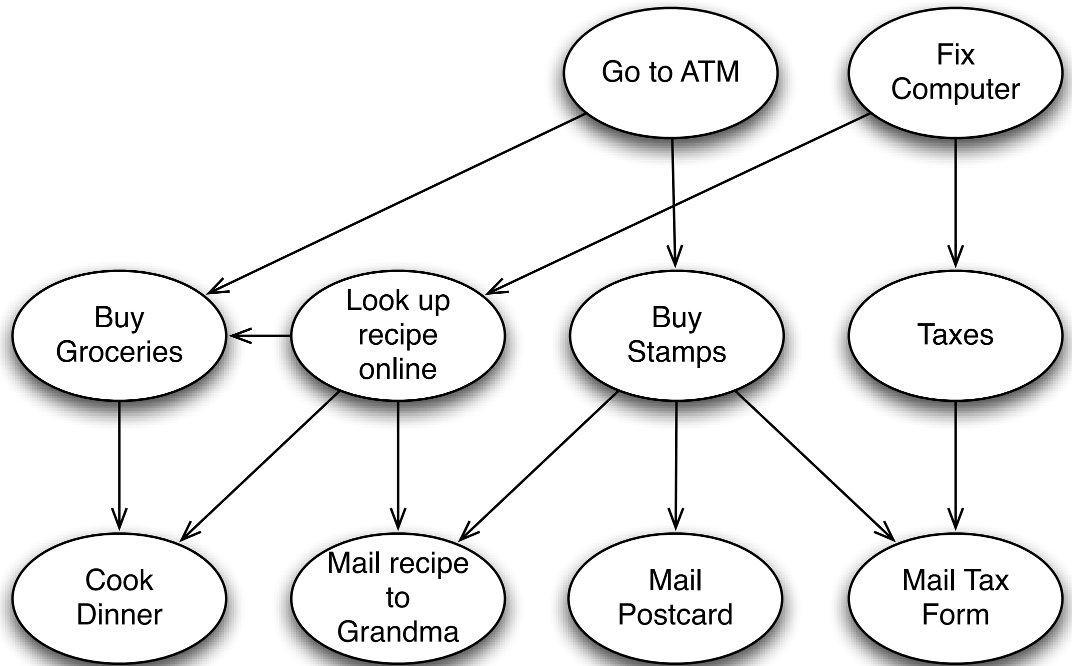
ATM	comp	grocer-ies	recipe	stamps	taxes	cook	grand-ma	post-card	mail taxes
0	1	-	4	2	3	~~2~~ 1	-	-	-

Queue



ATM	comp	grocer-ies	recipe	stamps	taxes	cook	grand-ma	post-card	mail taxes
0	1	8	4	2	3	~~2~~ ~~1~~ 0	7	5	6

Queue



ATM	comp	grocer-ies	recipe	stamps	taxes	cook	grand-ma	post-card	mail taxes
0	1	8	4	2	3	9	7	5	6

Topological Sort Running time

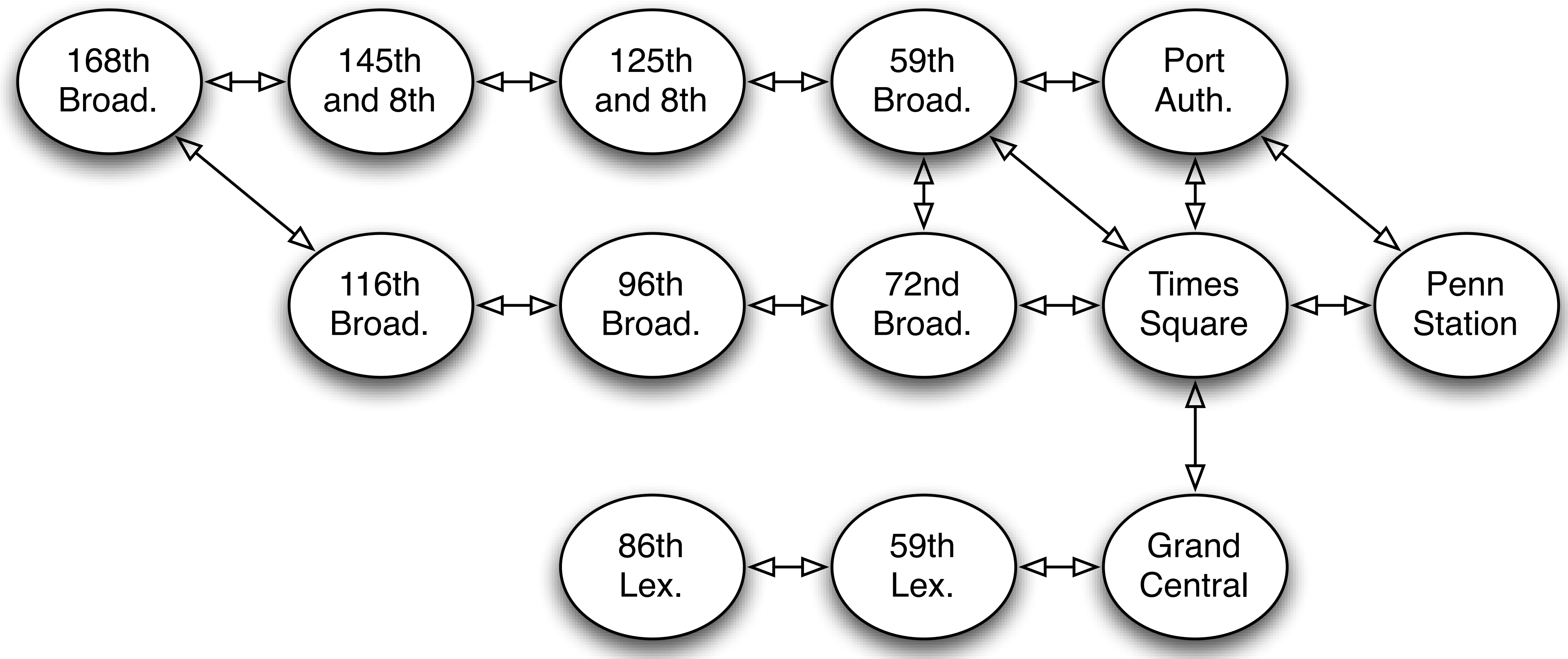
- Initial indegree computation: O(|E|)
  - Unless we update indegree as we build graph
- |V| nodes must be enqueued/dequeued
- Dequeue requires operation for outgoing edges
- Each edge is used, but never repeated
- Total running time O(|V| + |E|)

Shortest Path

- Given **G = (V,E)**, and a node **s** ∈ **V**, find the shortest (weighted) path from **s** to every other vertex in **G**.
- Motivating example: subway travel
  - Nodes are junctions, transfer locations
  - Edge weights are estimated time of travel

Approximate MTA Express Stop Subgraph

- A few inaccuracies (don't use this to plan any trips)

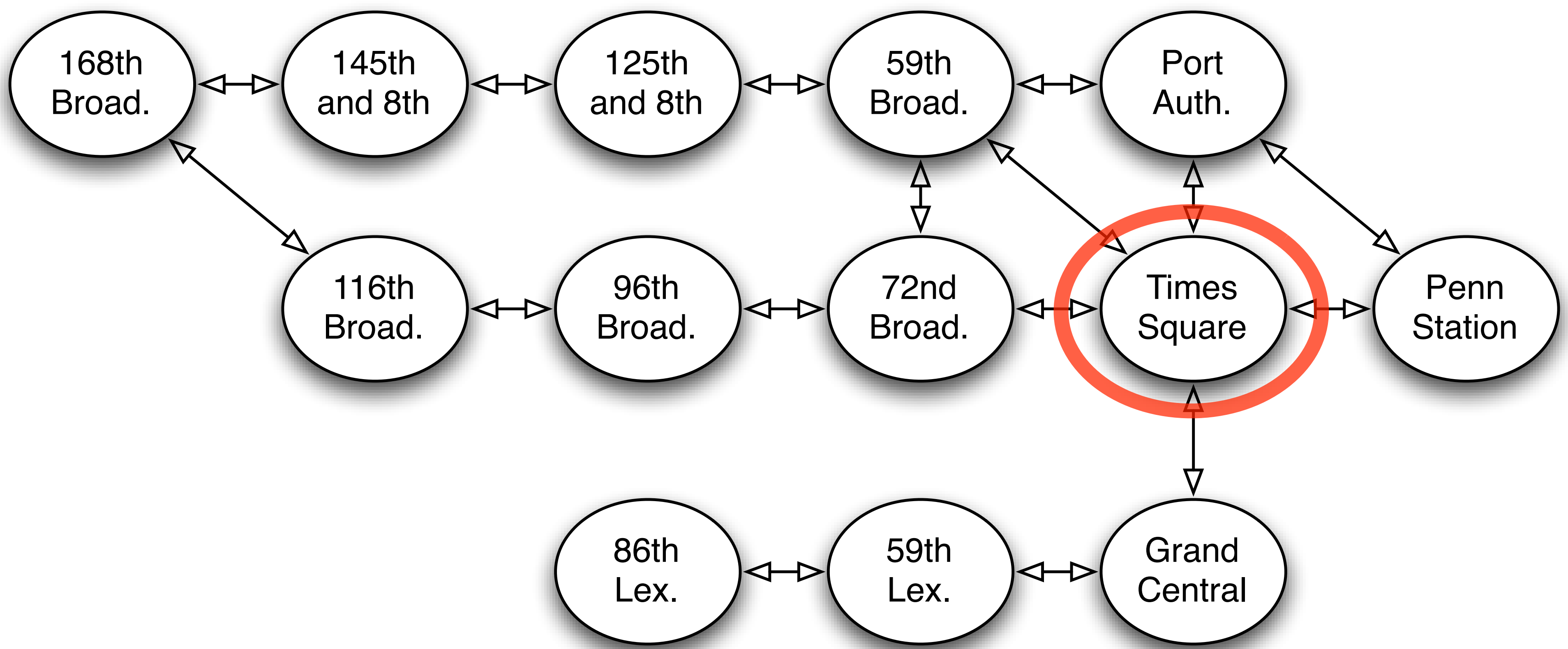


Breadth First Search

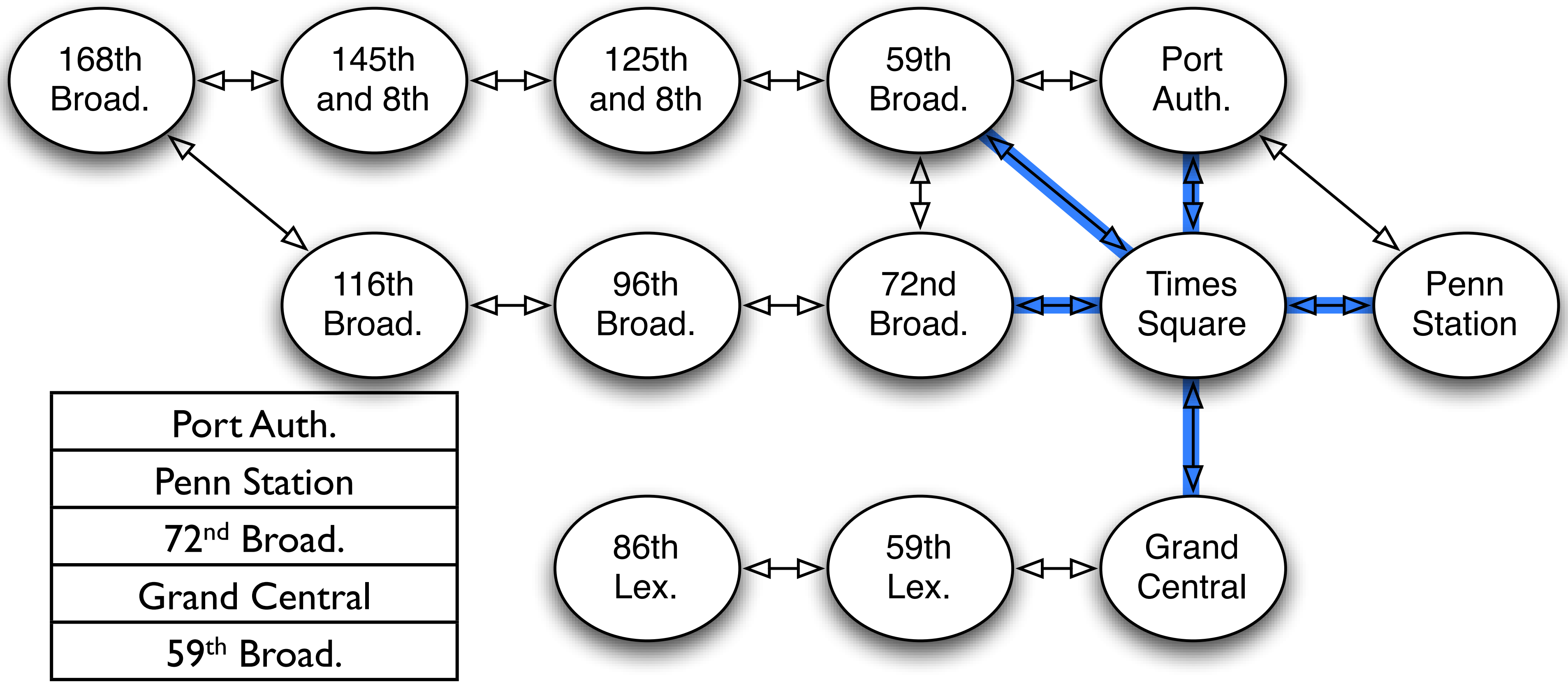
- Like a level-order traversal
- Find all adjacent nodes (level 1)
- Find *new* nodes adjacent to level 1 nodes (level 2)
- ... and so on
- We can implement this with a queue

