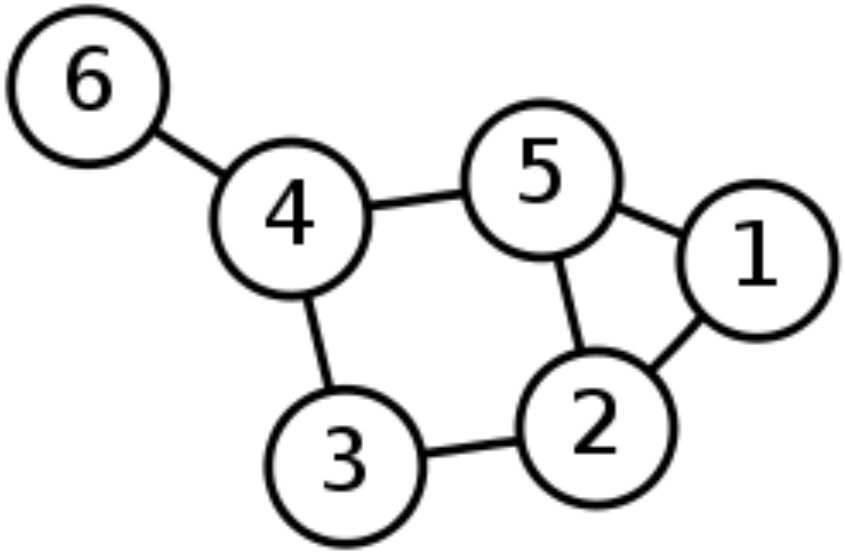
Todo

- Bridges and node labels
- Make a graph to throw at students
- If they can find a path of at least 4

*Just like a tree without the rules

Graphs II

CS 106B

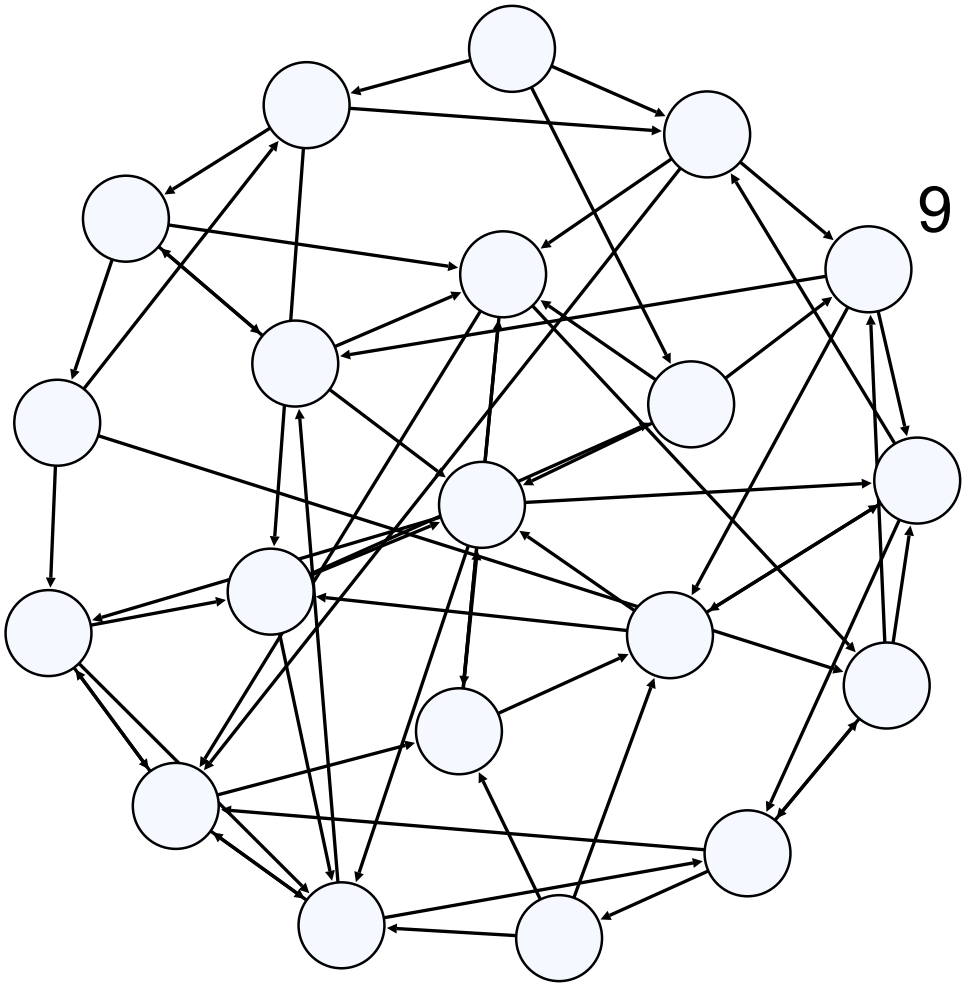


Programming Abstractions
Fall 2016
Stanford University
Computer Science Department

Play a Game

Directed Graph
20 Nodes

Each node has a unique id



Today's Goals

1. More practice with graphs

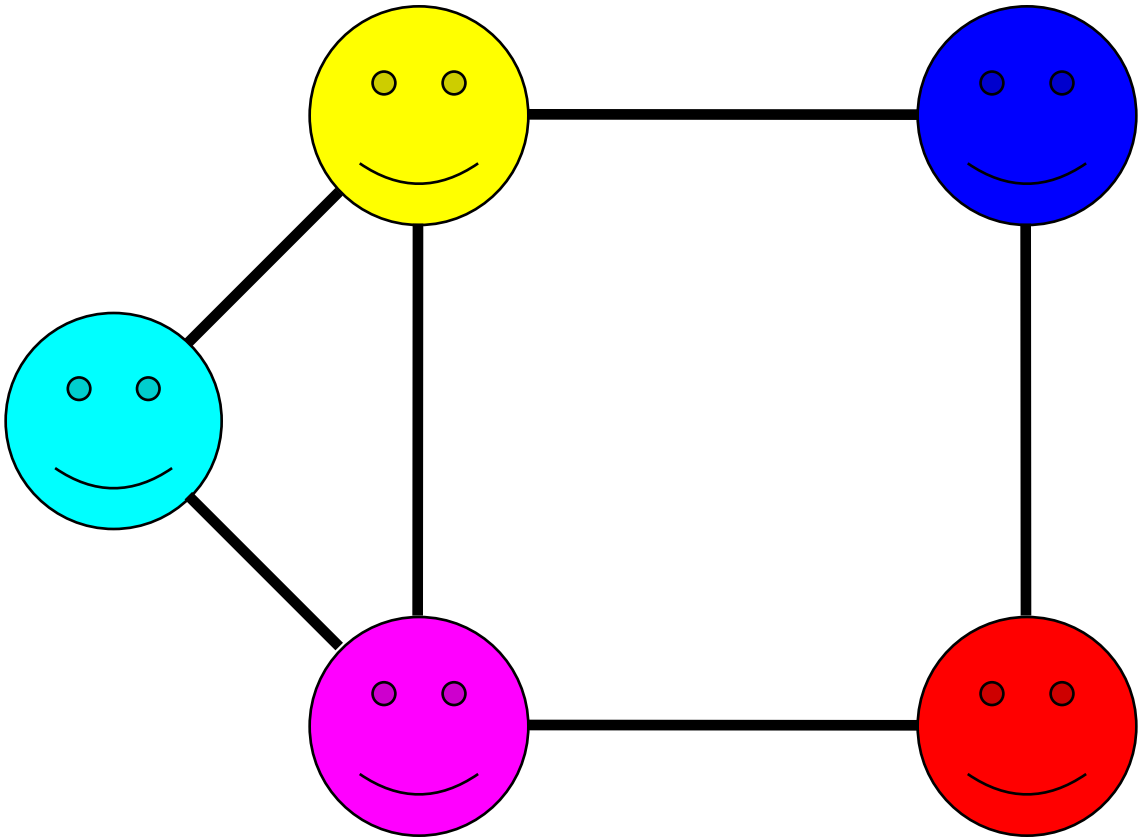
Some terms:

Graph Definition

A **graph** is a mathematical structure
for representing relationships using nodes and edges.

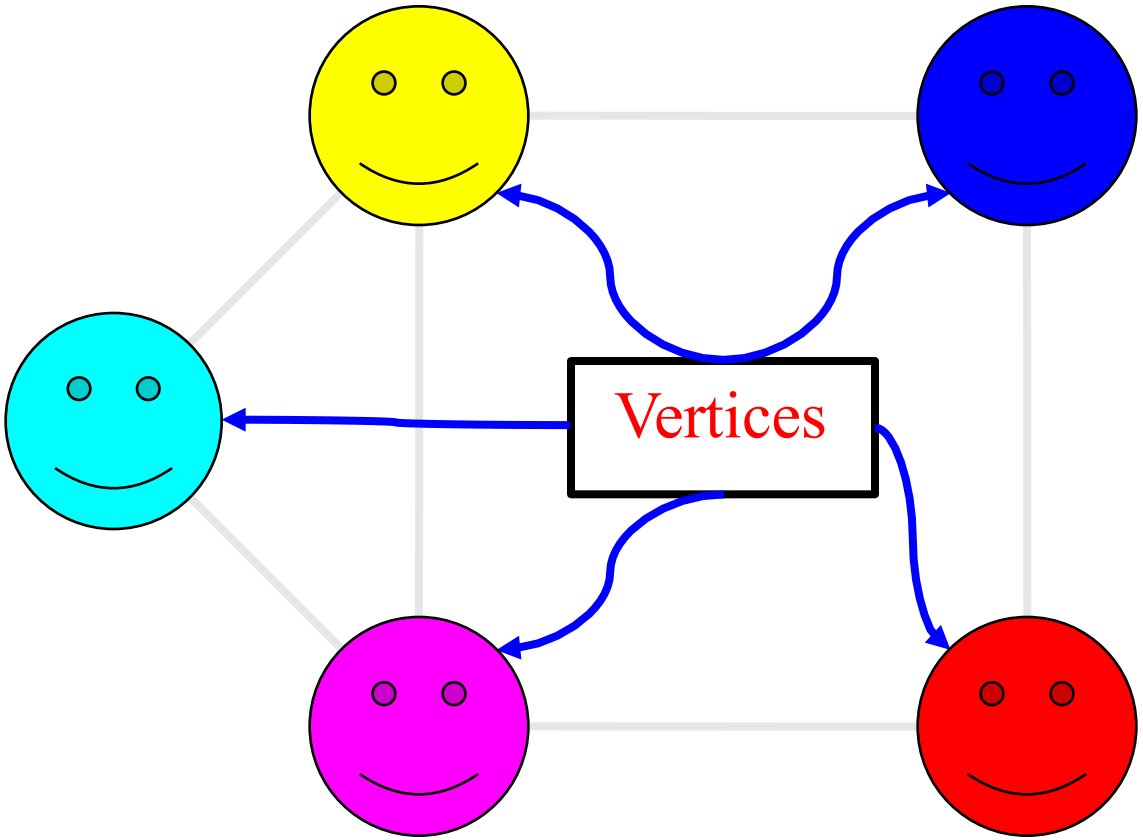
*Just like a tree without the rules

Graph



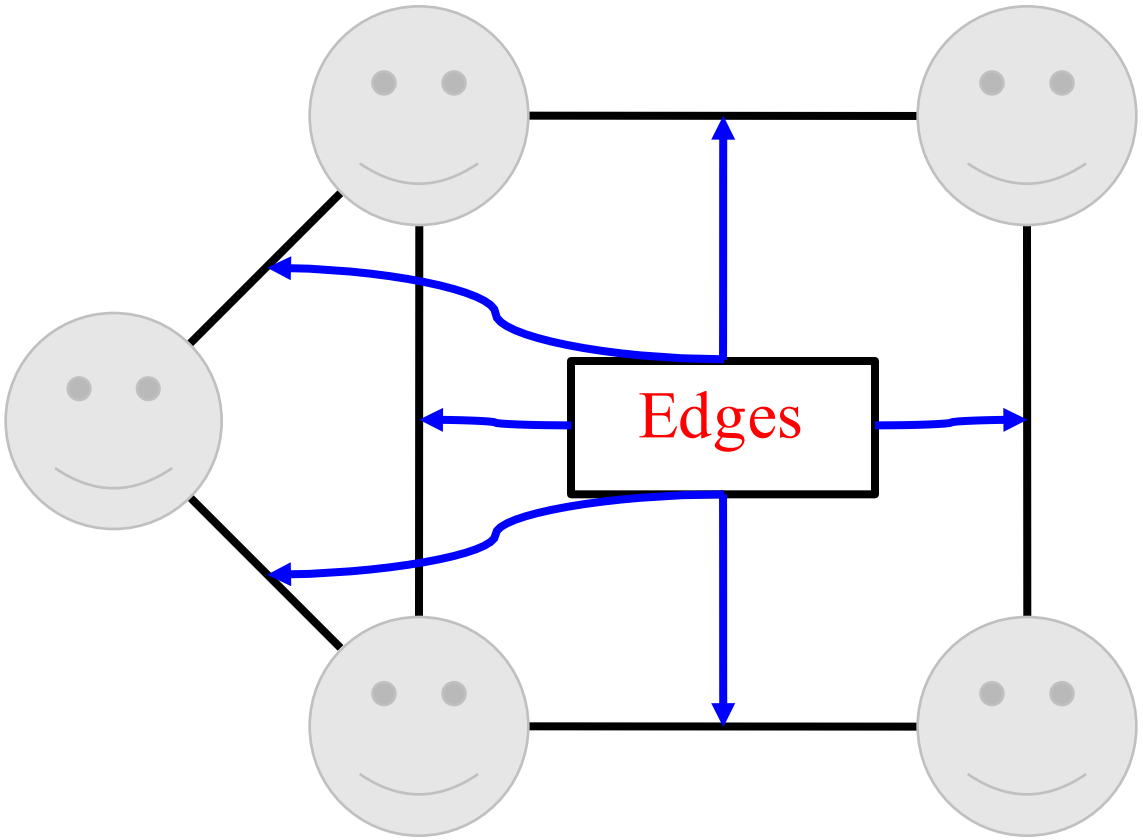
A graph consists of a set of **nodes** connected by **edges**.

Nodes Vertices



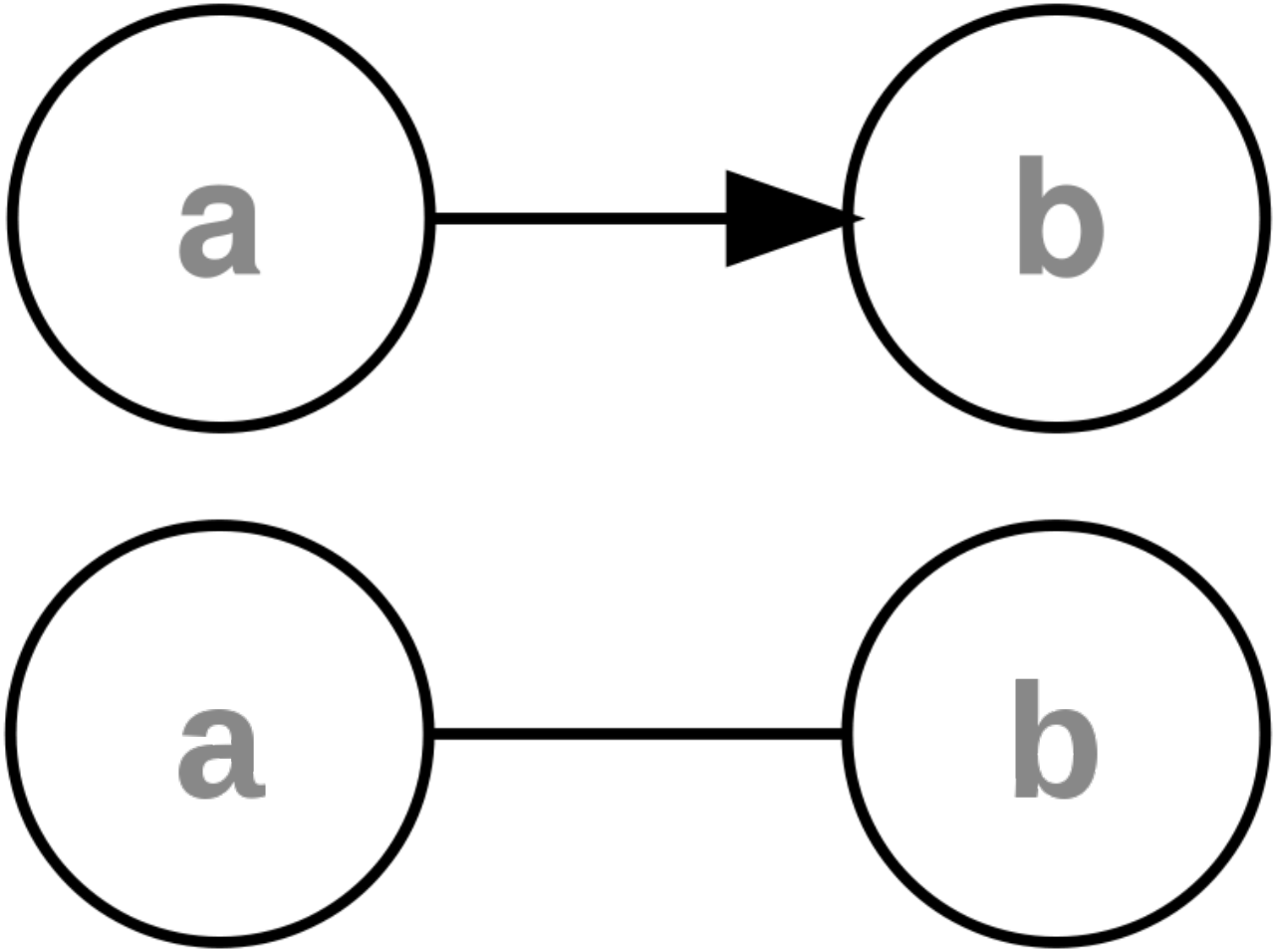
A graph consists of a set of **nodes** connected by **edges**.

