Graph Traversals

Slides by Carl Kingsford

Feb. 1, 2013

Based on/Reading: Chapter 3 of Kleinberg & Tardos

Breadth-First Search

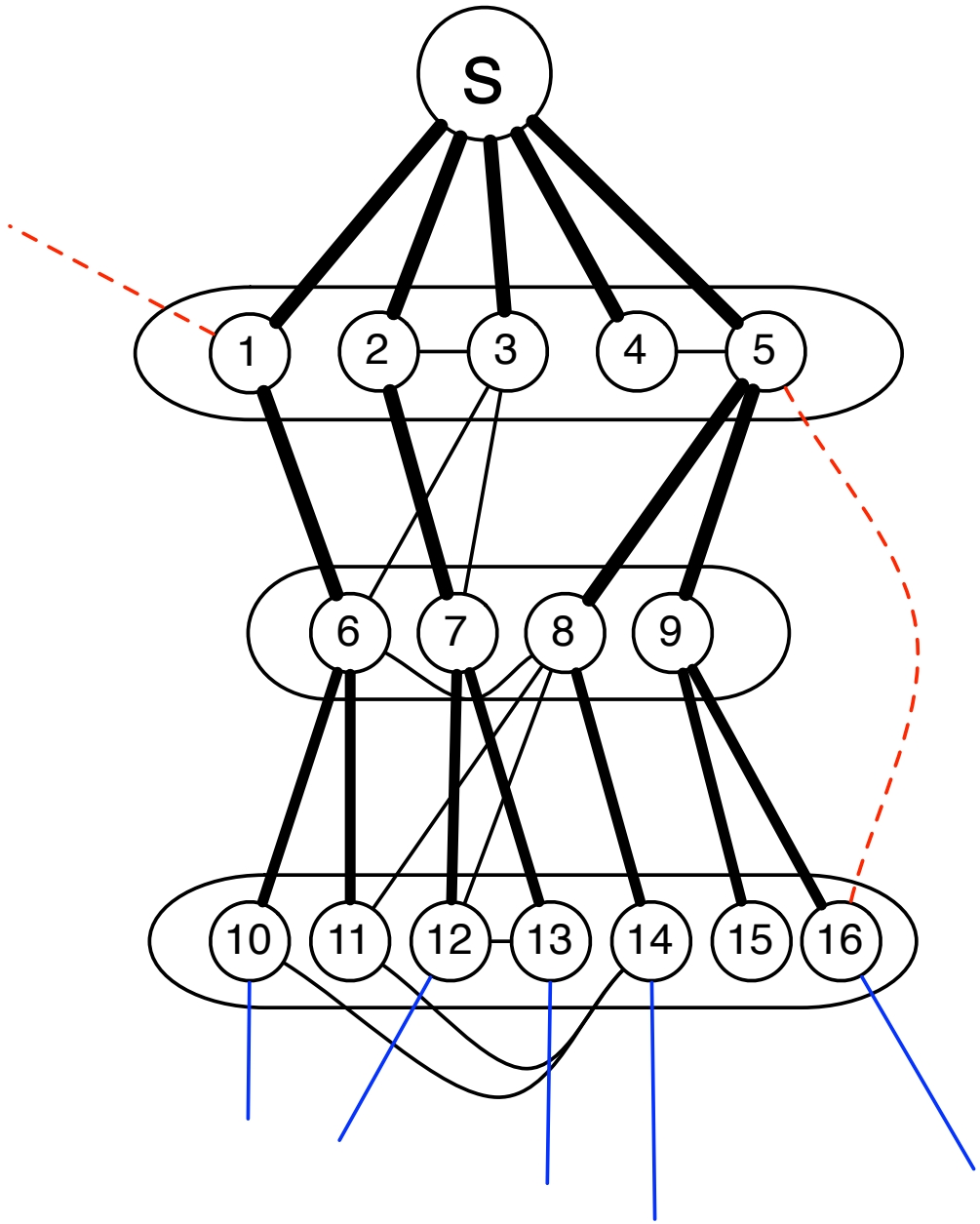
Breadth-first search explores the nodes of a graph in increasing distance away from some starting vertex s.

It decomposes the component into layers L_i such that the shortest path from s to each of nodes in L_i is of length i.

Breadth-First Search:

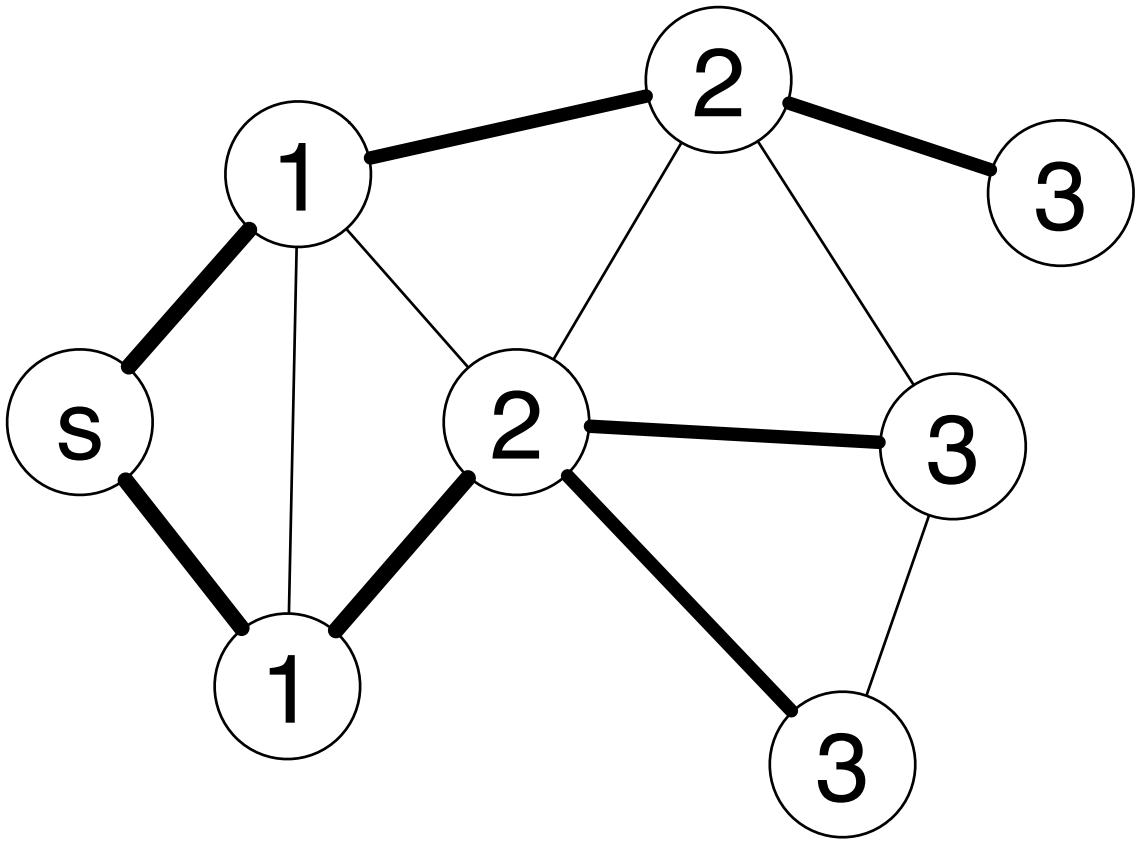
1. L_0 is the set $\{s\}$.
2. Given layers $L_0, L_1, \ldots, L_j$, then L_{j+1} is the set of nodes that are not in a previous layer and that have an edge to some node in layer L_j.

BFS Tree



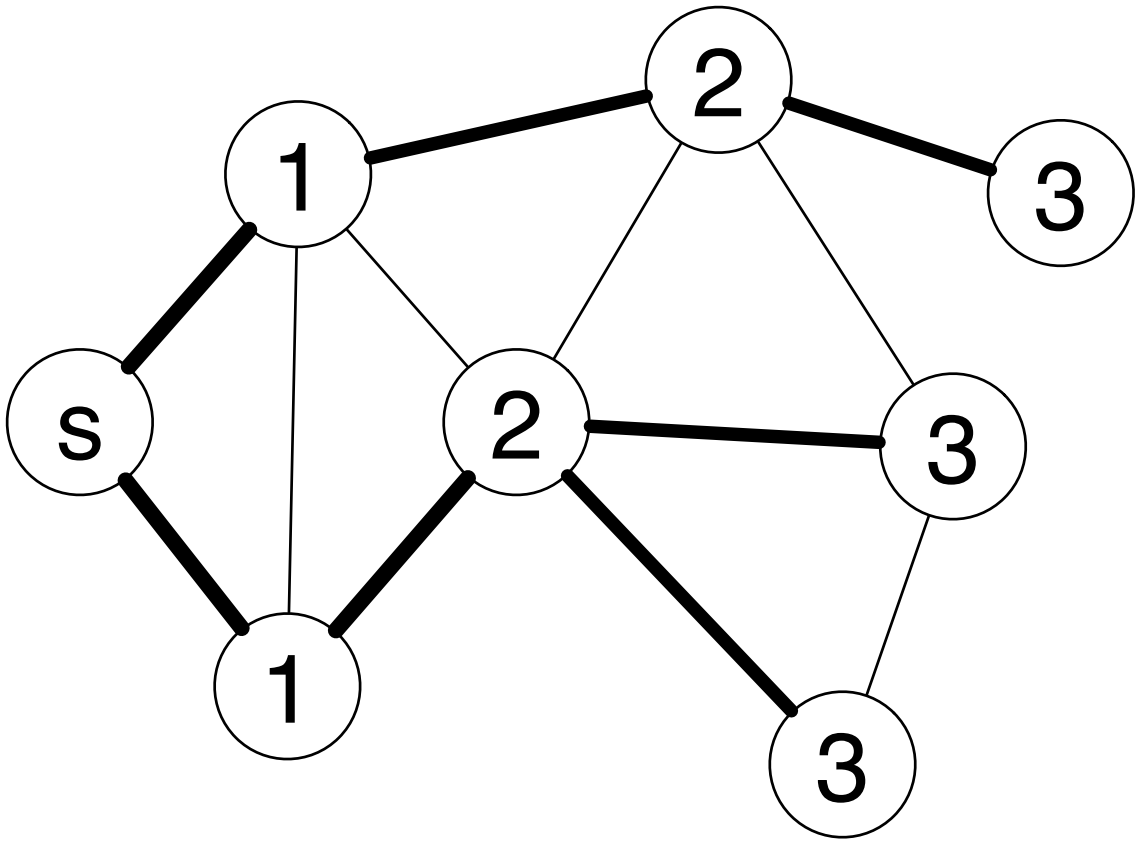
BFS Tree Example

A BFS traversal of a graph results in a breadth-first search tree:



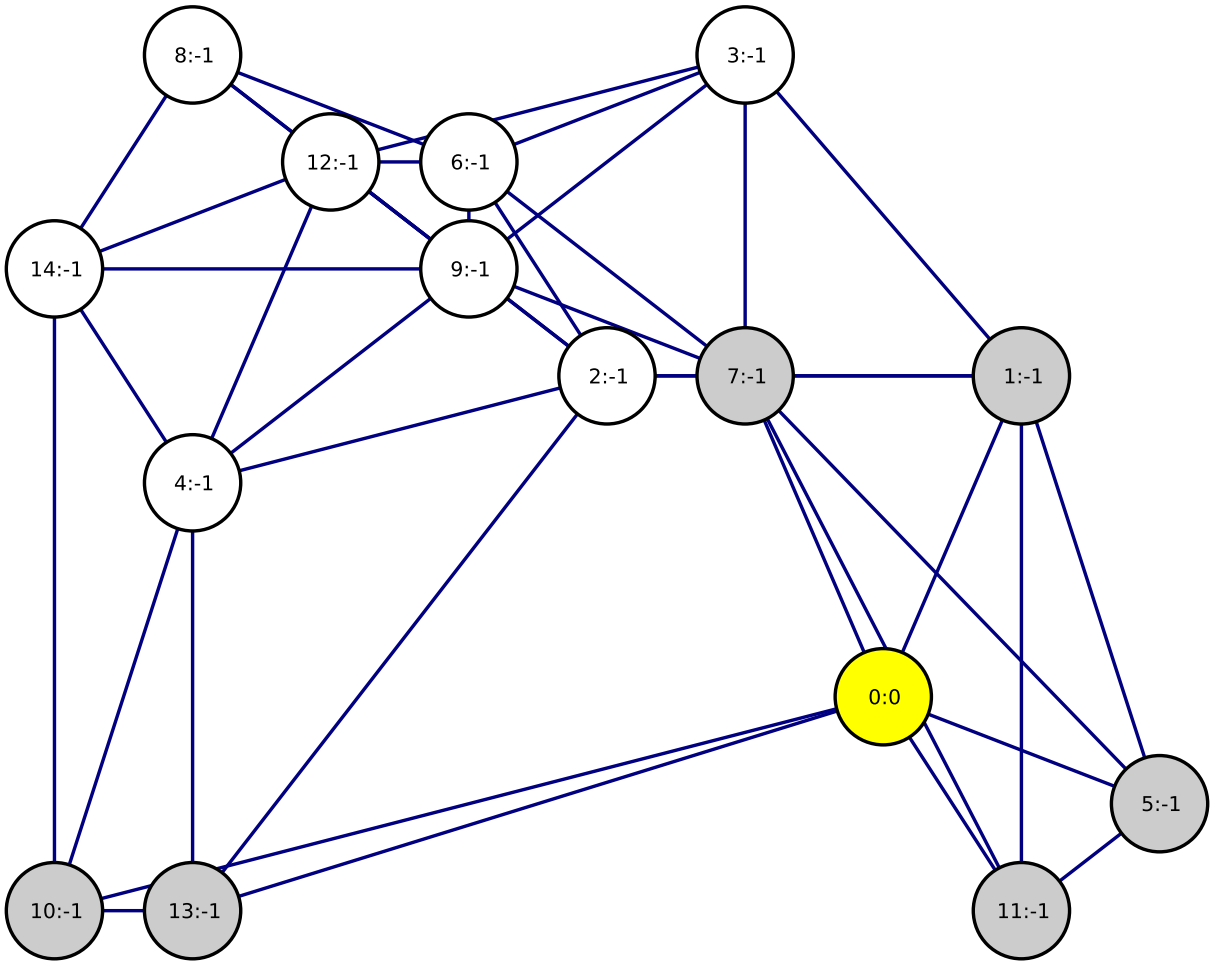
BFS Tree Example

A BFS traversal of a graph results in a breadth-first search tree:

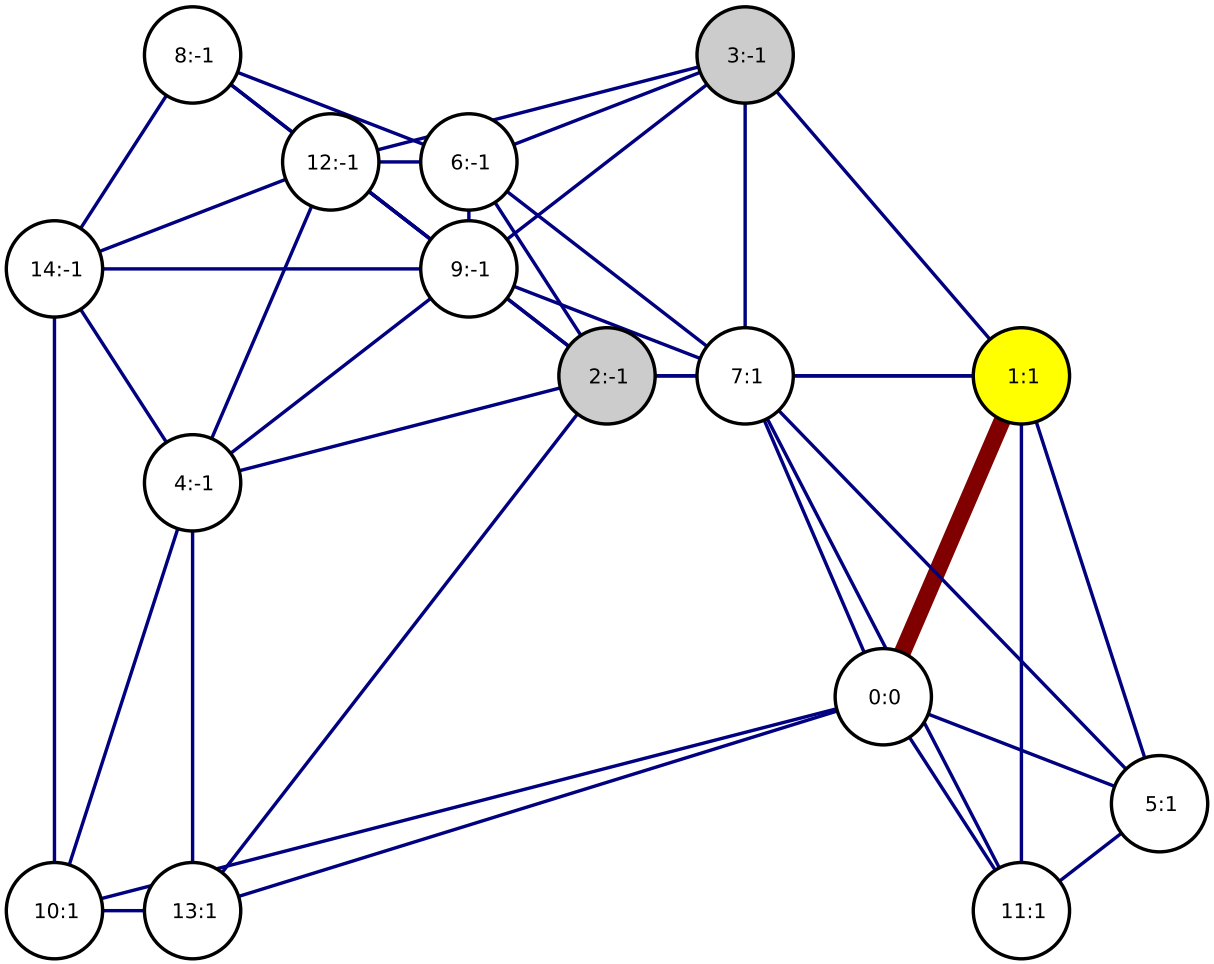


Can we say anything about the non-tree edges?

