Graphs Lecture 2

Today

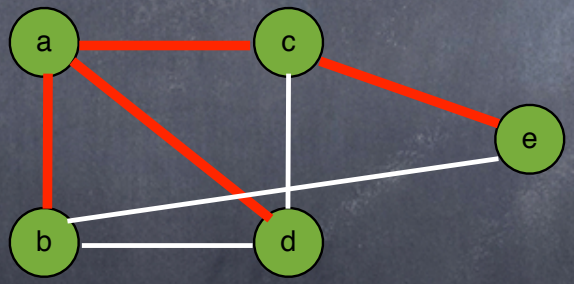
- BFS/DFS
  - Review; proof about DFS tree
  - Implementation
  - Running time
- Bipartite testing
- Topological sort

BFS/DFS Review

- Examples on board

Breadth-First Search

- Property. Let T be a BFS tree of G = (V, E), and let (x, y) be an edge of G. Then the layer of x and y differ by at most 1.

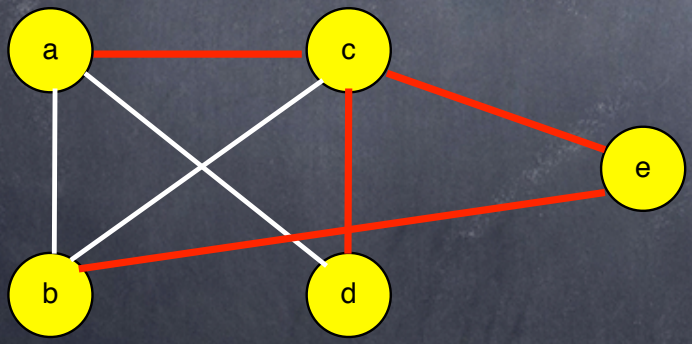


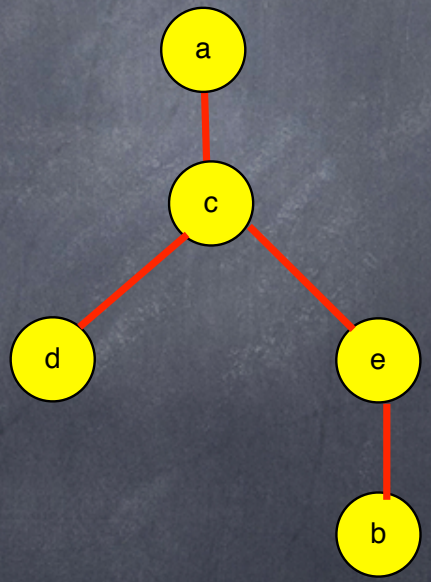
Layer 0: {a}
Layer 1: {b, c, d}
Layer 2: {e}

Proof?

Depth First Search

Theorem: Let T be a depth-first search tree. Let x and y be 2 nodes in the tree. Let (x, y) be an edge that is in G but not in T. Then either x is an ancestor of y or y is an ancestor of x in the DFS tree.





Proof on board

Graph Traversal

```
Set explored[u] to be false for all u
A = { s } // set of discovered but not explored nodes
while A is not empty
    Take a node u from A
    if explored[u] is false
        set explored[u] = true
        for each edge (u,v) incident to u
            add v to A
        end
    end
end
```

BFS: A is a queue (FIFO)

DFS: A is a stack (LIFO)

BFS: Alternate Implementation

```
Set discovered[u] to be false for all u
A = queue { s } // set of discovered but not explored nodes
layer[s] = 0
discovered[s] = true
while A is not empty
    Take a node u from A
    for each edge (u,v) incident to u
        if discovered[v] is false
            add v to A
            layer[v] = layer[u] + 1
            discovered[v] = true
            add (u,v) to T
        end
    end
end
```