Graph Edges



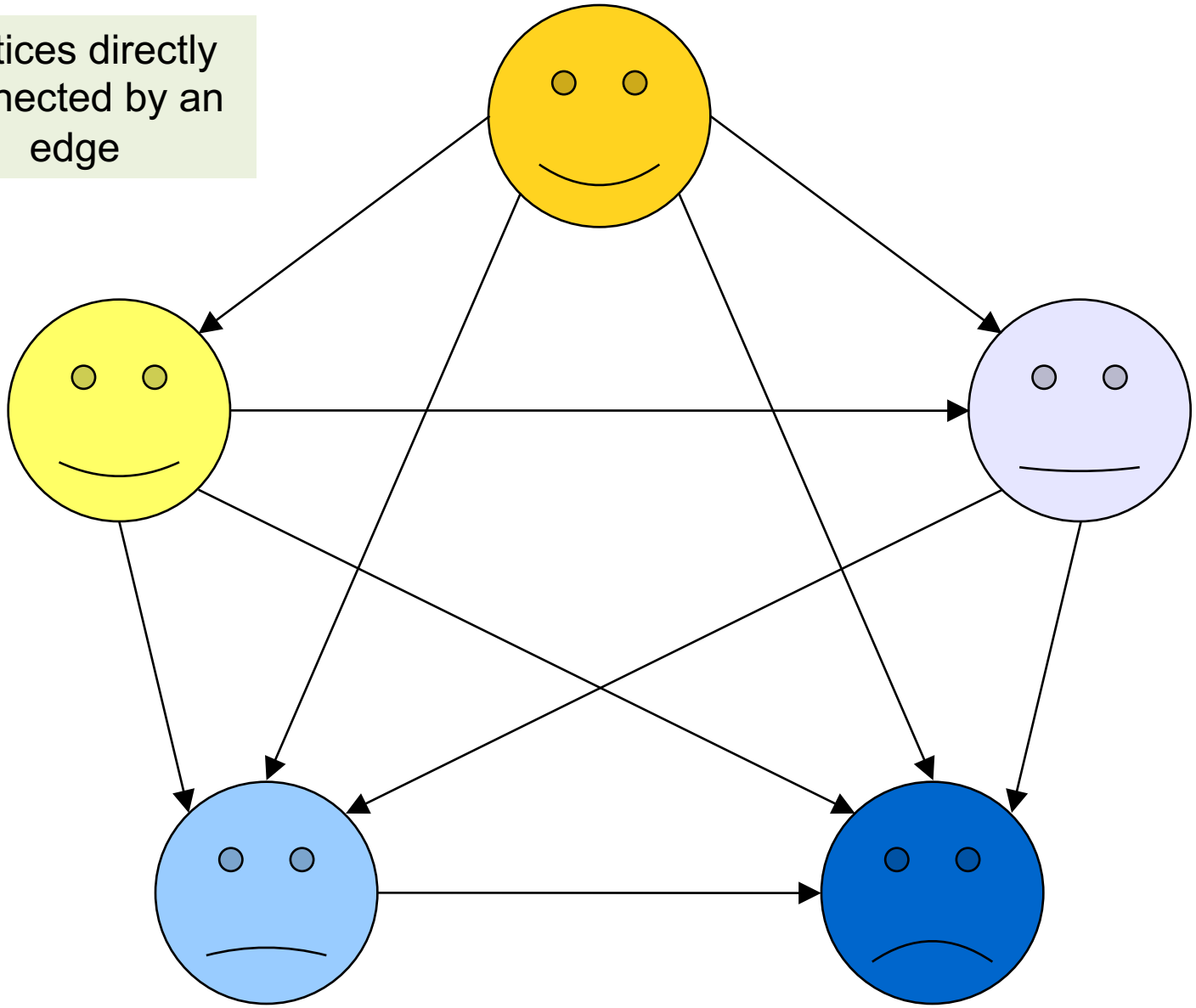
A graph consists of a set of **nodes** connected by **edges**.

Directed vs Undirected



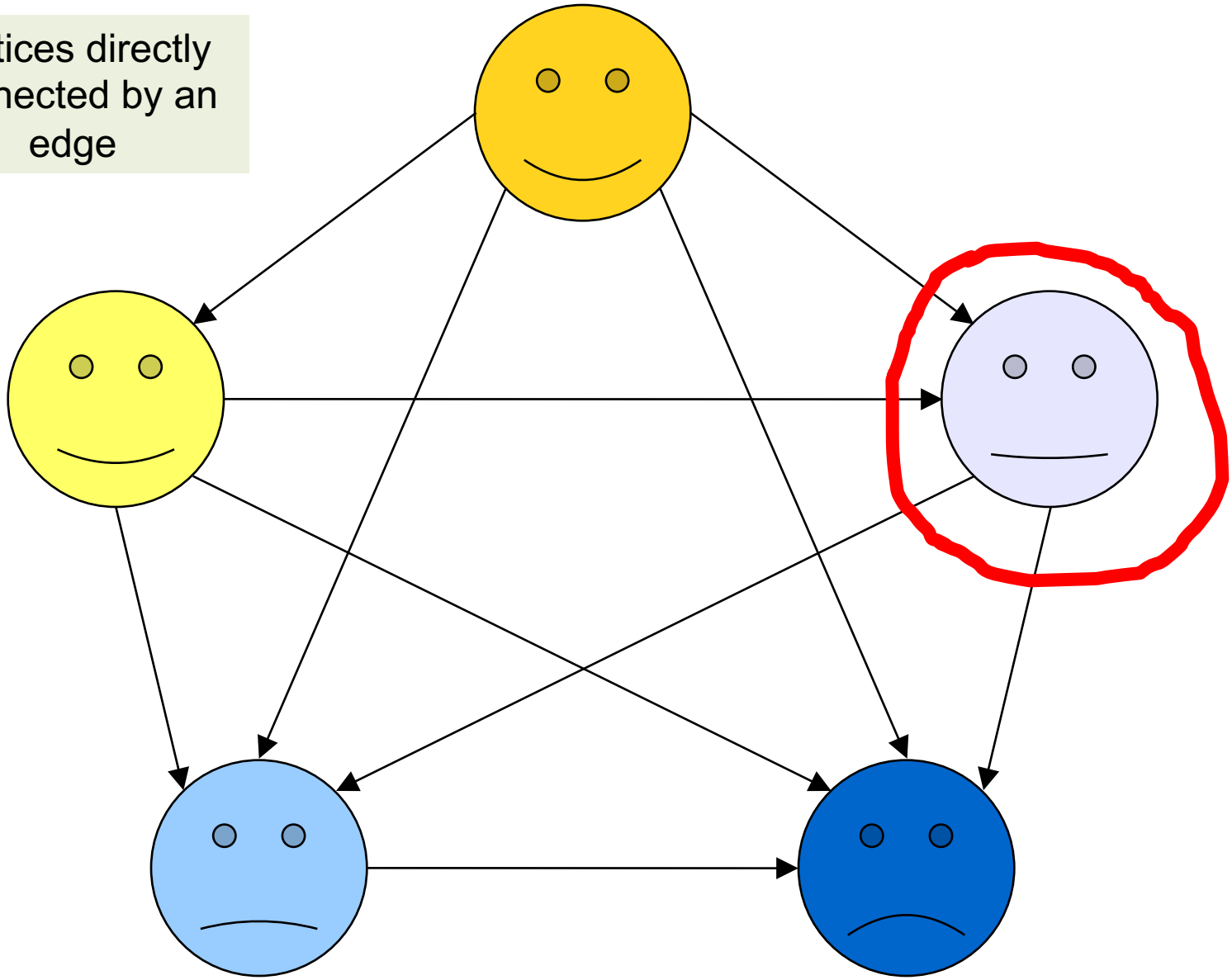
Neighbors

Vertices directly connected by an edge



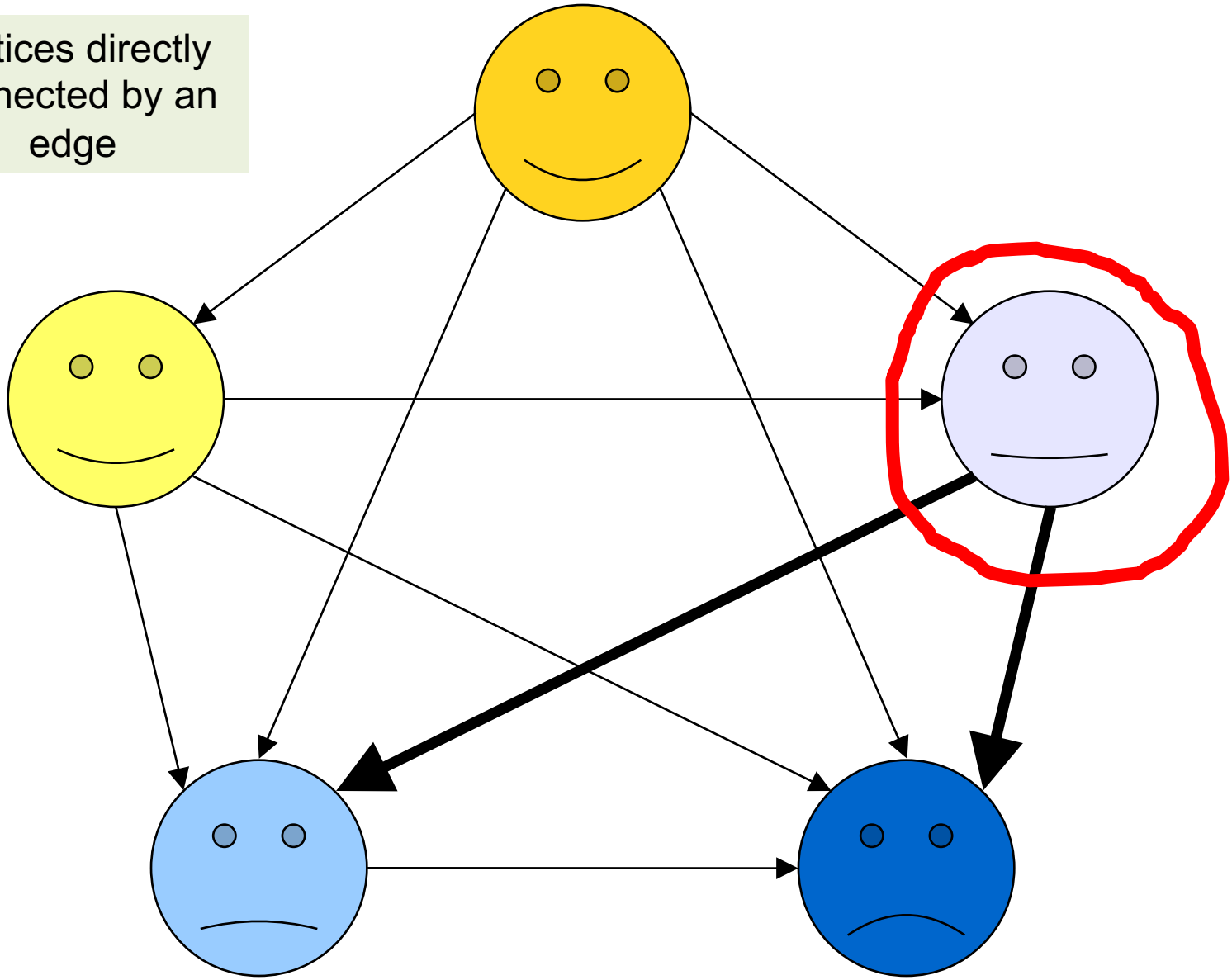
Neighbors

Vertices directly connected by an edge



Neighbors

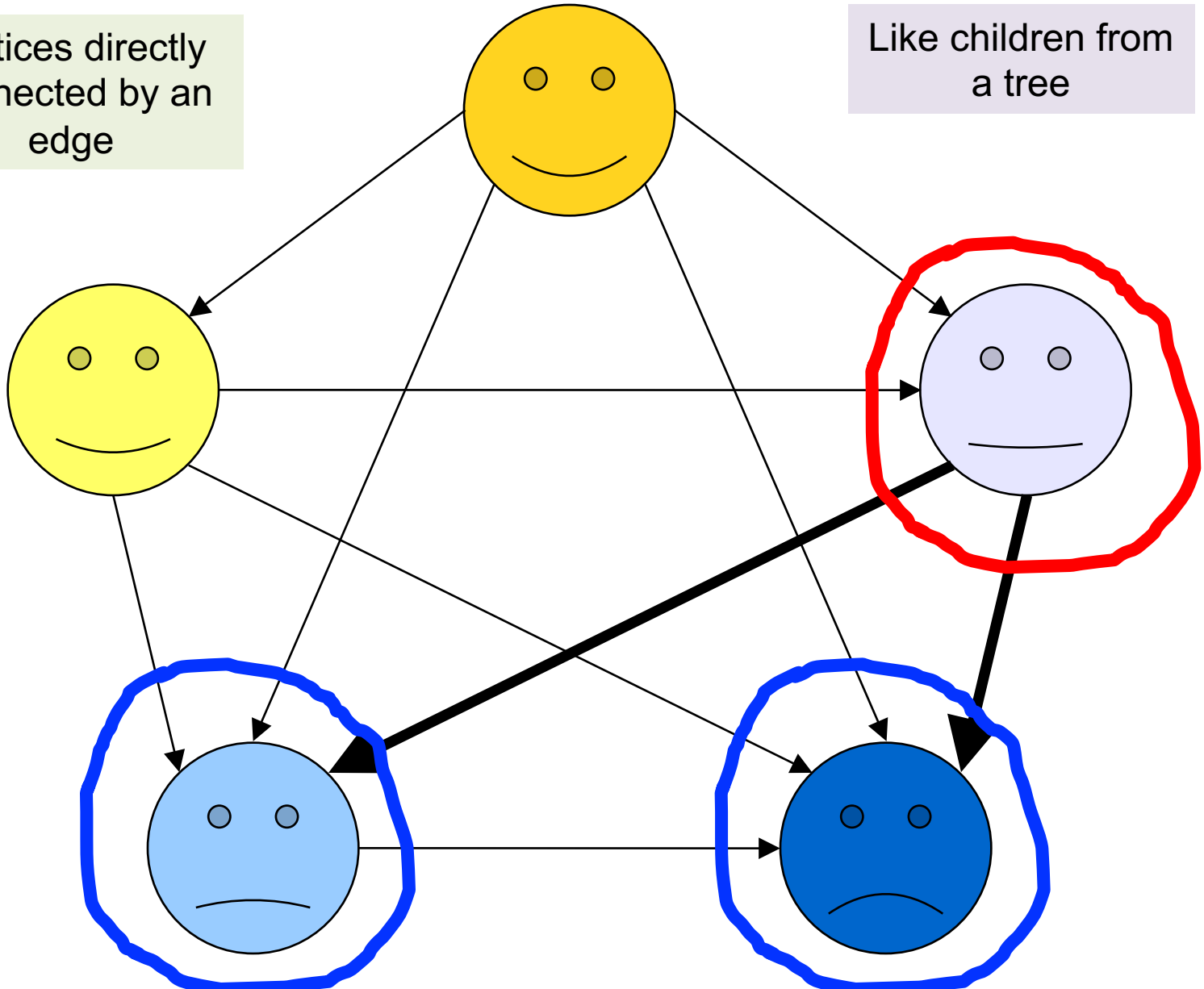
Vertices directly connected by an edge



Neighbors

Vertices directly connected by an edge

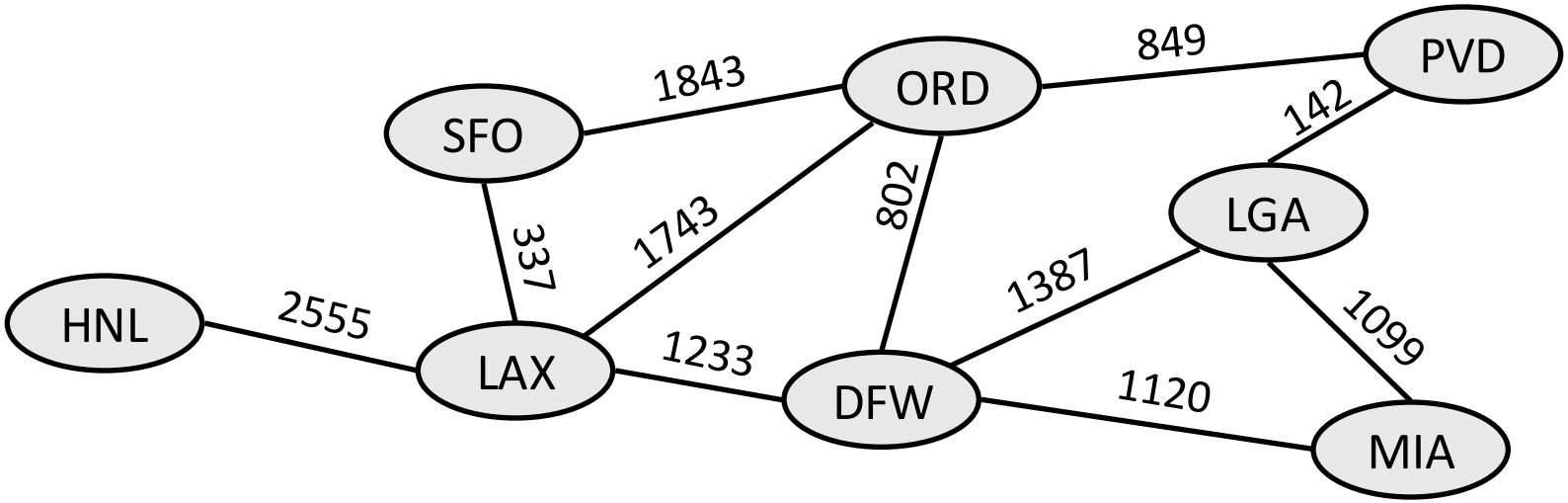
Like children from a tree



Edge Weights

weight: Cost associated with a given edge.

example: graph of airline flights, weighted by miles between cities:



Graph Representations

Most Straightforward

```
struct Vertex{
   string value;
   Vector<Vertex *> edges;
};

struct Graph {
   Set<Vertex *> nodes;
}
```

Weighted Graph

```
struct Vertex{
   string value;
   Vector<Edge *> edges;
};

struct Edge{
   Vertex * start;
   Vertex * end;
};

struct Graph{
   Set<Vertex *> nodes;
   Set<Edge*> edges;
};
```

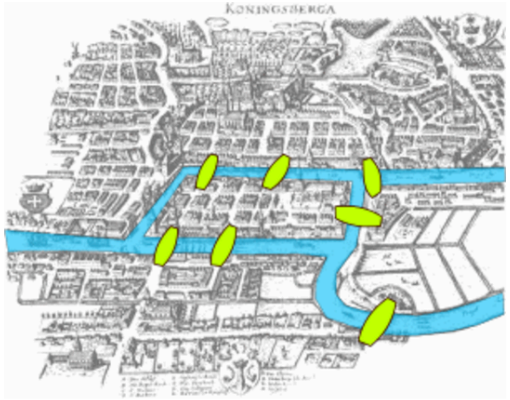
BasicGraph members

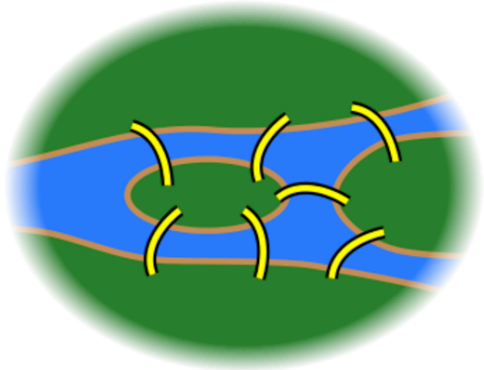
```
#include "basicgraph.h"    // a directed, weighted graph
```