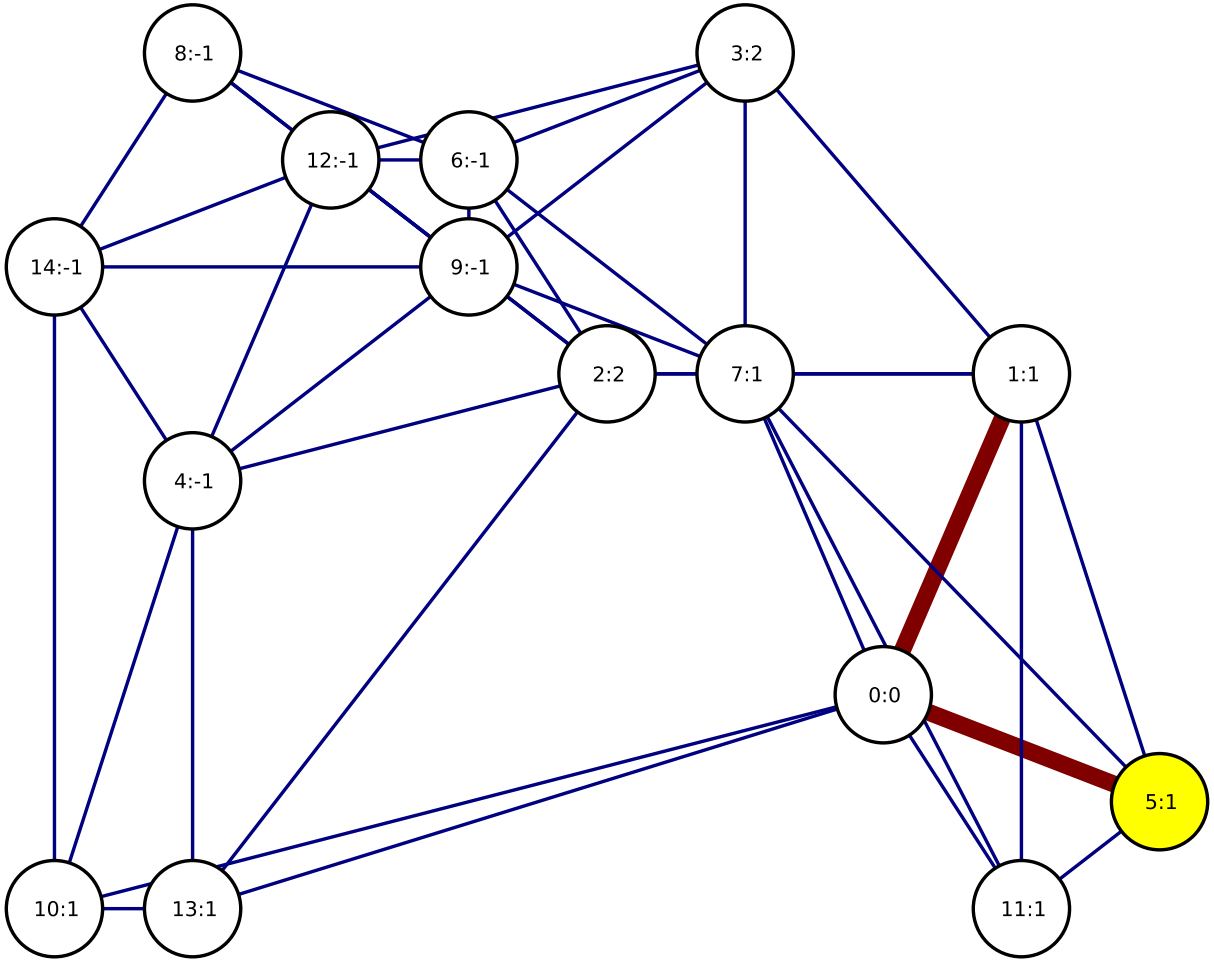
Example BFS



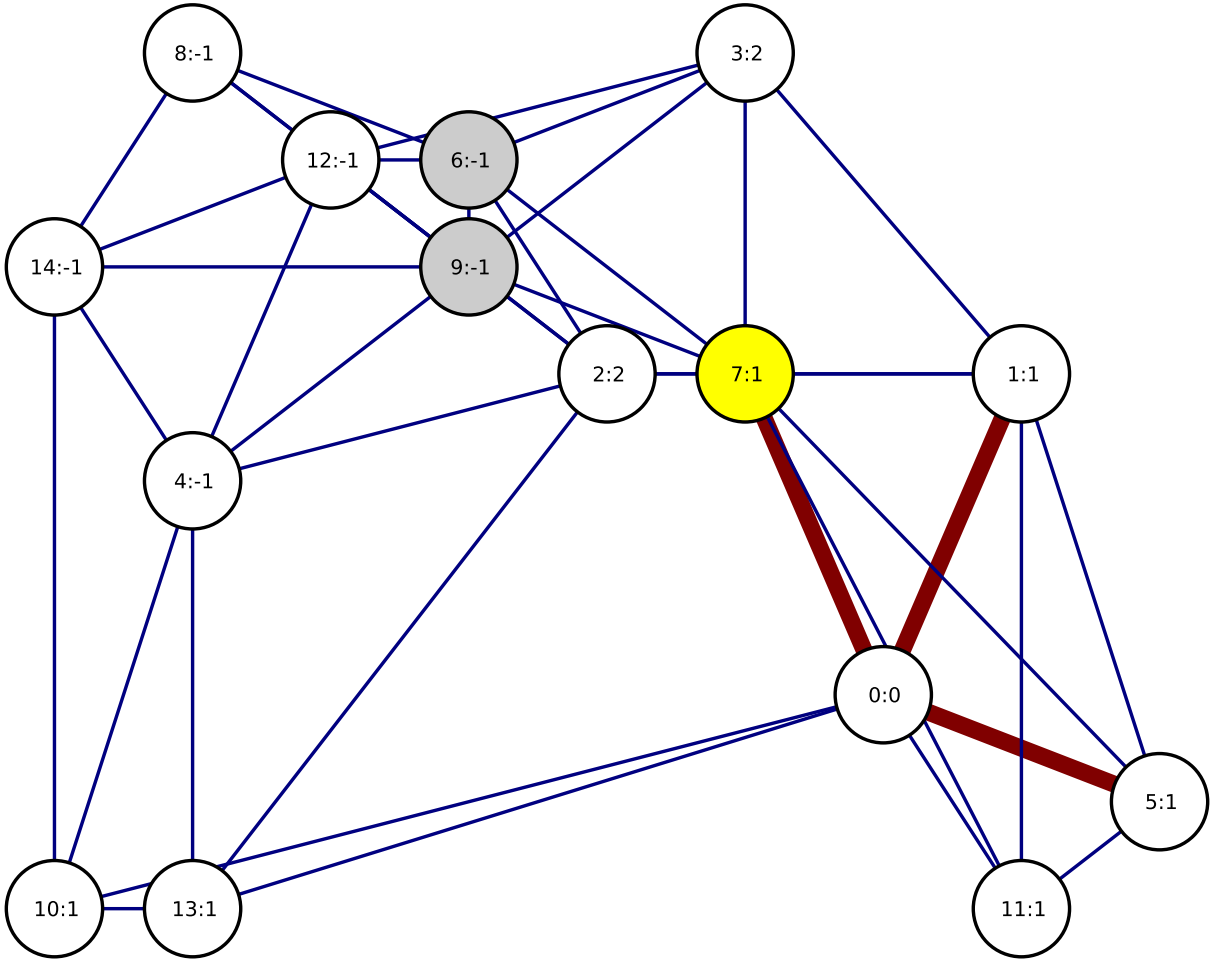
Example BFS



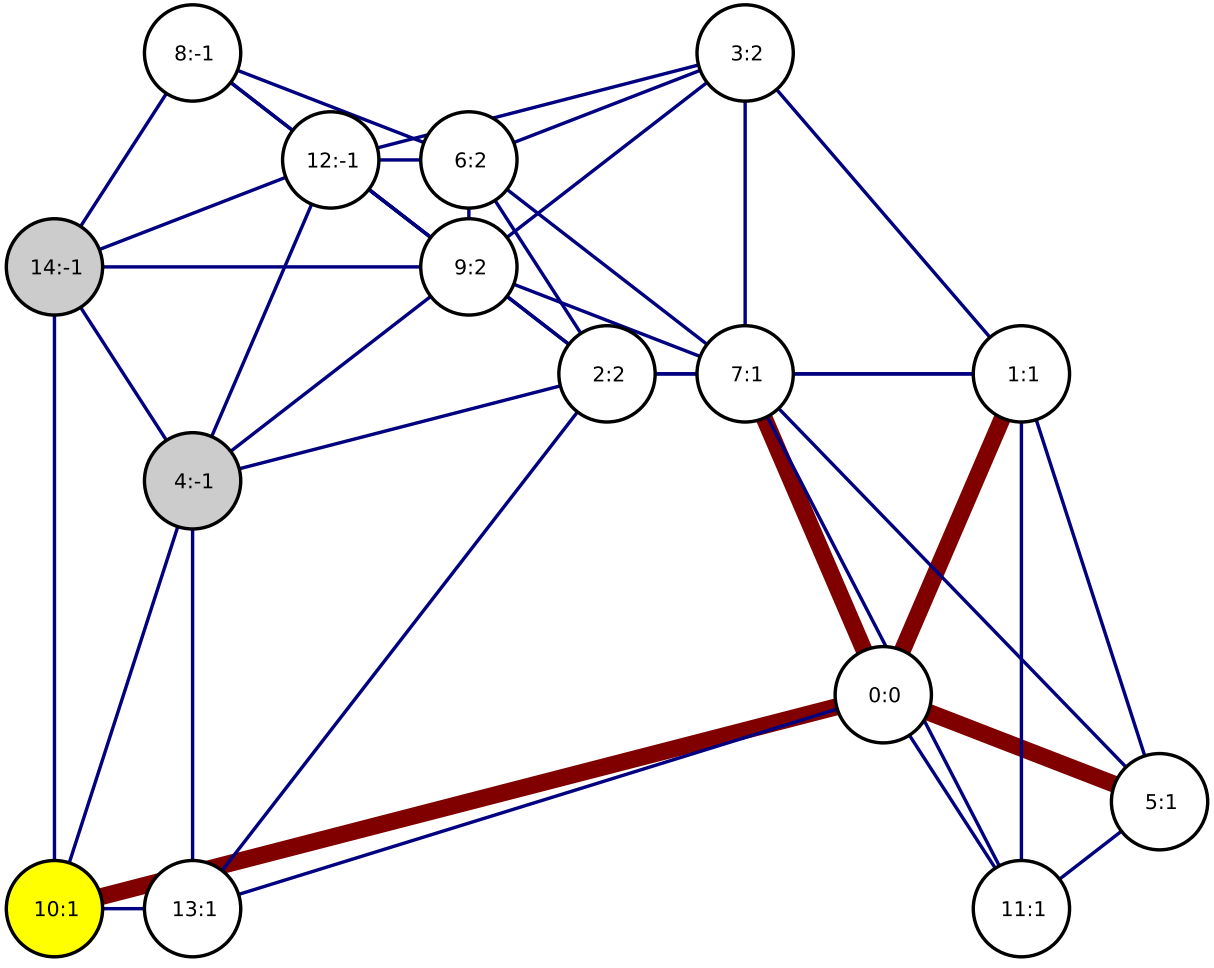
Example BFS



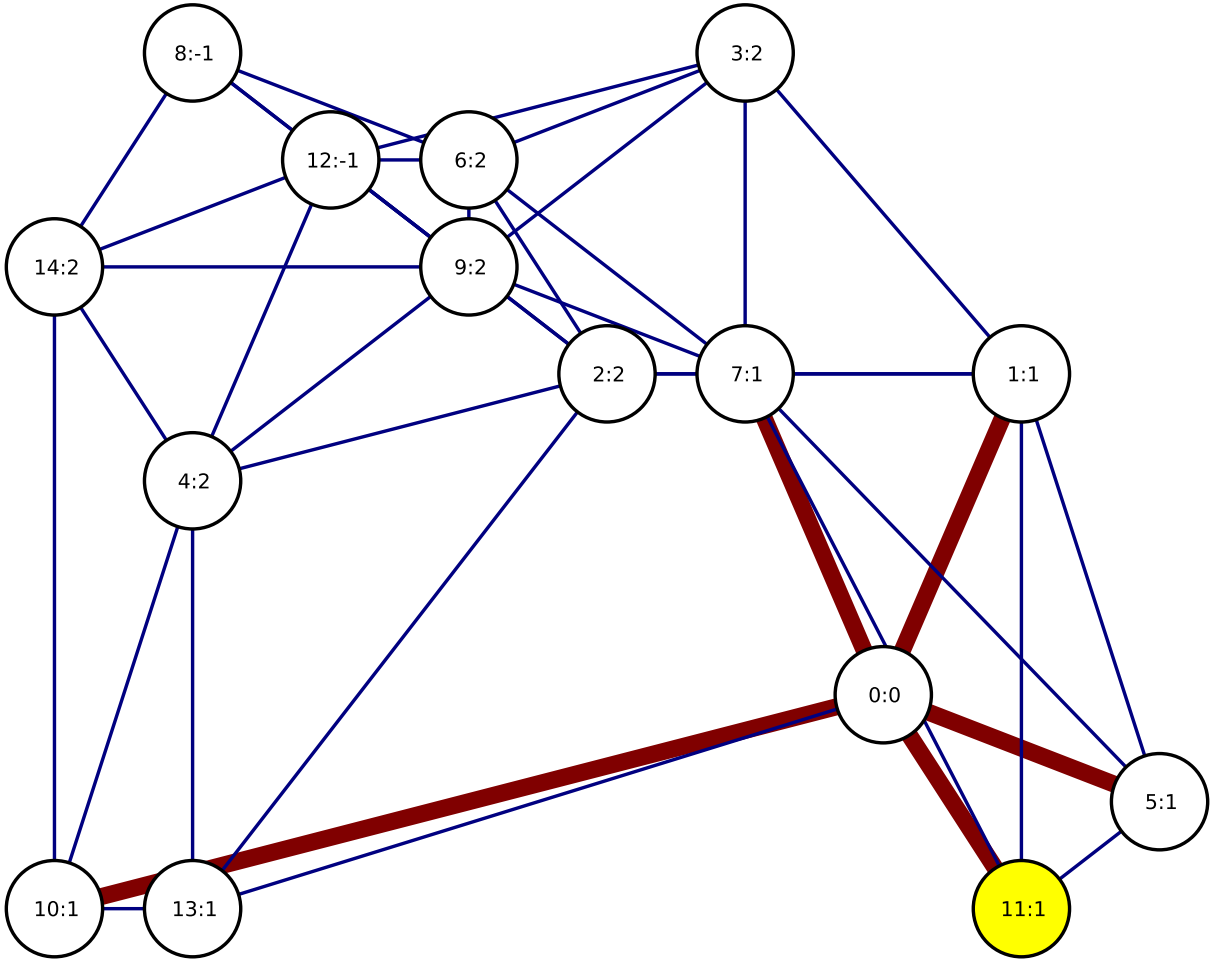
Example BFS



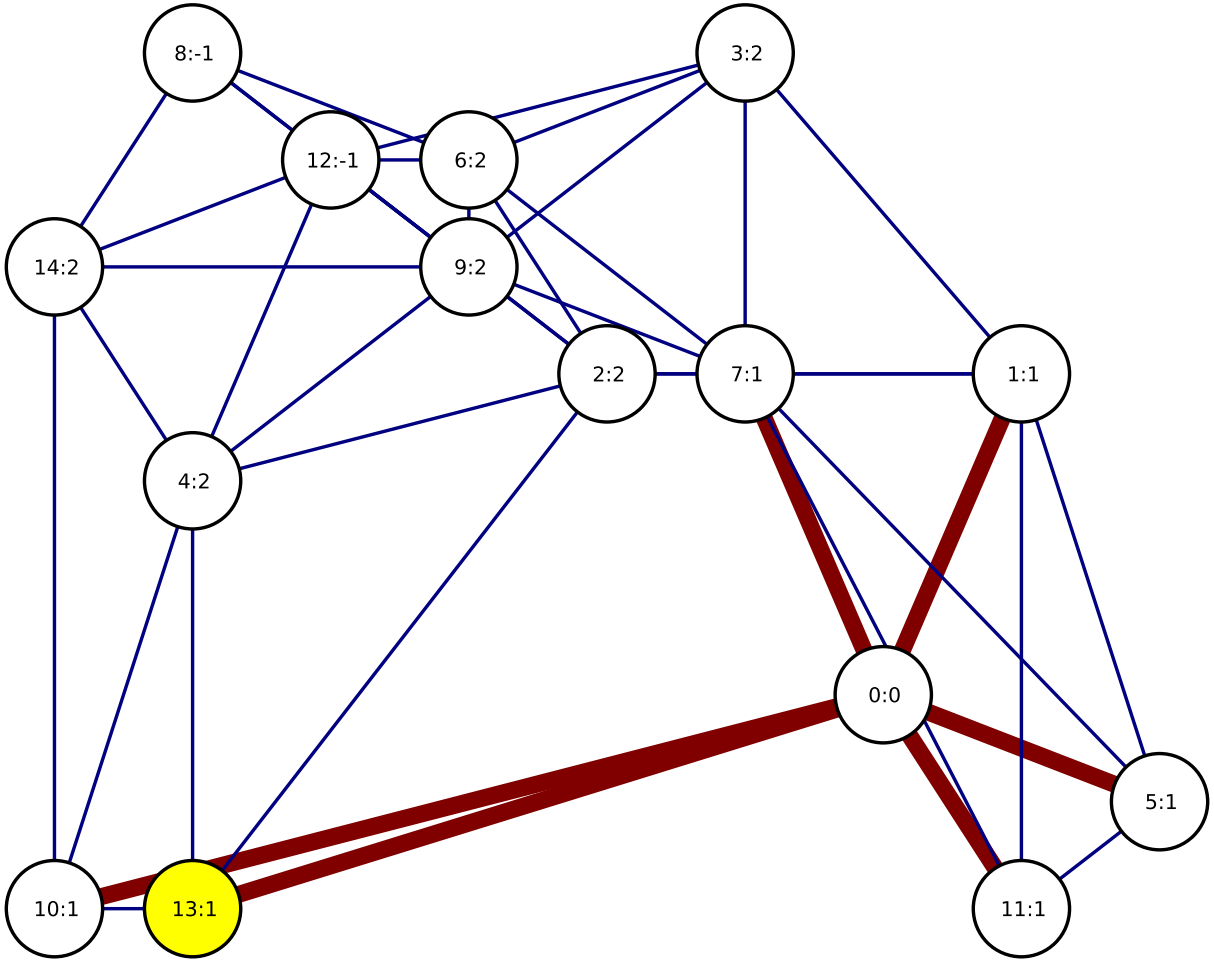
Example BFS



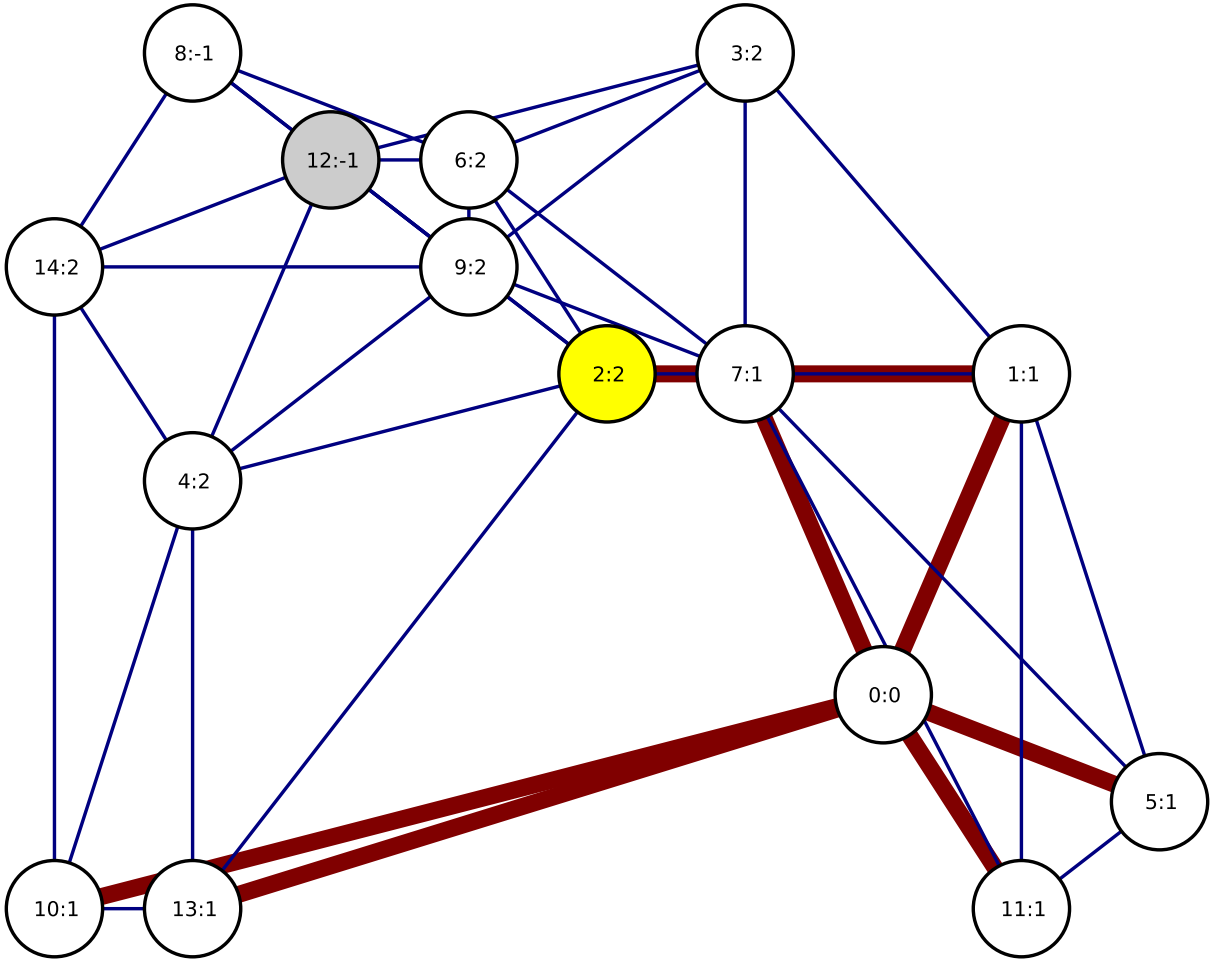
Example BFS