When A is a queue, it is equivalent to check for duplicates when adding to A

Running Time

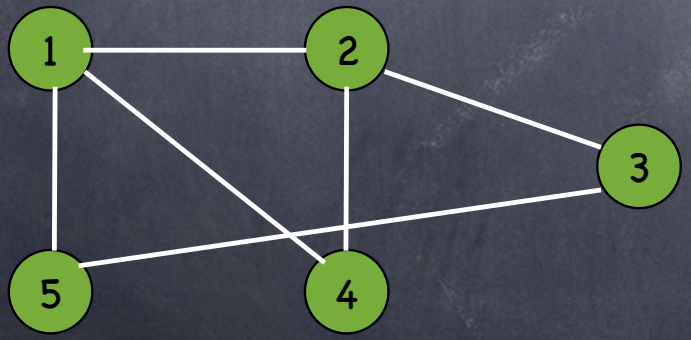
```
Set explored[u] to be false for all u
A = { s } // set of discovered but not explored nodes
while A is not empty
    Take a node u from A
    if explored[u] is false
        set explored[u] = true
        for each edge (u,v) incident to u
            add v to A
        end
    end
end
```

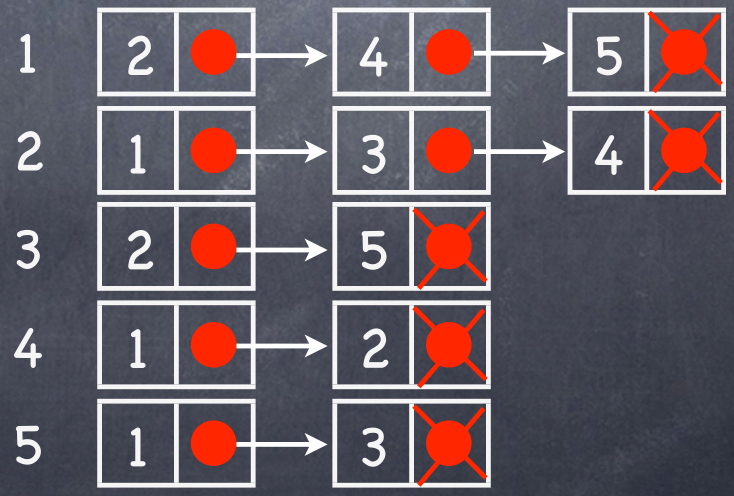
Discuss on board: running time is O(n+m), pending correct data structure...

Representing Graphs: Adjacency List

Adjacency list. Node indexed array of lists.

- Two representations of each edge.
- How much memory?
- How long to find a specific edge?
- How long to find all edges incident on a node?





Finding all Connected Components in a Graph

- Running BFS or DFS find all nodes connected to the start node
- How do we find all connected components?
- How expensive is that?

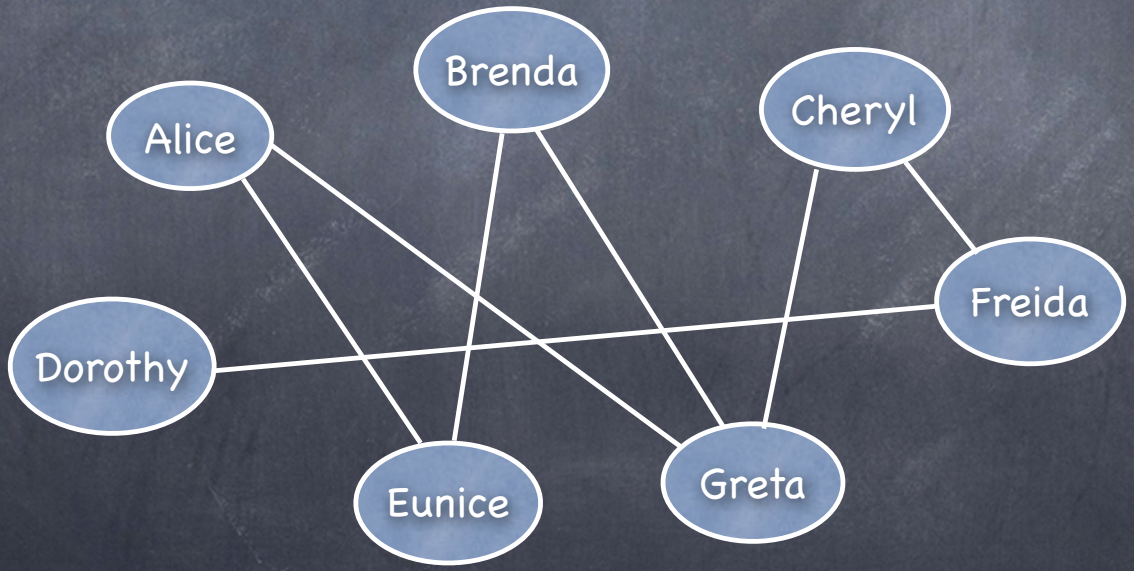
Party Problem

- You want to throw a party at which there are no pairs of guests that do not get along.
- You want to invite as many guests as possible.
- How would you solve this?