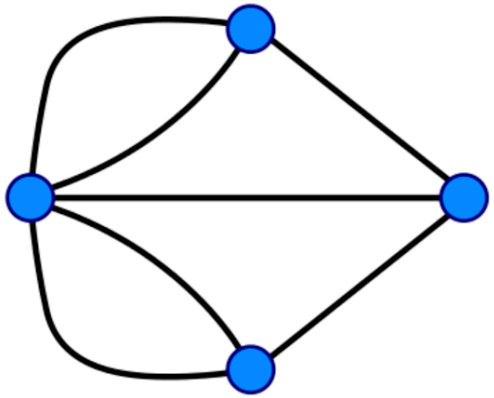
`g.addEdge(v1, v2);`	adds an edge between two vertexes
`g.addVertex(name);`	adds a vertex to the graph
`g.clear();`	removes all vertexes/edges from the graph
`g.getEdgeSet()` `g.getEdgeSet(v)`	returns all edges, or all edges that start at ***v***, as a `Set` of pointers
`g.getNeighbors(v)`	returns a set of all vertices that ***v*** has an edge to
`g.getVertex(name)`	returns pointer to vertex with the given name
`g.getVertexSet()`	returns a set of all vertexes
`g.isNeighbor(v1, v2)`	returns `true` if there is an edge from vertex ***v1*** to ***v2***
`g.isEmpty()`	returns `true` if queue contains no vertexes/edges
`g.removeEdge(v1, v2);`	removes an edge from the graph
`g.removeVertex(name);`	removes a vertex from the graph
`g.size()`	returns the number of vertexes in the graph
`g.toString()`	returns a string such as `"{a, b, c, a -> b}"`

Algorithms


© Historic Cities Research
Die Fürstliche Haupt Statt Königsberg in Preussen
S. Niclas
Schlos thurn
Der Cragheim
Der Schlos Teich
Edelleut stat
S. Barbara


KONINGSBERGA



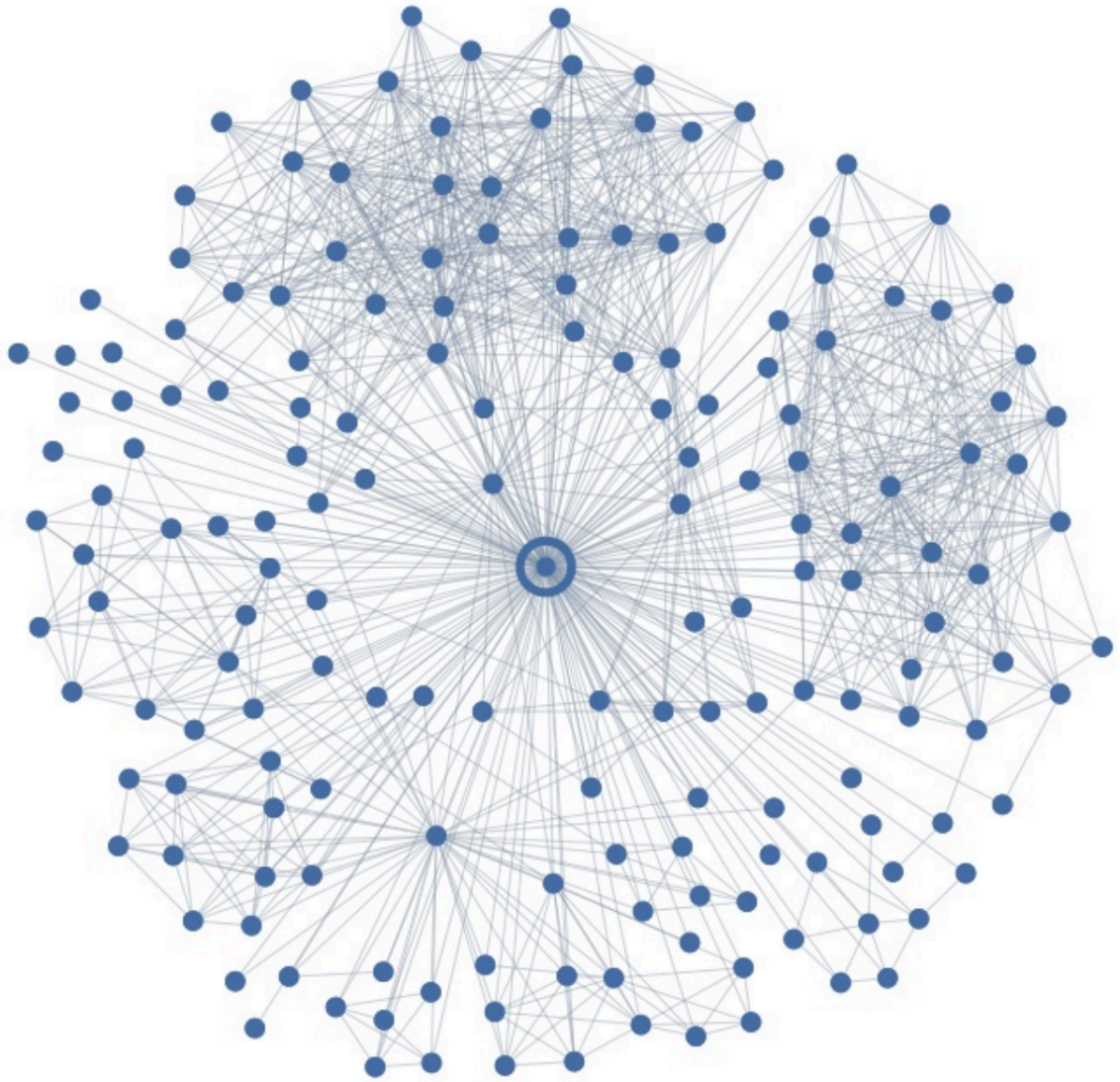



Der Kneiphofer holtzwiese

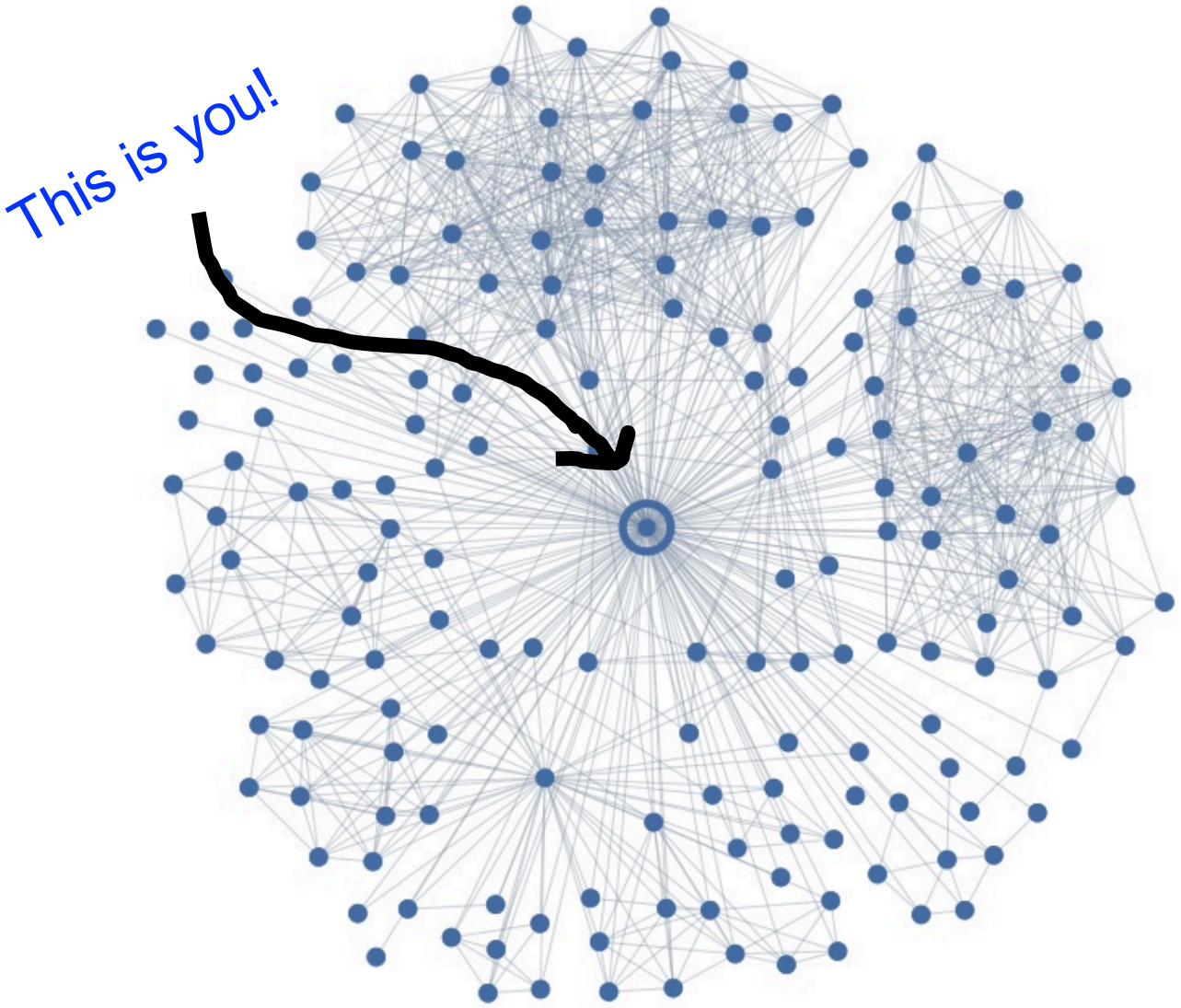
Who Do You Love

And how does Facebook know?