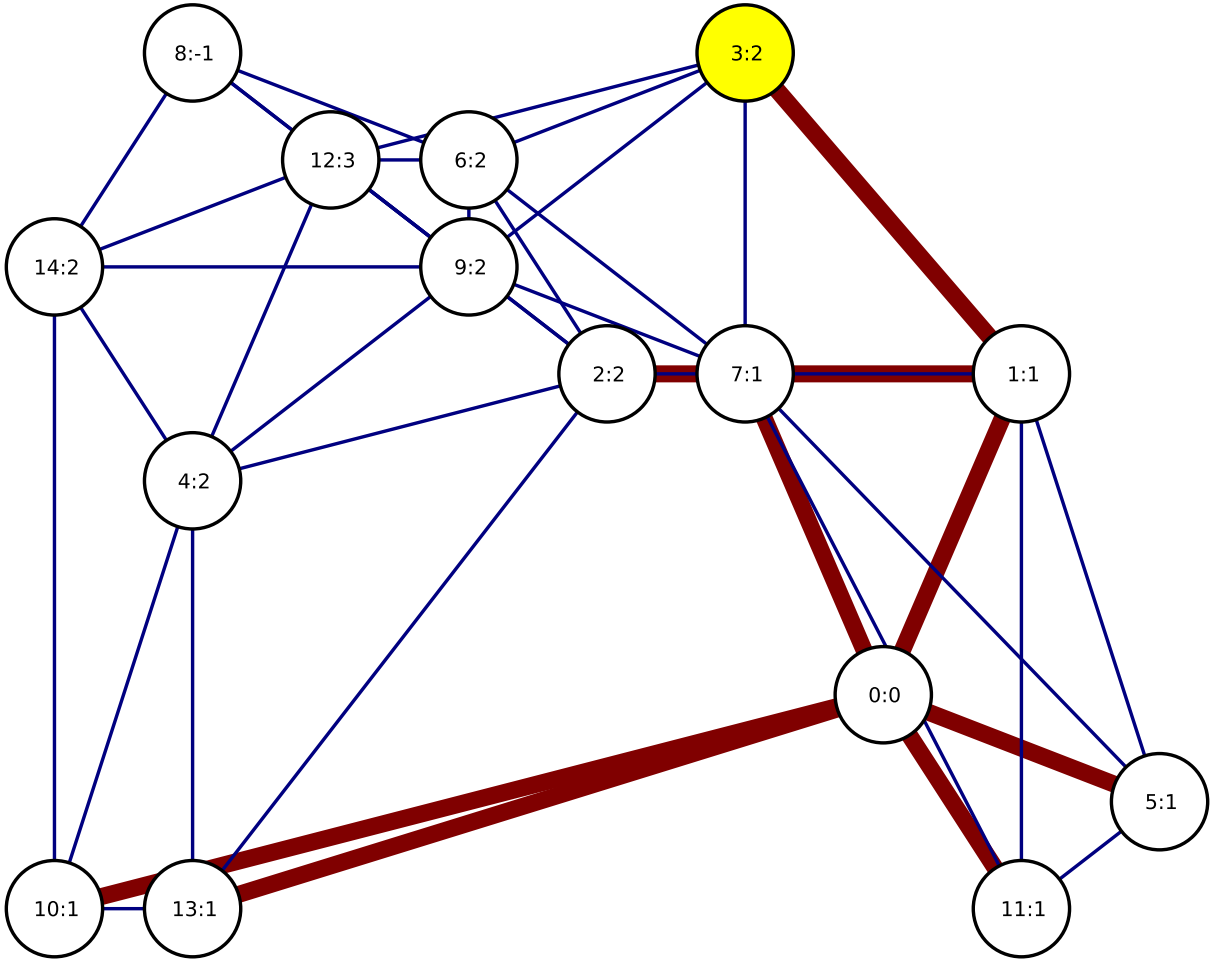
Example BFS



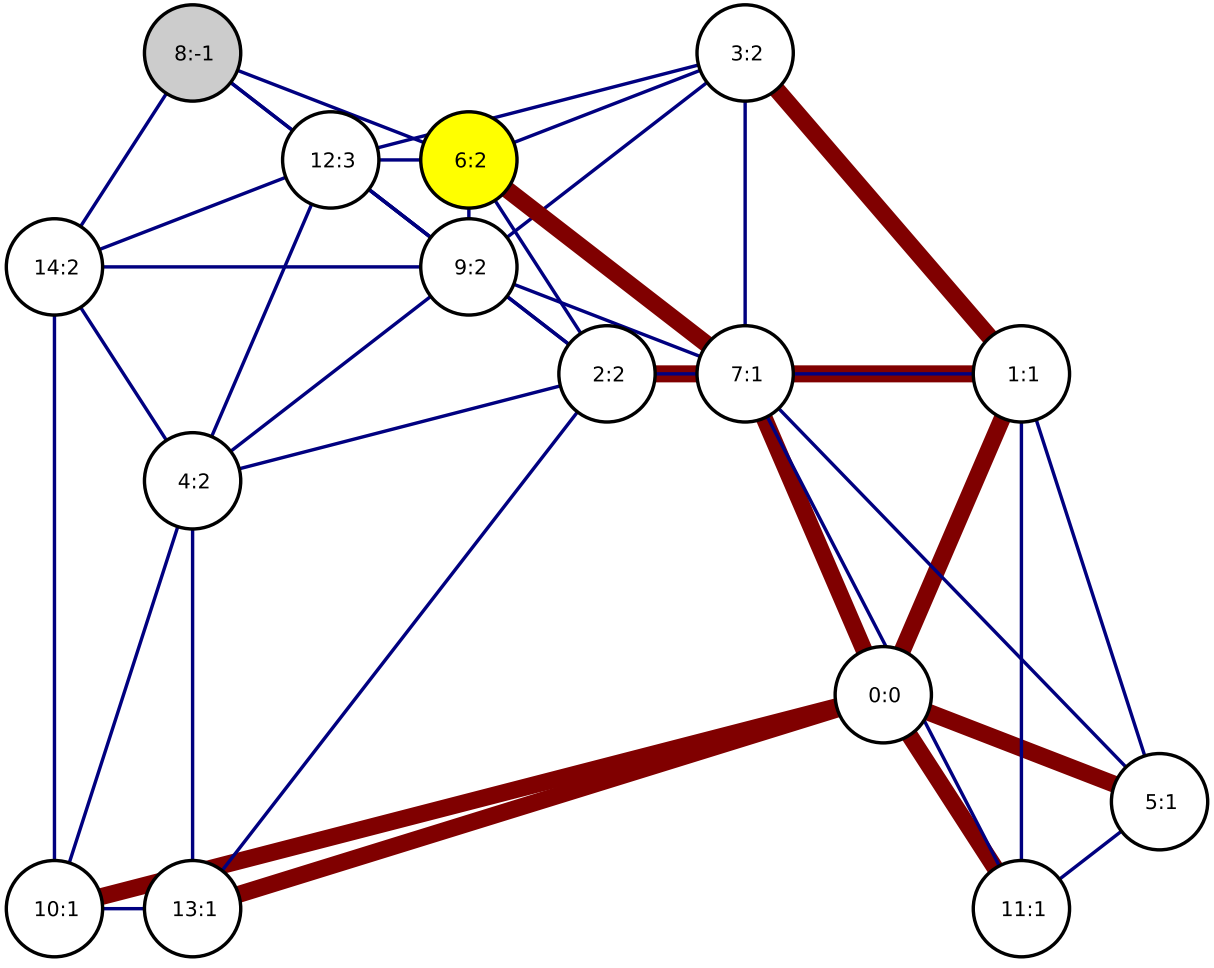
Example BFS



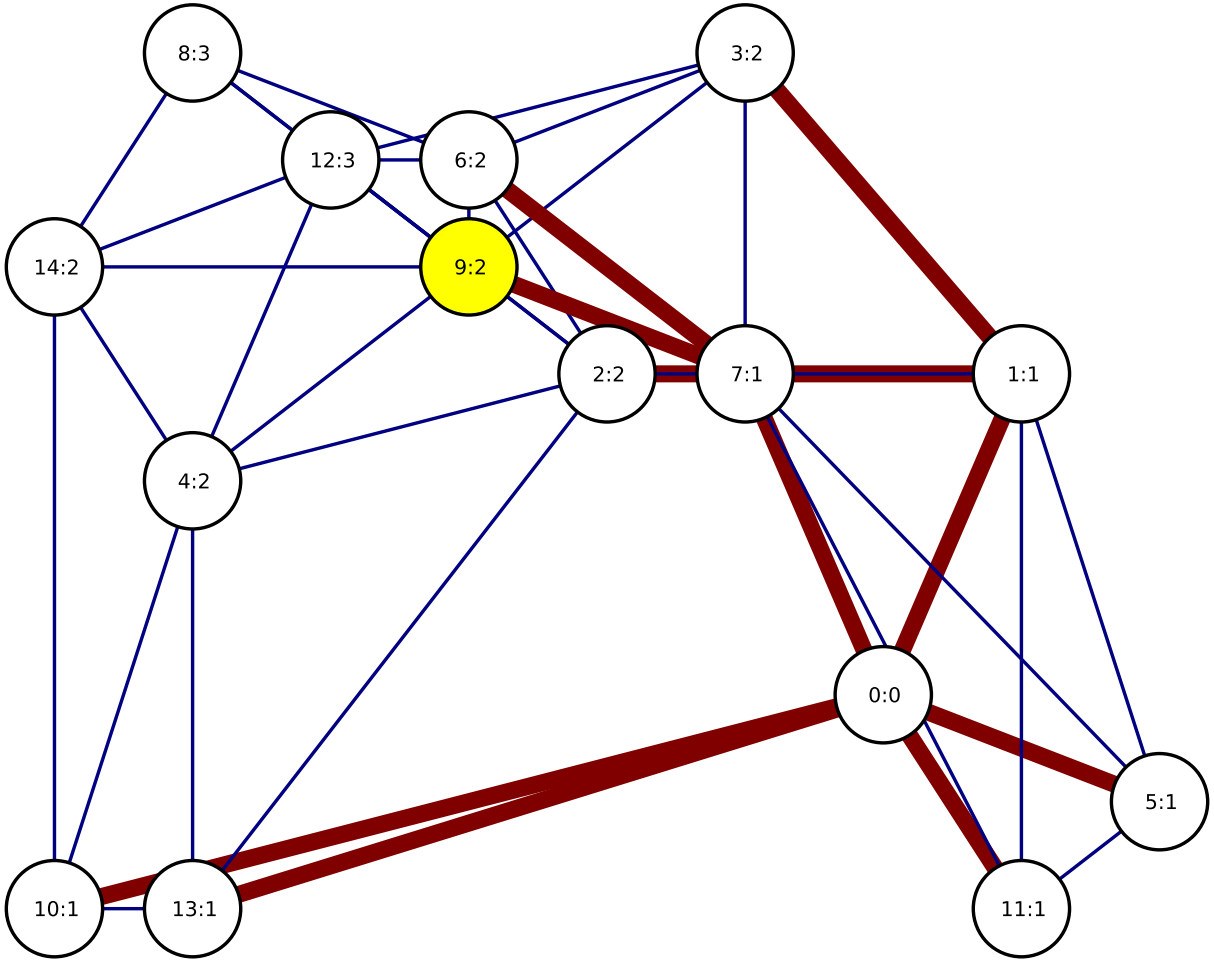
Example BFS



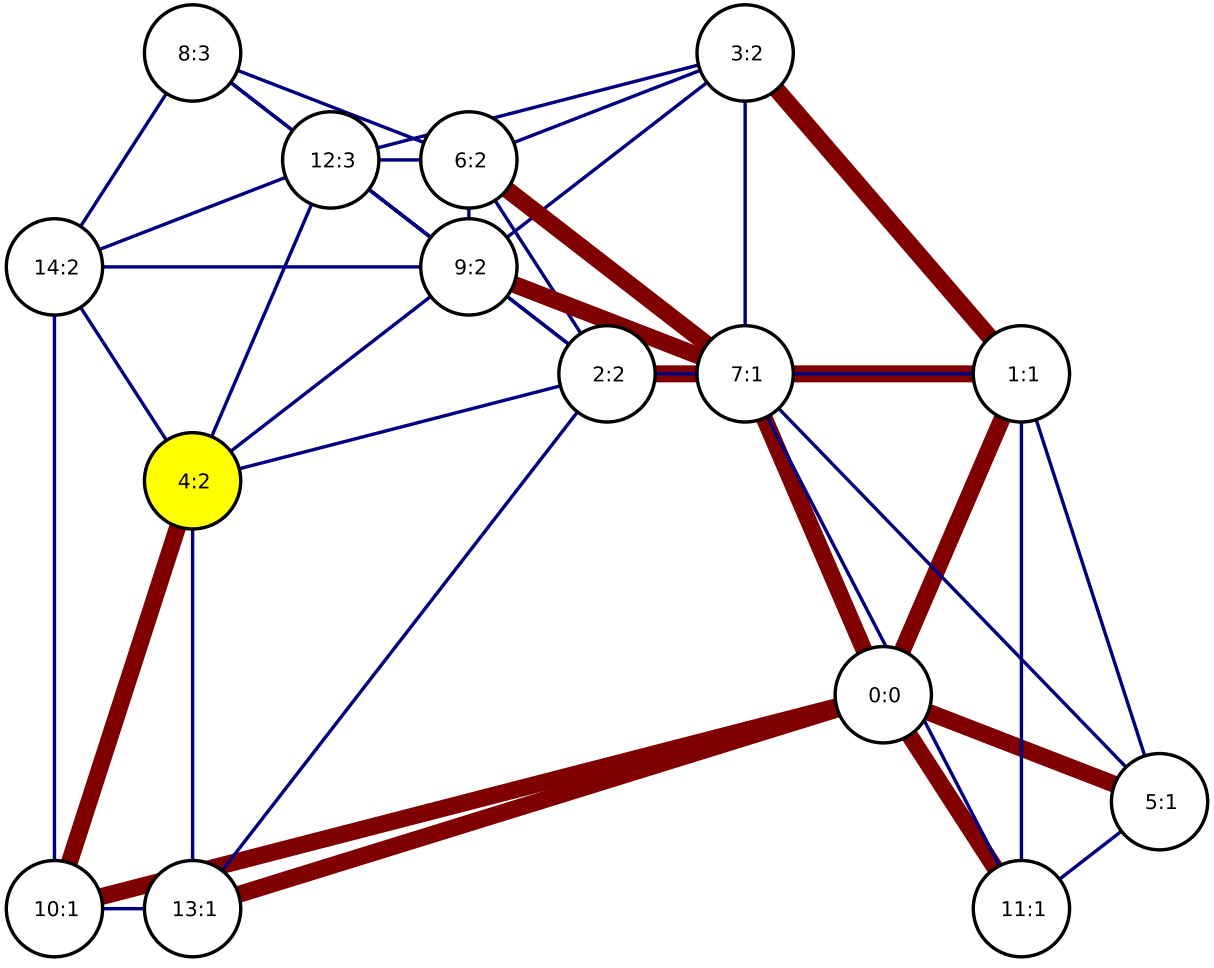
Example BFS



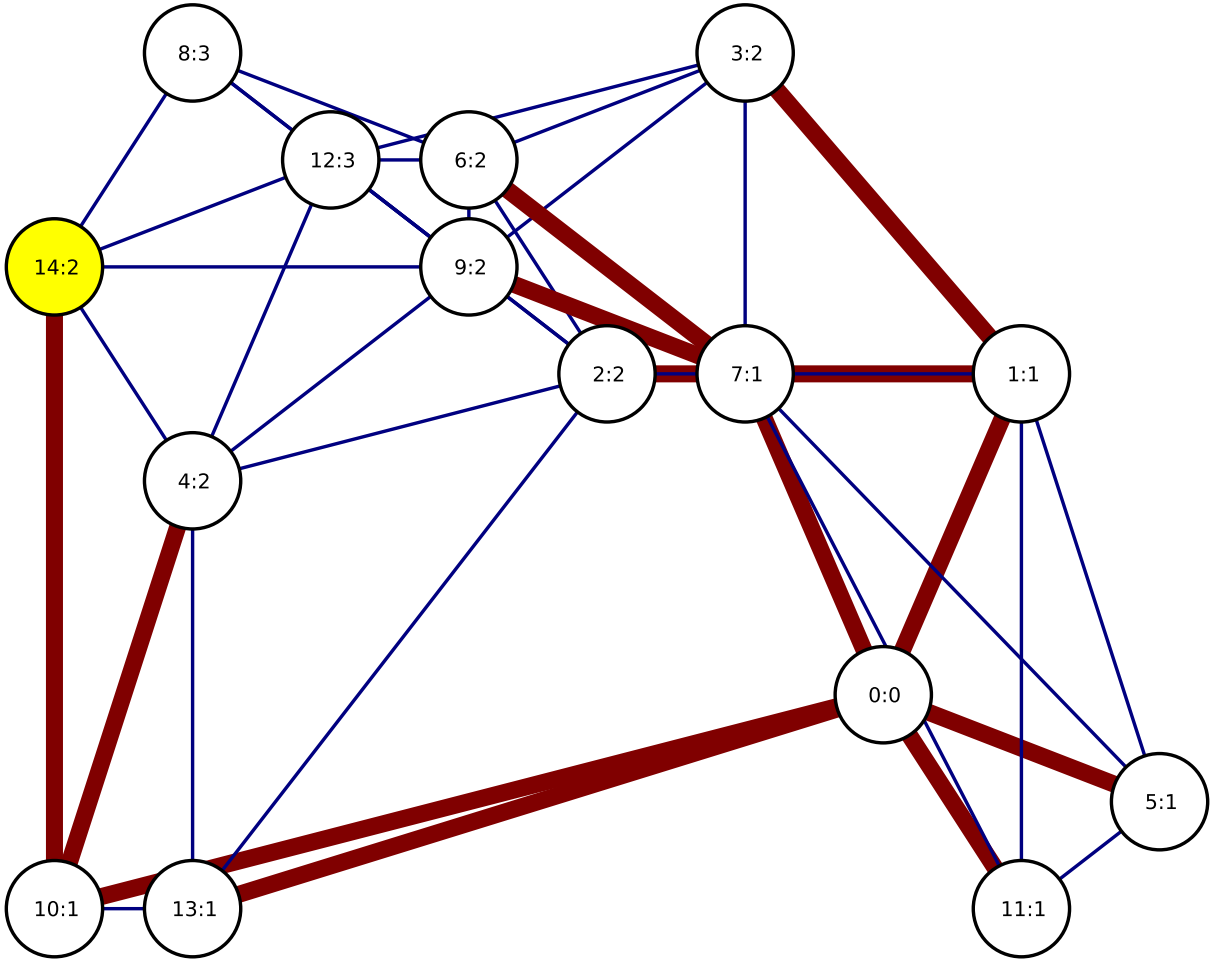
Example BFS



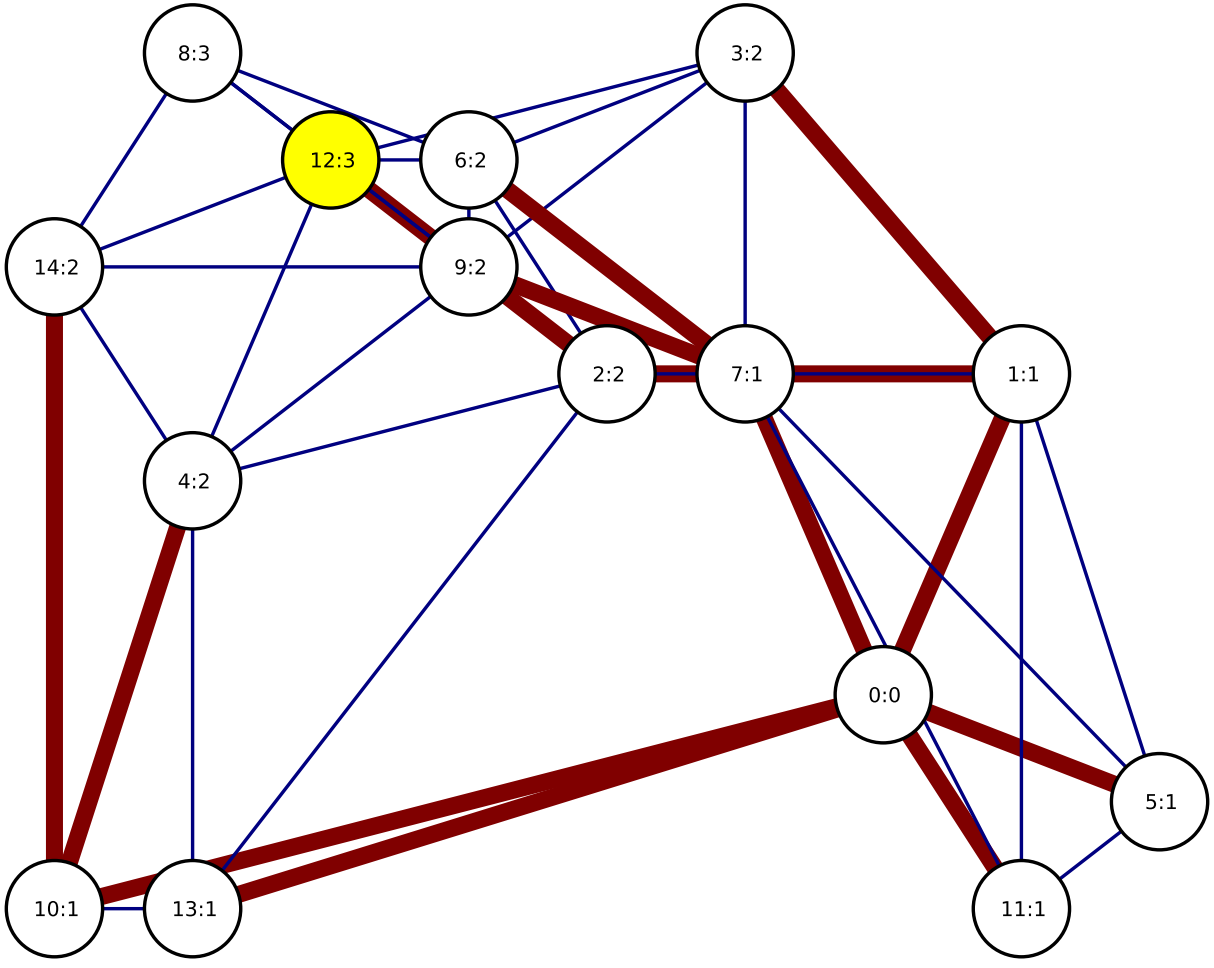
Example BFS