Unweighted Shortest Path Algorithm

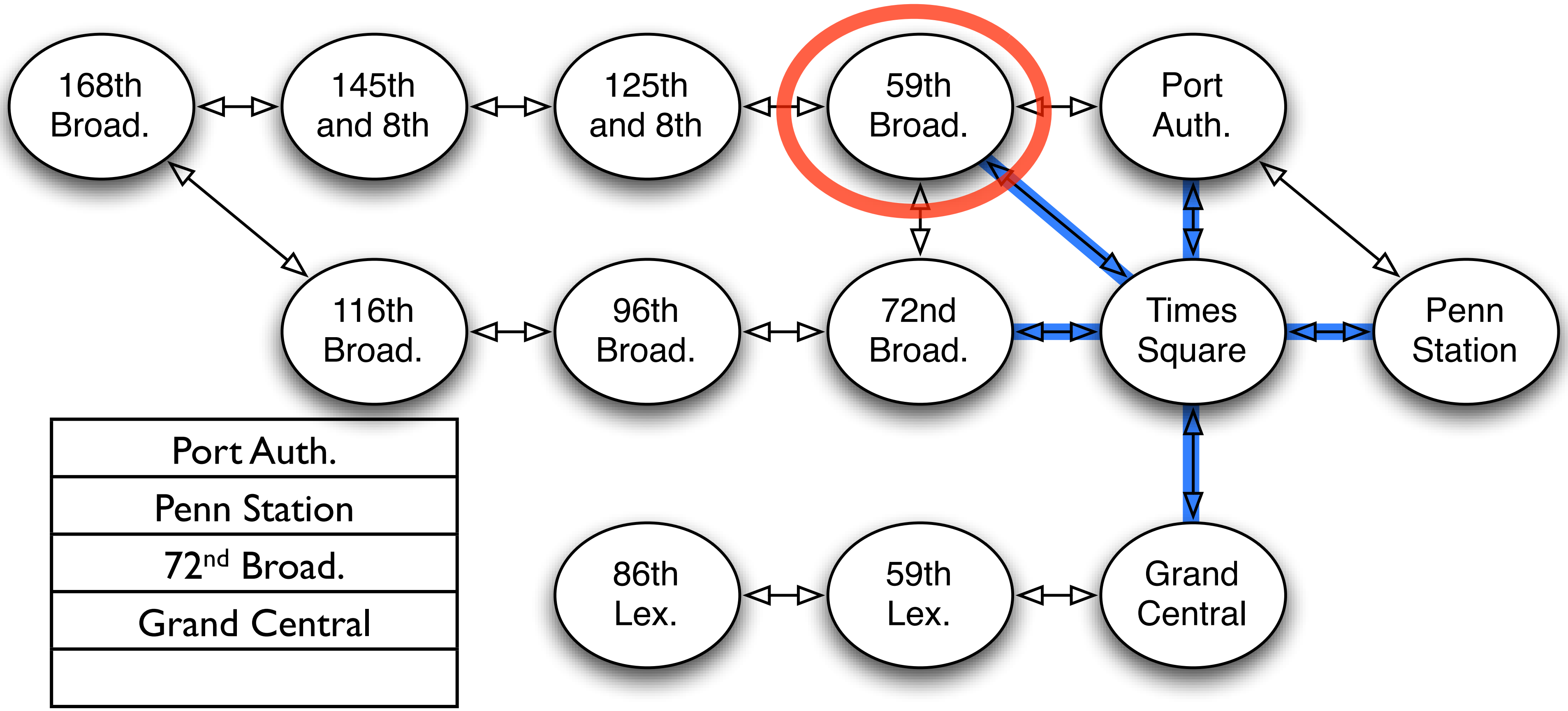
- Set node s’ distance to 0 and enqueue s.
- Then repeat the following:
  - Dequeue node **v**. For unset neighbor **u**:
    - set neighbor **u**’s distance to **v**’s distance +1
    - mark that we reached **v** from **u**
    - enqueue **u**



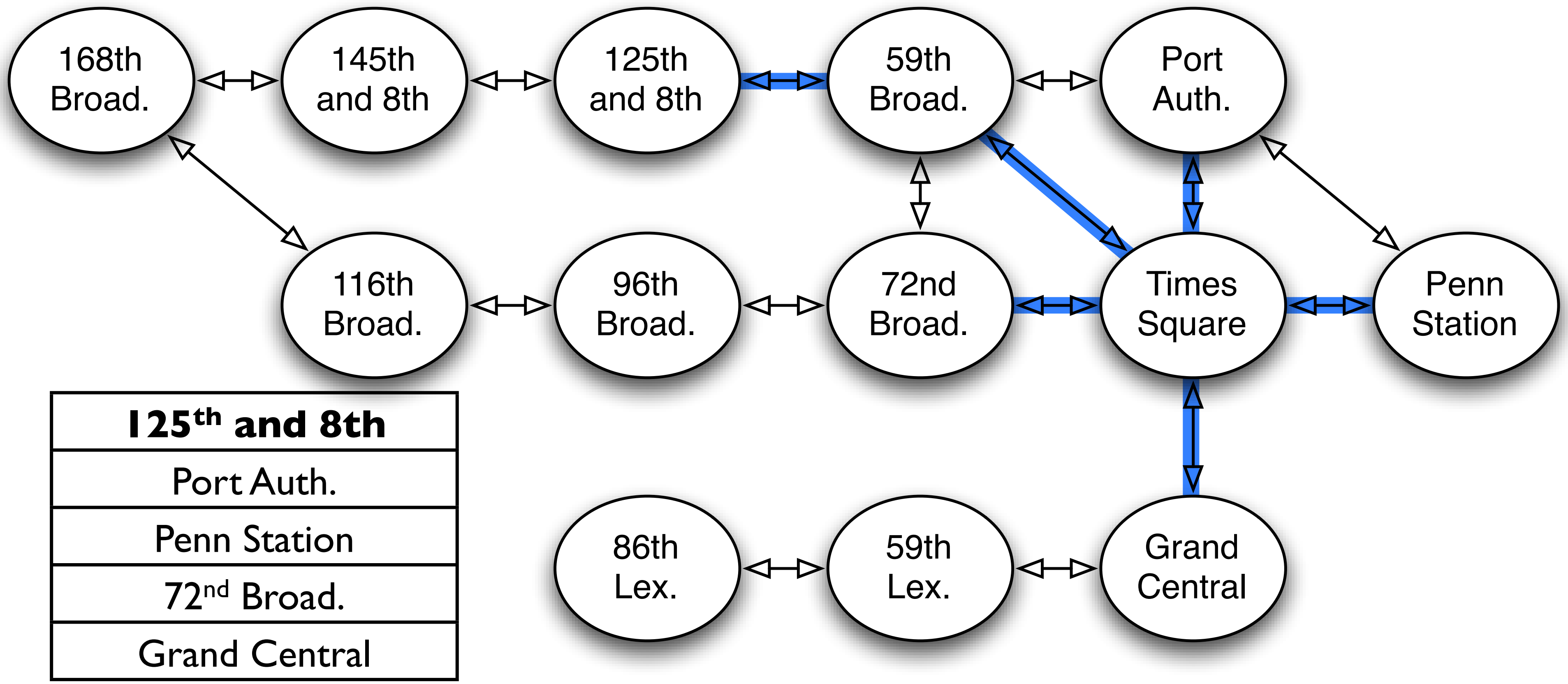
	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist									0				
prev									source				



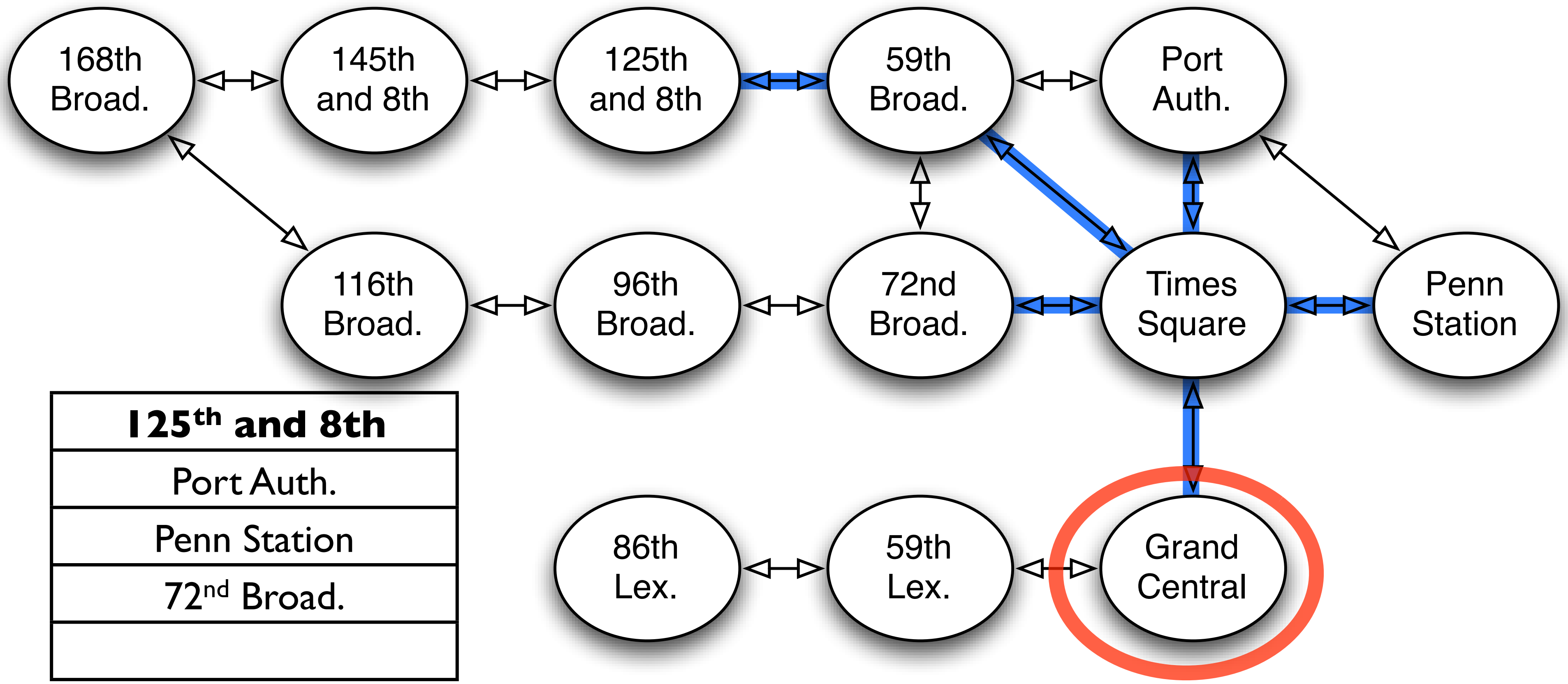
	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist				1	1			1	0	1			1
prev				Times Sq.	Times Sq.			Times Sq.	source	Times Sq.			Times Sq.



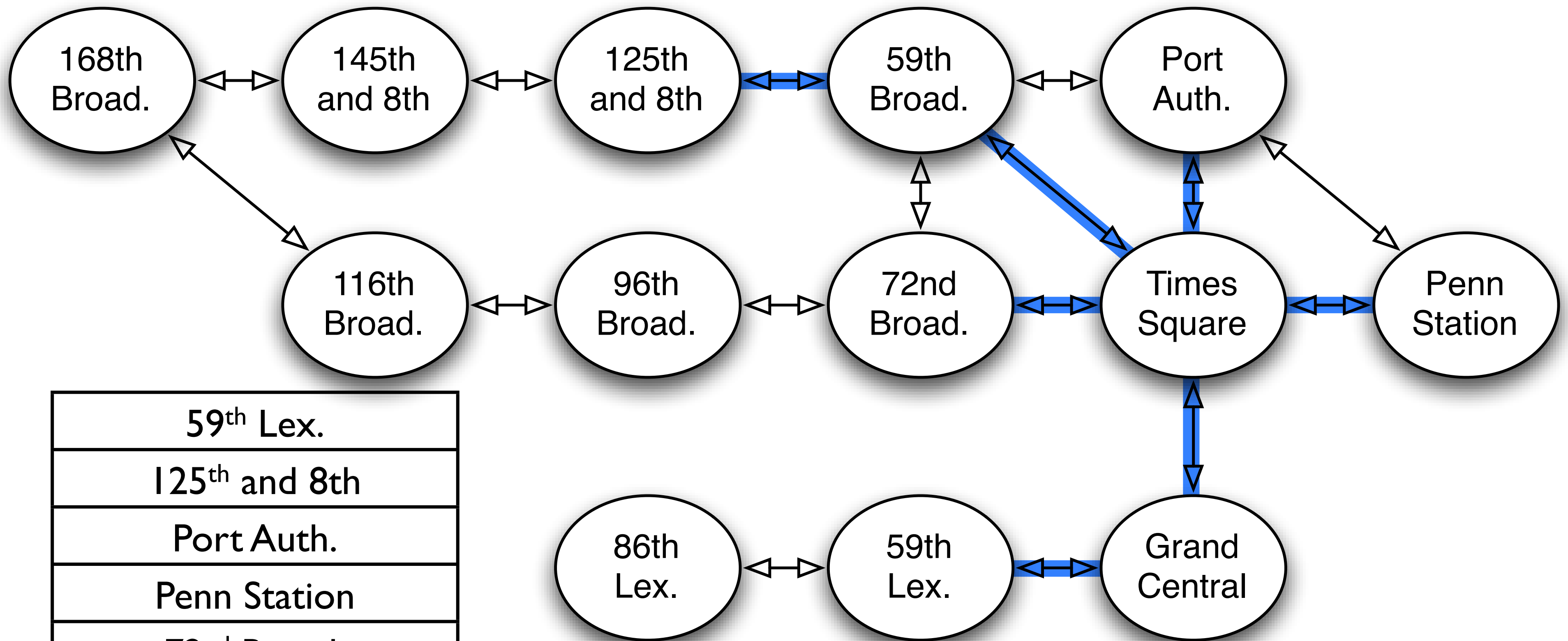
	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist				1	1			1	0	1			1
prev				Times Sq.	Times Sq.			Times Sq.	source	Times Sq.			Times Sq.