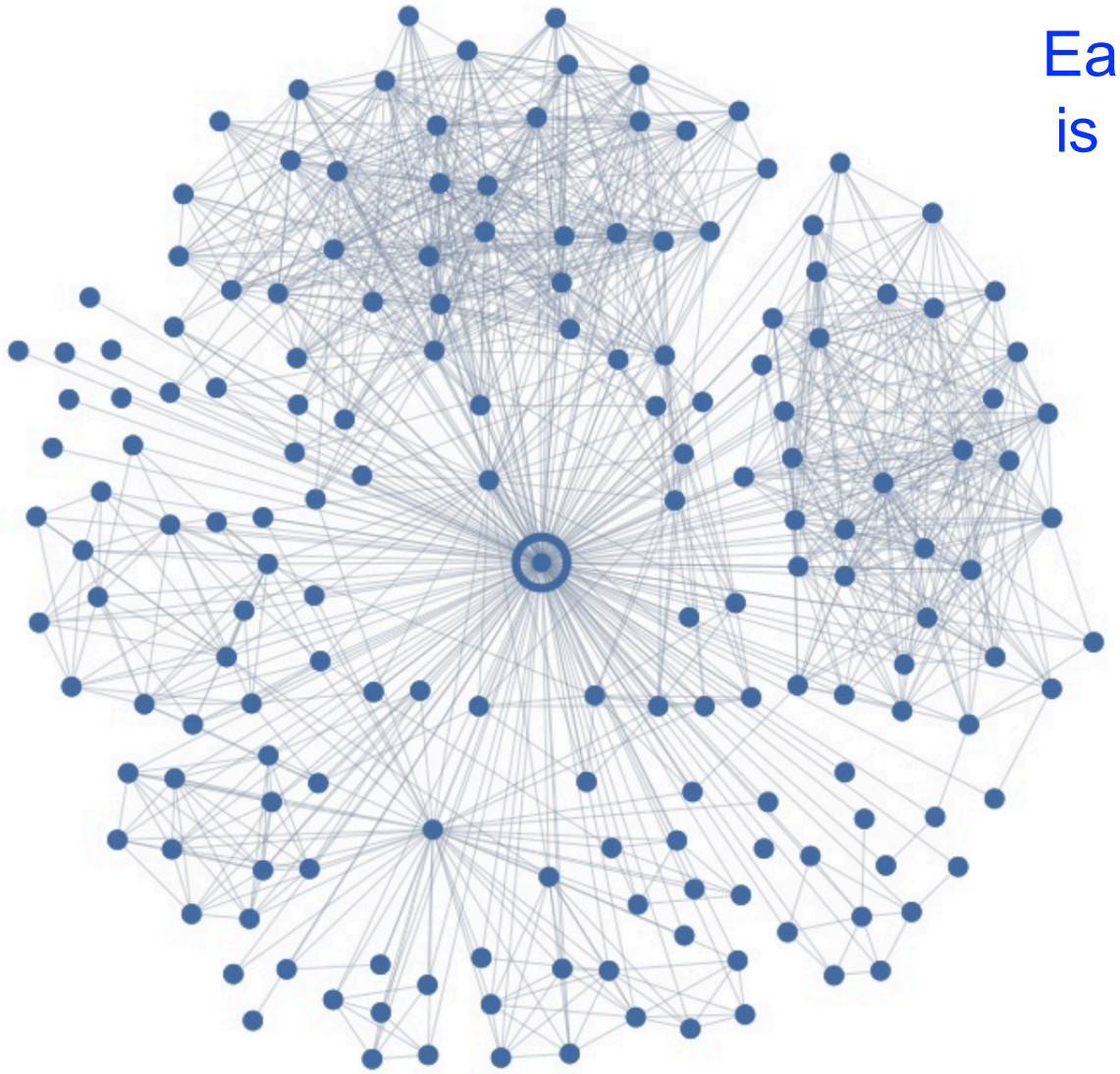
Ego Graph



Ego Graph

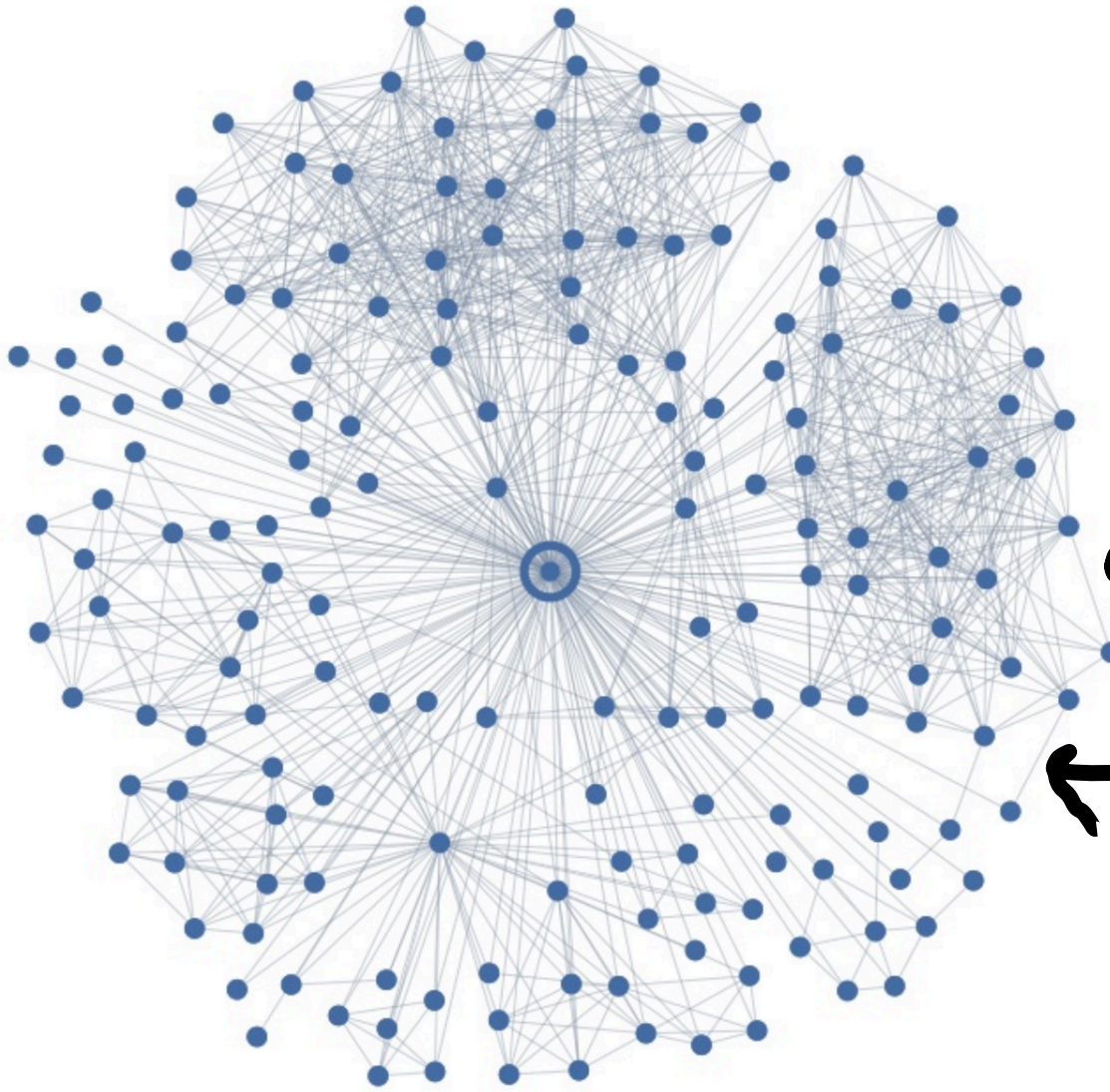


Ego Graph



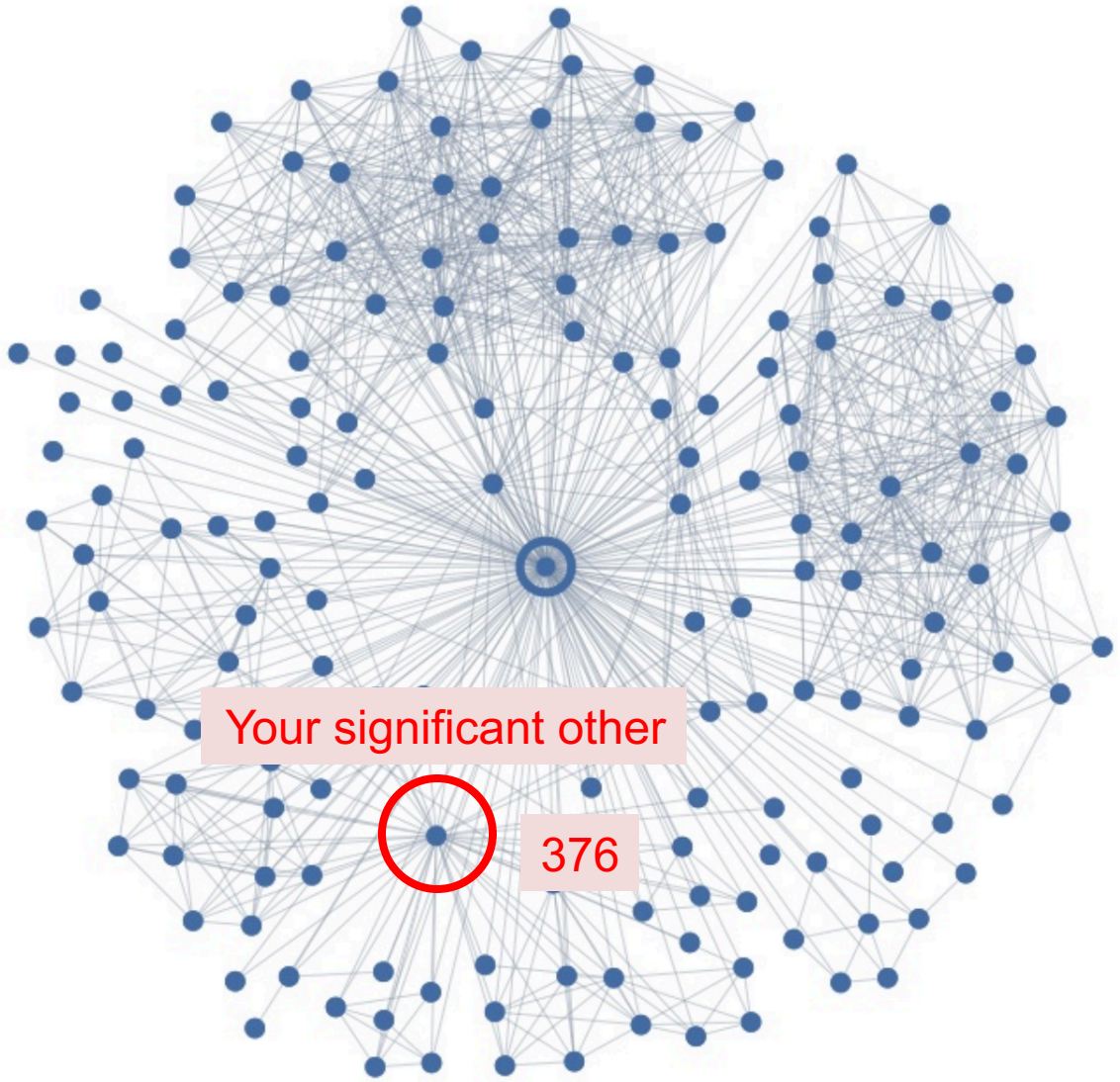
Each node is a friend

Ego Graph

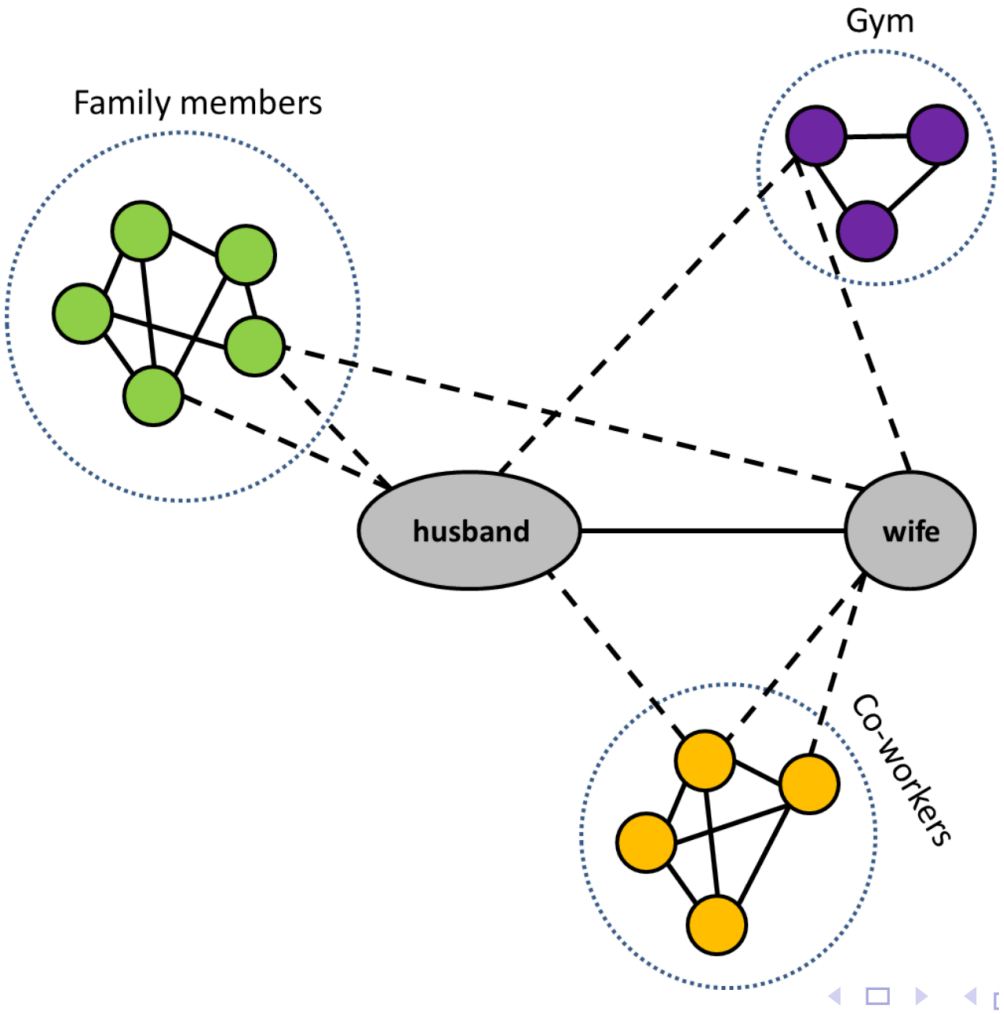


Some friends know each other

I Love This Person



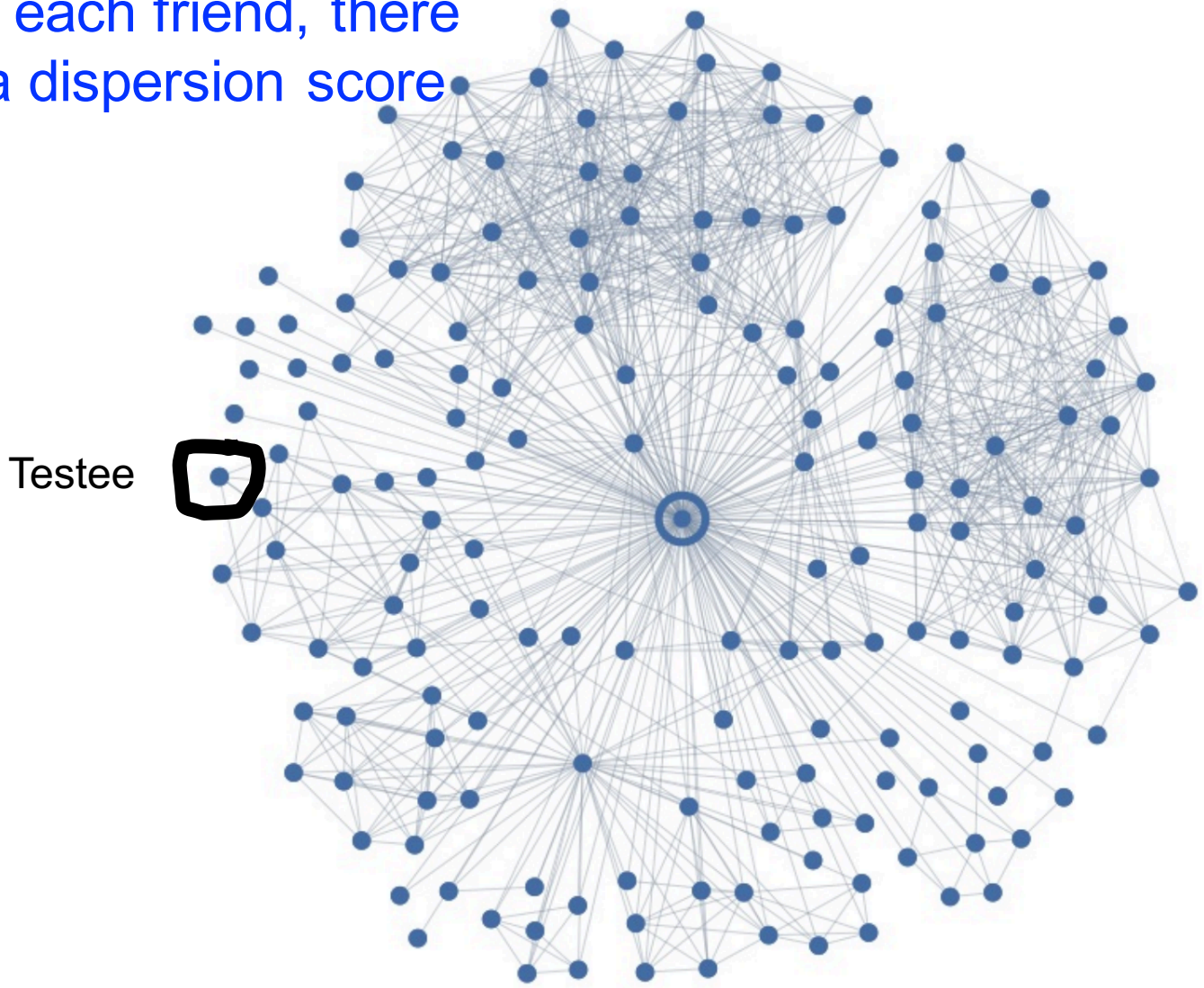
Dispersion Insight



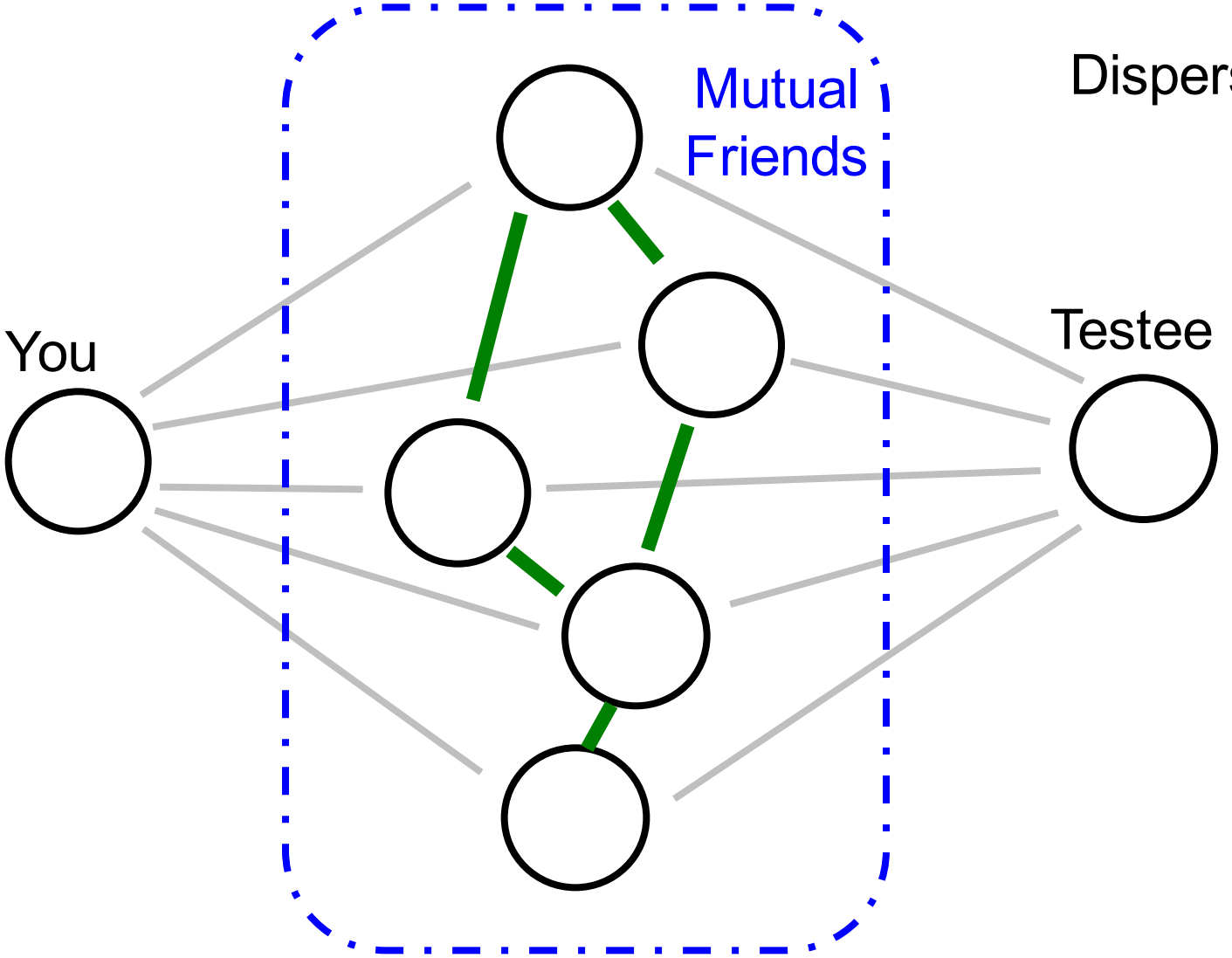
Dispersion: The extent to which two people's mutual friends are not directly connected

Dispersion Score

For each friend, there is a dispersion score

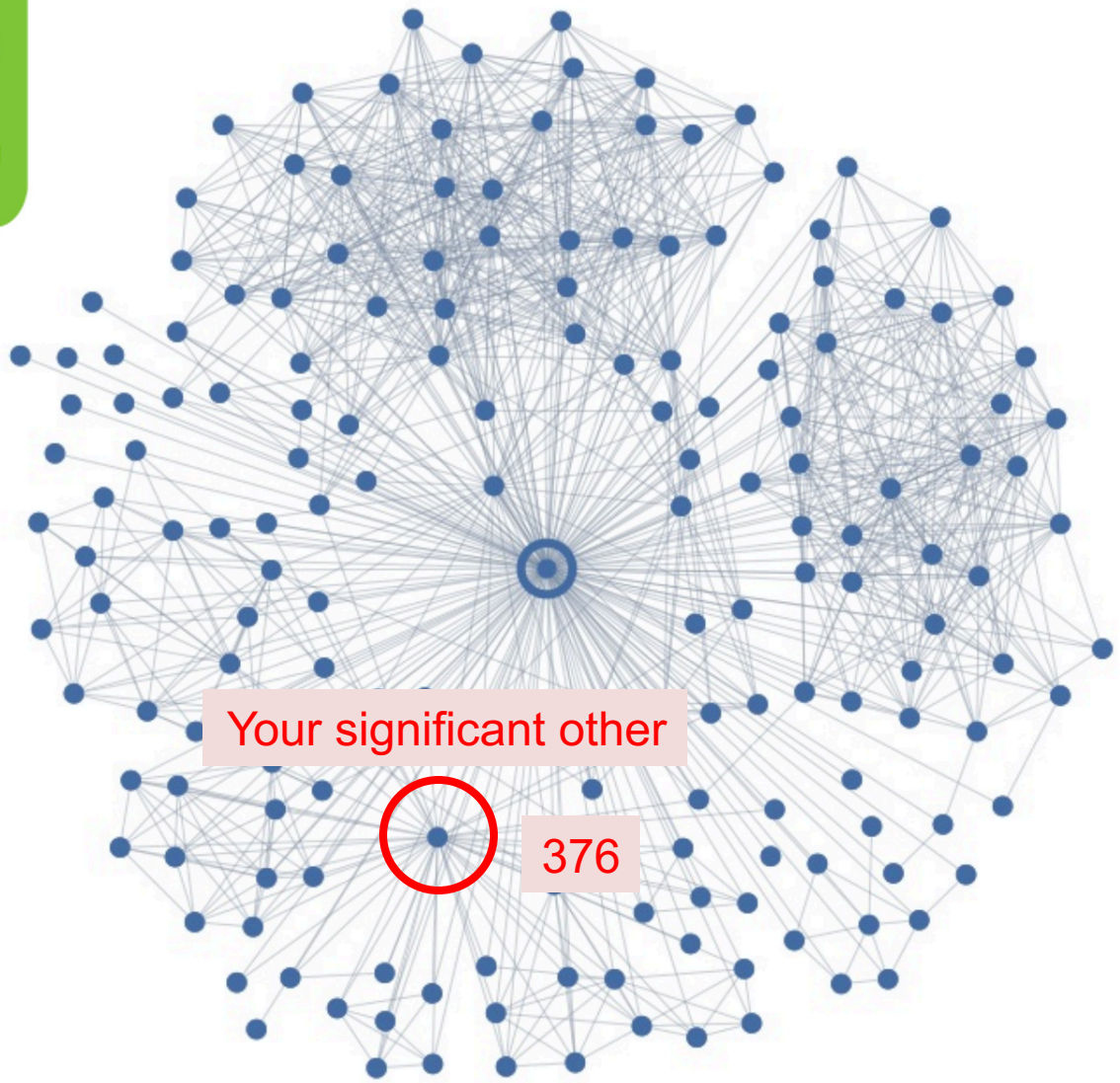


Dispersion



Dispersion: The extent to which two people's mutual friends are not directly connected

Who Do You Love?

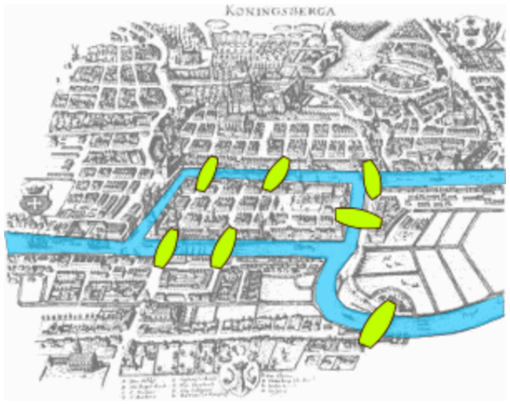


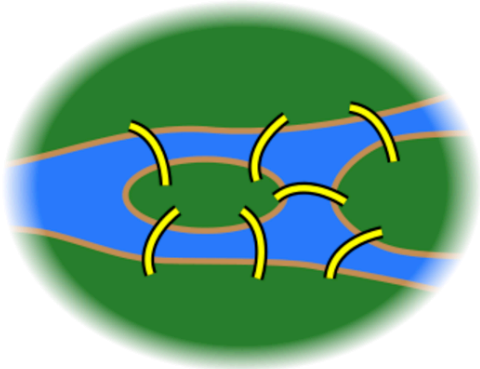
More problems

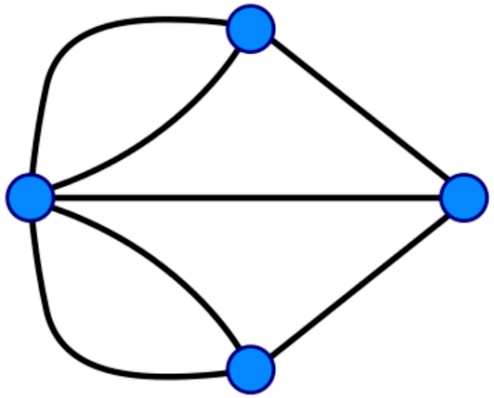
Spoiler alert:
These math problems become awesome.