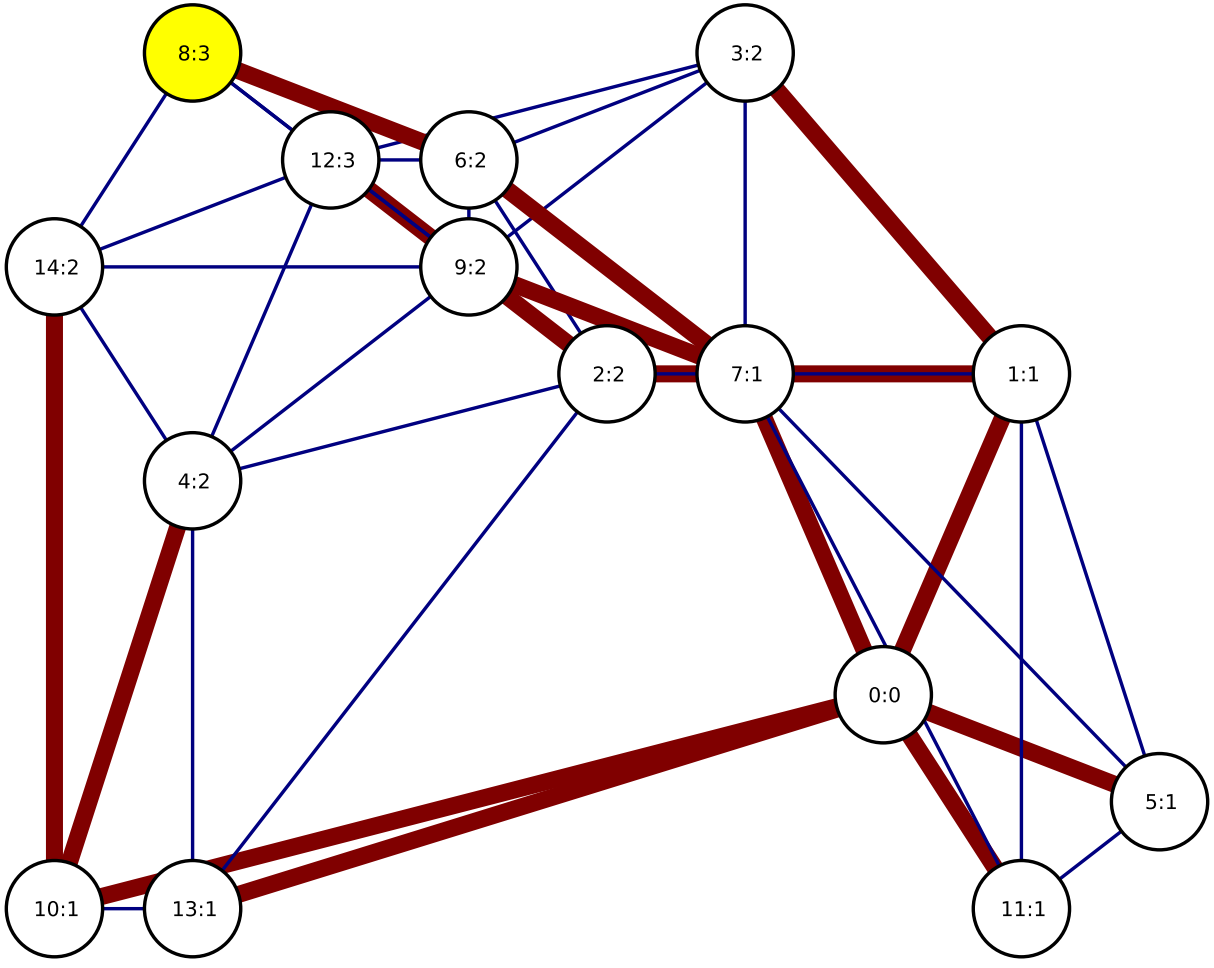
Example BFS



Example BFS



Example BFS



Depth-First Search

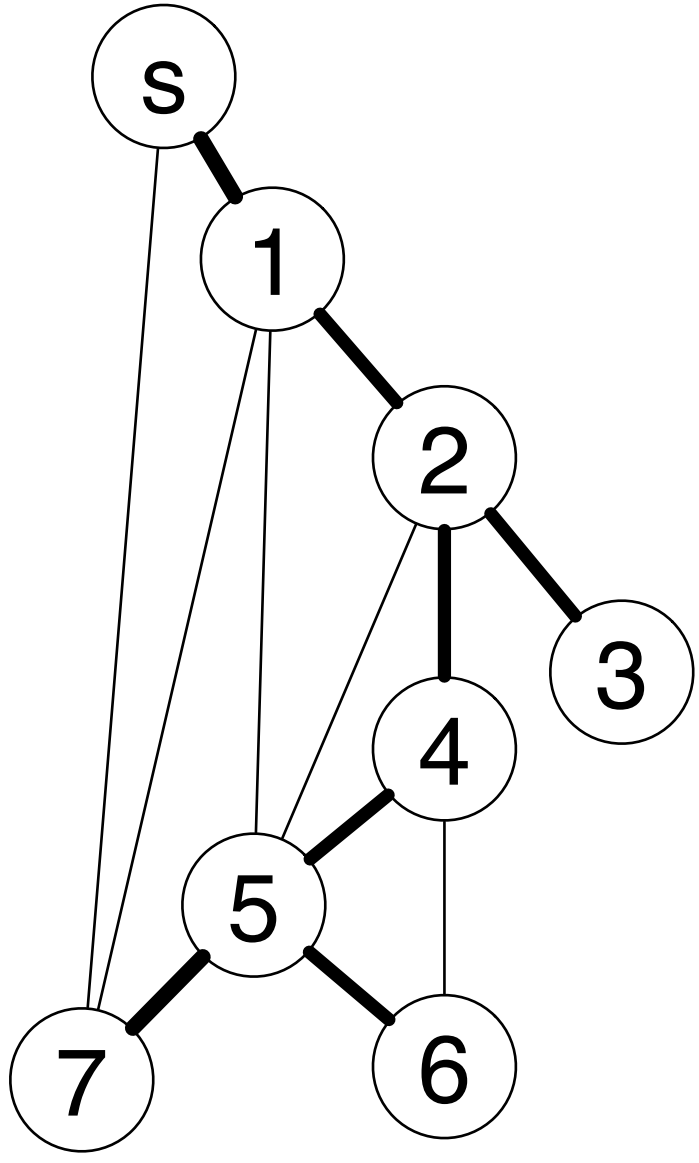
DFS keeps walking down a path until it is forced to backtrack.

It backtracks until it finds a new path to go down.

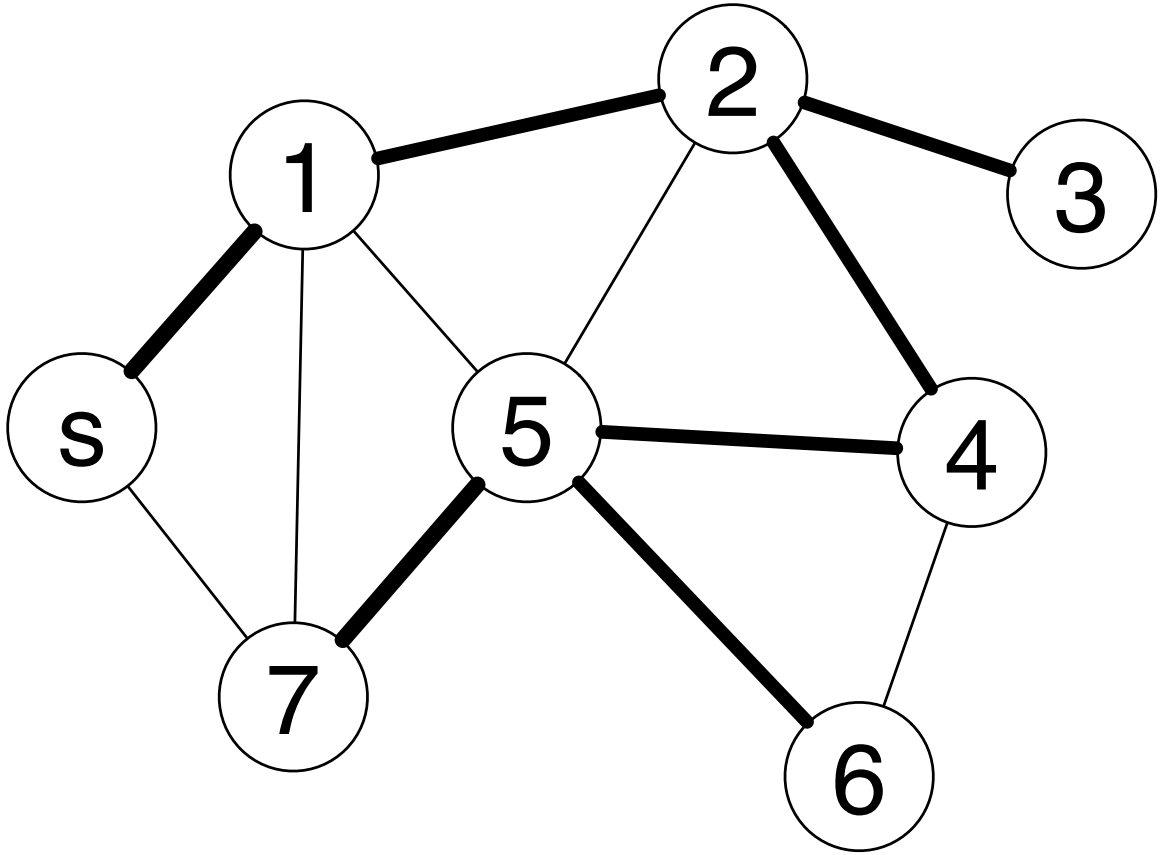
Think: Solving a maze.