	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist			2	1	1			1	0	1			1
prev			59th Broad.	Times Sq.	Times Sq.			Times Sq.	source	Times Sq.			Times Sq.

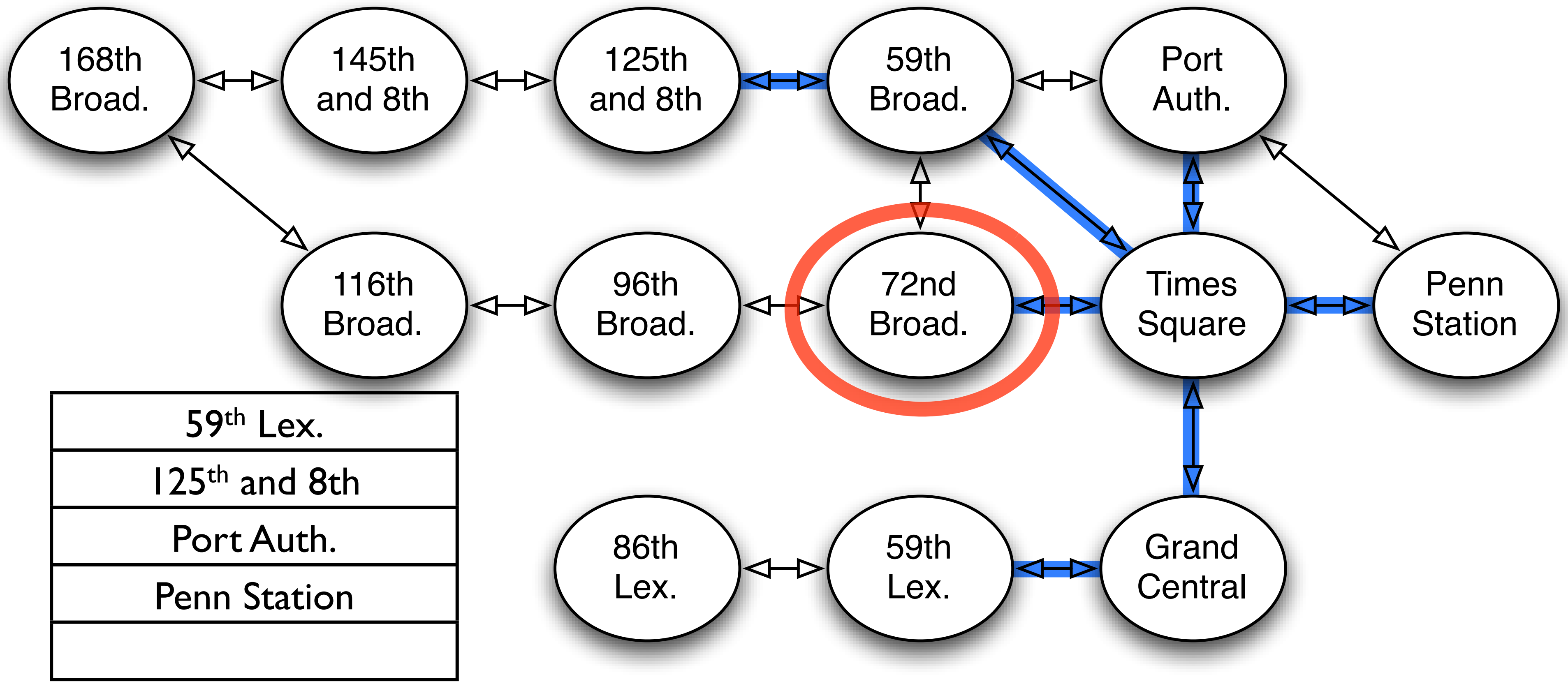


	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist			2	1	1			1	0	1			1
prev			59th Broad.	Times Sq.	Times Sq.			Times Sq.	source	Times Sq.			Times Sq.

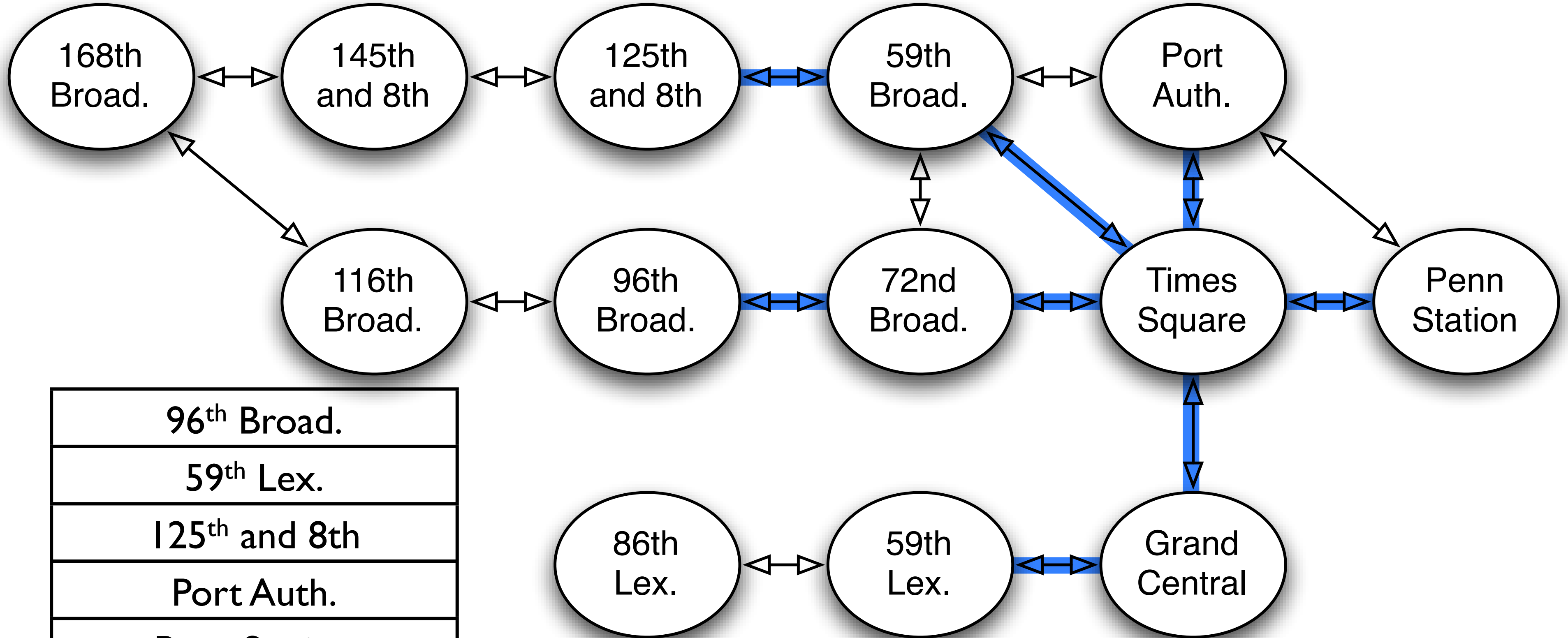


59th Lex.
125th and 8th
Port Auth.
Penn Station
72nd Broad.

	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist			2	1	1			1	0	1		2	1
prev			59th Broad.	Times Sq.	Times Sq.			Times Sq.	source	Times Sq.		Grand Centr.	Times Sq.

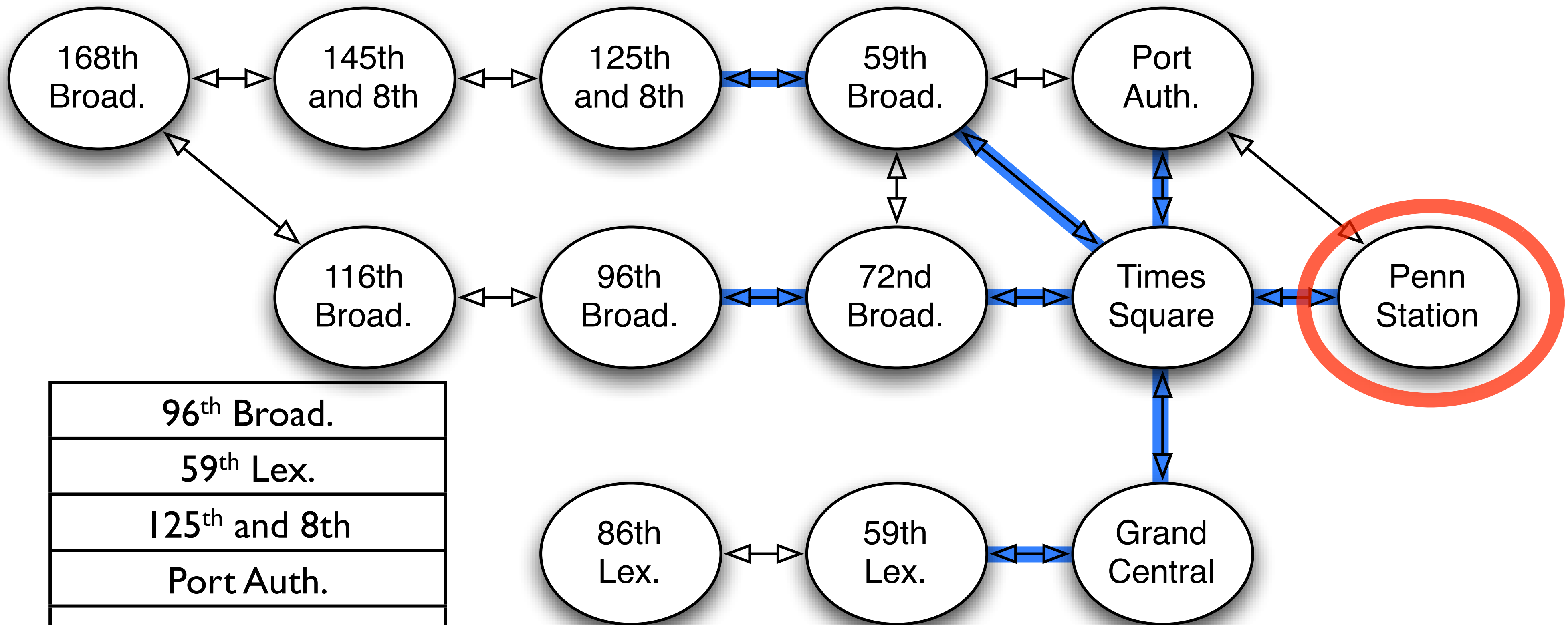


	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist			2	1	1			1	0	1		2	1
prev			59th Broad.	Times Sq.	Times Sq.			Times Sq.	source	Times Sq.		Grand Centr.	Times Sq.



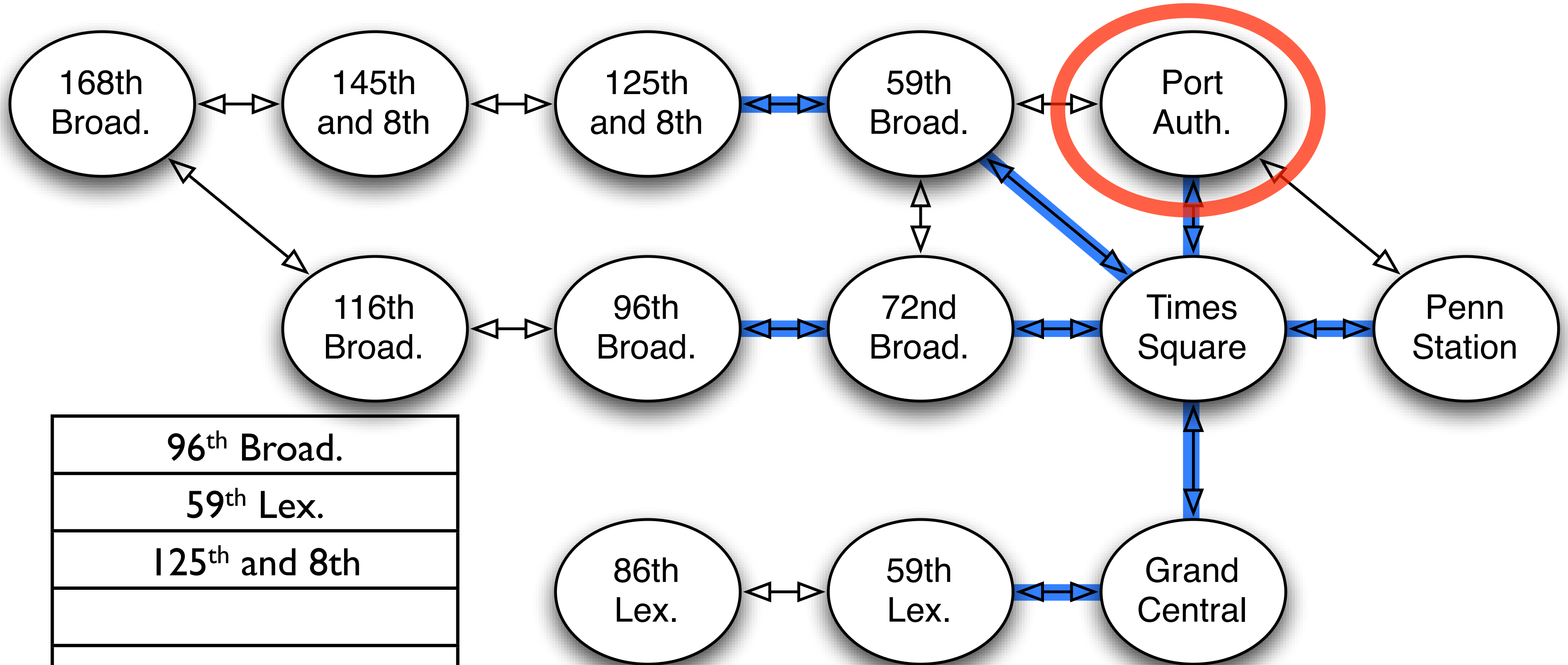
96th Broad.
59th Lex.
125th and 8th
Port Auth.
Penn Station

	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist			2	1	1		2	1	0	1		2	1
prev			59th Broad.	Times Sq.	Times Sq.		72nd Broad.	Times Sq.	source	Times Sq.		Grand Centr.	Times Sq.