Seven Bridges of Königsberg

1736





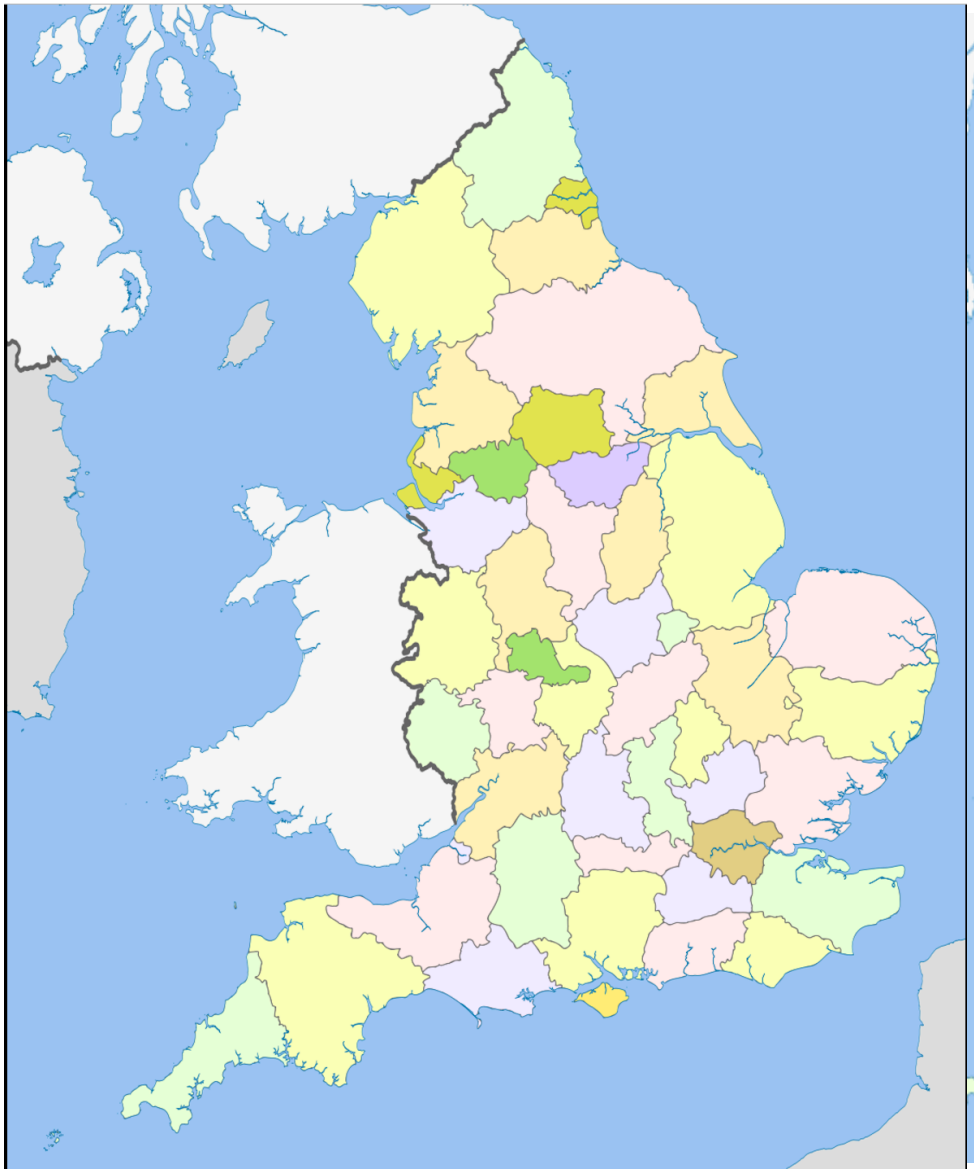


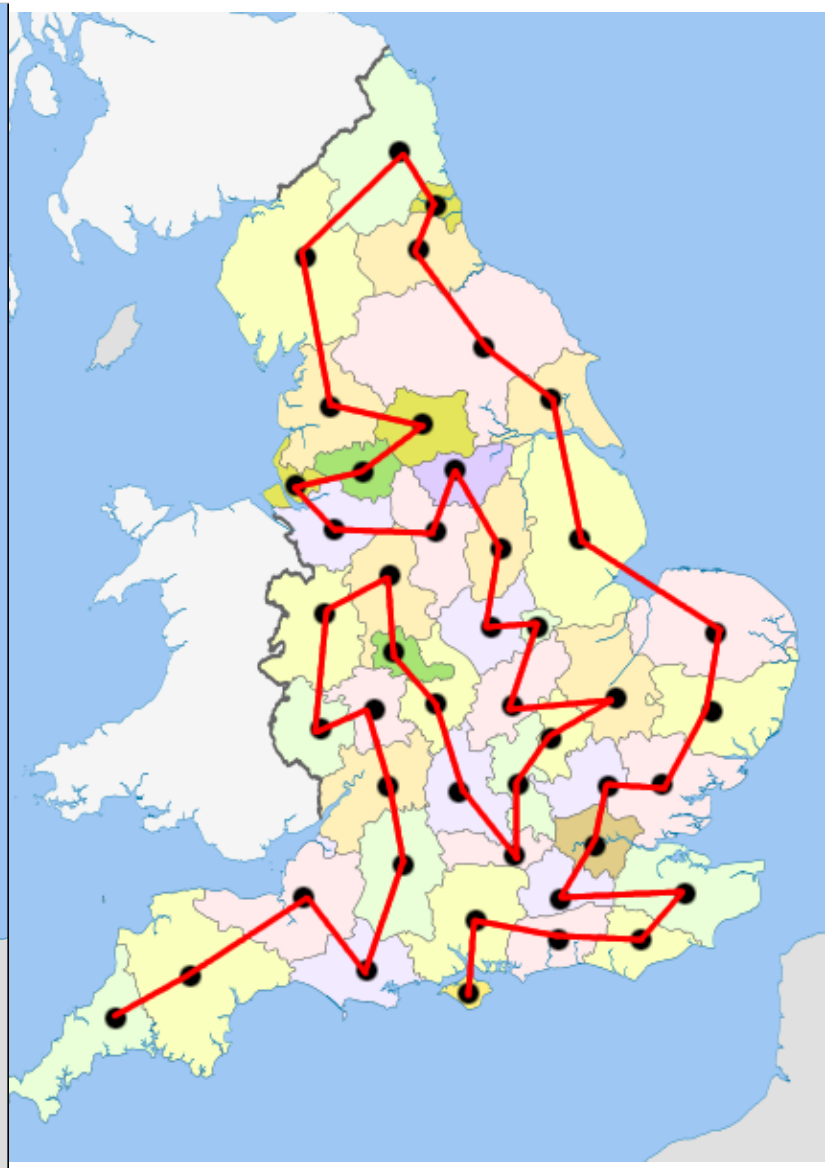
Seven Bridges of Königsberg

```
/* Function: Euler
 * Usage: euler(egoGraph)
 * ----------------------
 * Tests if there is an Euler path in a given graph. Euler
 * claimed without proof that if a graph has either 0 or 2
 * nodes with an odd number of neighbors, it has a path that
 * touches all edges (now called an Euler path).
 */
bool euler(BasicGraph & g) {
    int numOdd = 0;
    for(Vertex * v : g.getVertices()) {
        int numNeighbors = g.getNeighbors(v);
        if(numNeighbors % 2 == 1) numOdd++;
    }
    return numOdd == 0 || numOdd == 2;
}
```


HAMILTON
AN AMERICAN MUSICAL

Hamiltonian Path



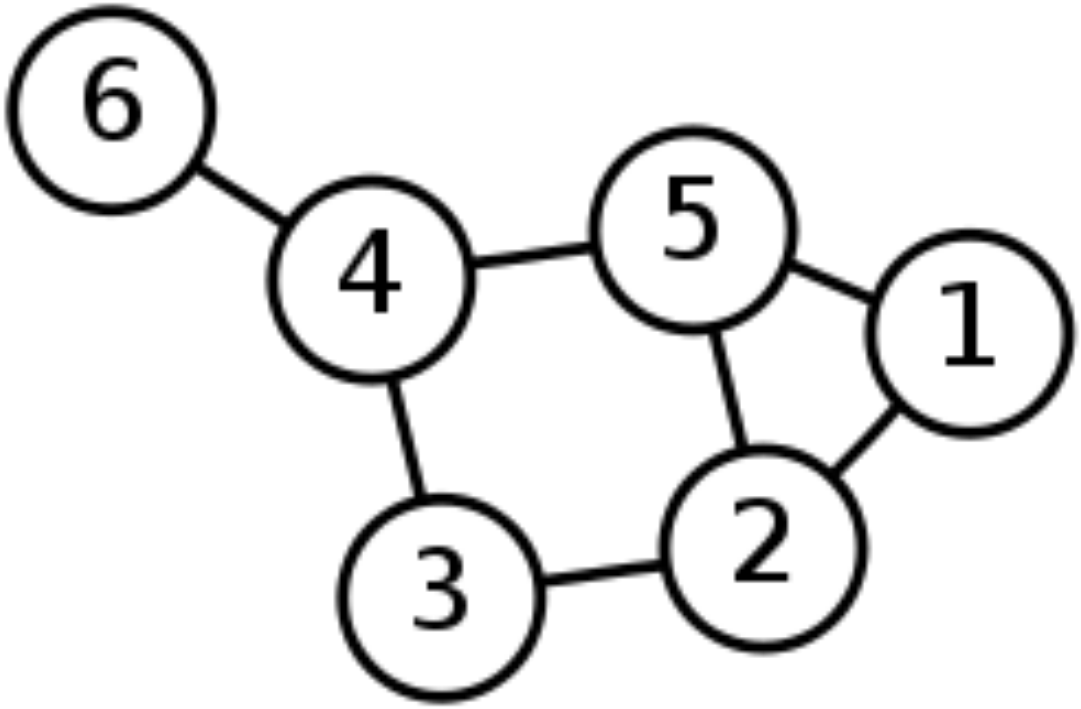


* Similar to an Euler path! But now we need to touch each node once (not each edge).

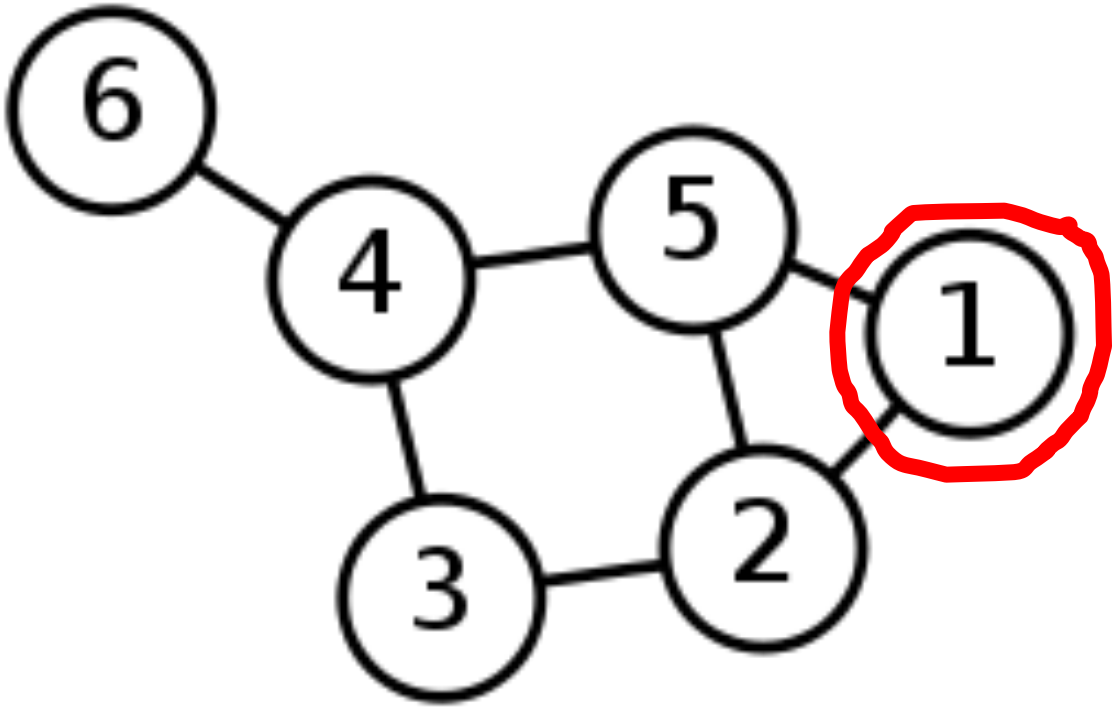
```
bool hamilton(BasicGraph & g);
```

Solution: Try all *decisions*
Look for any combination of decisions that lead to a hamilton path

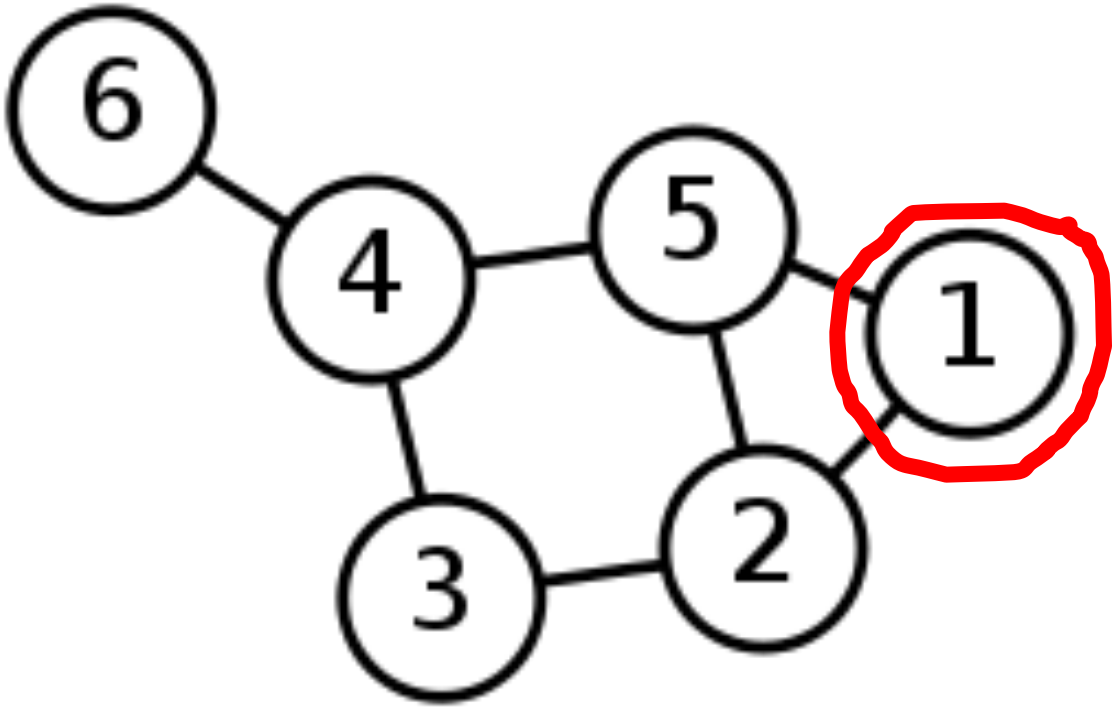
Decision #1: Where to start



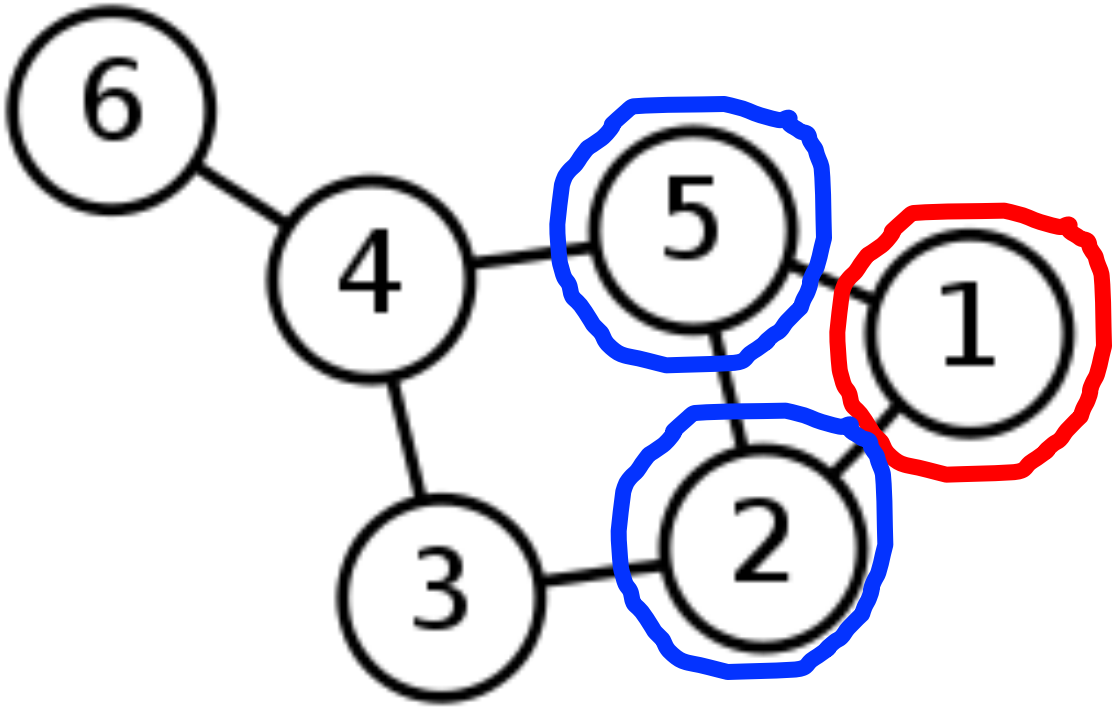
Decision #1: Where to start



Decision #2: Next node?



Decision #2: Next node?