It results in a search tree, called the depth-first search tree.

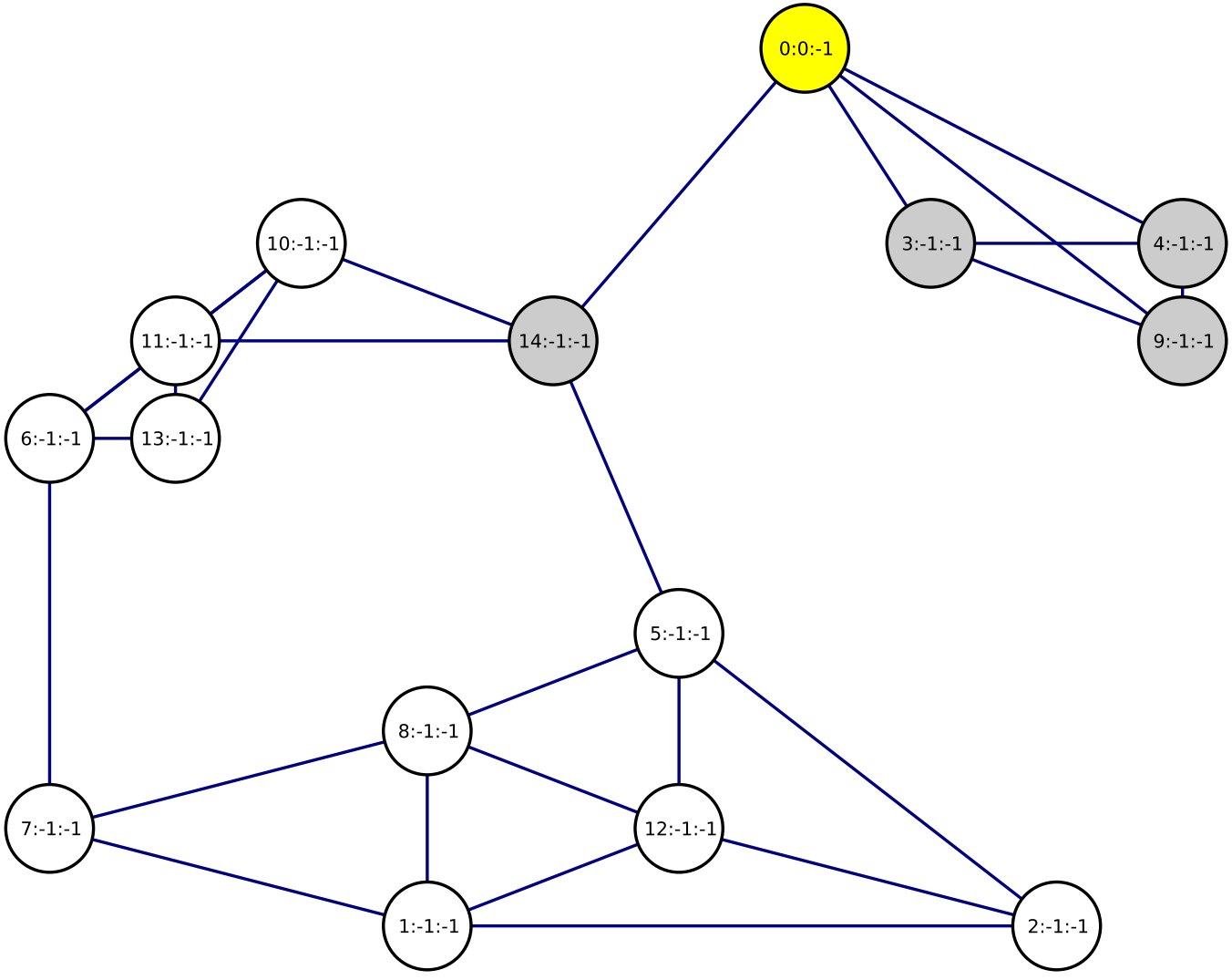
In general, the DFS tree will be very different than the BFS tree.



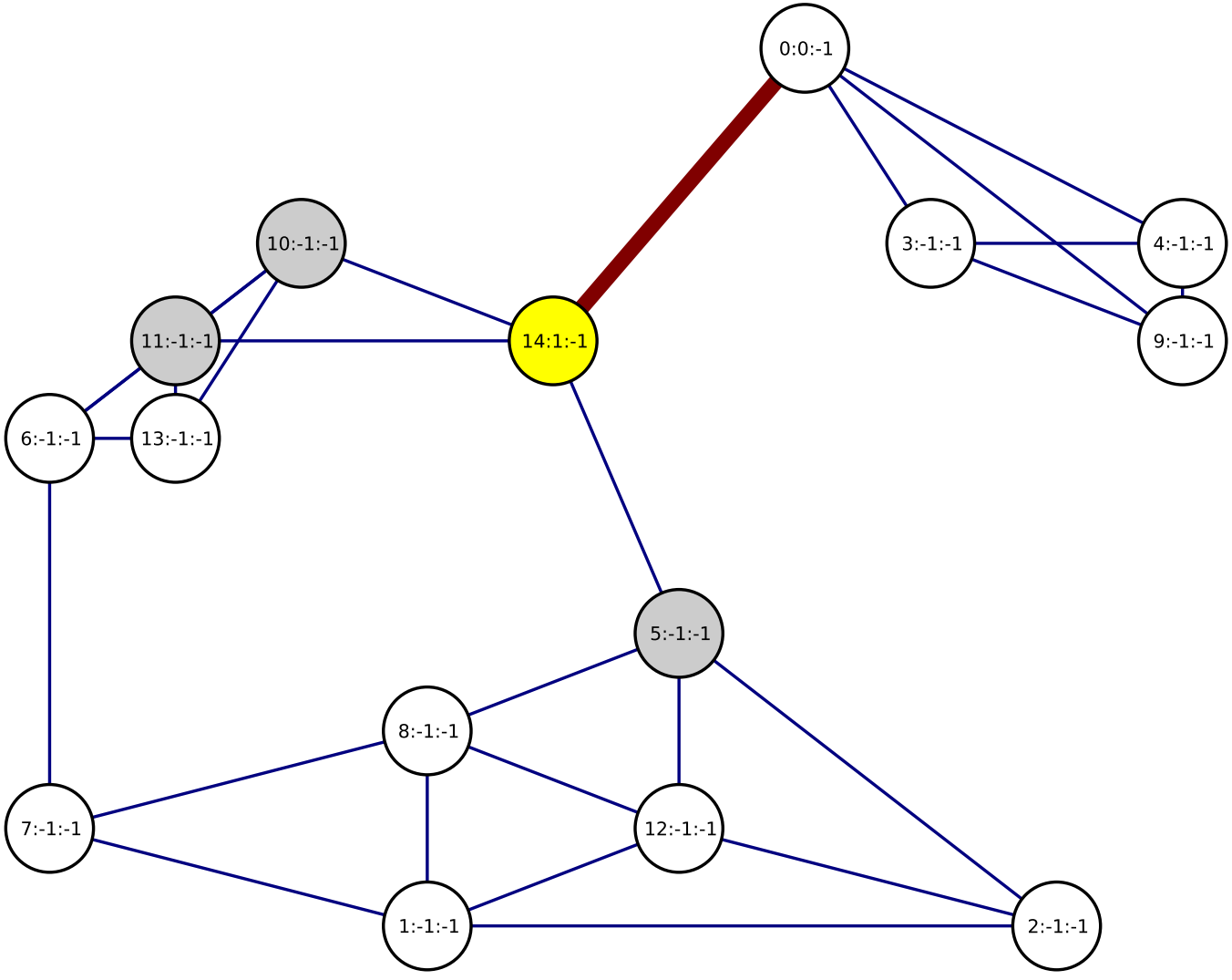
Depth-First Search



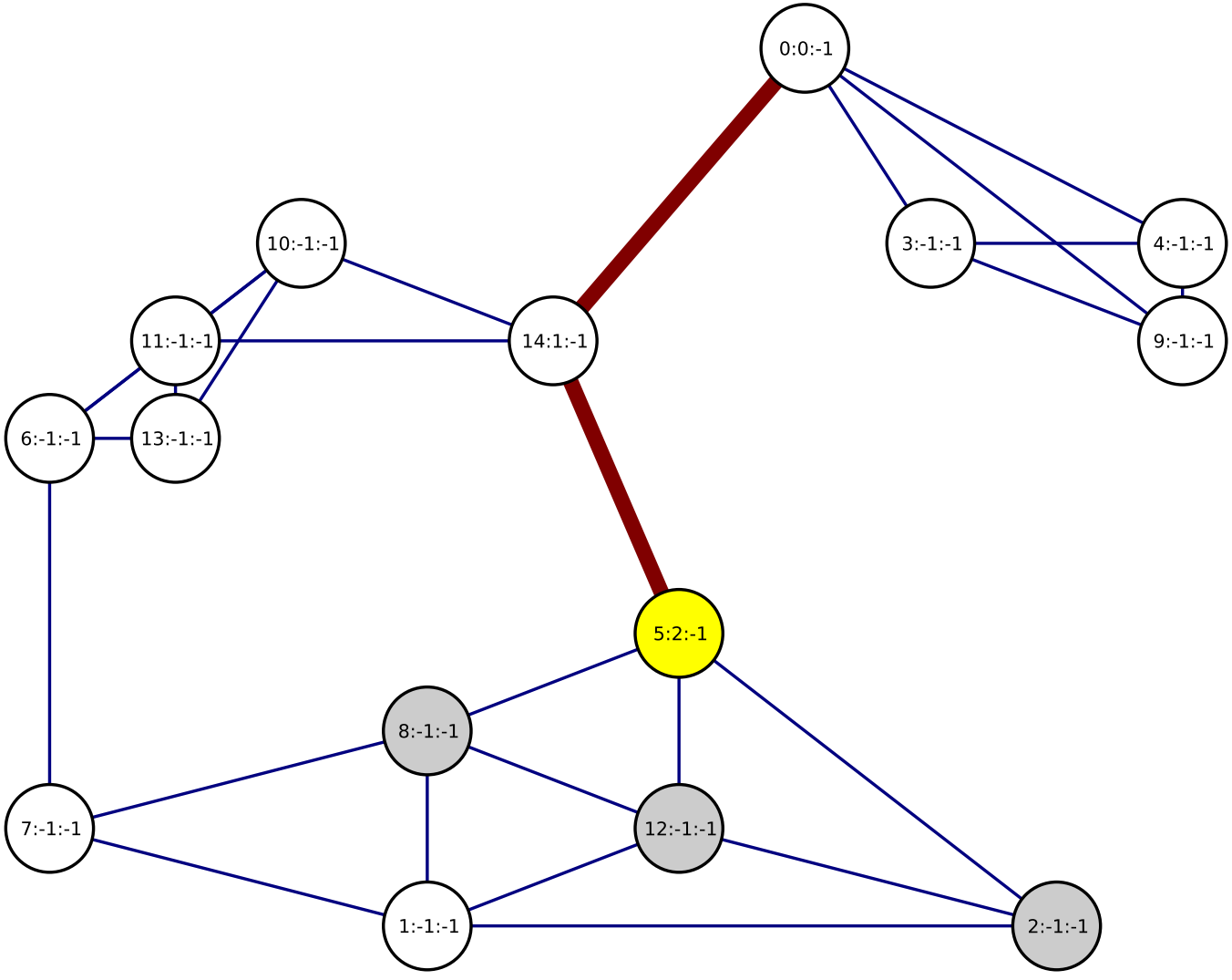
Example DFS


0:0:-1
10:-1:-1
3:-1:-1
4:-1:-1
11:-1:-1
14:-1:-1
9:-1:-1
6:-1:-1
13:-1:-1
5:-1:-1
8:-1:-1
7:-1:-1
12:-1:-1
1:-1:-1
2:-1:-1

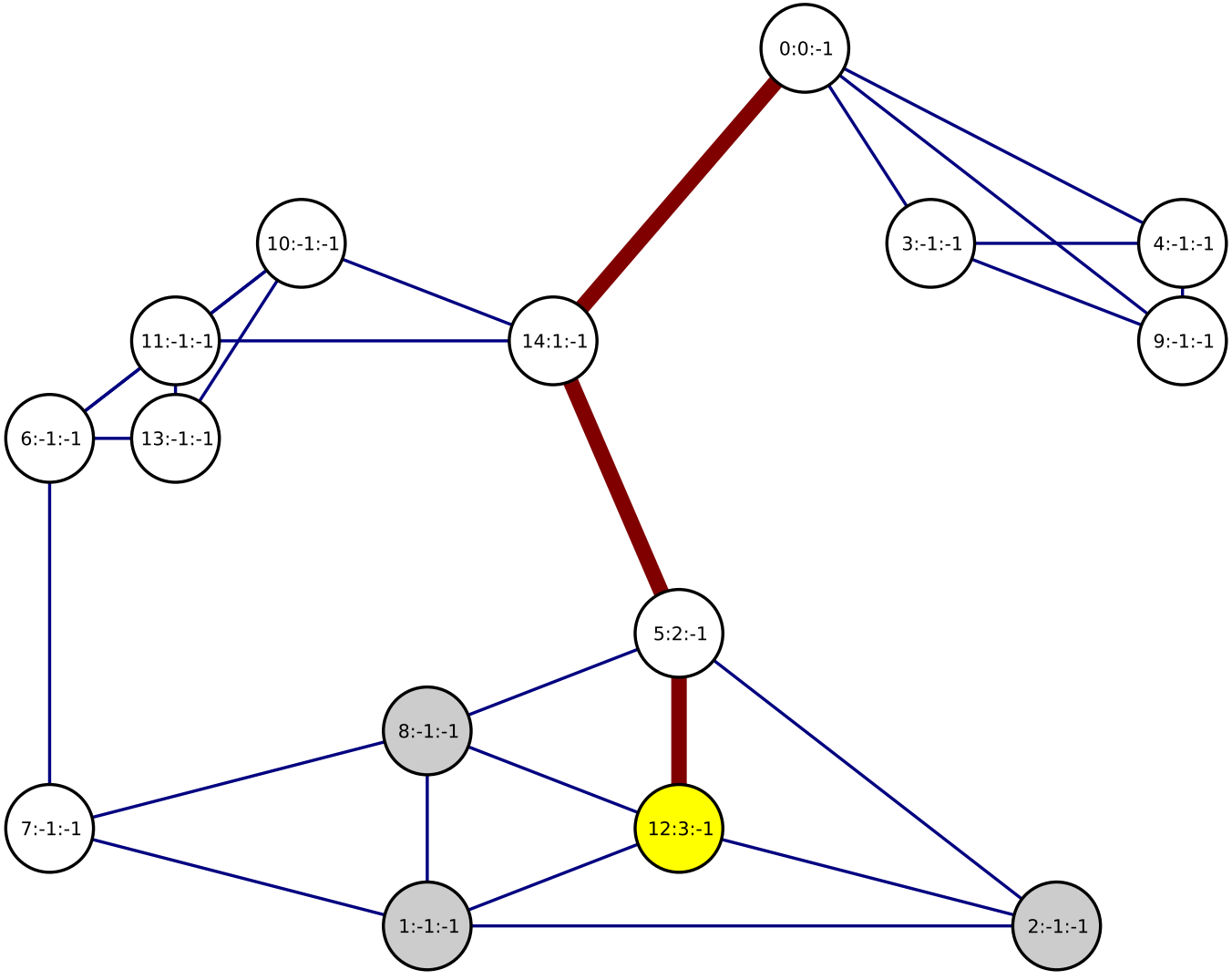
Example DFS


0:0:-1
10:-1:-1
3:-1:-1
4:-1:-1
11:-1:-1
14:1:-1
9:-1:-1
6:-1:-1
13:-1:-1
5:-1:-1
8:-1:-1
7:-1:-1
12:-1:-1
1:-1:-1
2:-1:-1

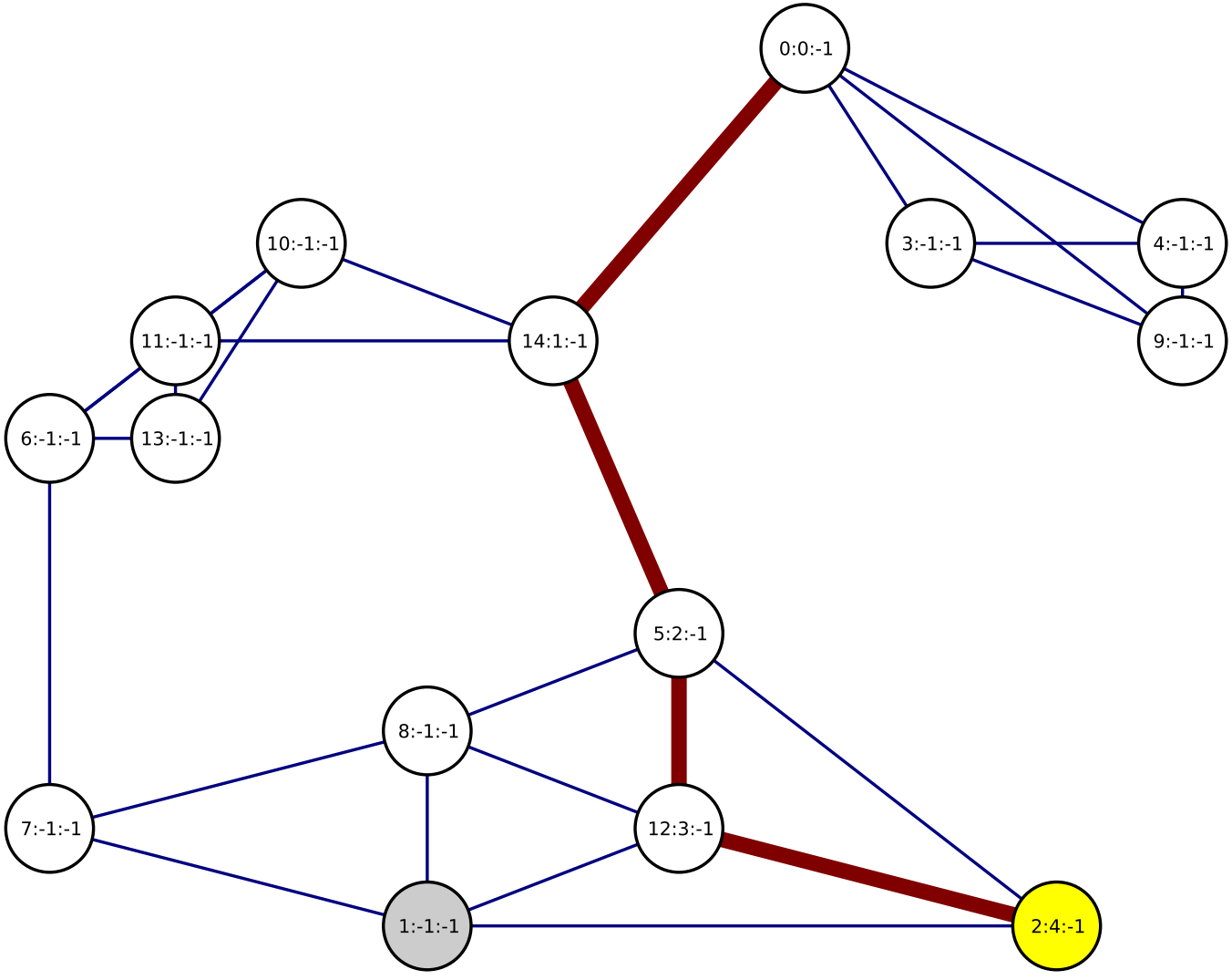
Example DFS


0:0:-1
10:-1:-1
3:-1:-1
4:-1:-1
11:-1:-1
14:1:-1
9:-1:-1
6:-1:-1
13:-1:-1
5:2:-1
8:-1:-1
7:-1:-1
12:-1:-1
1:-1:-1
2:-1:-1