Application: Bipartite Testing

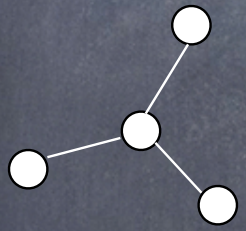
The party problem

- Represent each guest as a node
- Draw an edge between guests who do not get along
- Find the largest set of nodes where there is no edge between any pair of nodes in the set

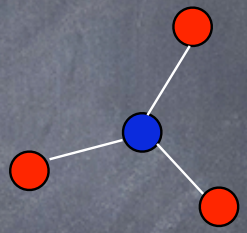


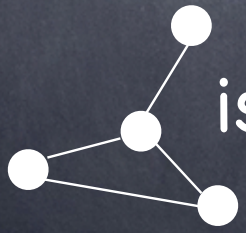
Bipartite Graphs

A bipartite graph is an undirected graph $G = (V, E)$ in which the nodes can be colored red or blue such that every edge has one red and one blue end.

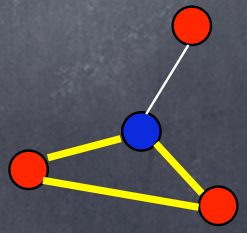


is a bipartite graph

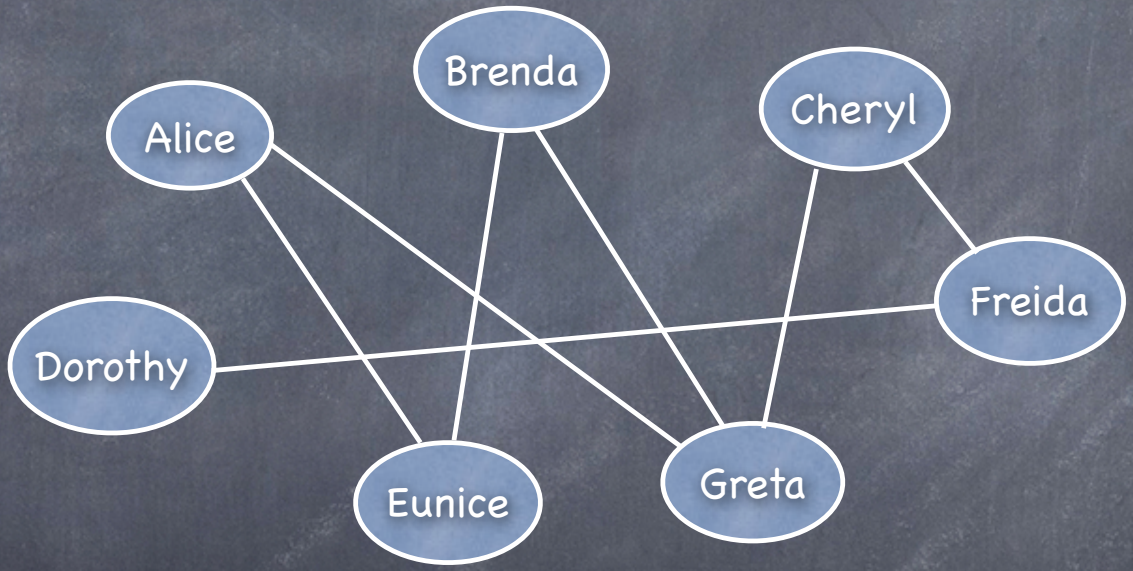




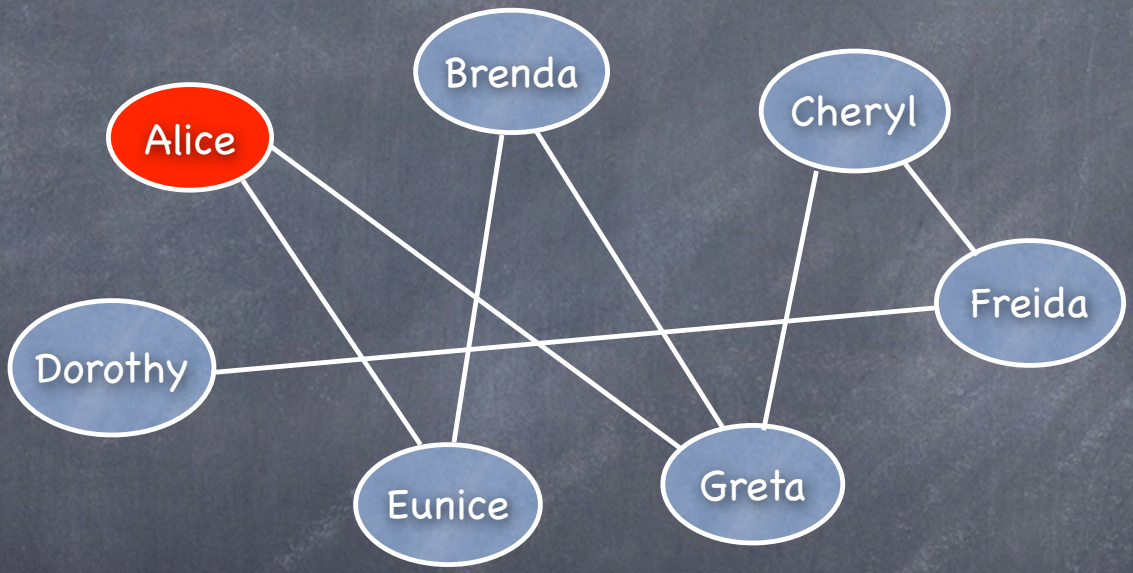
is NOT a bipartite graph



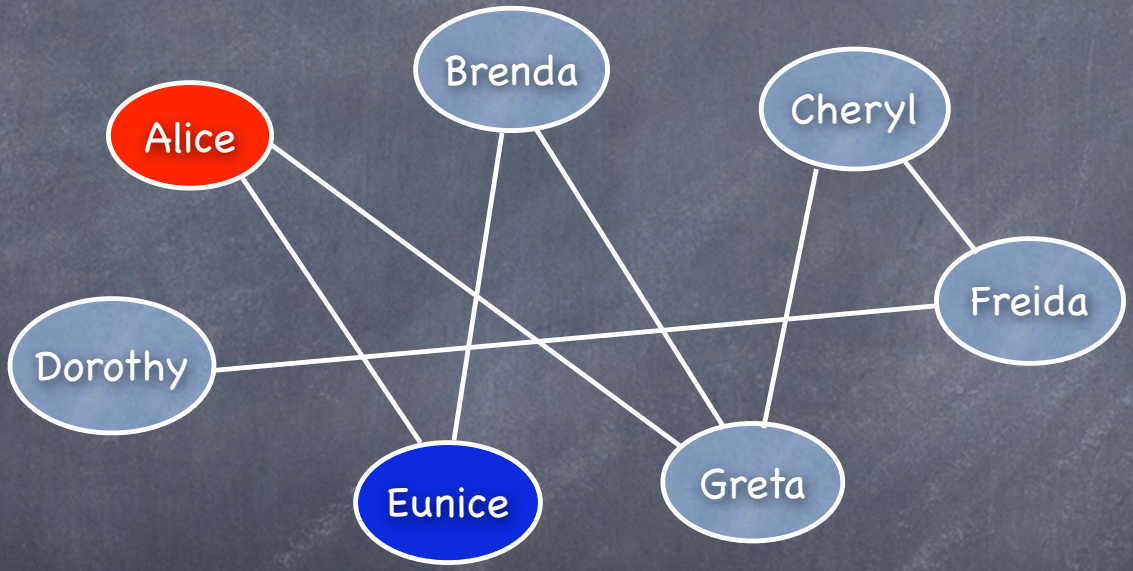