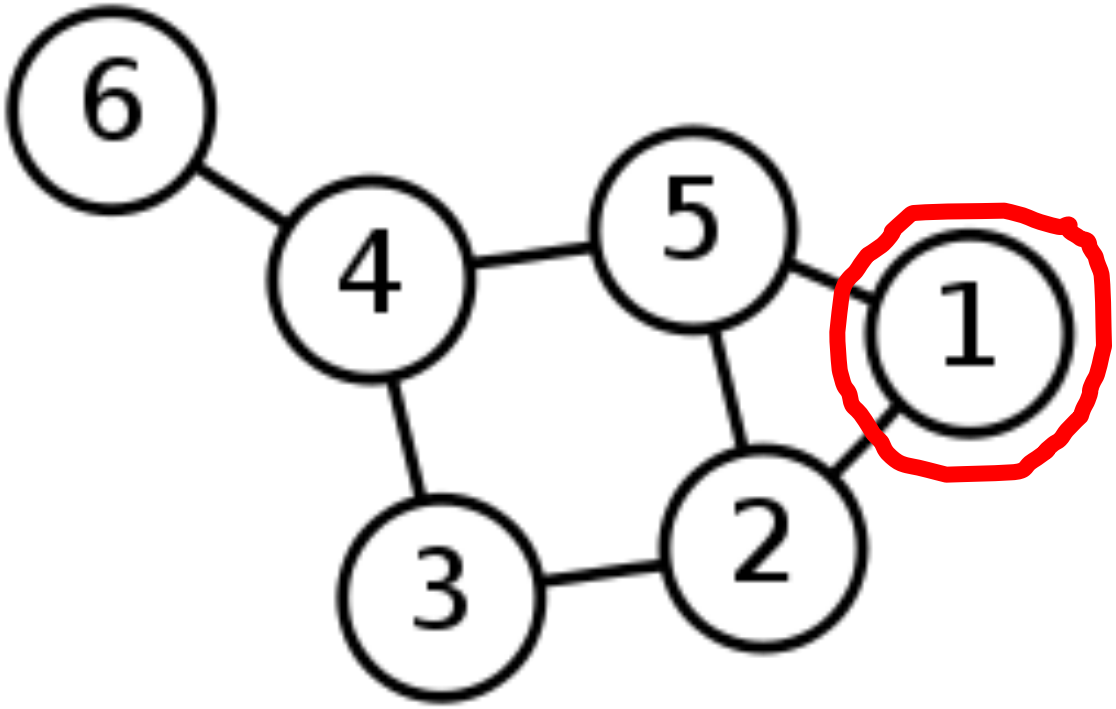


First. Let’s use a helper method to look for a path on each node.

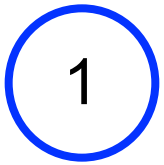
Done? Great!

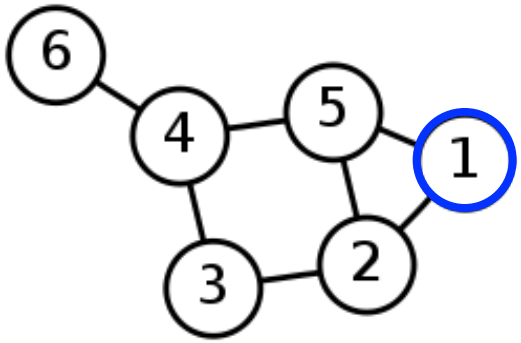
Next: Find

Hamiltoninan Path from Here?