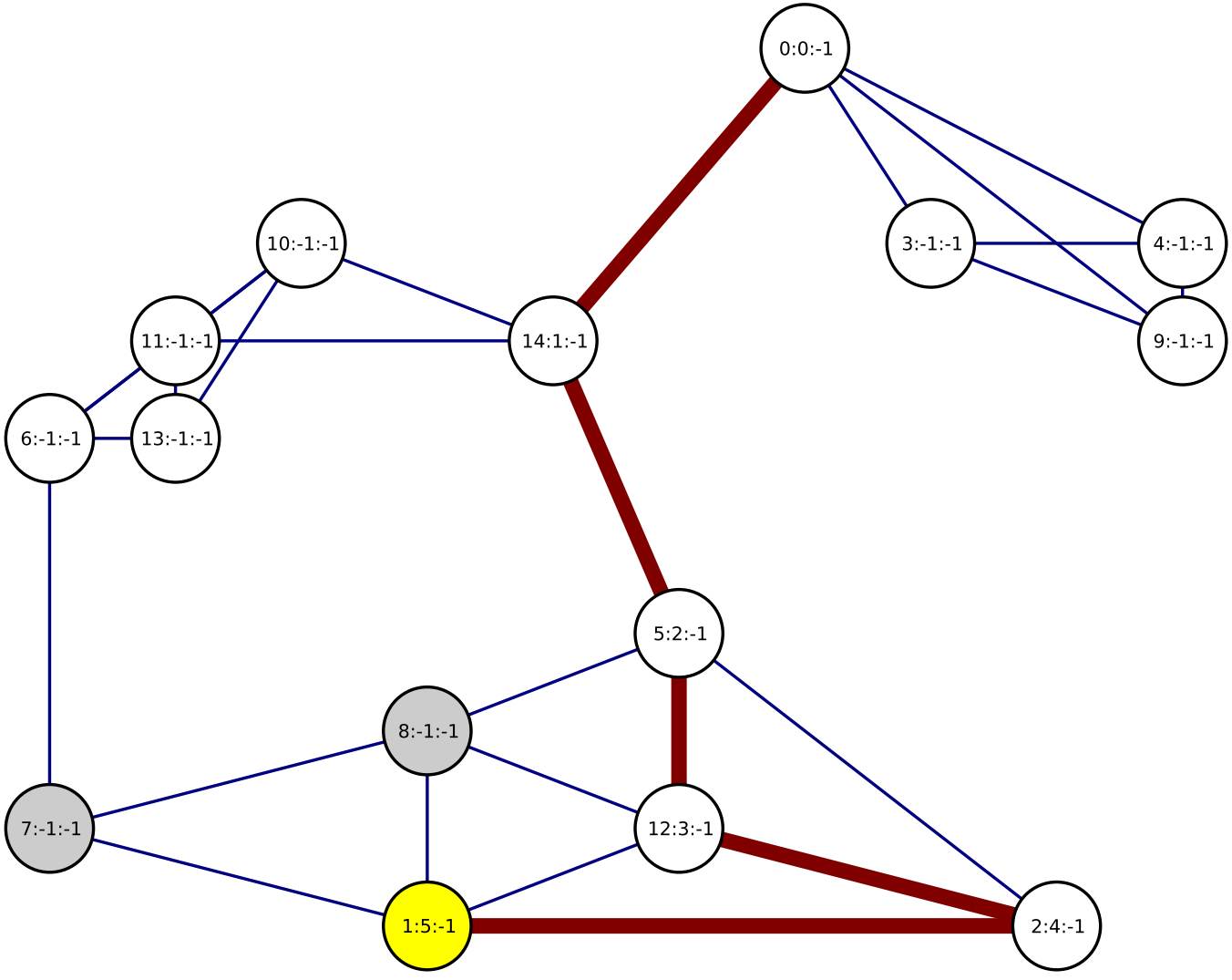
Example DFS


0:0:-1
10:-1:-1
3:-1:-1
4:-1:-1
11:-1:-1
14:1:-1
9:-1:-1
6:-1:-1
13:-1:-1
5:2:-1
8:-1:-1
7:-1:-1
12:3:-1
1:-1:-1
2:-1:-1

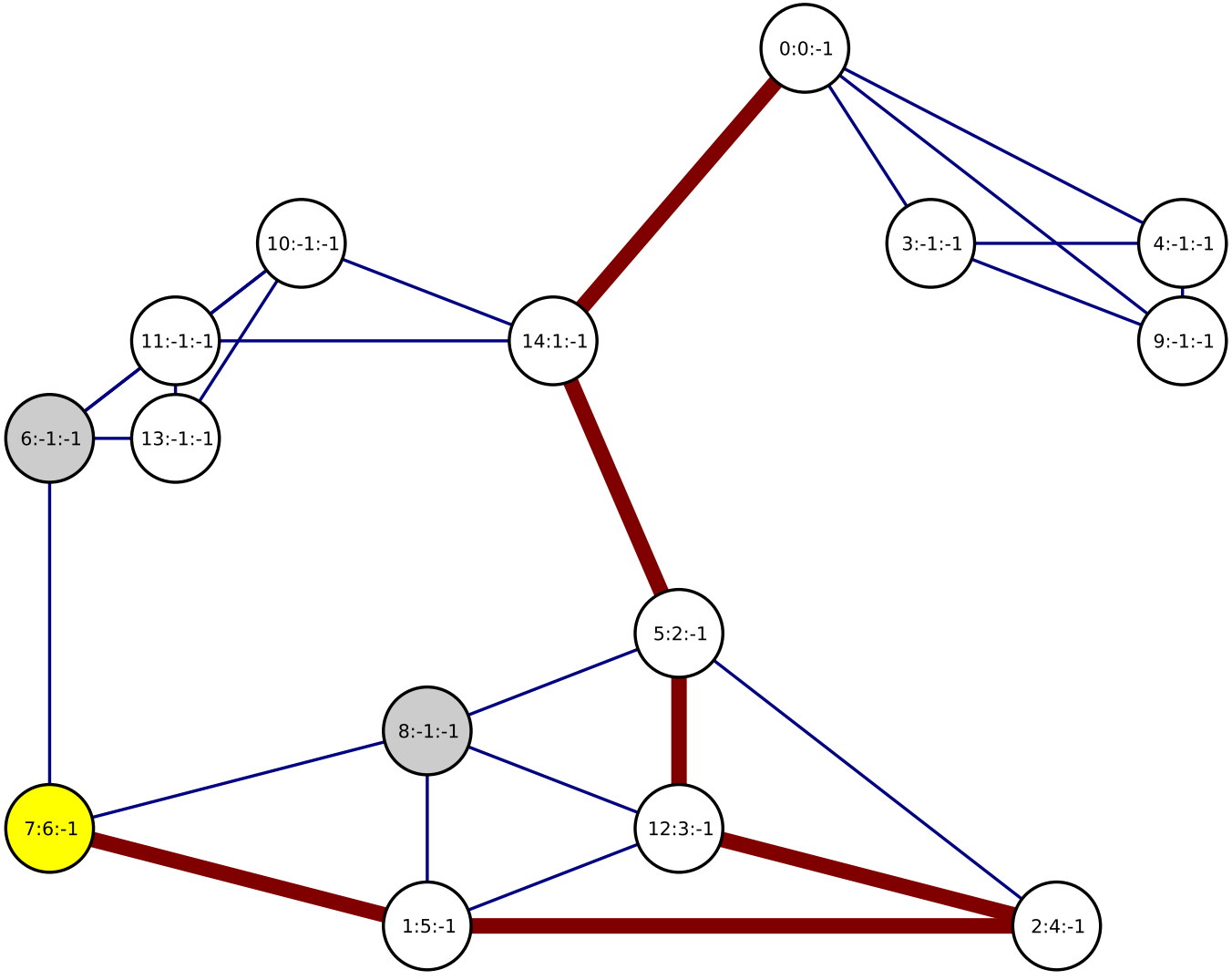
Example DFS


0:0:-1
10:-1:-1
3:-1:-1
4:-1:-1
11:-1:-1
14:1:-1
9:-1:-1
6:-1:-1
13:-1:-1
5:2:-1
8:-1:-1
7:-1:-1
12:3:-1
1:-1:-1
2:4:-1

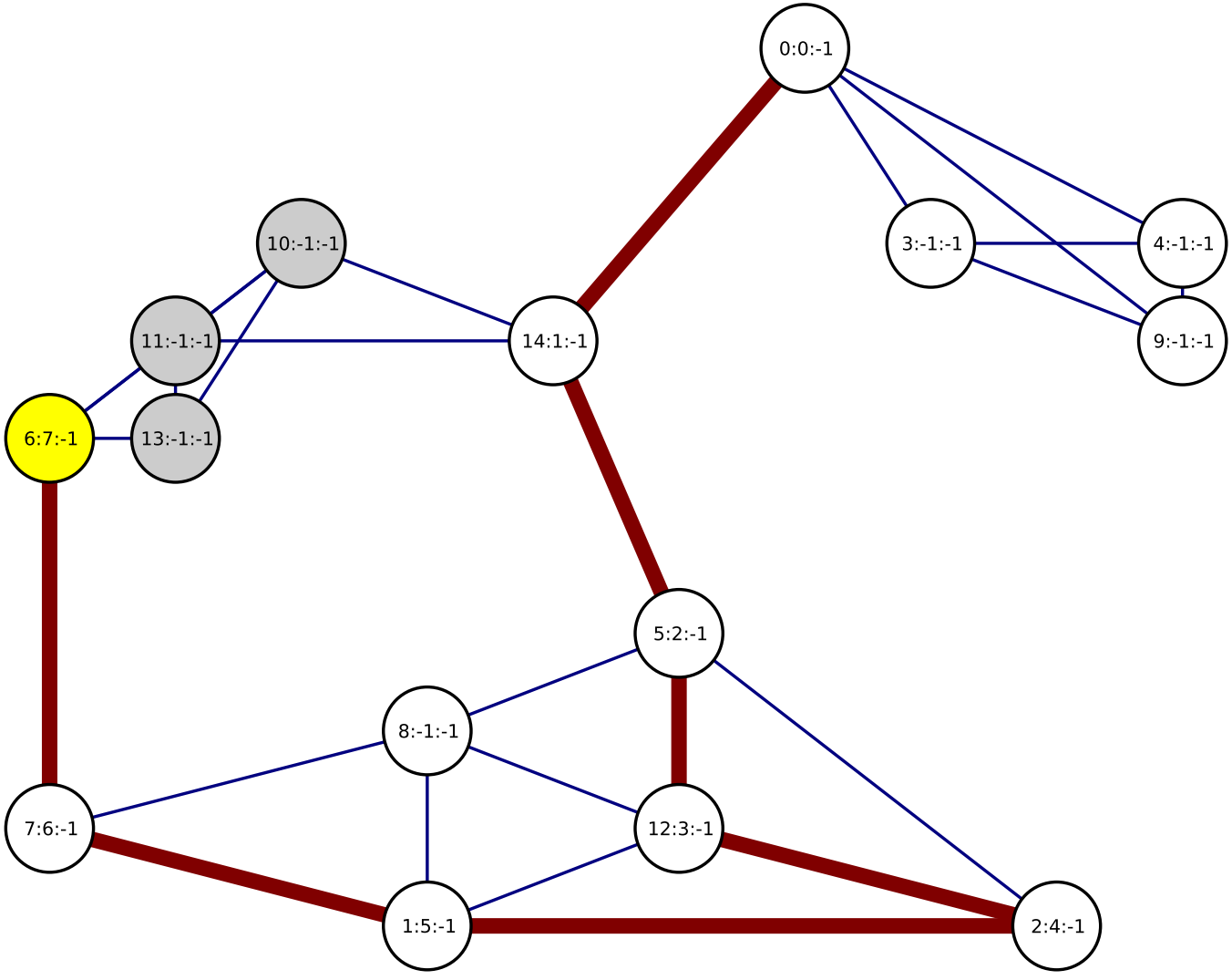
Example DFS


0:0:-1
10:-1:-1
3:-1:-1
4:-1:-1
11:-1:-1
14:1:-1
9:-1:-1
6:-1:-1
13:-1:-1
5:2:-1
8:-1:-1
7:-1:-1
12:3:-1
1:5:-1
2:4:-1

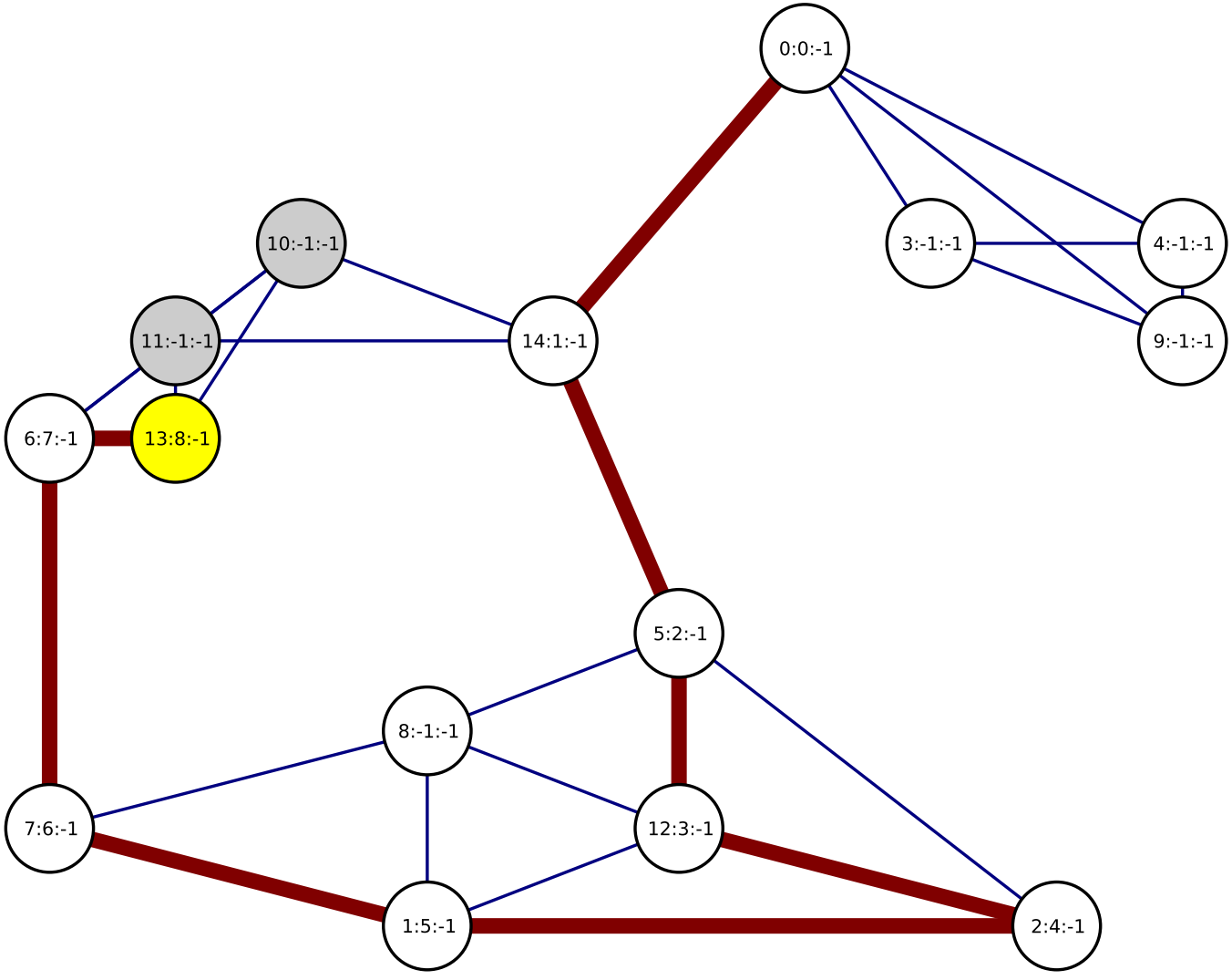
Example DFS


0:0:-1
10:-1:-1
3:-1:-1
4:-1:-1
11:-1:-1
14:1:-1
9:-1:-1
6:-1:-1
13:-1:-1
5:2:-1
8:-1:-1
7:6:-1
12:3:-1
1:5:-1
2:4:-1