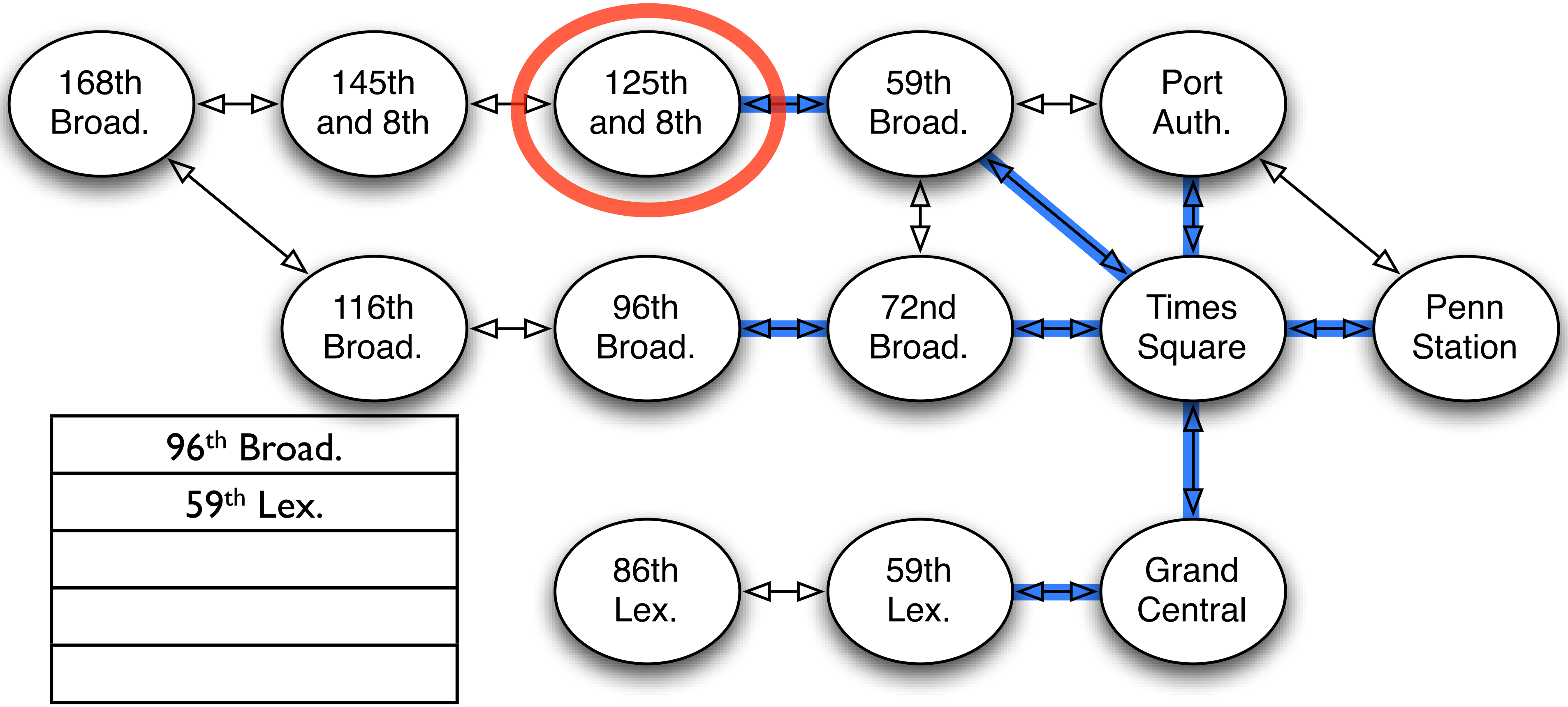
96th Broad.
59th Lex.
125th and 8th
Port Auth.

	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist			2	1	1		2	1	0	1		2	1
prev			59th Broad.	Times Sq.	Times Sq.		72nd Broad.	Times Sq.	source	Times Sq.		Grand Centr.	Times Sq.

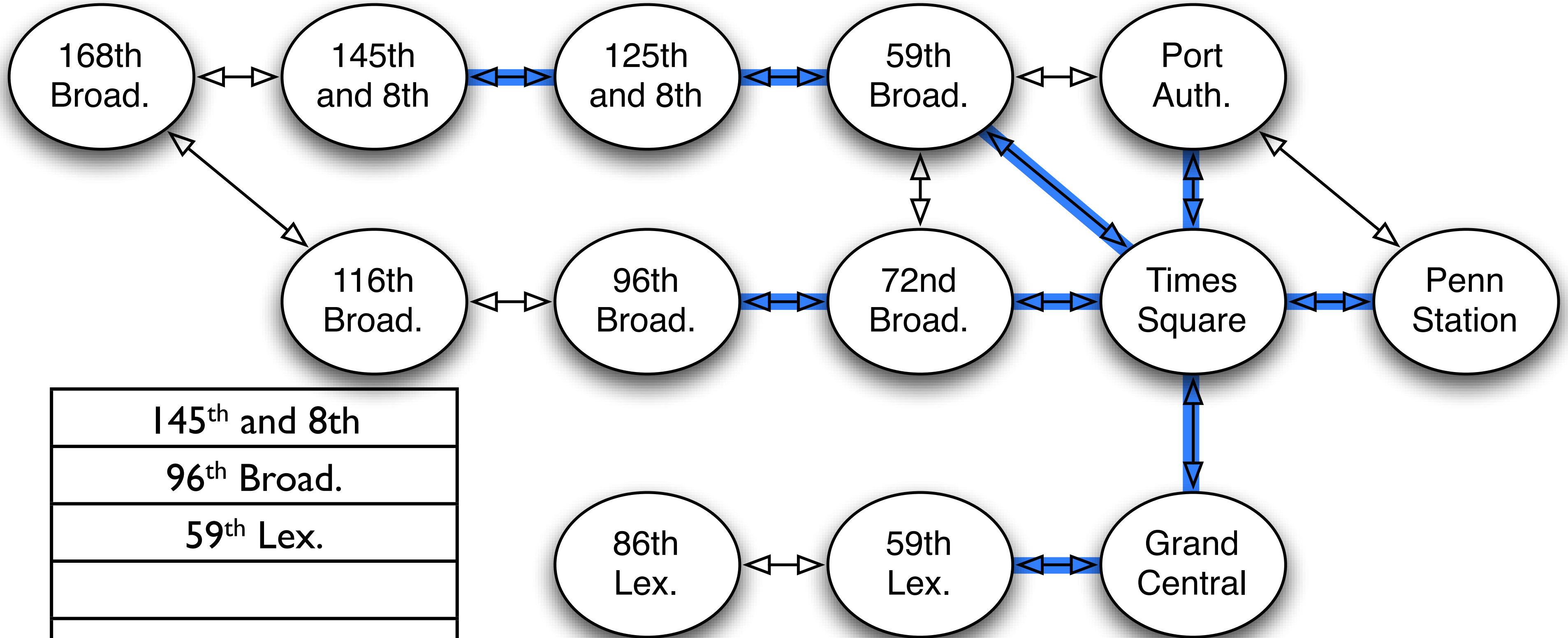


96th Broad.
59th Lex.
125th and 8th

	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist			2	1	1		2	1	0	1		2	1
prev			59th Broad.	Times Sq.	Times Sq.		72nd Broad.	Times Sq.	source	Times Sq.		Grand Centr.	Times Sq.

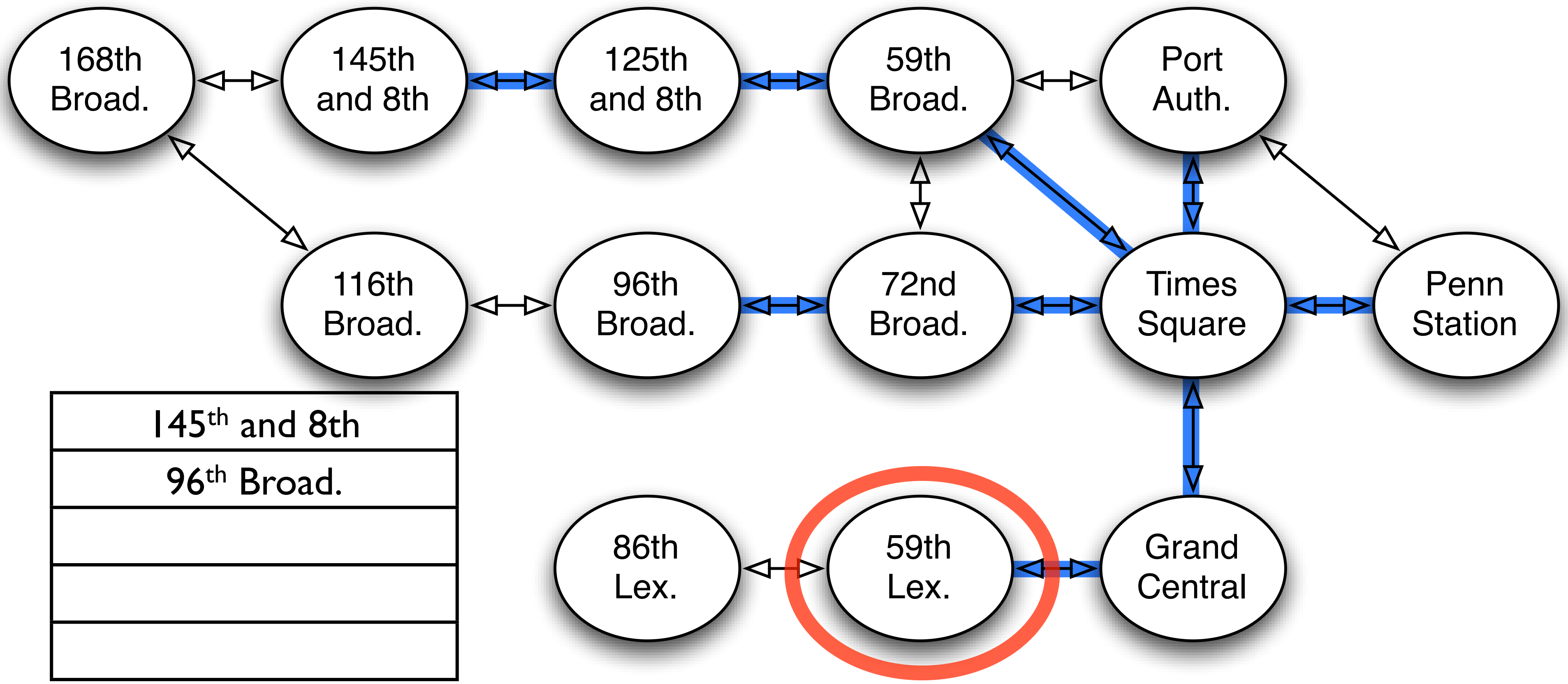


	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist			2	1	1		2	1	0	1		2	1
prev			59th Broad.	Times Sq.	Times Sq.		72nd Broad.	Times Sq.	source	Times Sq.		Grand Centr.	Times Sq.

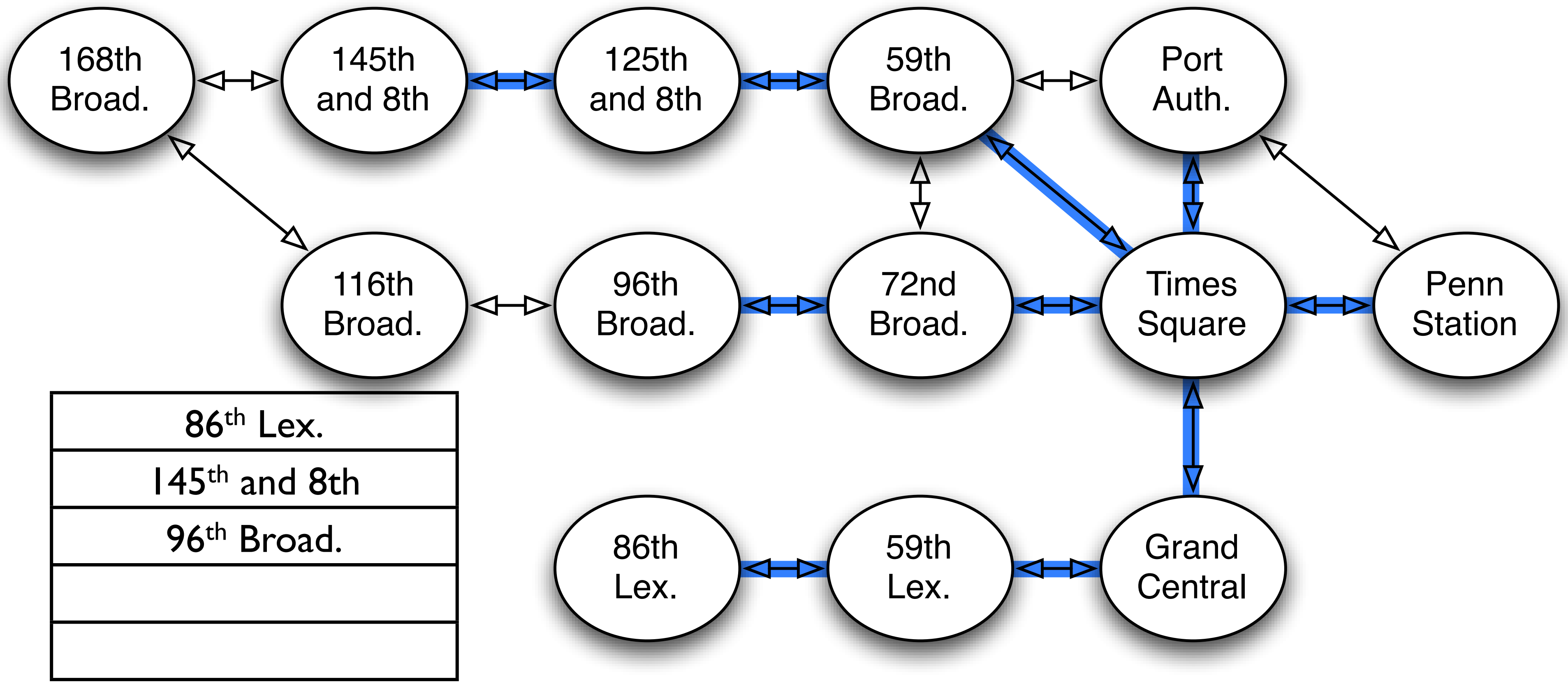


145th and 8th
96th Broad.
59th Lex.