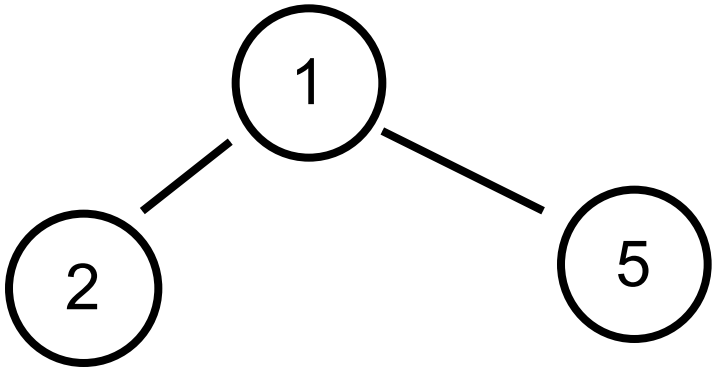


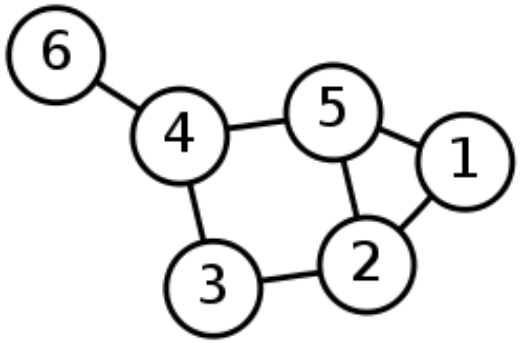
Decision Tree



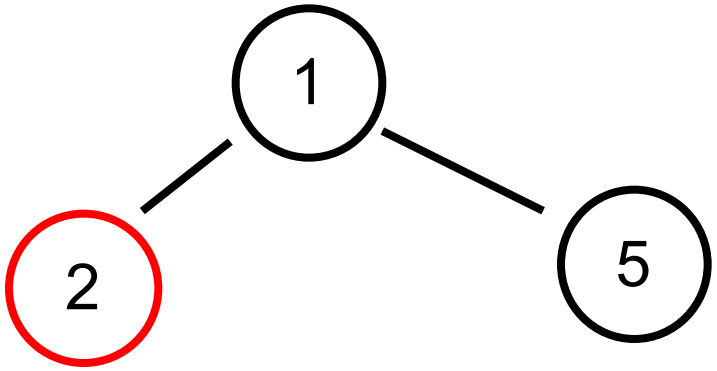


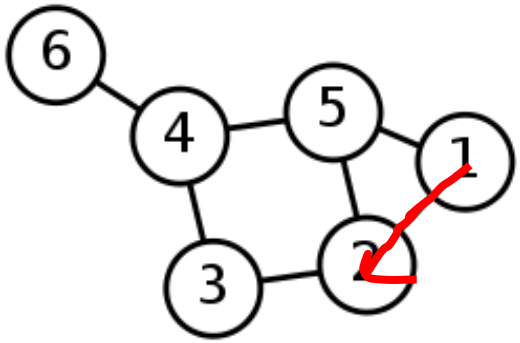
Decision Tree



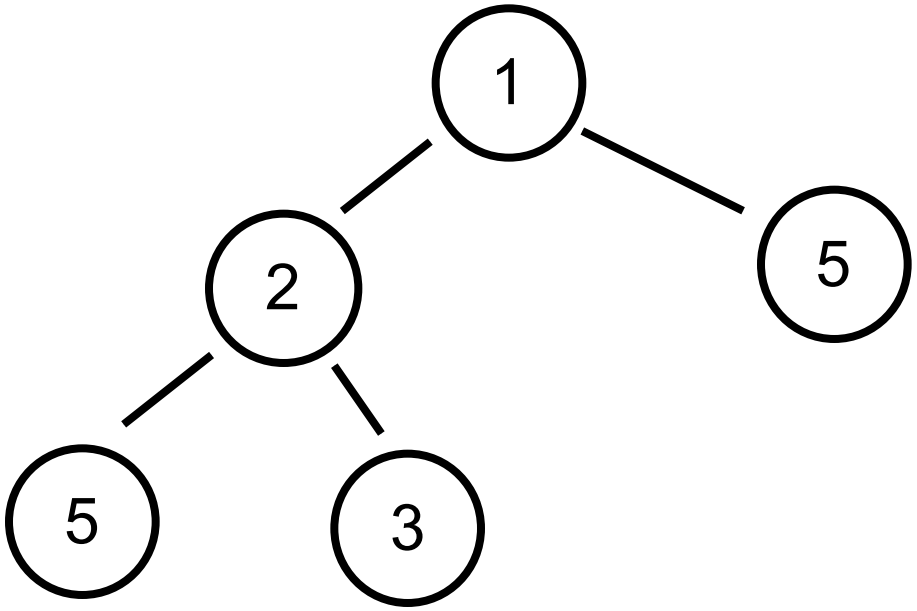


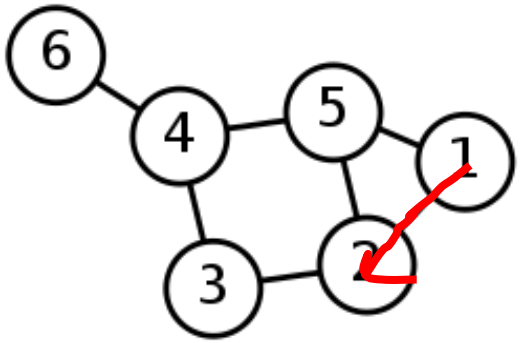
Decision Tree



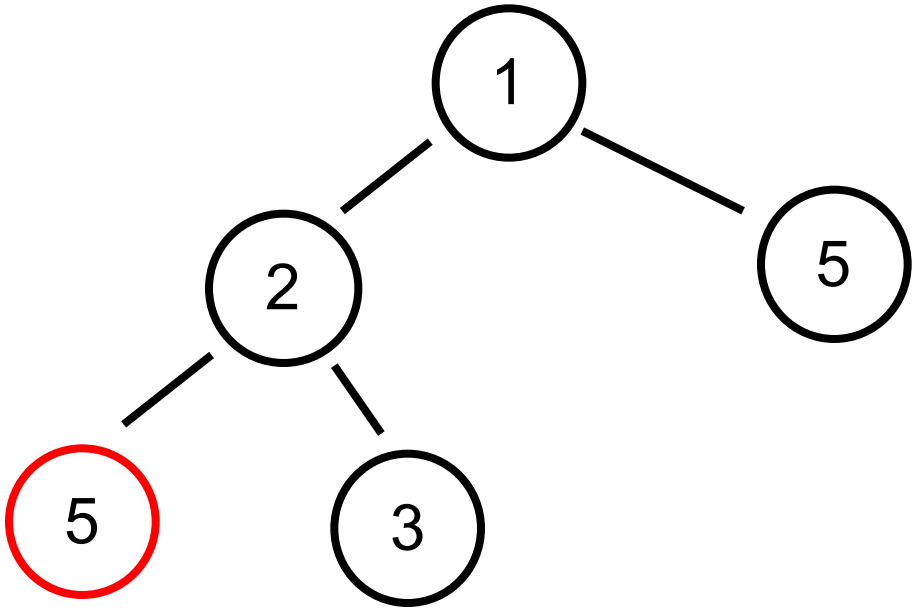


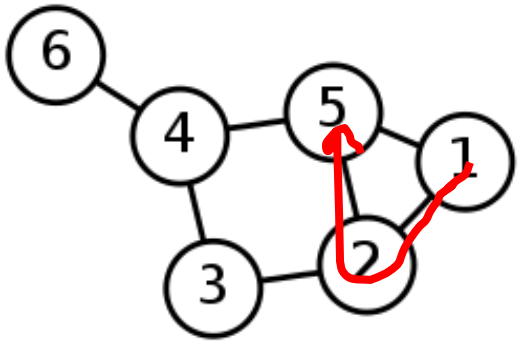
Decision Tree



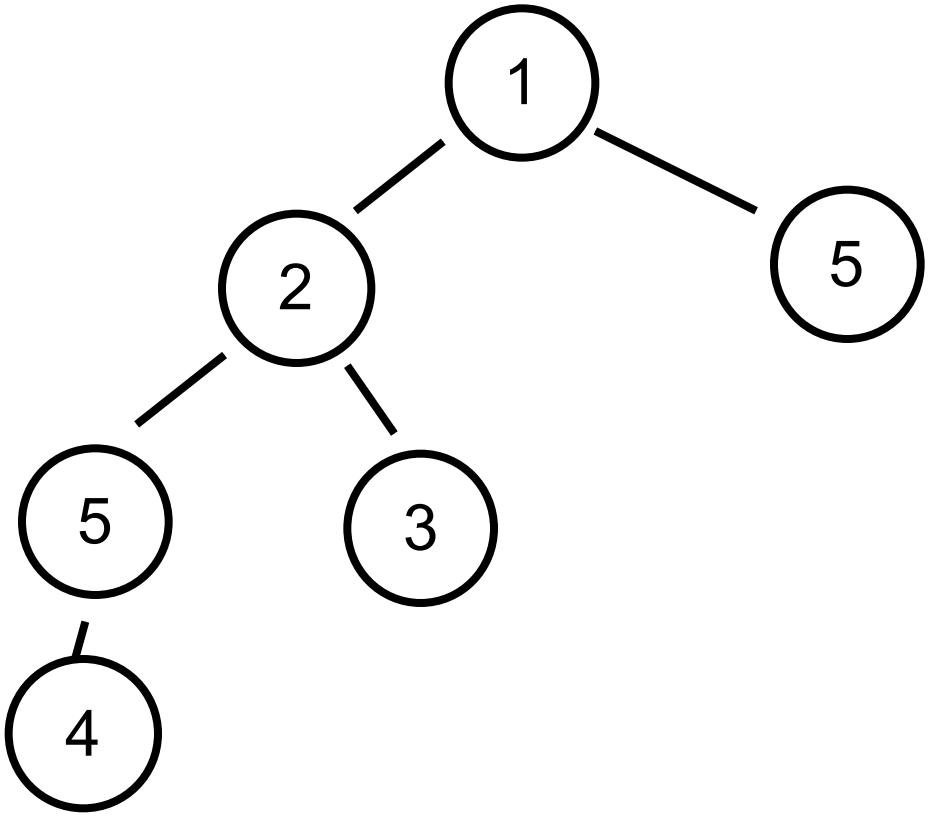


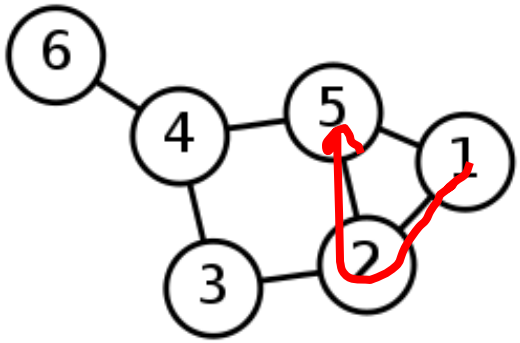
Decision Tree



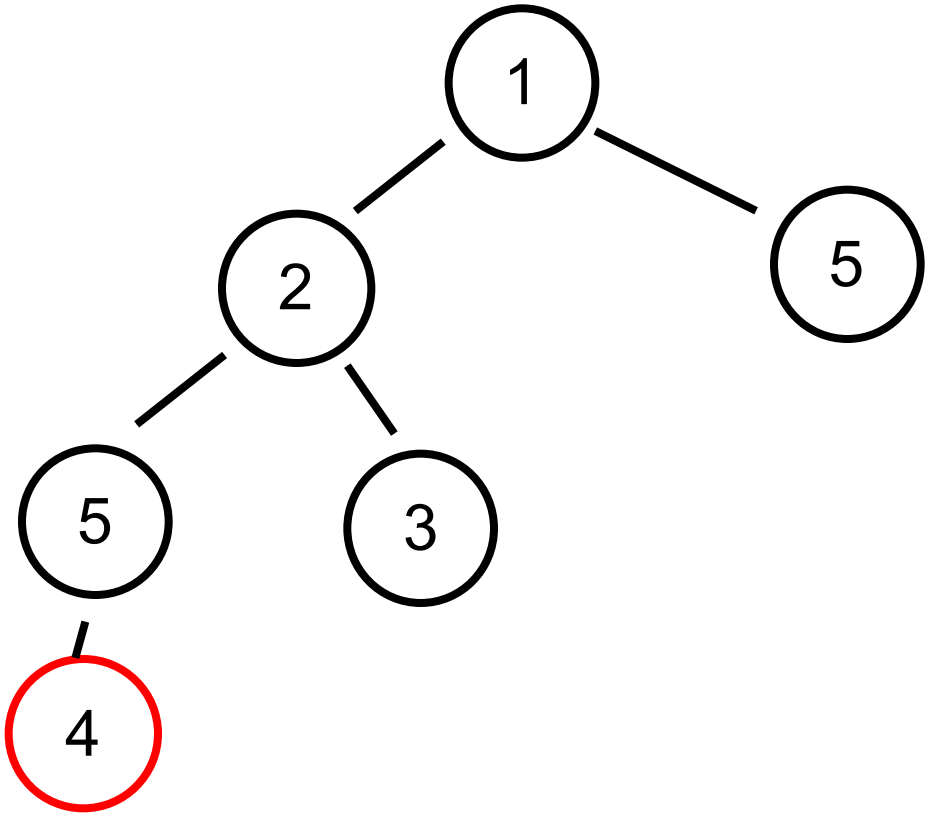


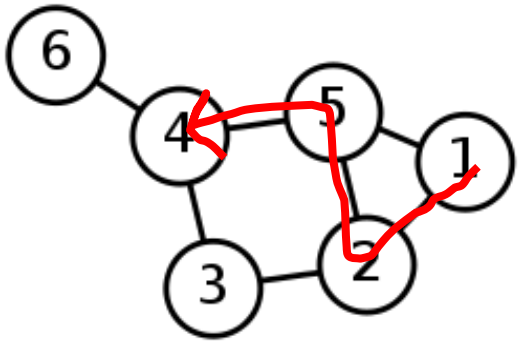
Decision Tree



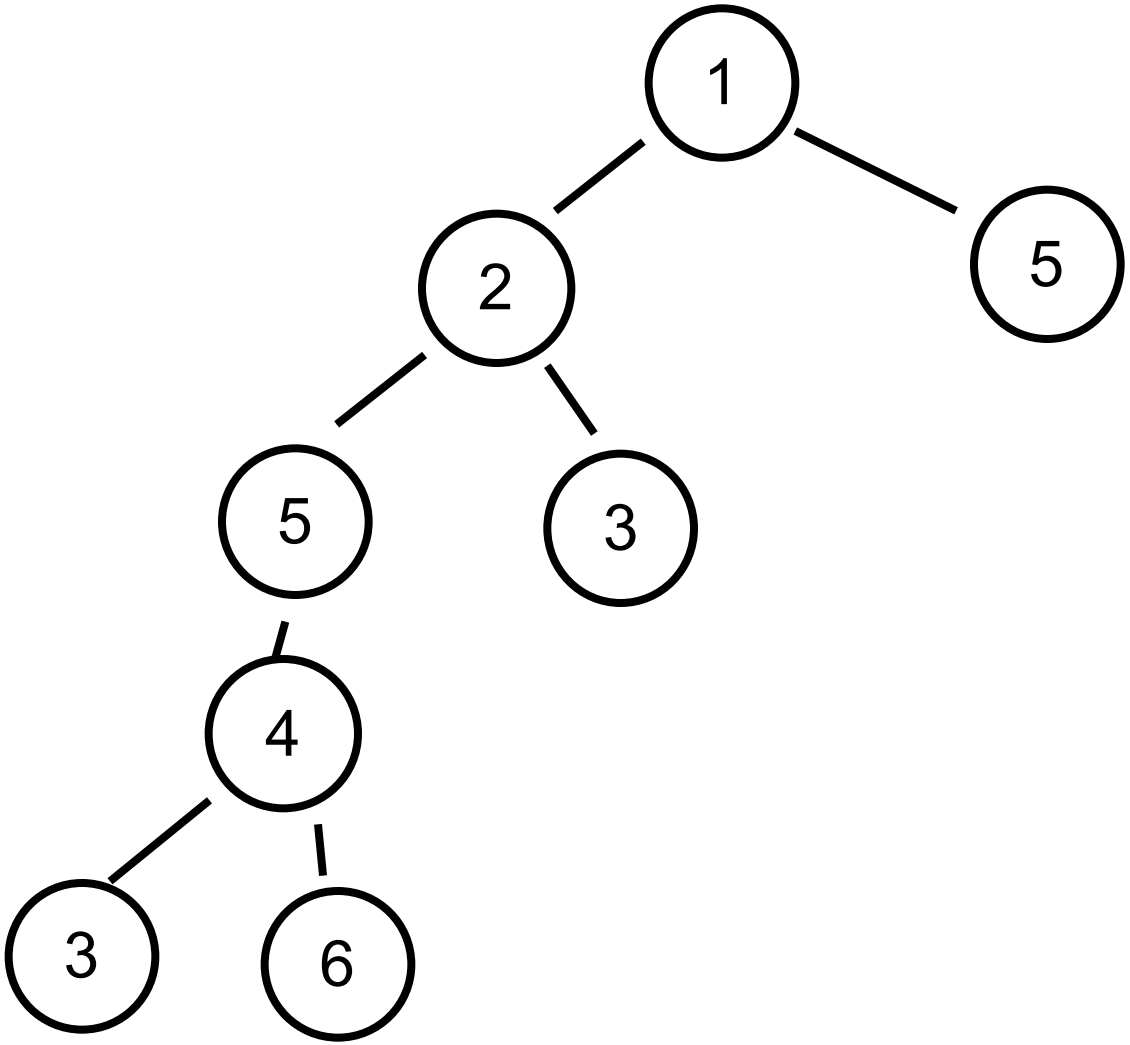


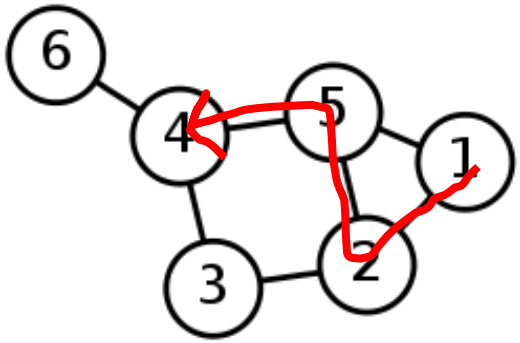
Decision Tree



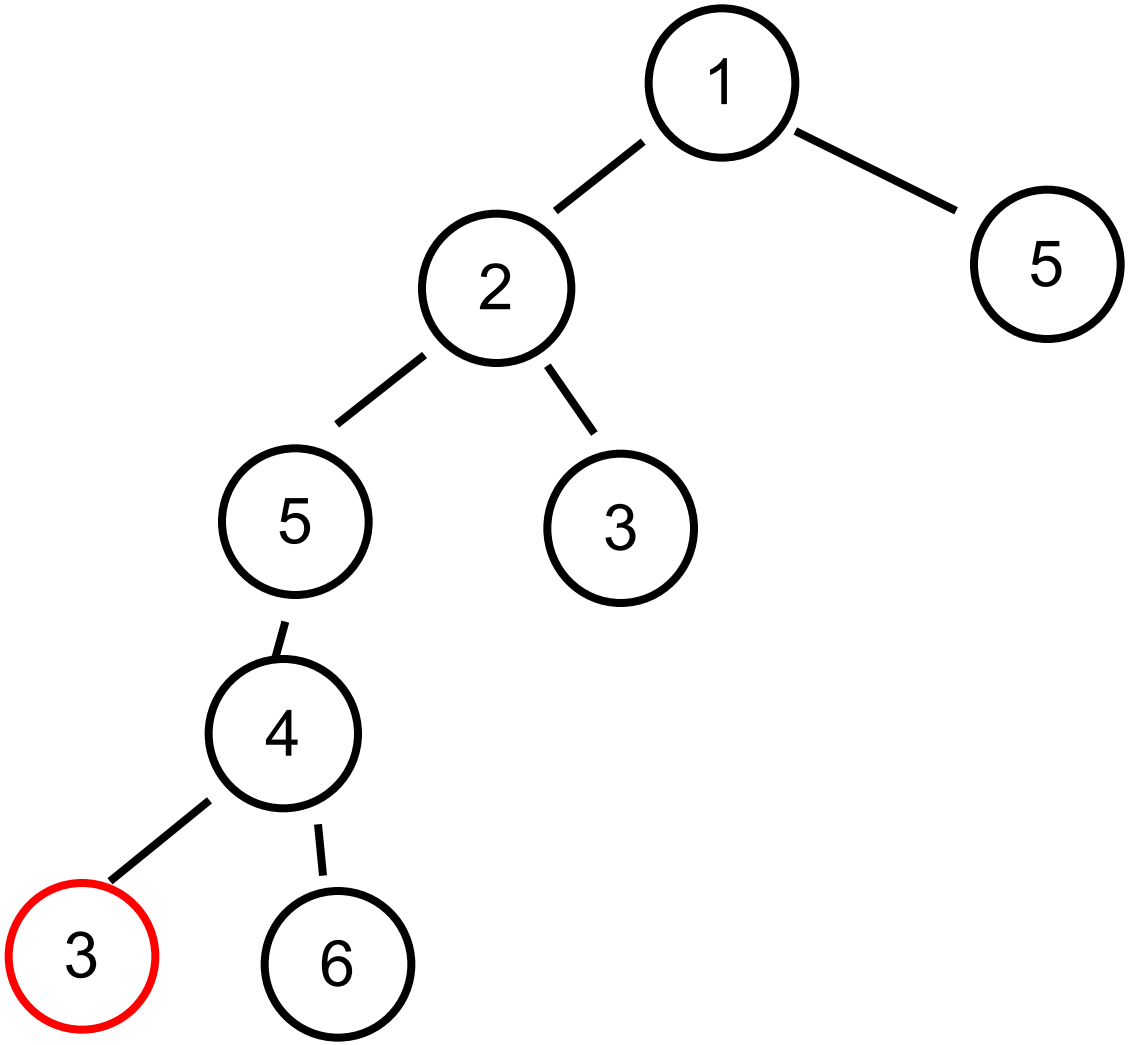


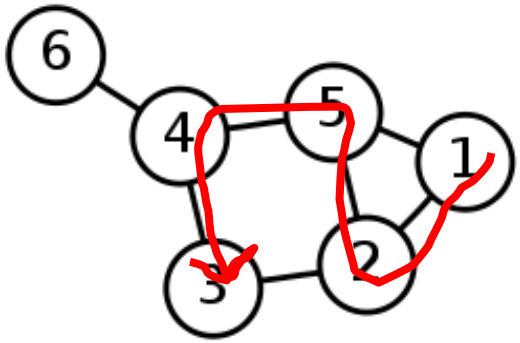
Decision Tree



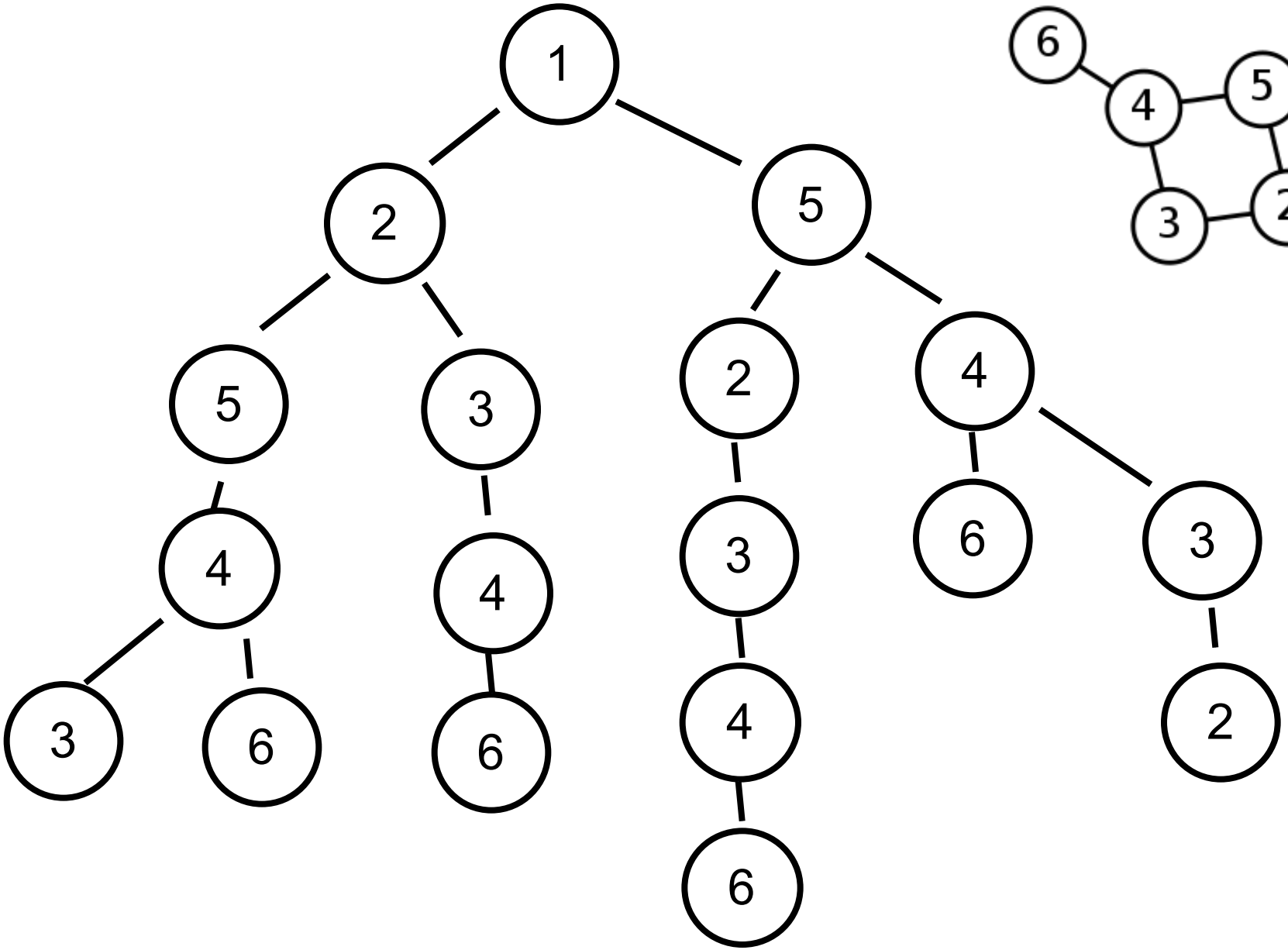


Decision Tree

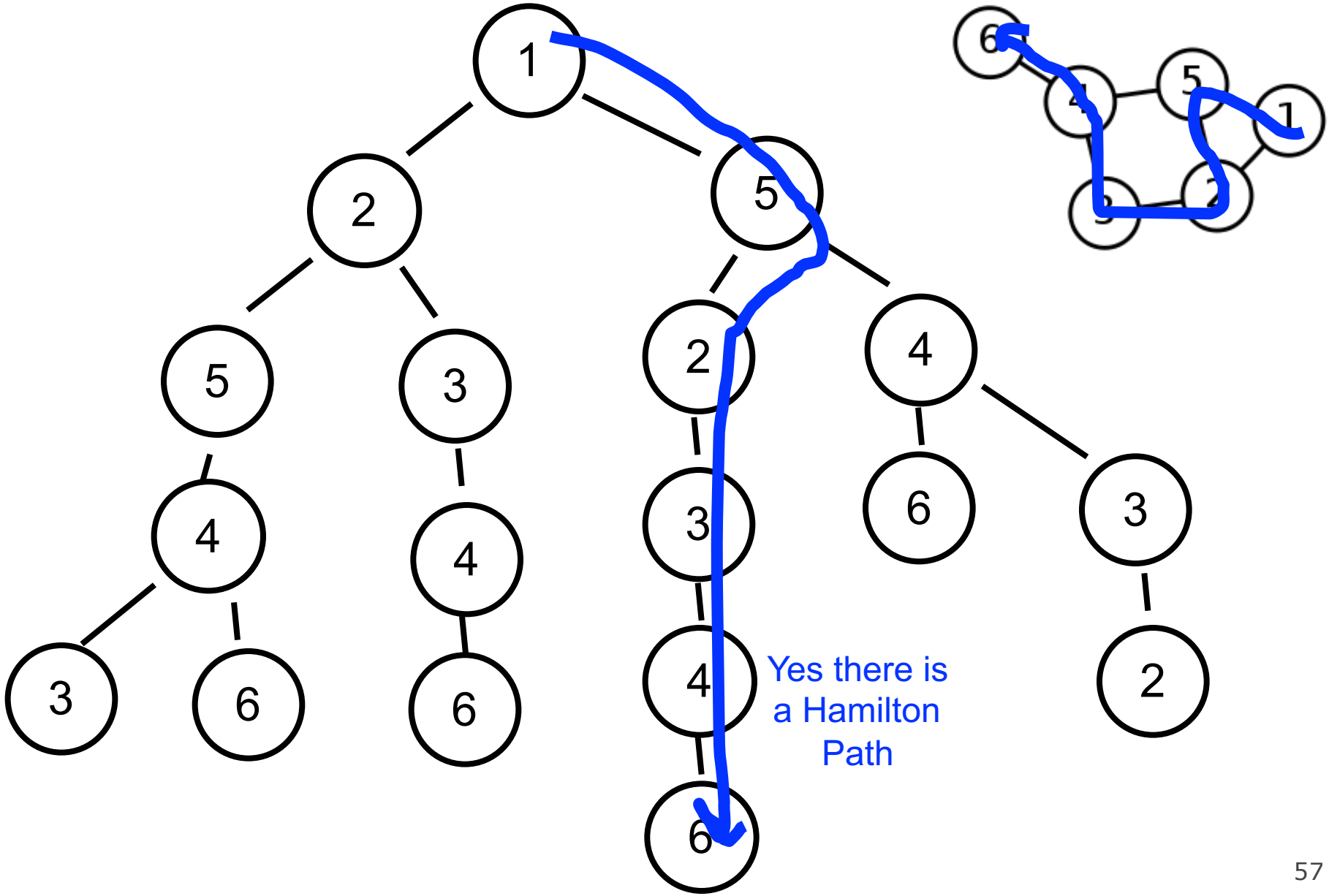




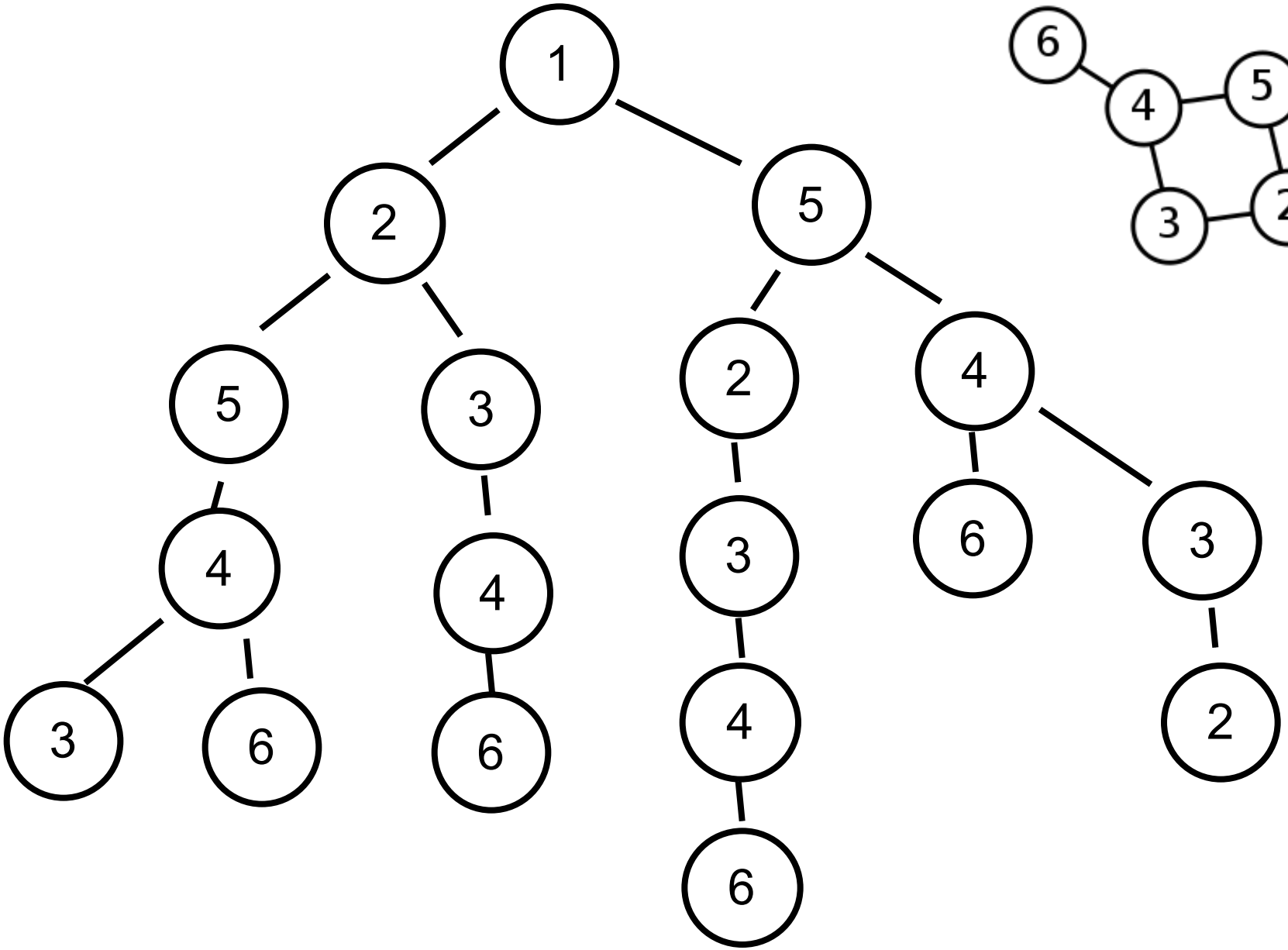
Decision Tree



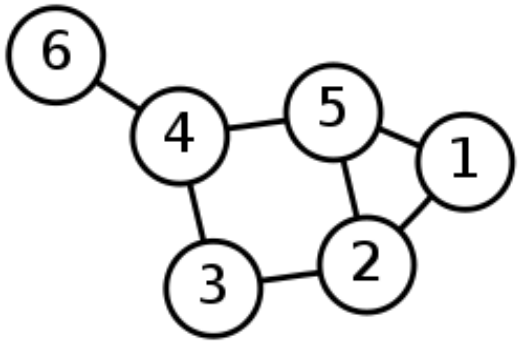
Decision Tree



To the Board!