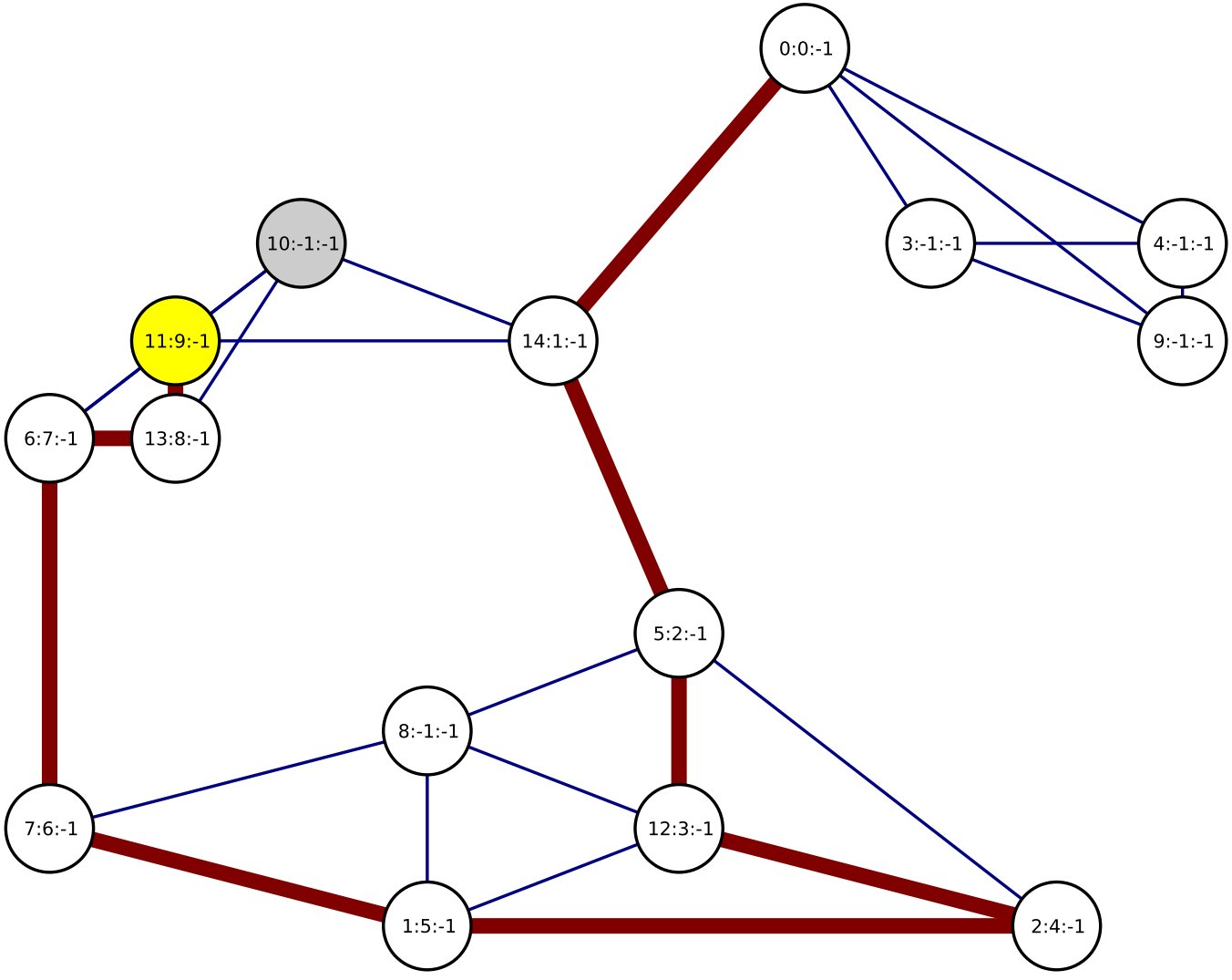
Example DFS


0:0:-1
10:-1:-1
3:-1:-1
4:-1:-1
11:-1:-1
14:1:-1
9:-1:-1
6:7:-1
13:-1:-1
5:2:-1
8:-1:-1
7:6:-1
12:3:-1
1:5:-1
2:4:-1

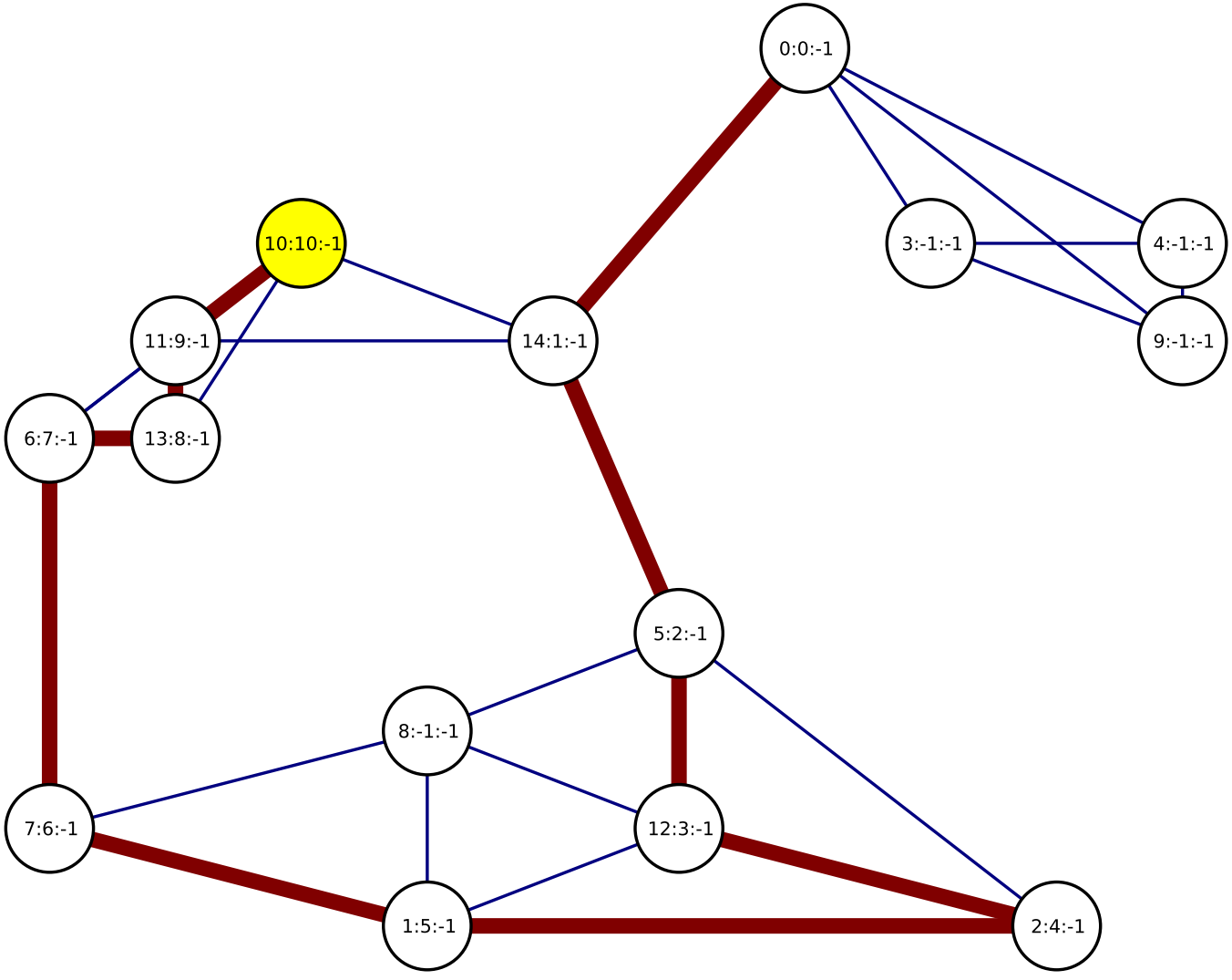
Example DFS


0:0:-1
10:-1:-1
3:-1:-1
4:-1:-1
11:-1:-1
14:1:-1
9:-1:-1
6:7:-1
13:8:-1
5:2:-1
8:-1:-1
7:6:-1
12:3:-1
1:5:-1
2:4:-1

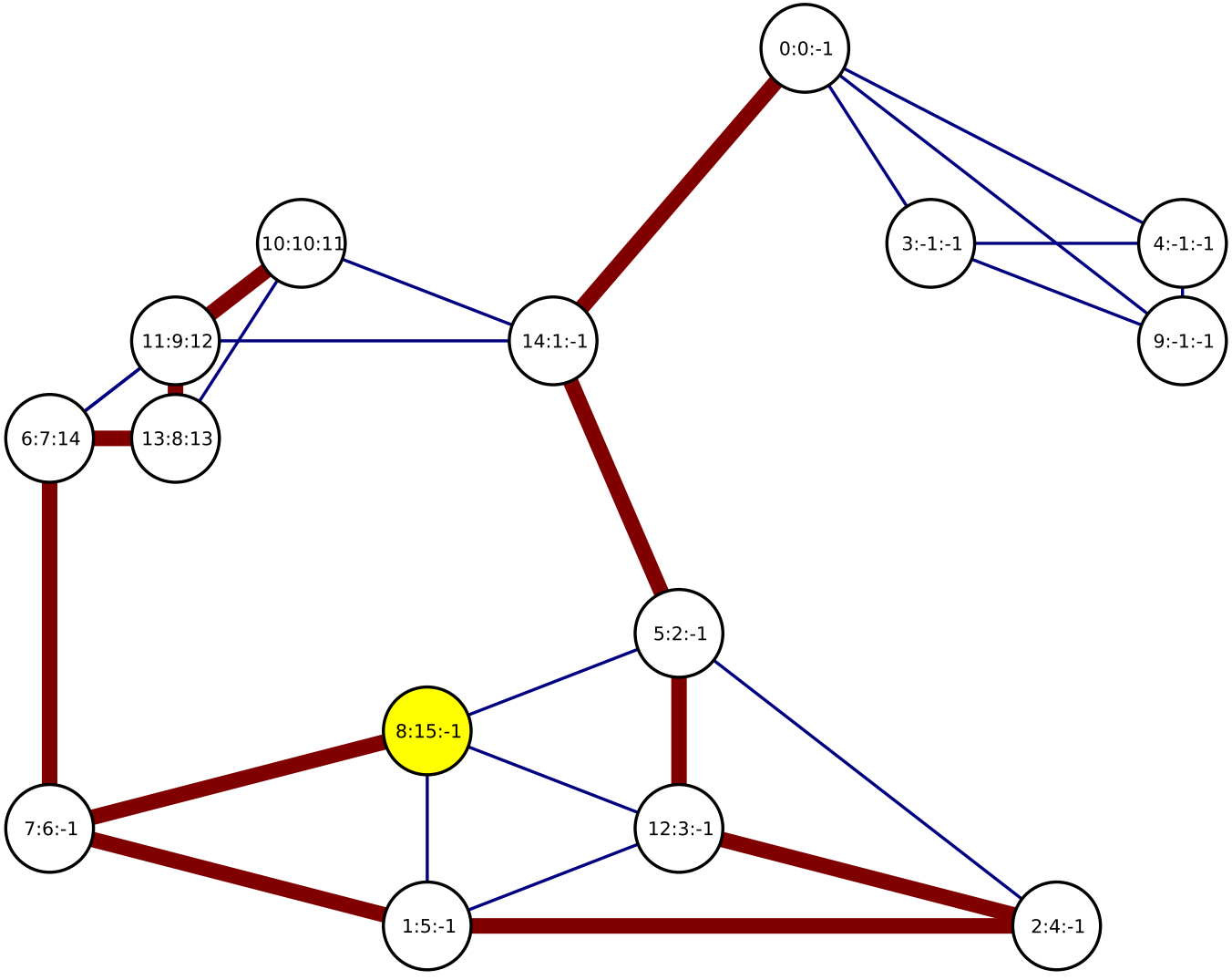
Example DFS


0:0:-1
10:-1:-1
3:-1:-1
4:-1:-1
11:9:-1
14:1:-1
9:-1:-1
6:7:-1
13:8:-1
5:2:-1
8:-1:-1
7:6:-1
12:3:-1
1:5:-1
2:4:-1

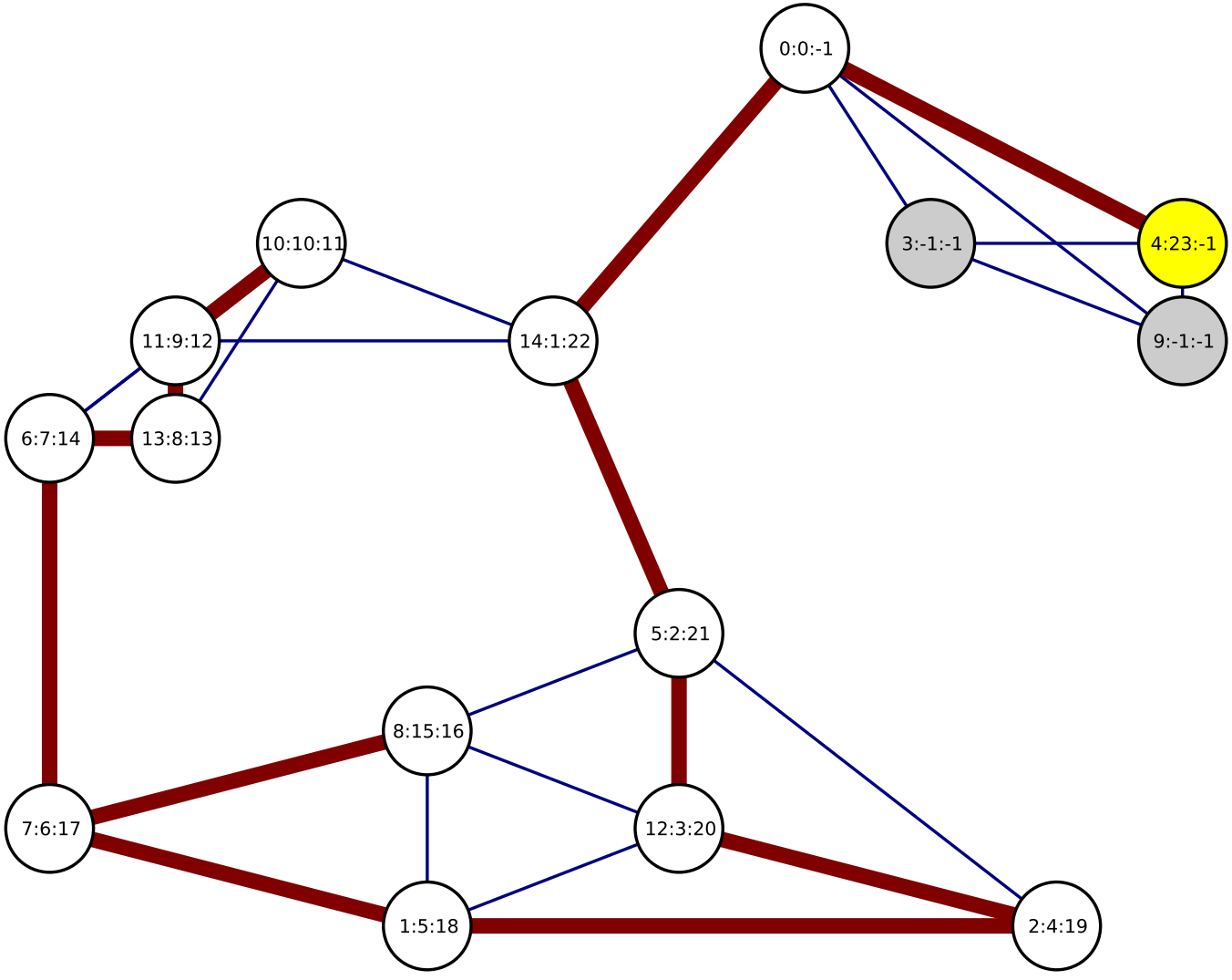
Example DFS


0:0:-1
10:10:-1
3:-1:-1
4:-1:-1
11:9:-1
14:1:-1
9:-1:-1
6:7:-1
13:8:-1
5:2:-1
8:-1:-1
7:6:-1
12:3:-1
1:5:-1
2:4:-1

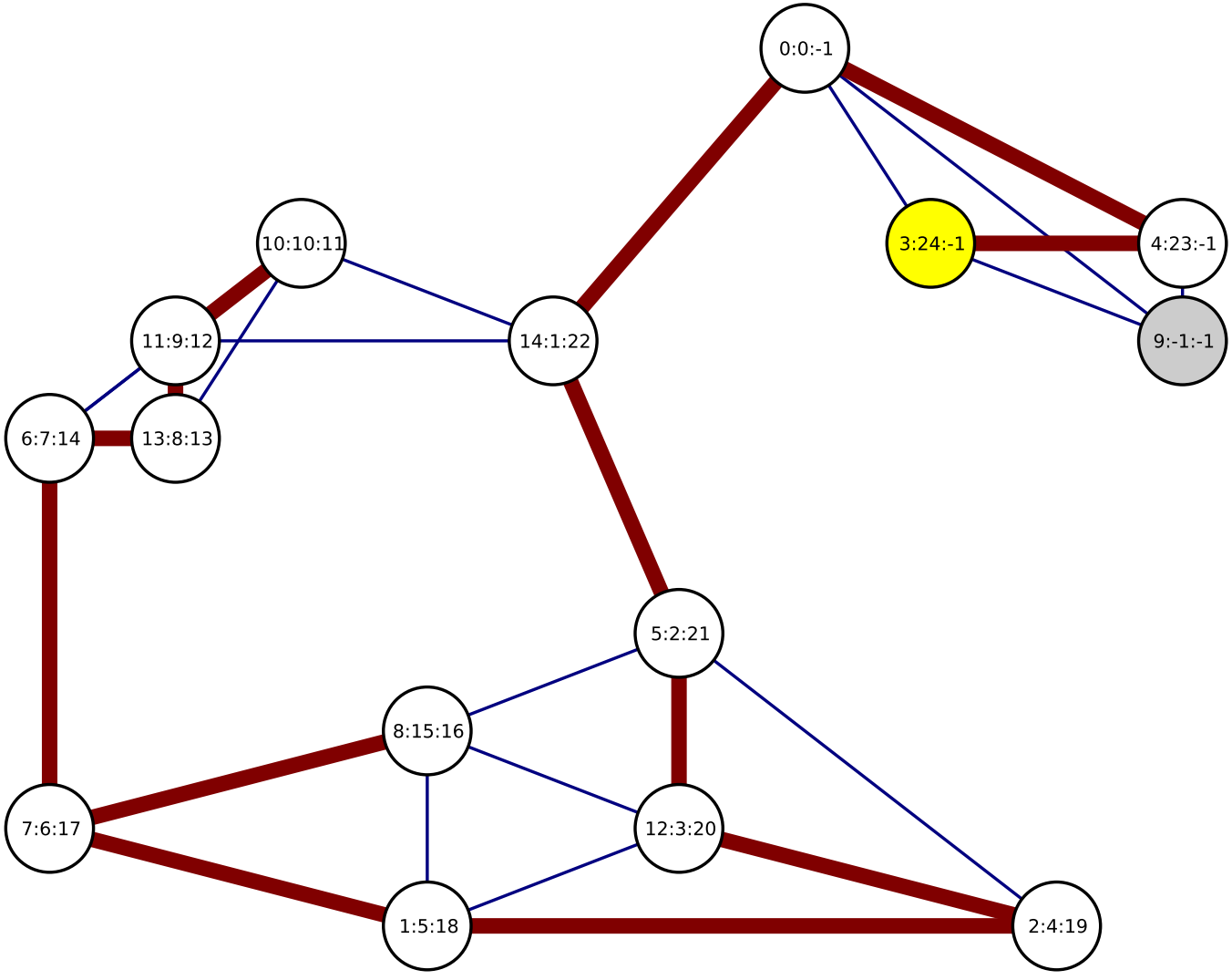
Example DFS


0:0:-1
3:-1:-1
4:-1:-1
9:-1:-1
10:10:11
11:9:12
14:1:-1
6:7:14
13:8:13
5:2:-1
8:15:-1
7:6:-1
12:3:-1
1:5:-1
2:4:-1