A bipartite solution



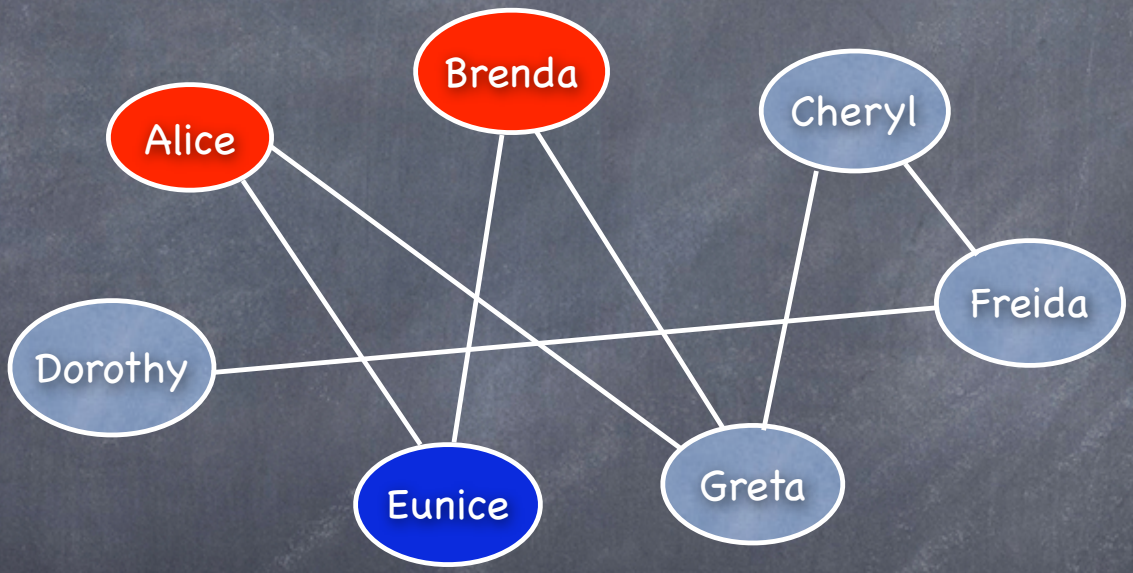
A bipartite solution



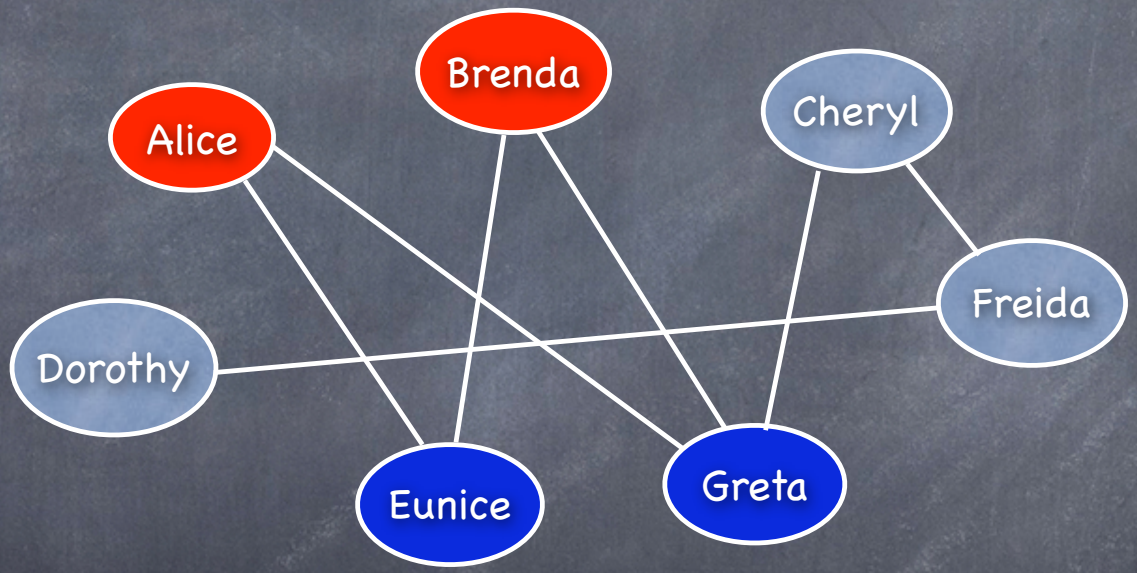
A bipartite solution



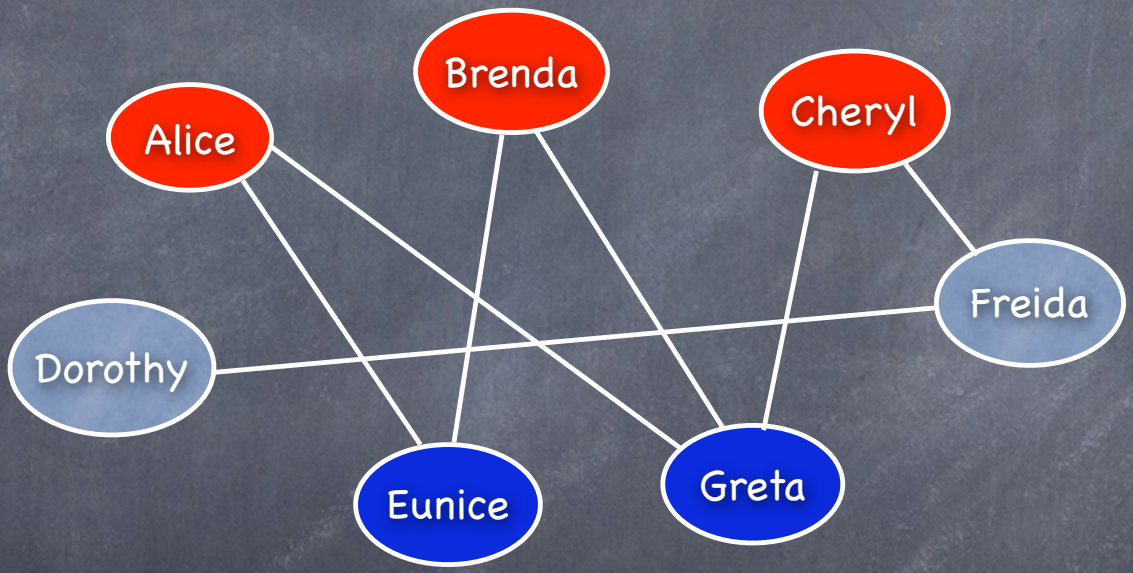
A bipartite solution