Hamiltonian Path (soln)

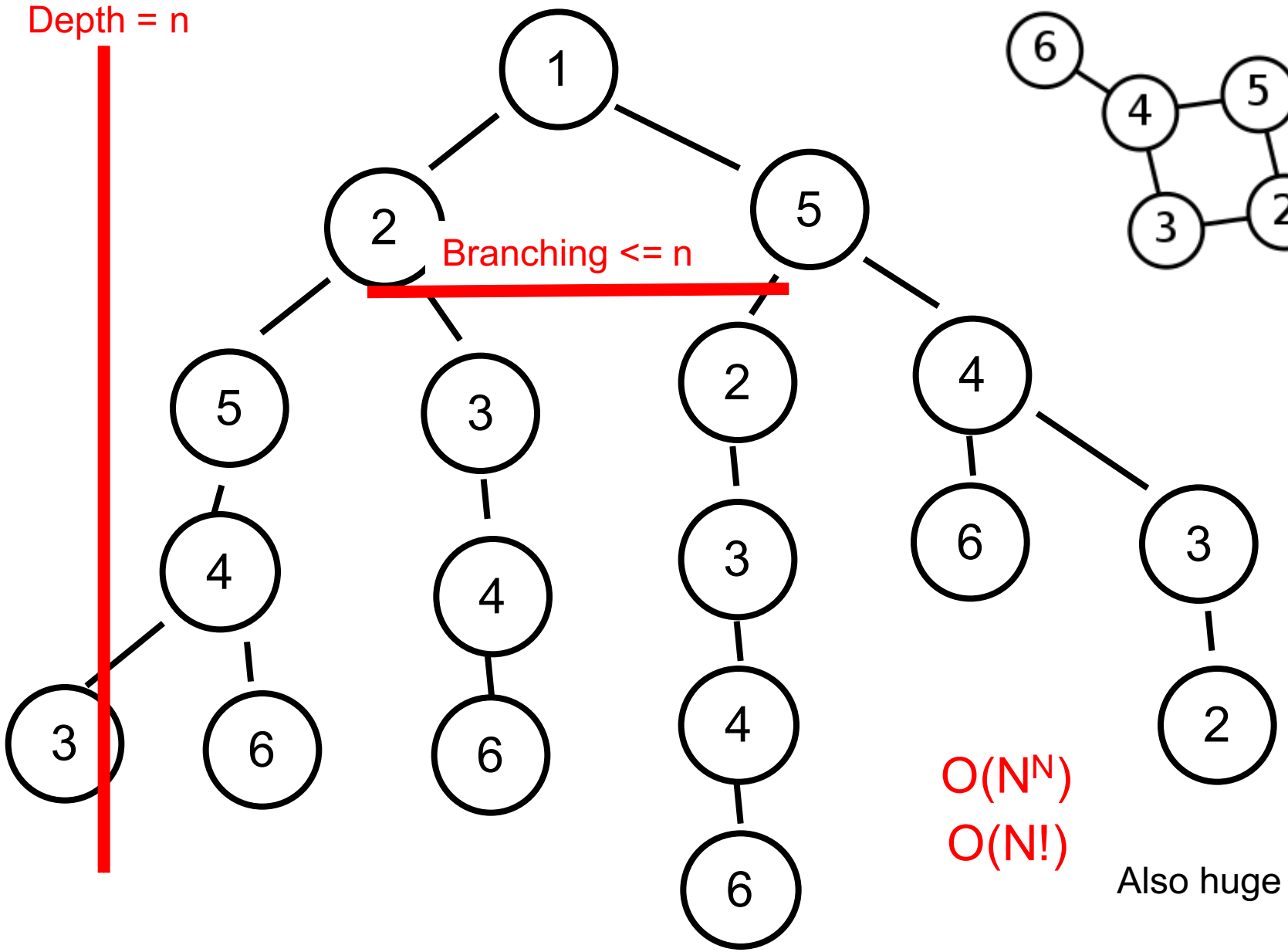



```
/* Function: Hamilton
 * Usage: hamilton(egoGraph)
 * -------------------------
 * Tests if there is a hamilton path starting at any vertex
 */
bool hamilton(BasicGraph & g) {
    for(Vertex * v : g.getVertices()) {
        Set<Vertex * v> seen;
        if(helper(g, seen, v)) return true;
    }
    return false;
}
```

Hamiltonian Path (soln)

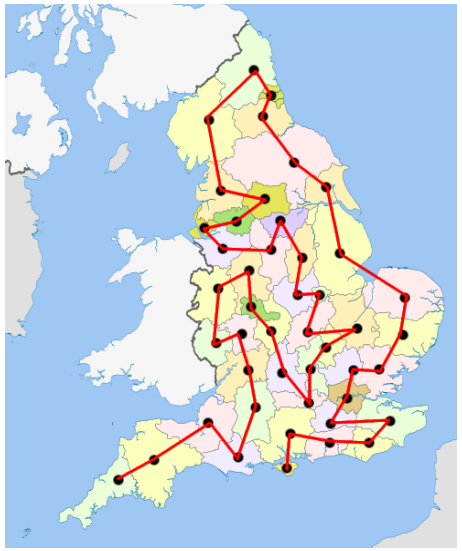
```
/* Function: Helper
 * Usage: helper(egoGraph, seen, start)
 * ------------------------
 * Recursive helper. Find a path from curr to all nodes?
 */
bool helper(BasicGraph & g, Set<Vertex *> seen, Vertex * curr) {
    if(g.size() == seen.size()) return true;
    seen.add(curr);

    for(Vertex * neighbor : g.getNeighbors()) {
        if(!seen.contains(neighbor)) {
            if(helper(g, seen, neighbor)) return true;
        }
    }

    return false;
}
```

Big O?

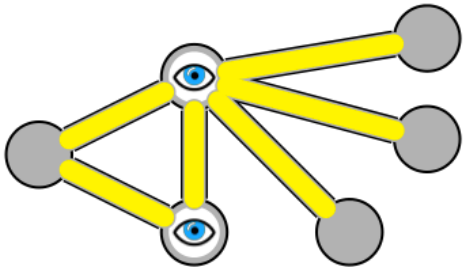

Depth = n
Branching <= n
1
2
5
5
3
2
4
4
6
3
3
4
6
6
3
4
6
2
6
4
5
1
3
2
O(N^N)
O(N!)
Also huge

Non Deterministic Polynomial Problems

Hamilton Path

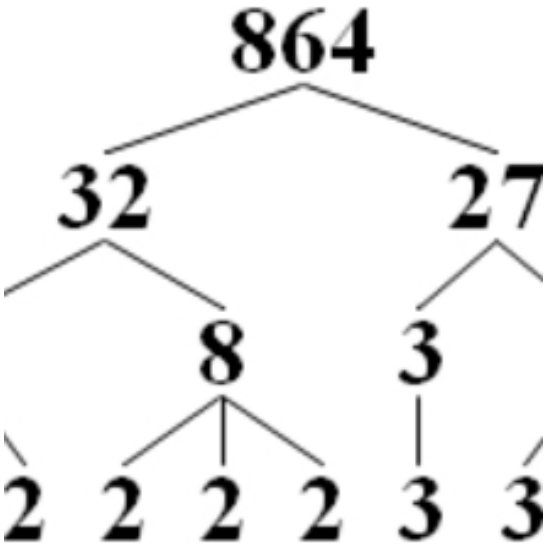


Vertex Cover

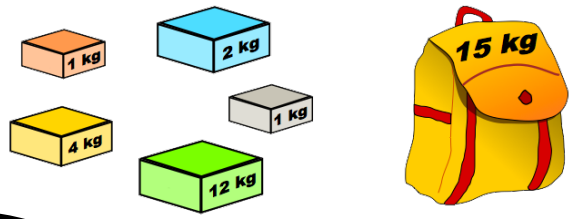


Same class of problem

Integer Factorization



Knapsack Problem



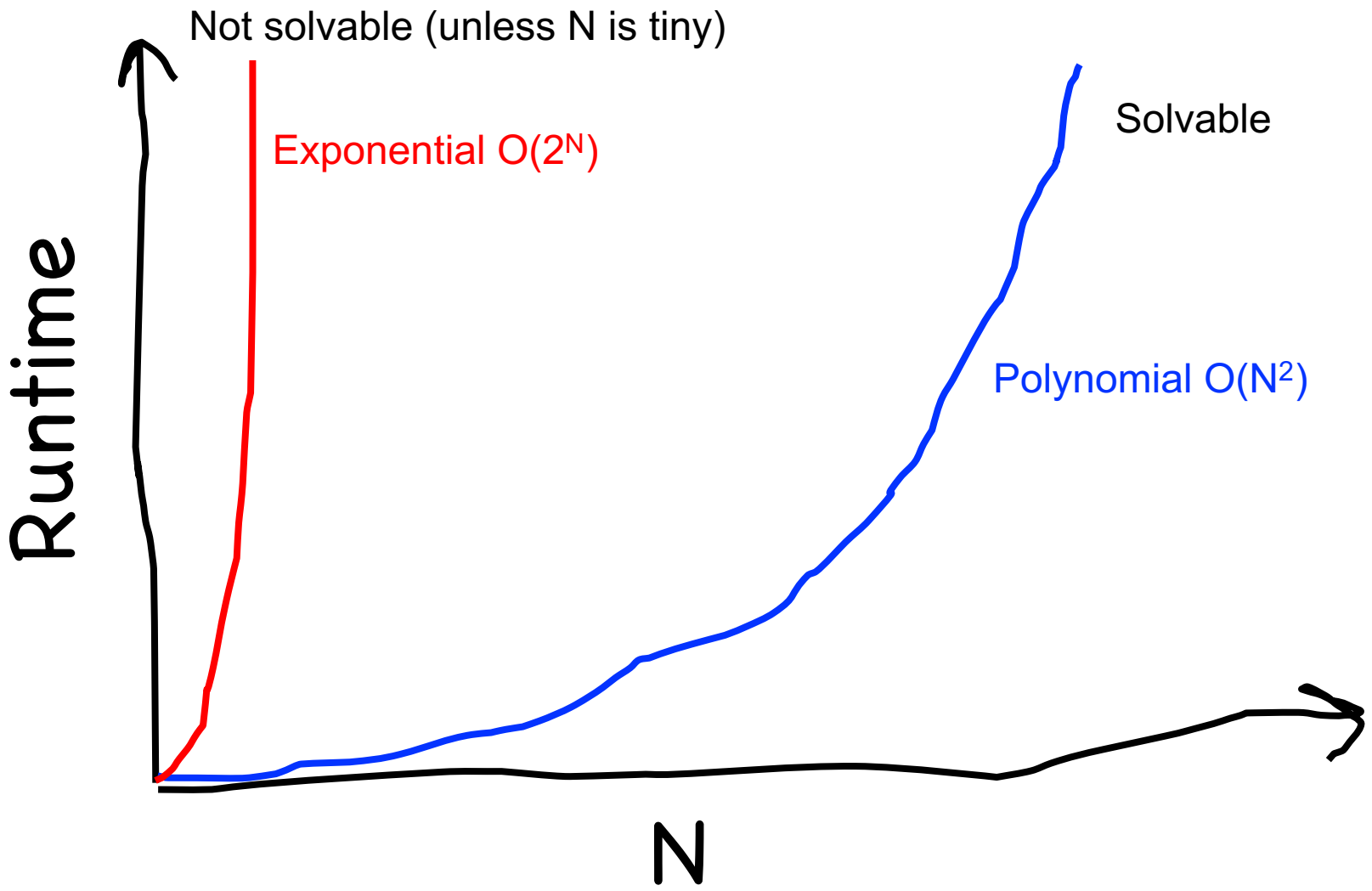
Polynomial: $O(N^k)$

* Where k is a number. Like 8. Or 2.

Exponential: $O(k^N)$

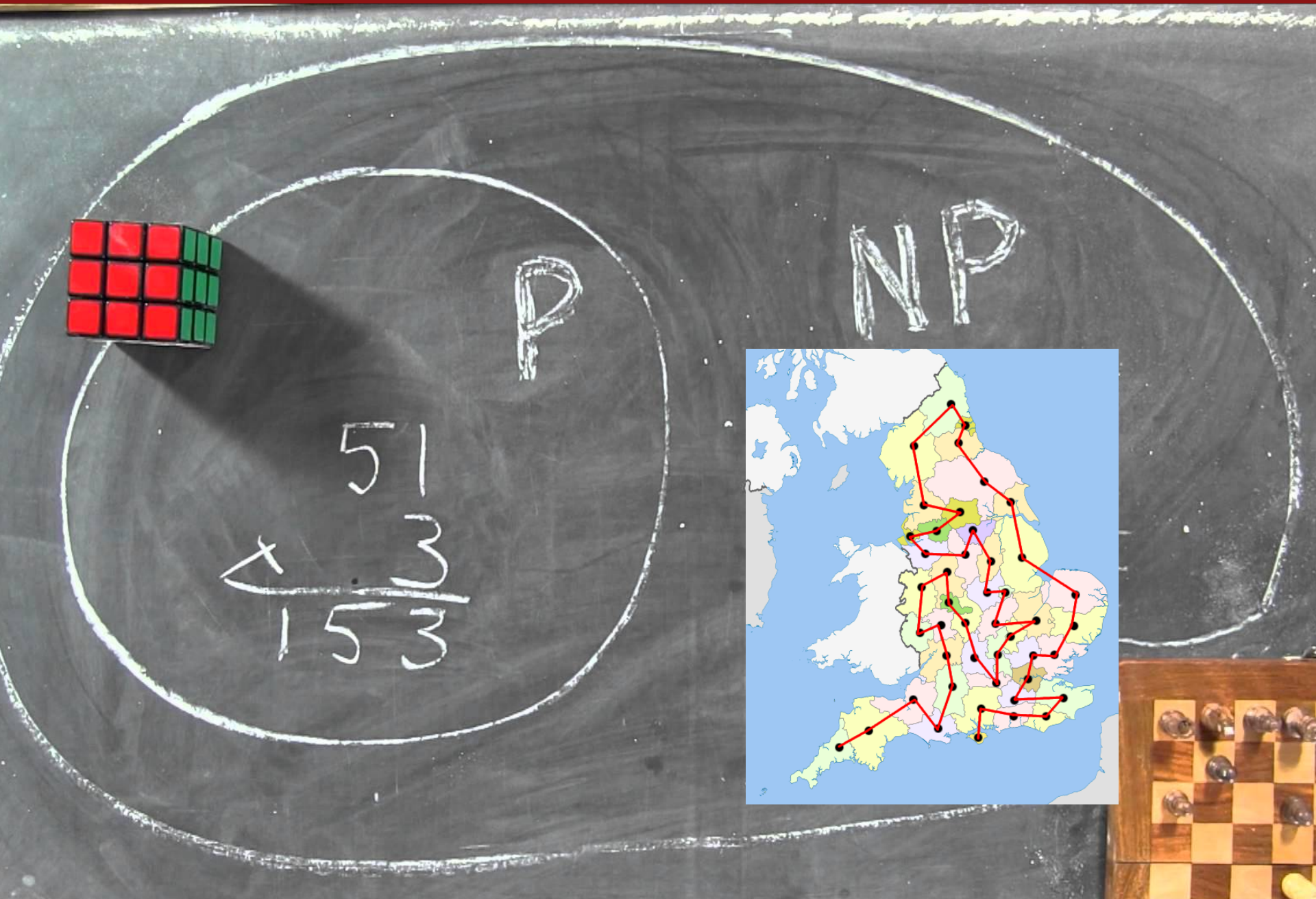
* Where k is a number. Like 8. Or 2.

The Big O Matters



When n = 300, 2^n is larger than the # atoms in the observable universe

P VS NP



Great Unsolved Problems in CS

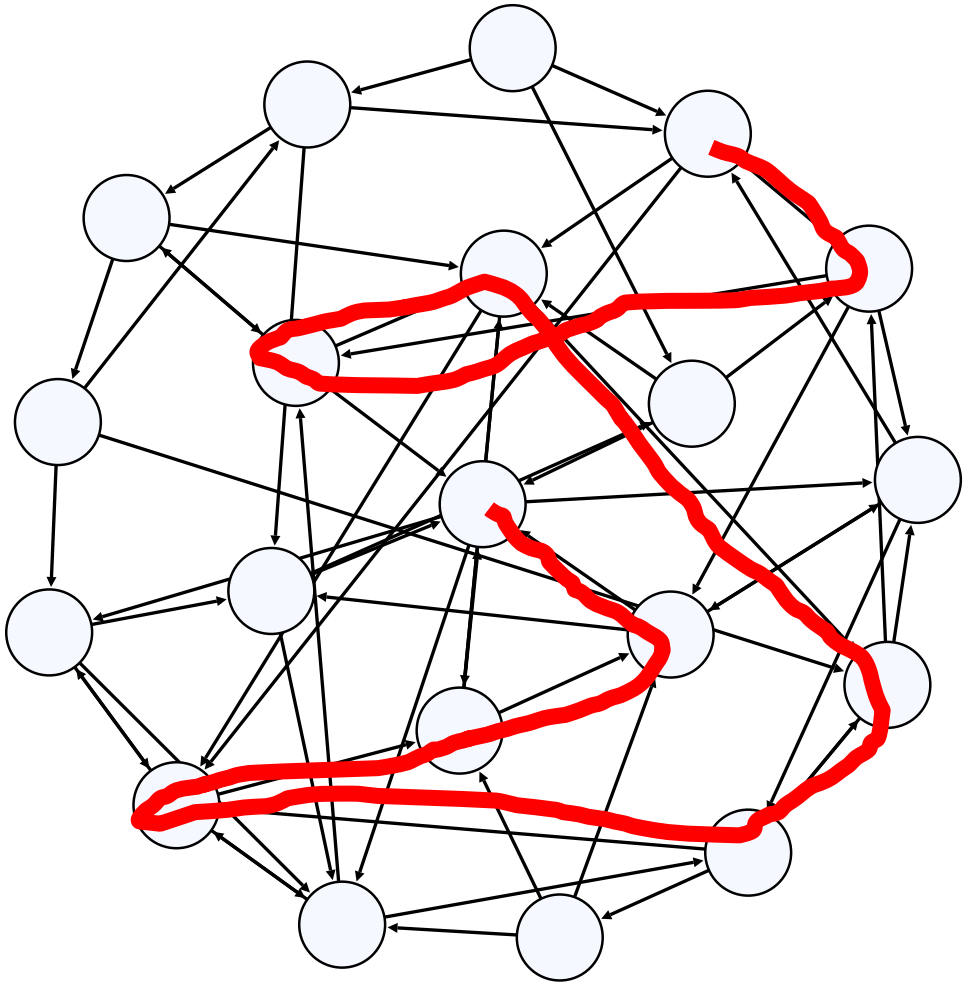
* First posed in 1956

Prize $1 million

Play a Game

Directed Graph
20 Nodes

Find a path of length at least 3? Email Chris P with each of the node numbers (and the students in the path)



There will be a prize for whomever can find the longest path (each person on the path) without repeating nodes.

Happy Thanksgiving

* You can come to LAiR or Class. But we won't be there! LAiR is back on the Sunday the 27th of November