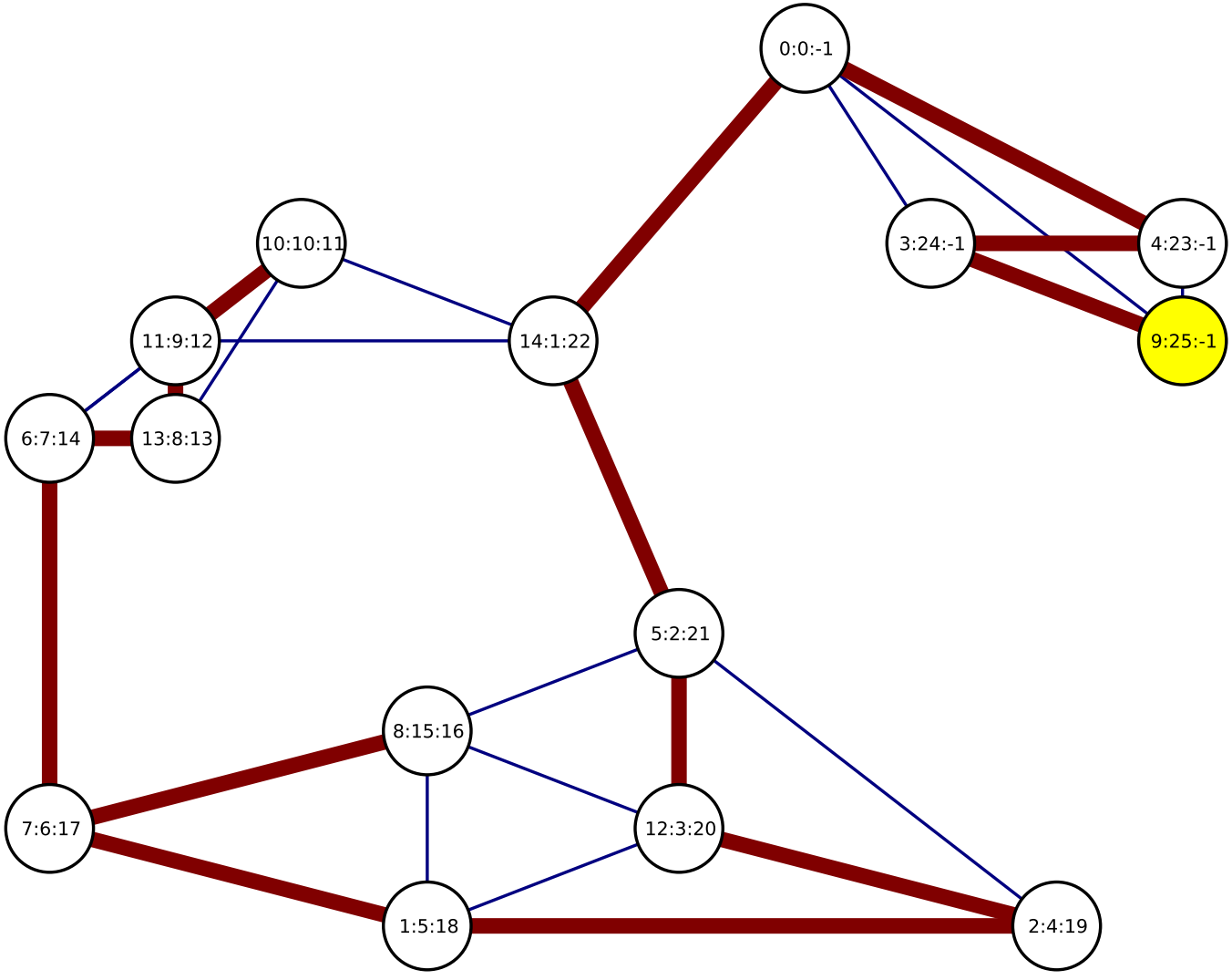
Example DFS


0:0:-1
4:23:-1
3:-1:-1
9:-1:-1
10:10:11
11:9:12
14:1:22
6:7:14
13:8:13
5:2:21
8:15:16
7:6:17
12:3:20
1:5:18
2:4:19

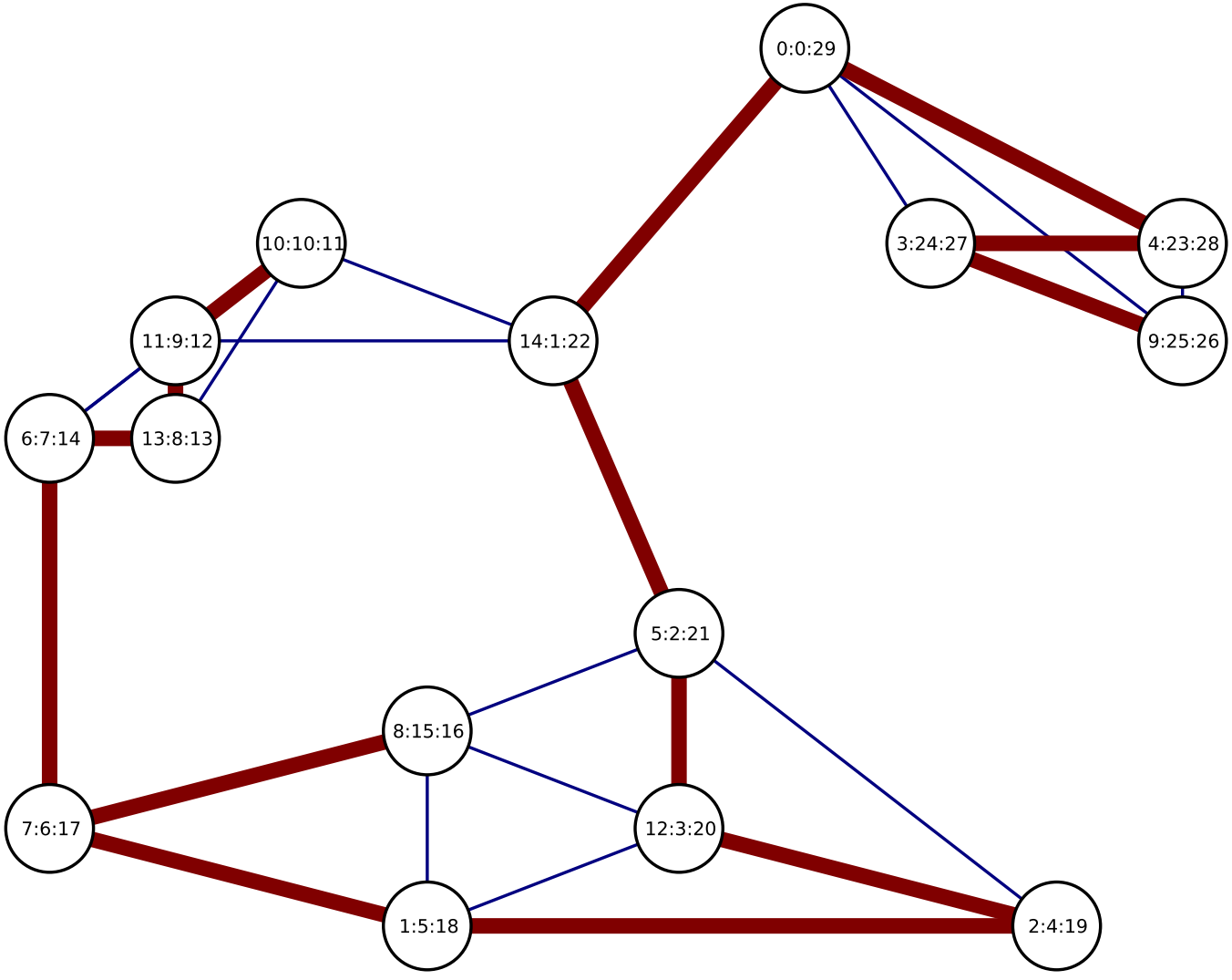
Example DFS


0:0:-1
3:24:-1
4:23:-1
10:10:11
11:9:12
14:1:22
9:-1:-1
6:7:14
13:8:13
5:2:21
8:15:16
7:6:17
12:3:20
1:5:18
2:4:19

Example DFS


0:0:-1
3:24:-1
4:23:-1
9:25:-1
10:10:11
11:9:12
14:1:22
6:7:14
13:8:13
5:2:21
8:15:16
7:6:17
12:3:20
1:5:18
2:4:19

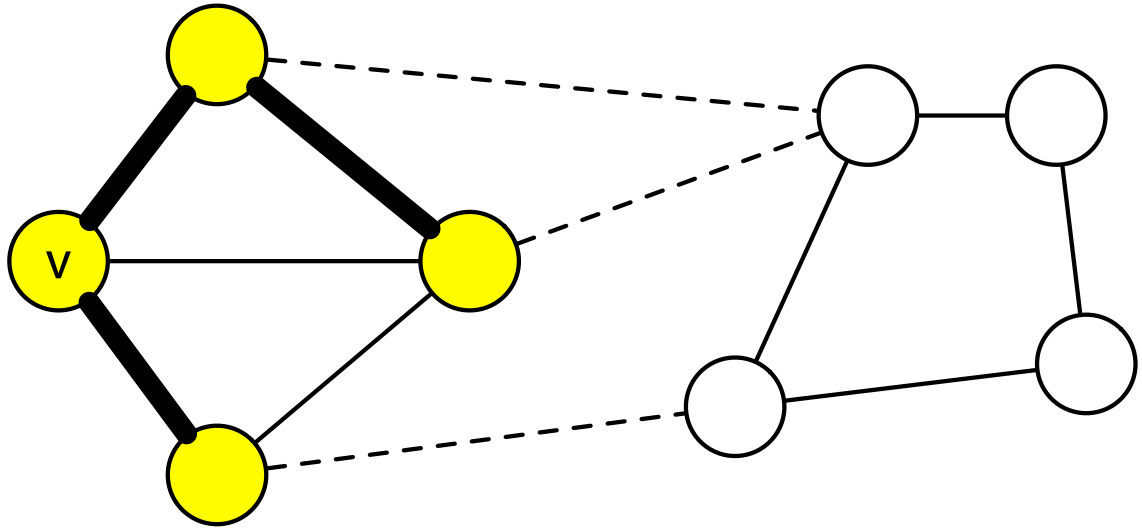
Example DFS


0:0:29
10:10:11
3:24:27
4:23:28
11:9:12
14:1:22
9:25:26
6:7:14
13:8:13
5:2:21
8:15:16
7:6:17
12:3:20
1:5:18
2:4:19

General Tree Growing (following Gross & Yellen)

We can think of BFS and DFS (and several other algorithms) as special cases of tree growing:

- Let T be the current tree T, and
- Maintain a list of frontier edges: the set of edges of G that have one endpoint in T and one endpoint not in T:



- Repeatedly choose a frontier edge (somehow) and add it to T.

Tree Growing

```
TreeGrowing(graph G, vertex v, func nextEdge):
    T = (v,∅)
    S = set of edges incident to v
    While S is not empty:
        e = nextEdge(G, S)
        T = T + e                    // add edge e to T
        S = updateFrontier(G, S, e)
    return T
```

- The function `nextEdge(G, S)` returns a frontier edge from S.
- `updateFrontier(G, S, e)` returns the new frontier after we add edge e to T.

Tree Growing

These algorithms are all special cases / variants of Tree Growing, with different versions of `nextEdge`:

1. Depth-first search
2. Breadth-first search
3. Prim's minimum spanning tree algorithm
4. Dijkstra's shortest path
5. A*

BFS & DFS as Tree Growing

What's `nextEdge` for DFS?

What's `nextEdge` for BFS?

BFS & DFS as Tree Growing

What's `nextEdge` for DFS?

> Select a frontier edge whose tree endpoint was discovered most recently.

Why? We can use a stack to implement DFS.
Runtime: $O(|\text{Edges}|)$

What's `nextEdge` for BFS?

BFS & DFS as Tree Growing

What's `nextEdge` for DFS?

> Select a frontier edge whose tree endpoint was discovered most recently.

Why? We can use a stack to implement DFS.
Runtime: $O(|\text{Edges}|)$

What's `nextEdge` for BFS?

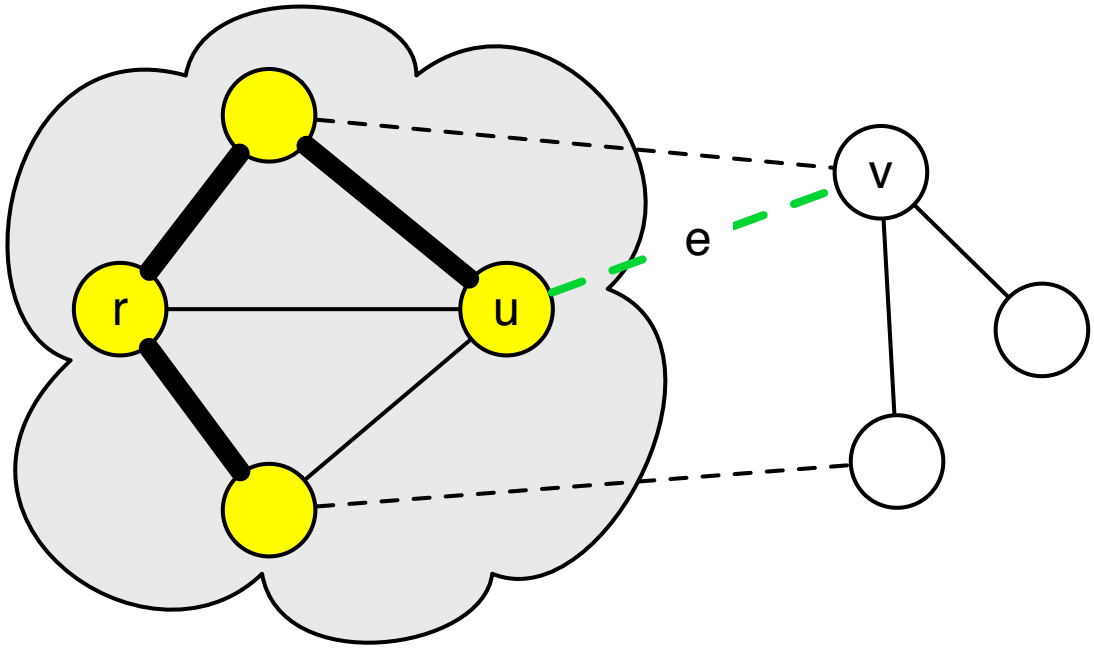
> Select a frontier edge whose tree endpoint was discovered earliest.

Why? We can use a queue to implement BFS.
Runtime: $O(|\text{Edges}|)$

Prim's Algorithm

Prim's Algorithm: Run `TreeGrowing` starting with any root node, adding the frontier edge with the smallest weight.

Theorem. *Prim's algorithm produces a minimum spanning tree.*



S = set of nodes already in the tree when e is added

Implementations of BFS and DFS

BFS implementation

```
procedure bfs(G, s):
    Q := queue containing only s
    while Q not empty
        v := Q.front(); Q.remove_front()
        for w ∈ G.neighbors(v):
            if w not seen:
                mark w seen
                Q.enqueue(w)
```

Recursive implementation of DFS

```
procedure dfs(G, u):
    while u has an unvisited neighbor in G
        v := an unvisited neighbor of u
        mark v visited
        dfs(G, v)
```

Stack-based implementation of DFS

```
procedure dfs(G, s):
    S := stack containing only s
    while S not empty
        v := S.pop()
        if v not visited:
            mark v visited
            for w ∈ G.neighbors(v): S.push(w)
```

Properties of BFS and DFS

Property of Non-BFS-Tree Edges

Theorem. *Choose $x \in L_i$ and $y \in L_j$ such that $\{x, y\}$ is an edge in undirected graph G. Then i and j differ by at most 1.*

In other words, edges of G that do not appear in the tree connect nodes either in the same layer or adjacent layer.

Proof. Suppose not, and that $i < j - 1$.

All the neighbors of x will be found by layer $i + 1$.

Therefore, the layer of y is less than $i + 1$, so $j \leq i + 1$, which contradicts $i < j - 1$. □

A property of Non-DFS-Tree Edges

Theorem. *Let x and y be nodes in the DFS tree T_G such that $\{x, y\}$ is an edge in undirected graph G. Then one of x or y is an ancestor of the other in T_G.*

Proof. Suppose, wlog, x is reached first in the DFS.

All the nodes that are marked explored between first encountering x and leaving x for the last time are descendants of x in T_G.

When we reach x, node y must not yet have been explored.

It must become explored before leaving x for the last time (otherwise, we should add $\{x, y\}$ to T_G). Hence, y is a descendent of x in T_G. □