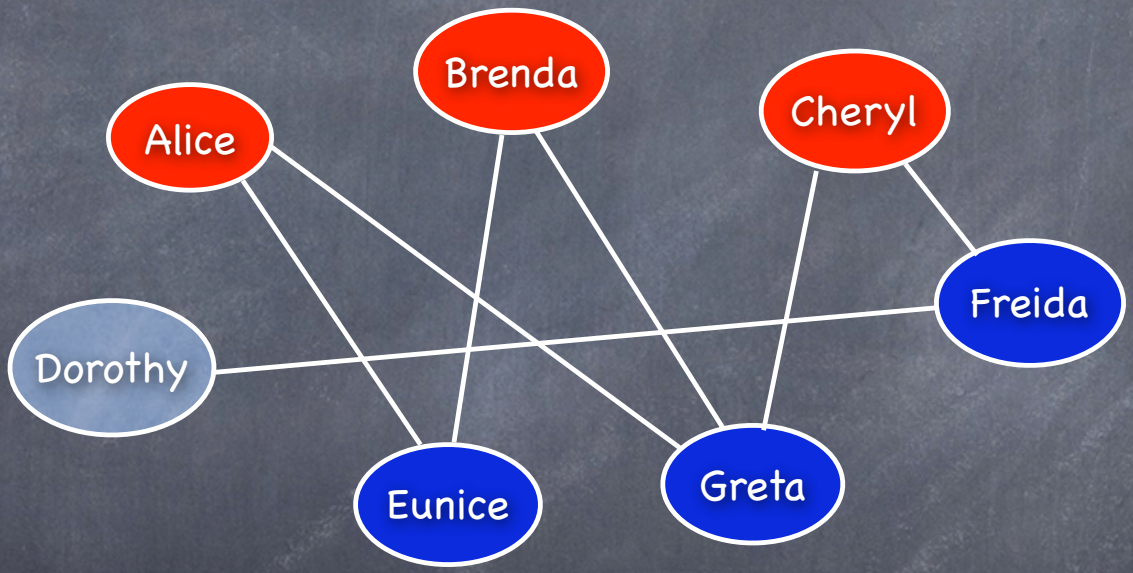
A bipartite solution



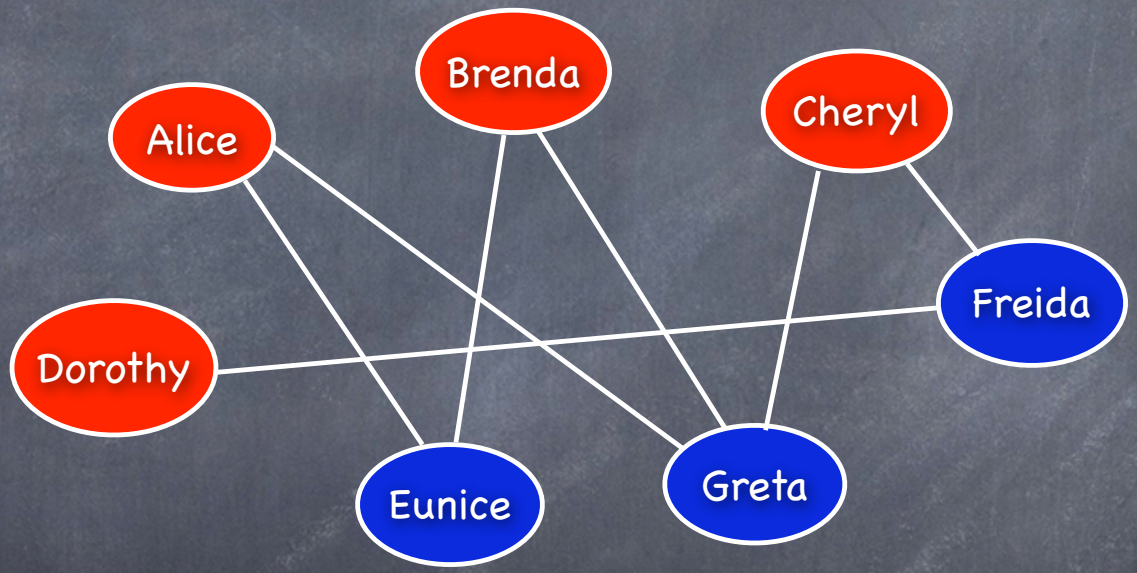
A bipartite solution



A bipartite solution



A bipartite solution



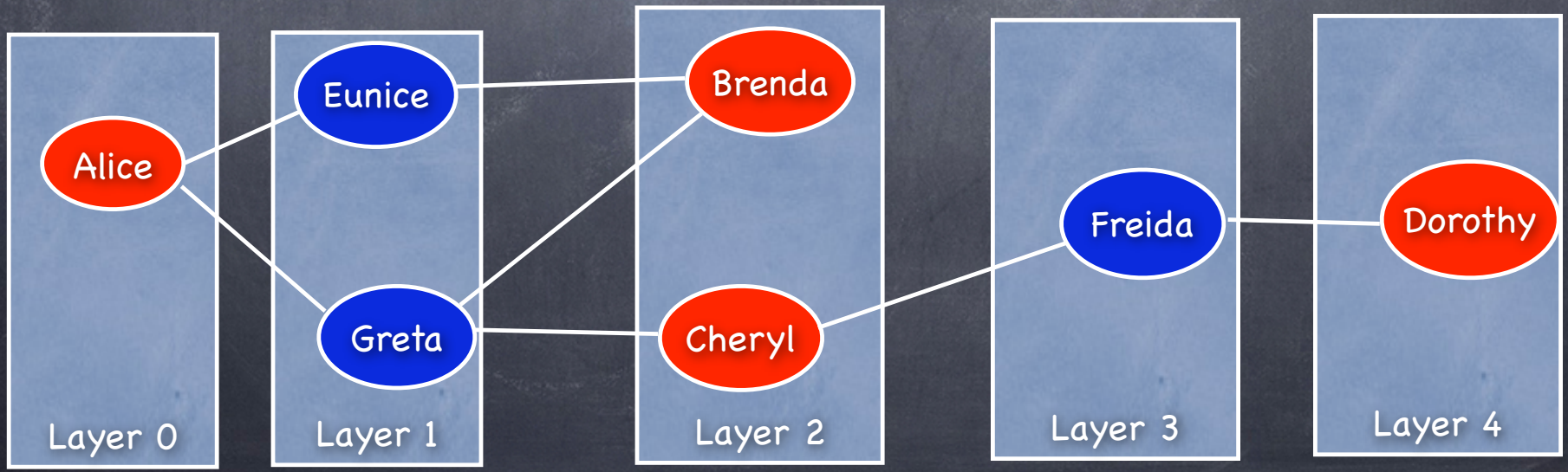
Who should you invite?