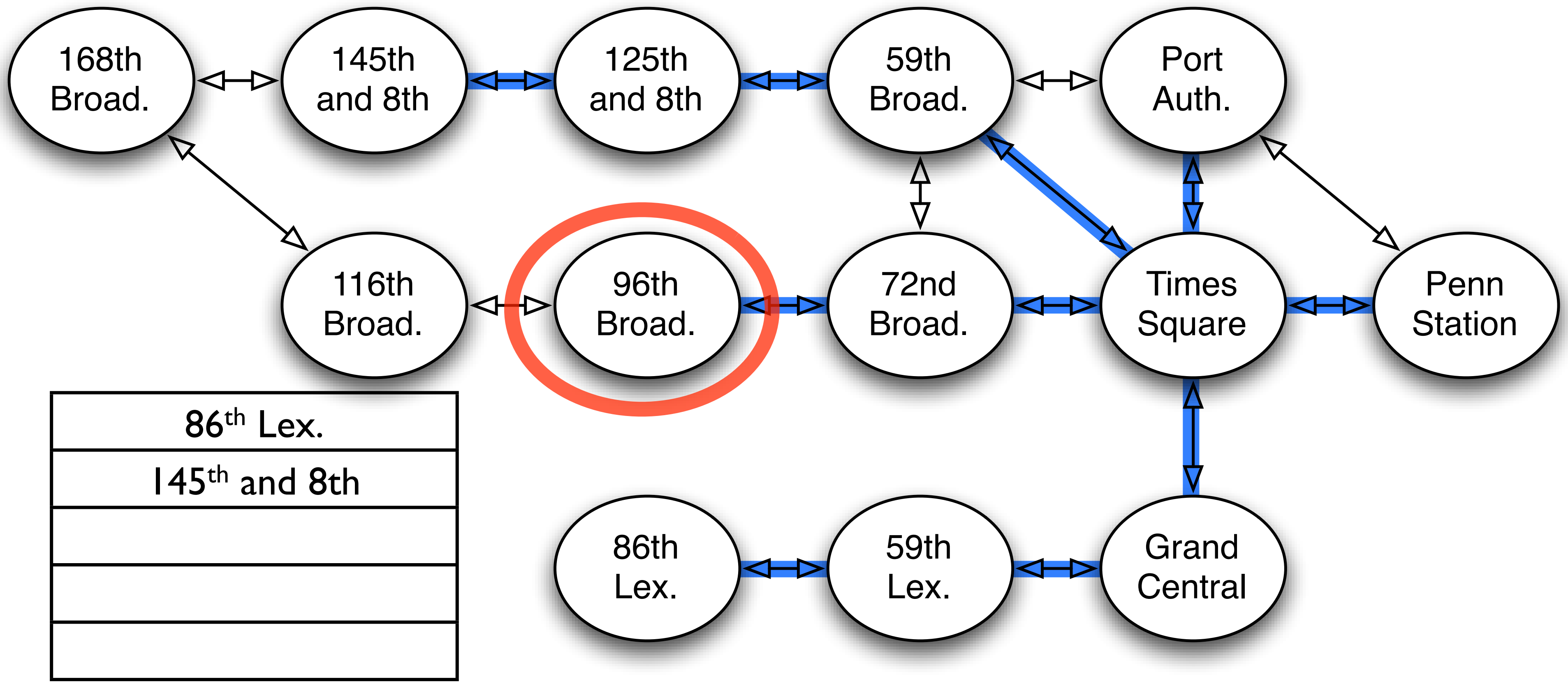
	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist		3	2	1	1		2	1	0	1		2	1
prev		125th 8th	59th Broad.	Times Sq.	Times Sq.		72nd Broad.	Times Sq.	source	Times Sq.		Grand Centr.	Times Sq.



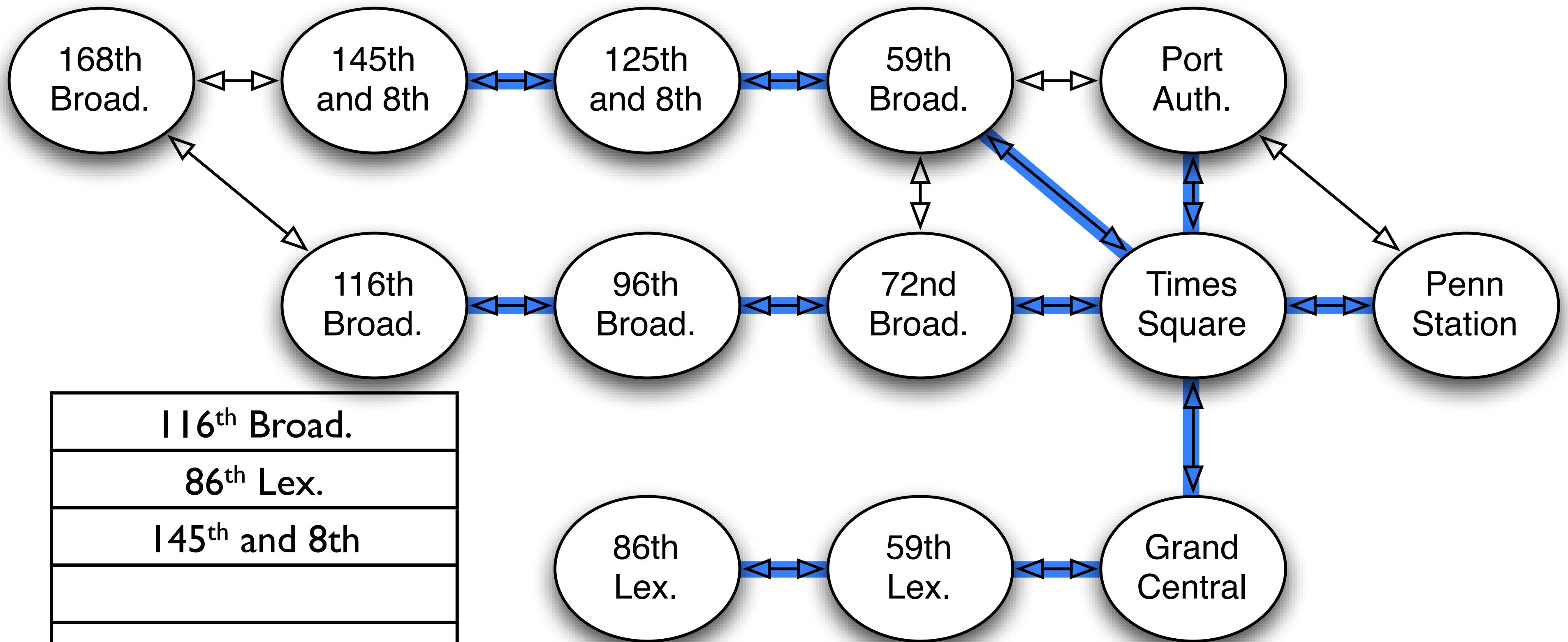
	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist		3	2	1	1		2	1	0	1		2	1
prev		125th 8th	59th Broad.	Times Sq.	Times Sq.		72nd Broad.	Times Sq.	source	Times Sq.		Grand Centr.	Times Sq.



	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist		3	2	1	1		2	1	0	1	3	2	1
prev		125th 8th	59th Broad.	Times Sq.	Times Sq.		72nd Broad.	Times Sq.	source	Times Sq.	59th Lex.	Grand Centr.	Times Sq.

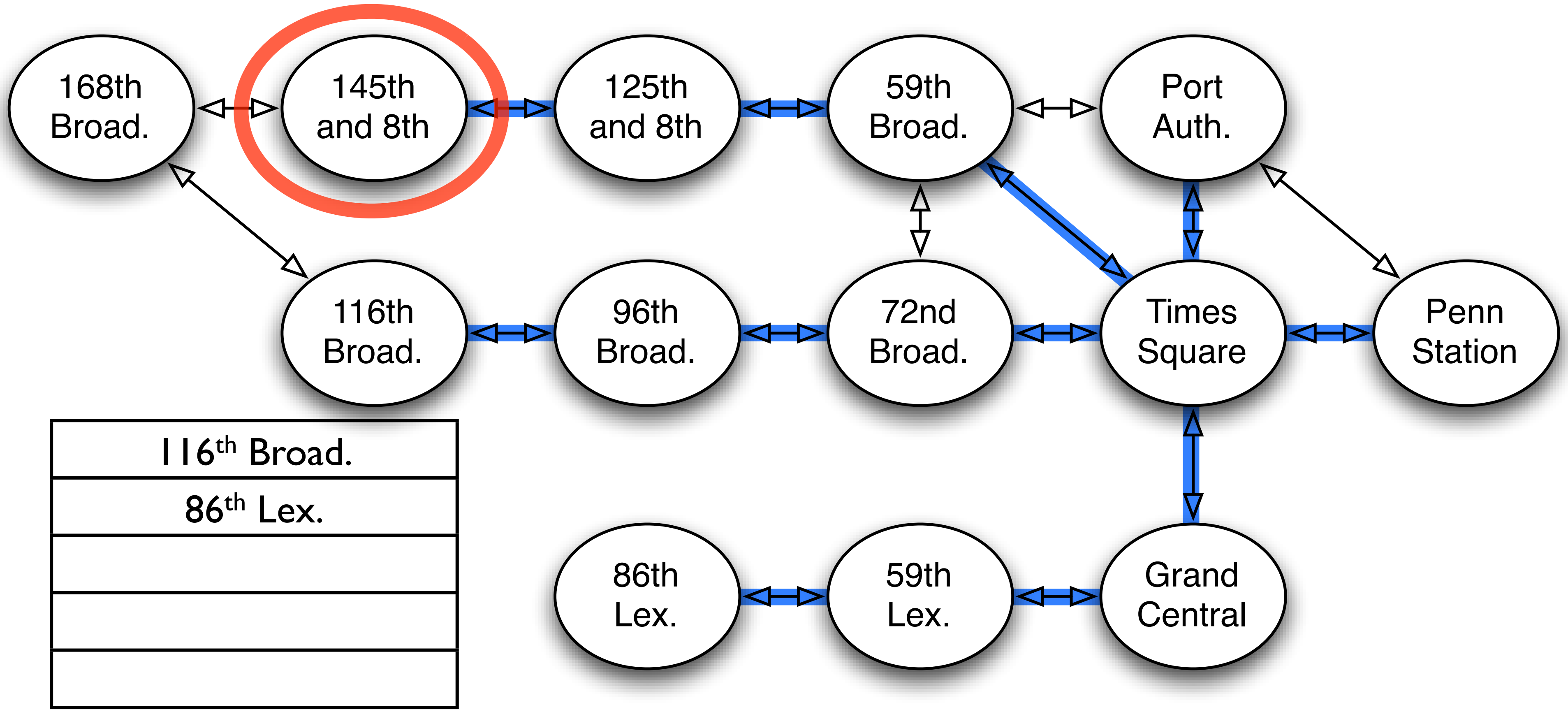


	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist		3	2	1	1		2	1	0	1	3	2	1
prev		125th 8th	59th Broad.	Times Sq.	Times Sq.		72nd Broad.	Times Sq.	source	Times Sq.	59th Lex.	Grand Centr.	Times Sq.