BFS and Bipartite Graphs

Let G be a connected graph

Lemma 1. G is bipartite if and only if G has no odd cycles

Lemma 2. Let T be a BFS tree of G. Then G is bipartite if and only if there is no edge between any two nodes in the same layer.

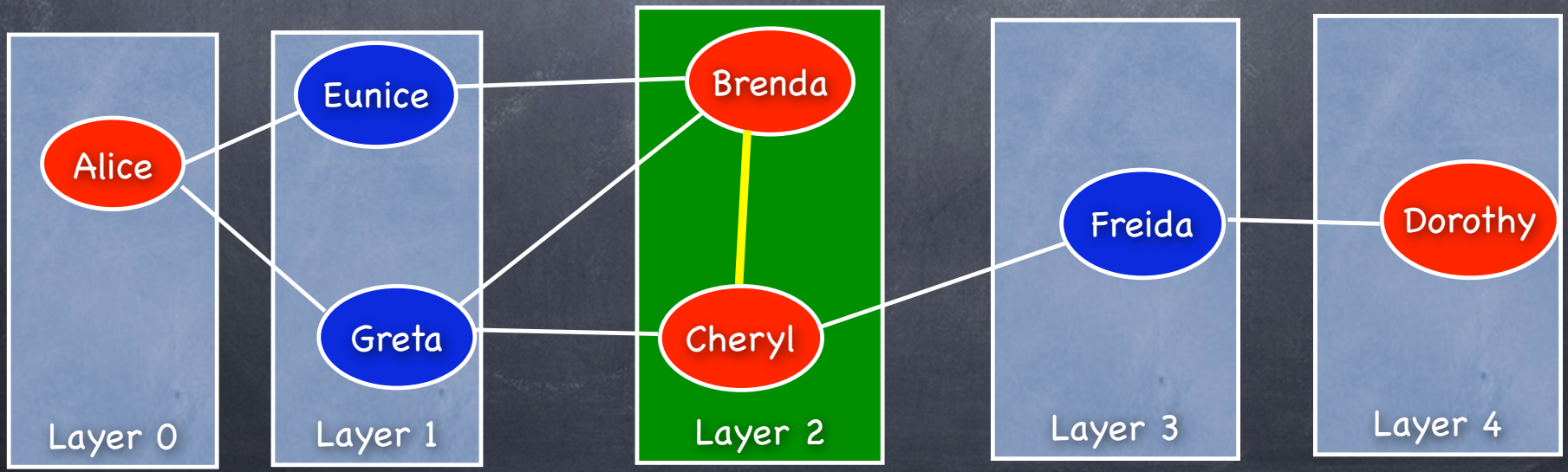


BFS and Bipartite Graphs

Let G be a connected graph

Lemma 1. G is bipartite if and only if G has no odd cycles

Lemma 2. Let T be a BFS tree of G. Then G is bipartite if and only if there is no edge between any two nodes in the same layer.



Directed Graphs

- Definitions: directed graph, DAG, topological order

Topological Sort

- Lemma 1. If G has a topological order, then G is a DAG.
- Lemma 2. If G is a DAG, then G has a topological order.
- Proof by algorithm
  - Find a node with no incoming edges
  - Repeat…