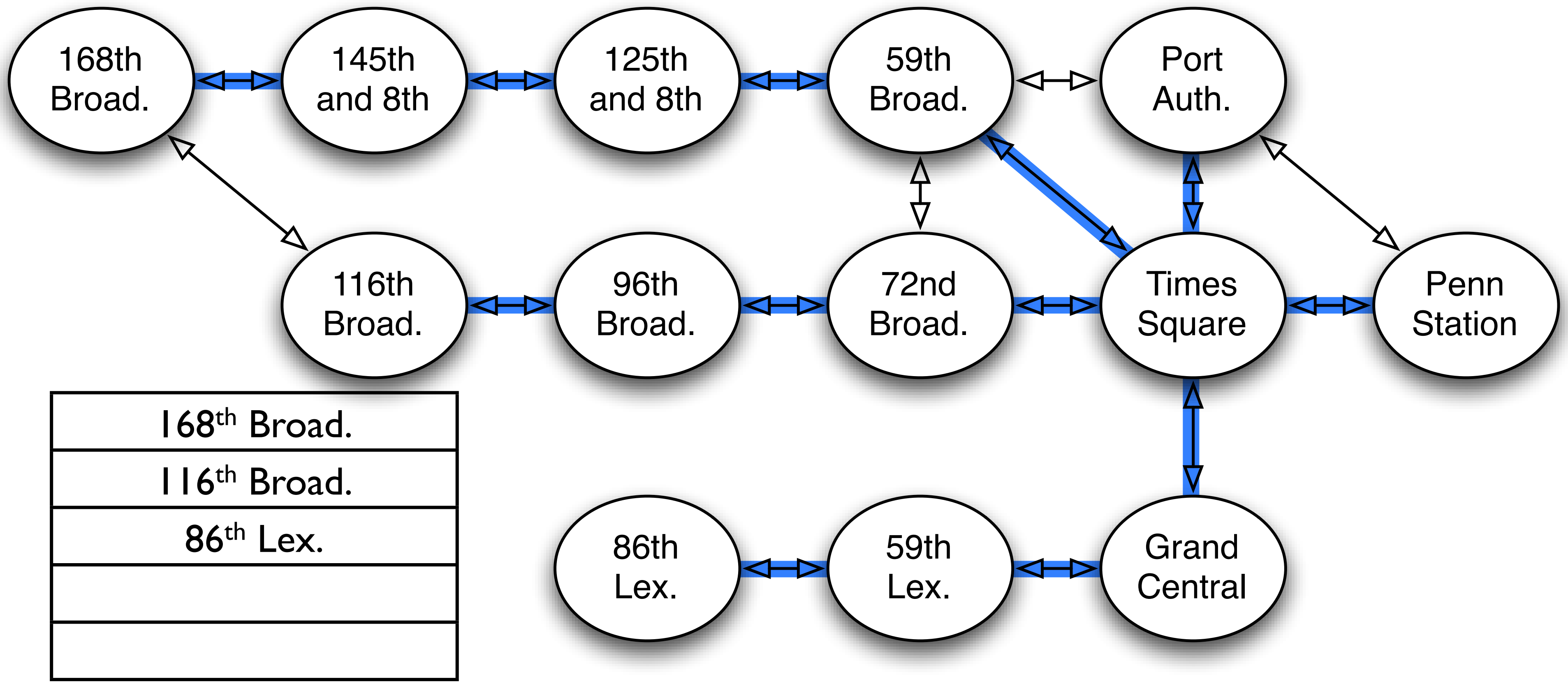


116th Broad.
86th Lex.
145th and 8th

	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist		3	2	1	1	3	2	1	0	1	3	2	1
prev		125th 8th	59th Broad.	Times Sq.	Times Sq.	96th Broad.	72nd Broad.	Times Sq.	source	Times Sq.	59th Lex.	Grand Centr.	Times Sq.



	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist		3	2	1	1	3	2	1	0	1	3	2	1
prev		125th 8th	59th Broad.	Times Sq.	Times Sq.	96th Broad.	72nd Broad.	Times Sq.	source	Times Sq.	59th Lex.	Grand Centr.	Times Sq.



	168th Broad.	145th Broad.	125th 8th	59th Broad.	Port Auth.	116th Broad.	96th Broad.	72nd Broad.	Times Sq.	Penn St.	86th Lex.	59th Lex.	Grand Centr.
dist	4	3	2	1	1	3	2	1	0	1	3	2	1
prev	145th 8th	125th 8th	59th Broad.	Times Sq.	Times Sq.	96th Broad.	72nd Broad.	Times Sq.	source	Times Sq.	59th Lex.	Grand Centr.	Times Sq.

Weighted Shortest Path

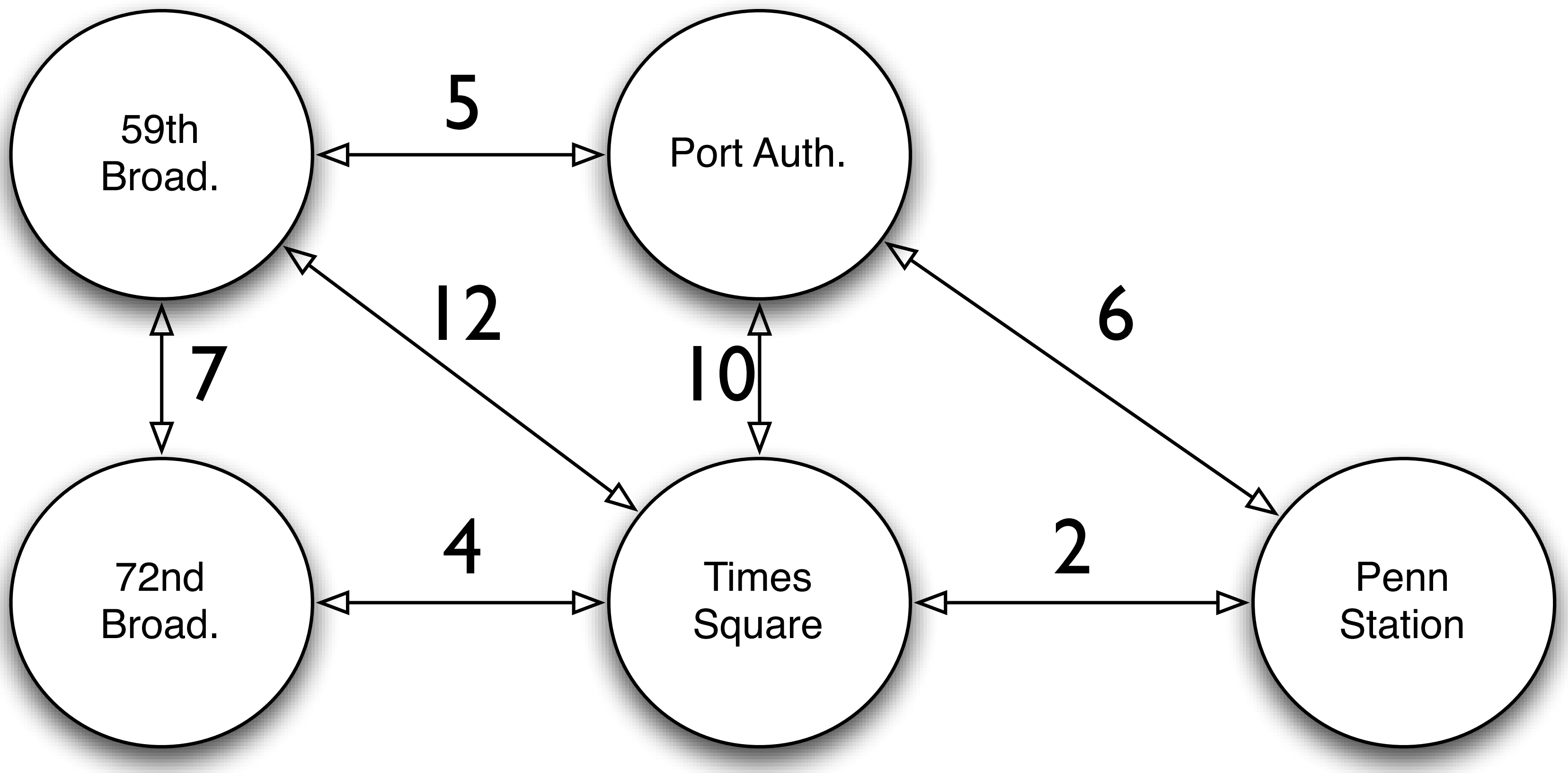
- The problem becomes more difficult when edges have different weights
- Weights represent different costs on using that edge
- Standard algorithm is **Dijkstra's Algorithm**

Dijkstra's Algorithm

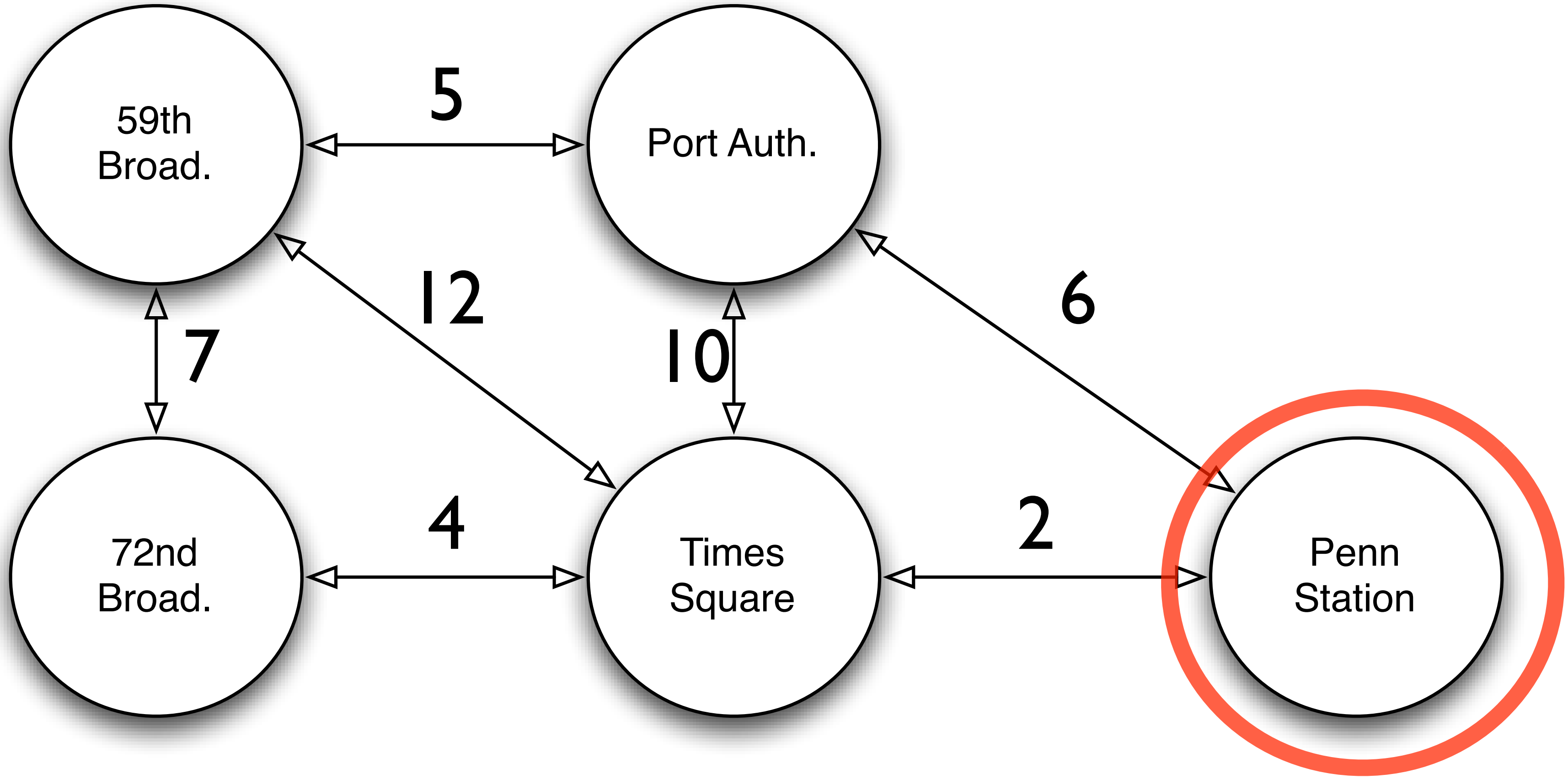
- Keep distance overestimates **D(v)** for each node **v** (all non-source nodes are initially infinite)
- 1. Choose node **v** with smallest *unknown* distance
- 2. Declare that **v**'s shortest distance is *known*
- 3. Update distance estimates for neighbors

Updating Distances

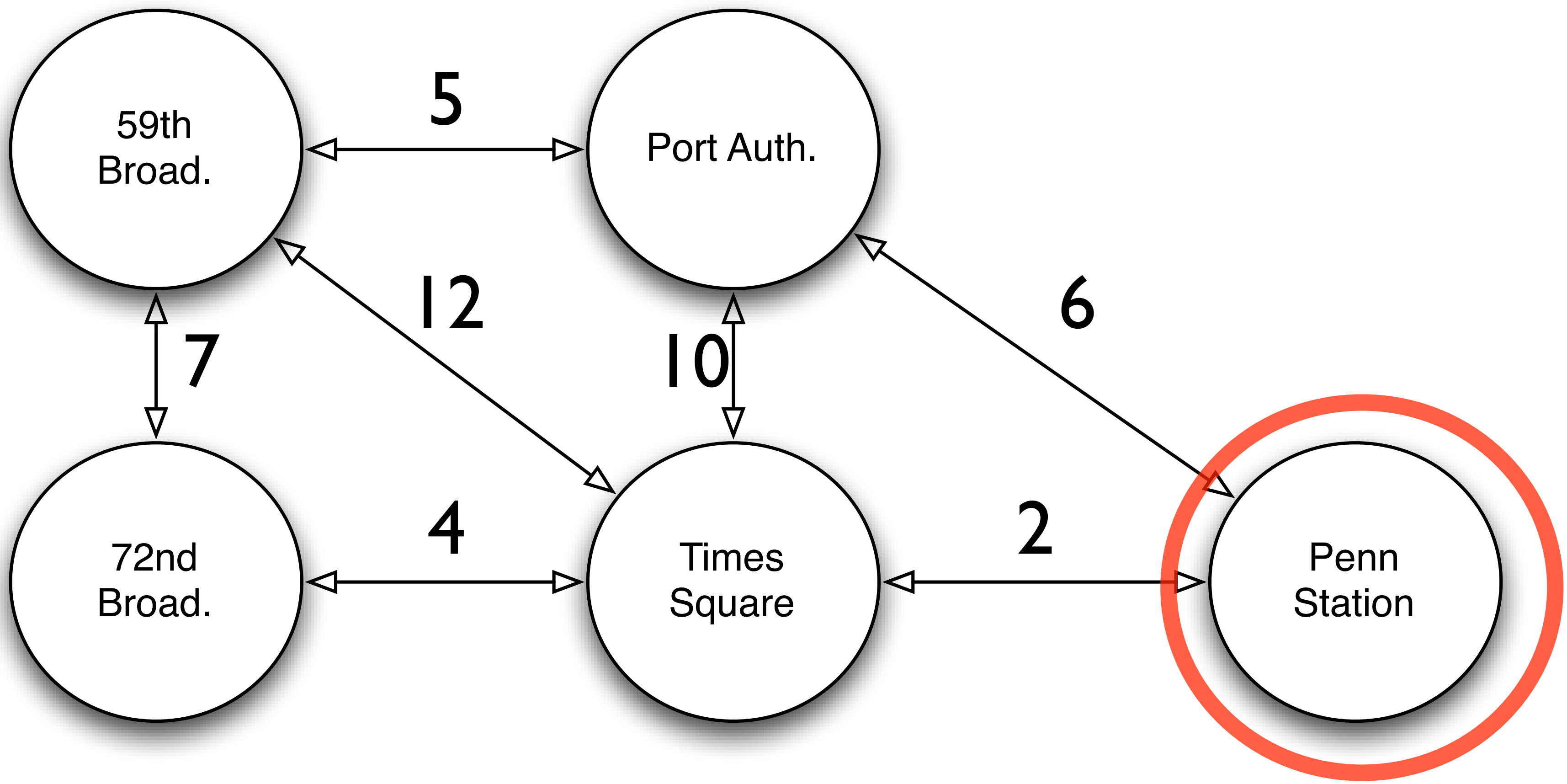
- For each of **v**’s neighbors, **w**,
- if min(**D(v)+ weight(v,w)**, **D(w)**)
  - i.e., update **D(w)** if the path going through **v** is cheaper than the best path so far to **w**



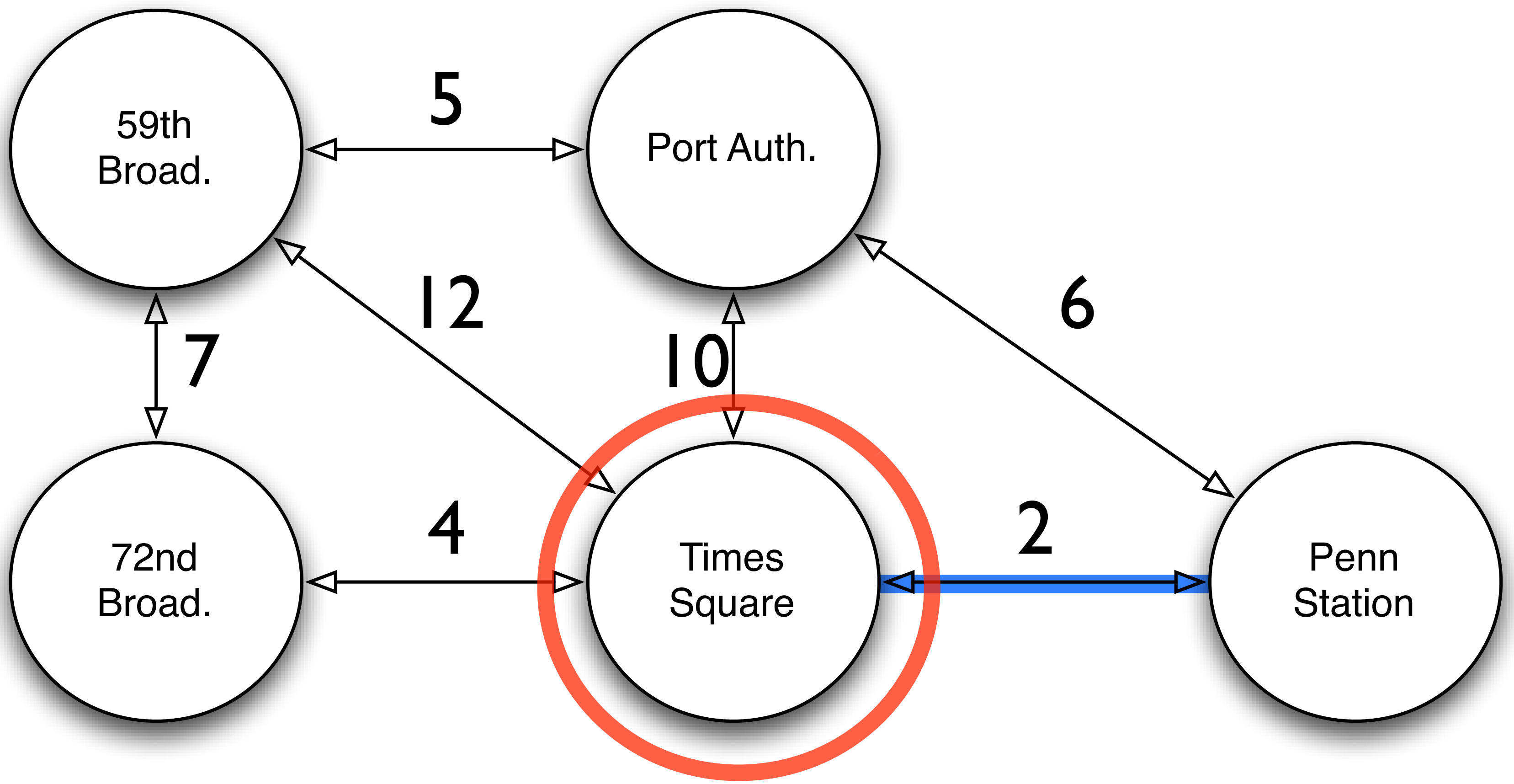
59th Broad.	Port Auth.	72nd Broad	Times Sq.	Penn St.
inf	inf	inf	inf	0
?	?	?	?	home



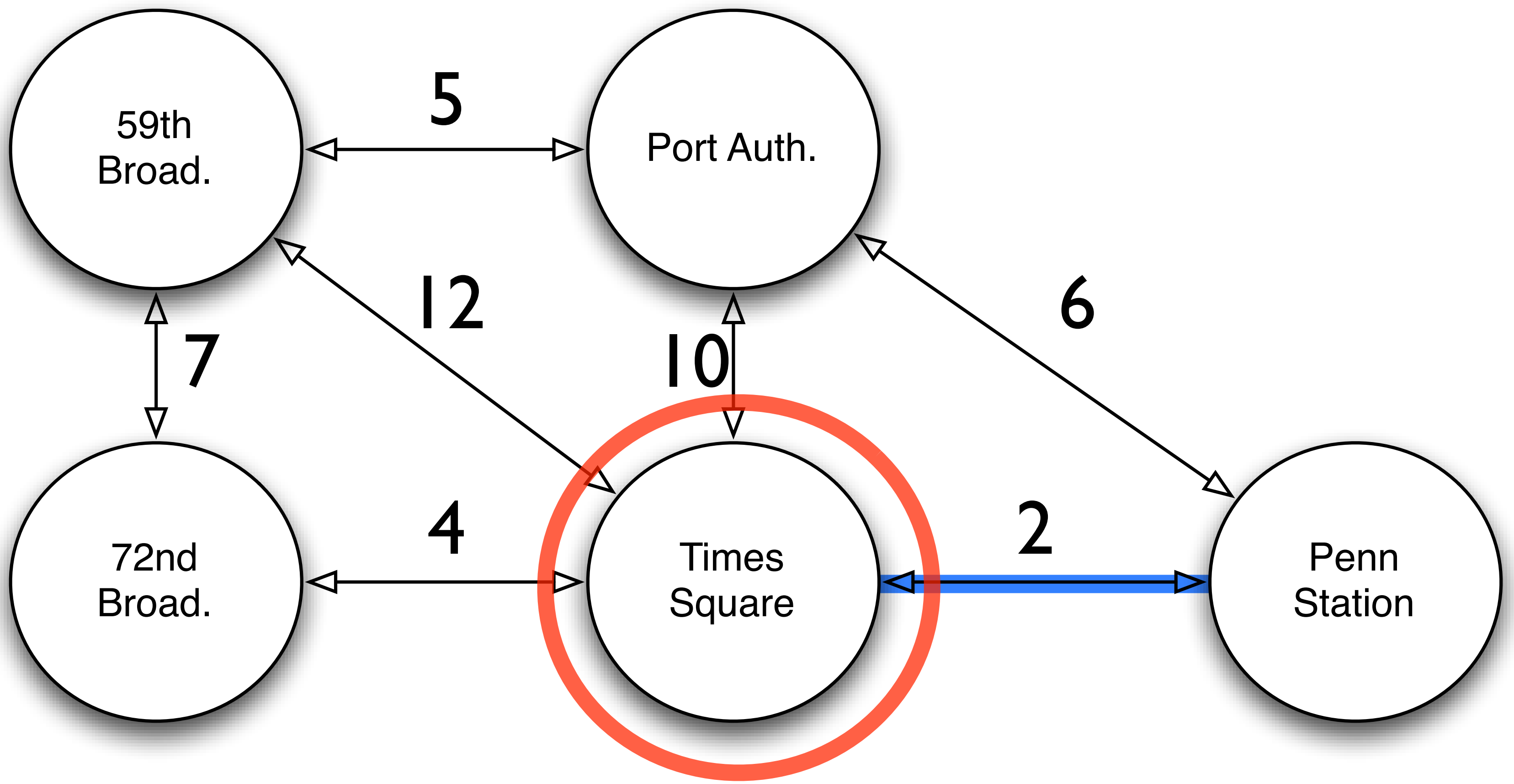
59th Broad.	Port Auth.	72nd Broad	Times Sq.	Penn St.
inf	inf	inf	inf	**0**
?	?	?	?	**home**



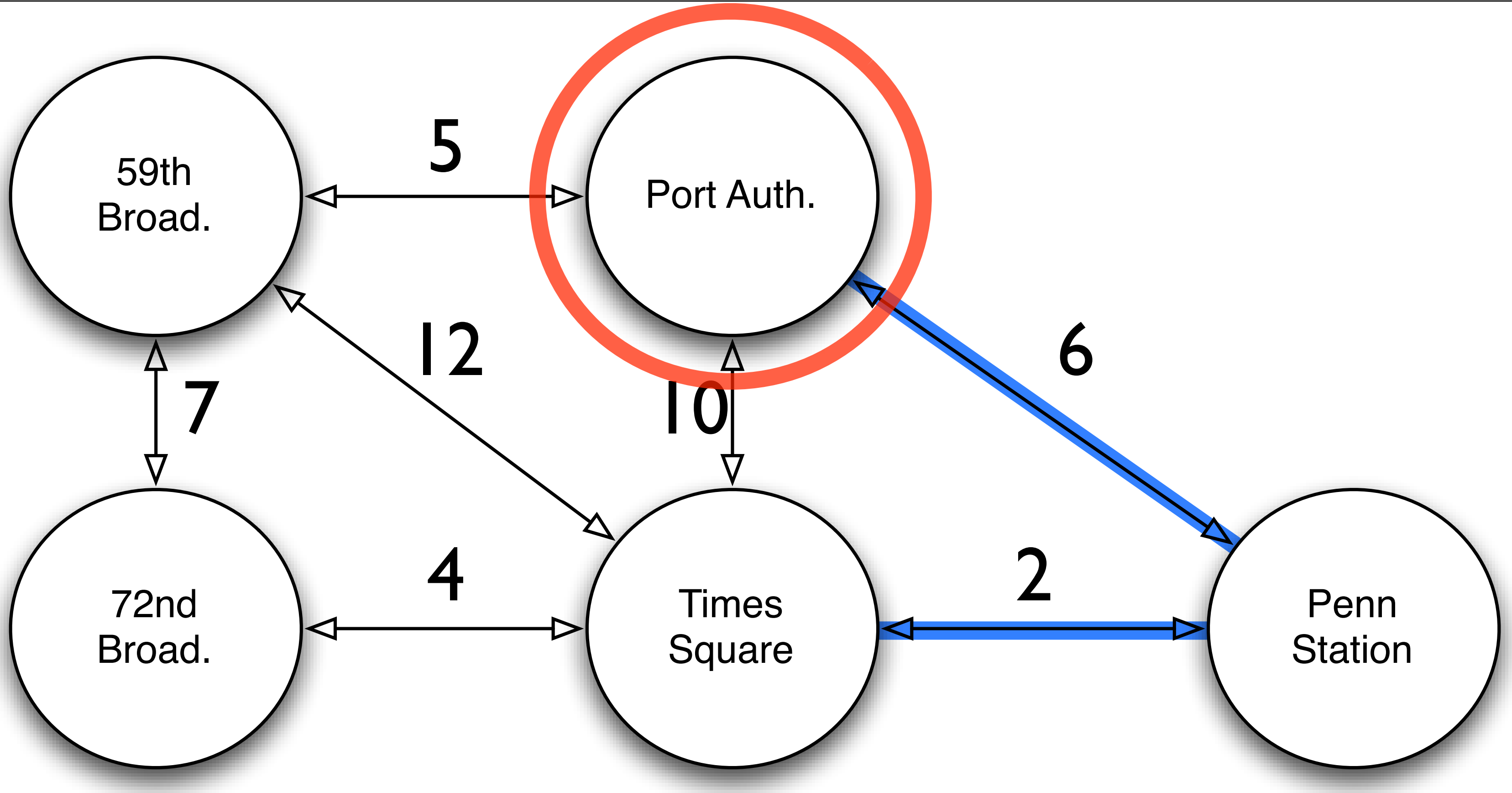
59th Broad.	Port Auth.	72nd Broad	Times Sq.	Penn St.
inf	6	inf	2	**0**
?	Penn St.?	?	Penn St.?	**home**



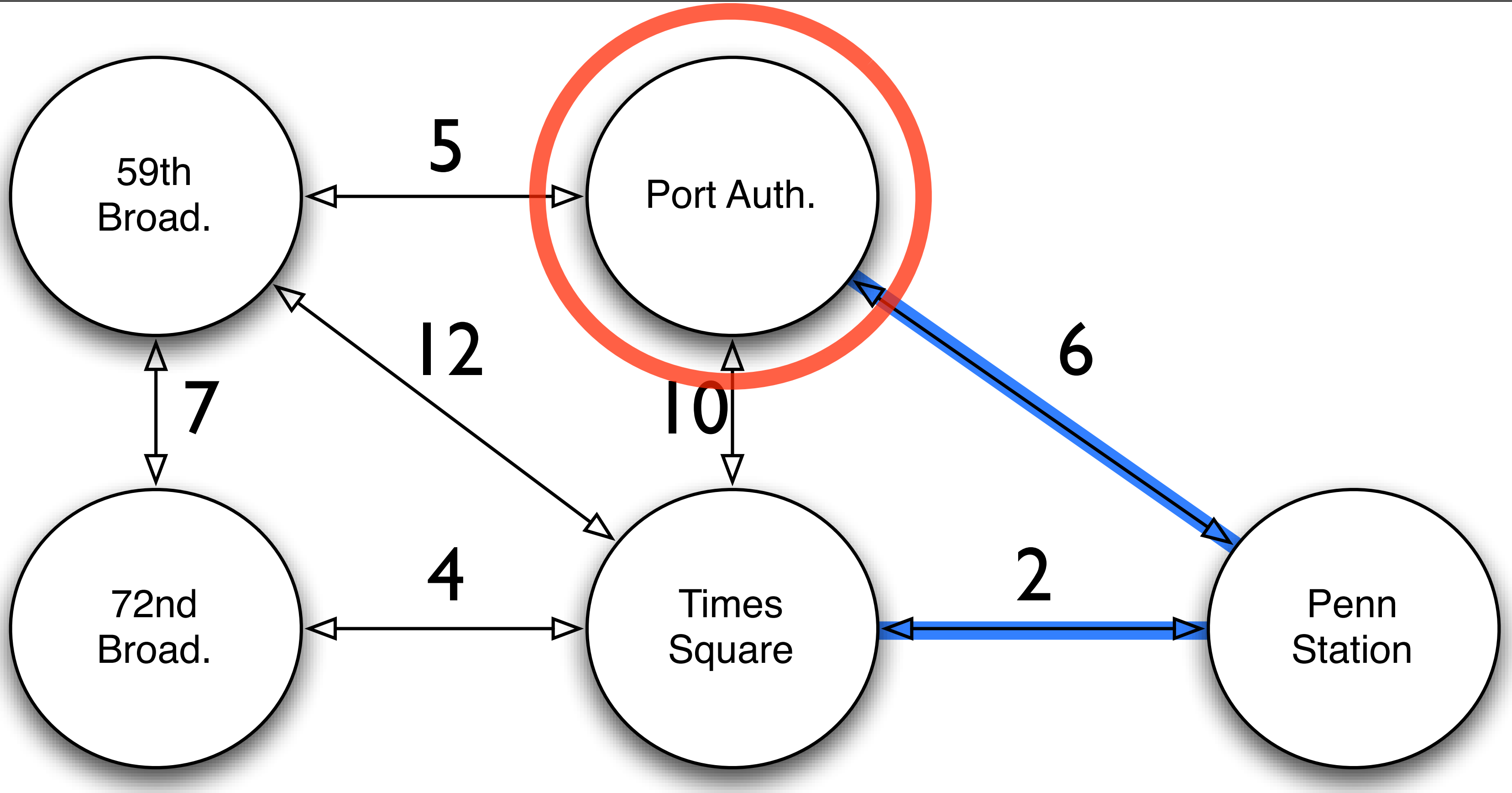
59th Broad.	Port Auth.	72nd Broad	Times Sq.	Penn St.
inf	6	inf	**2**	**0**
?	Penn St.?	?	**Penn St.**	**home**



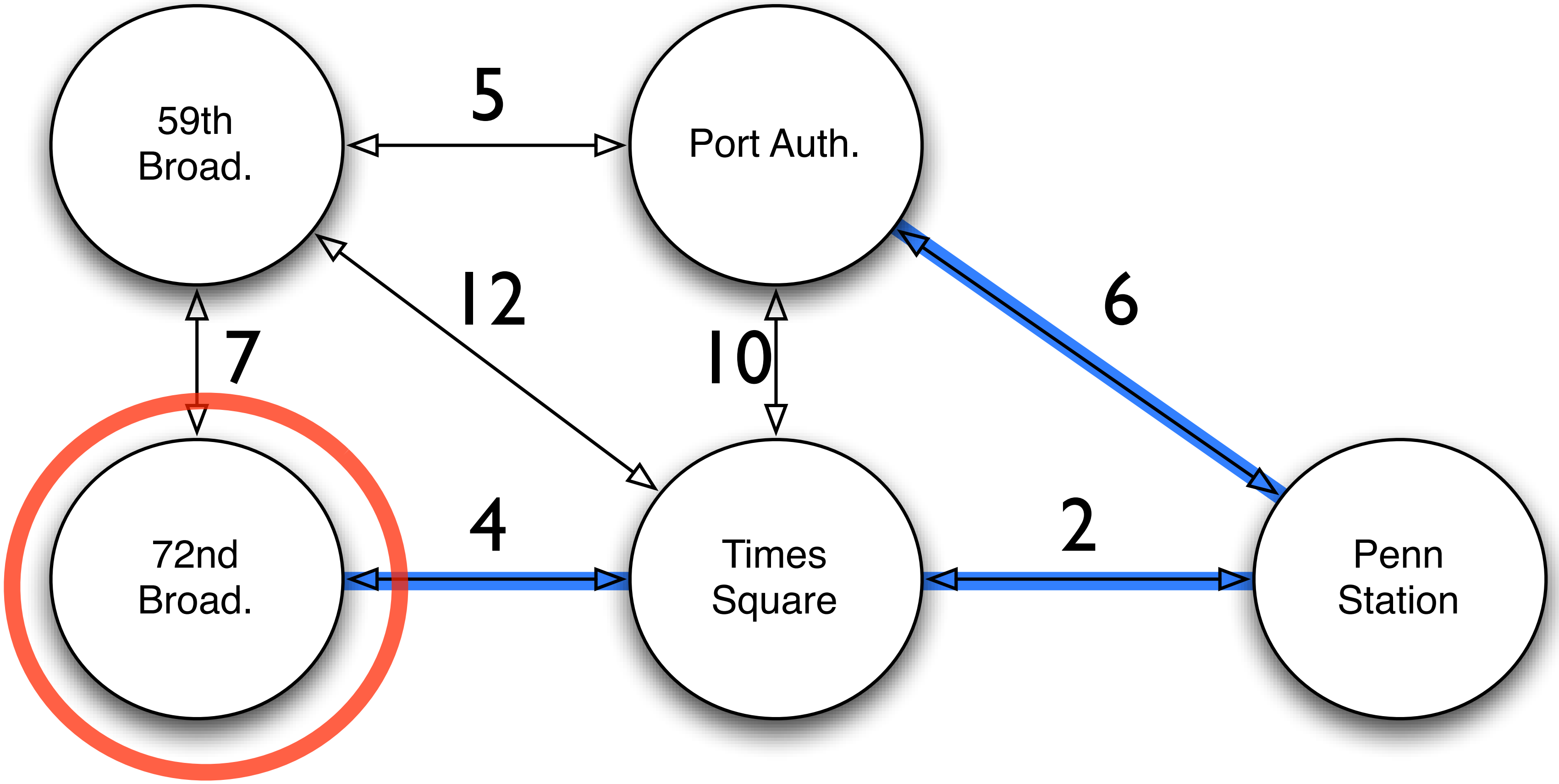
59th Broad.	Port Auth.	72nd Broad	Times Sq.	Penn St.
2+12=14	6	2+4=6	**2**	**0**
Times Sq?	Penn St.?	Times Sq?	**Penn St.**	**home**



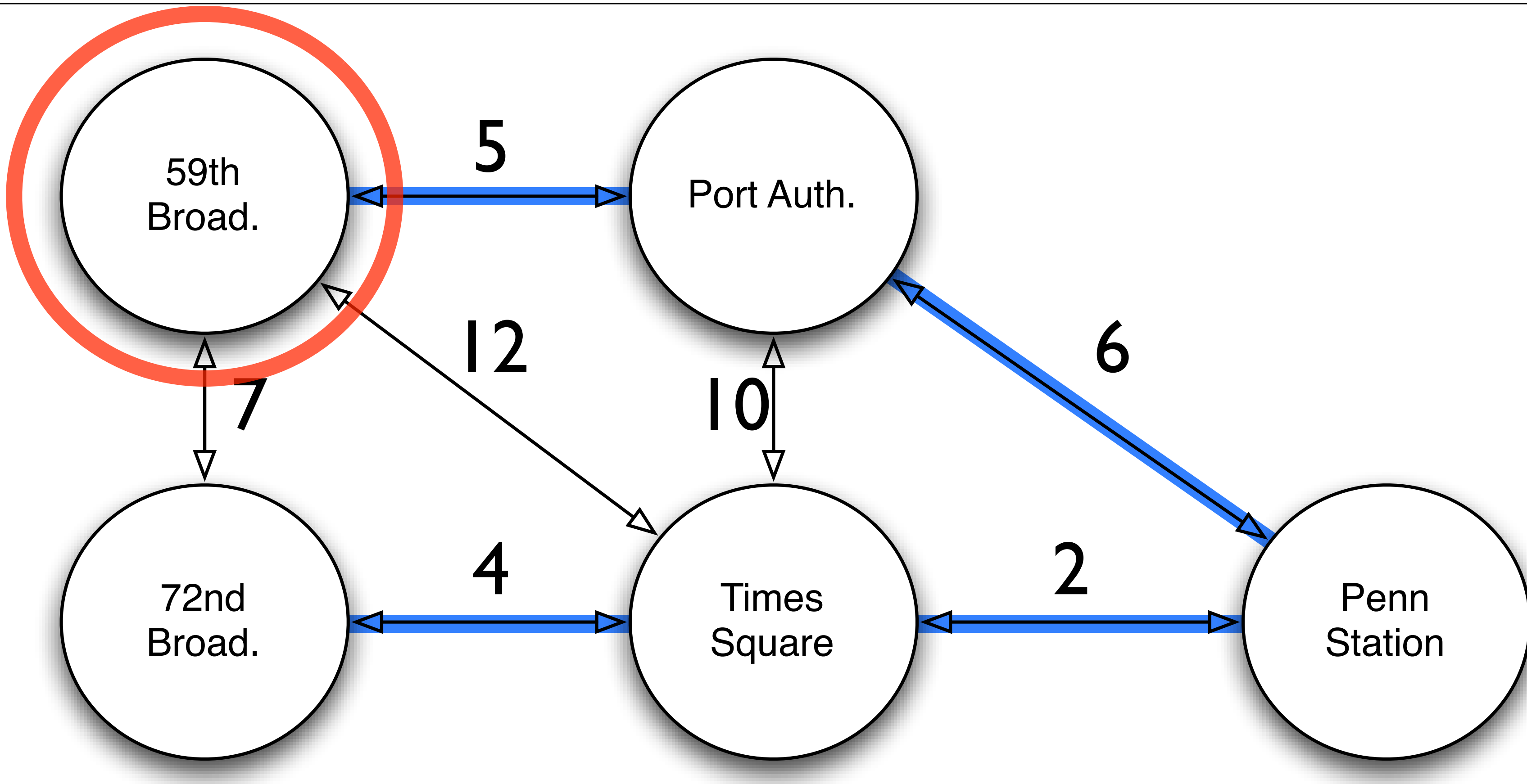
59th Broad.	Port Auth.	72nd Broad	Times Sq.	Penn St.
14	**6**	6	**2**	**0**
Times Sq?	**Penn St.**	Times Sq?	**Penn St.**	**home**



59th Broad.	Port Auth.	72nd Broad	Times Sq.	Penn St.
5+6=11	**6**	6	**2**	**0**
Port Auth?	**Penn St.**	Times Sq?	**Penn St.**	**home**



59th Broad.	Port Auth.	72nd Broad	Times Sq.	Penn St.
11	**6**	**6**	**2**	**0**
Port Auth?	**Penn St.**	**Times Sq**	**Penn St.**	**home**



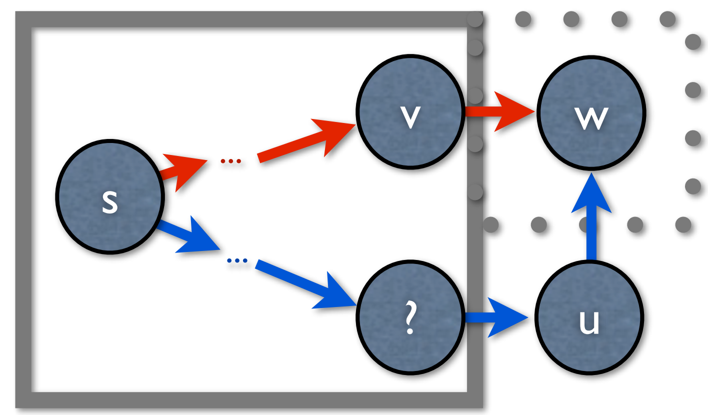
59^{th} Broad.	Port Auth.	72^{nd} Broad	Times Sq.	Penn St.
11	**6**	**6**	**2**	**0**
Port Auth	**Penn St.**	**Times Sq**	**Penn St.**	**home**

Dijkstra's Algorithm Analysis

- First, convince ourselves that the algorithm works.
- At each stage, we have a set of nodes whose shortest paths we know
- In the base case, the set is the source node.
- Inductive step: if we have a correct set, is greedily adding the shortest neighbor correct?

Proof by Contradiction (Sketch)

- Contradiction: Dijkstra's finds a shortest path to node **w** through **v**, but there exists an even shorter path
- This shorter path must pass from inside our known set to outside.
- Call the 1st node in cheaper path outside our set **u**



- The path to **u** must be shorter than the path to **w**
  - But then we would have chosen **u** instead

Computational Cost

- Keep a priority queue of all unknown nodes
- Each stage requires a **deleteMin**, and then some **decreaseKey**s (the # of neighbors of node)
- We call **decreaseKey** once per edge, we call **deleteMin** once per vertex
- Both operations are $O(\log |V|)$
- Total cost: $O(|E| \log |V| + |V| \log |V|) = O(|E| \log |V|)$

Reading

- Weiss Section 9.1-9.3 (today’s material)