

# Computer Science Lunch

Honors thesis???

◉ WHEN: 12:15pm, Thursday, Feb. 9      Masters???

◉ WHERE: Kendade 307

◉ TOPIC: Grad school -- is it for you? How to prepare for it?

PhD???

GREs???

1

```
BFS (G) {
  R = {s}
  layer0 = {s}
  i = 1
  while layeri-1 is not empty {
    for each node u in layeri-1 {
      while there is an edge (u,v) such that v ∉ R{
        add v to layeri and to R
      }
    }
    i = i + 1
  }
}

DFS(u) {
  mark u as "explored"
  add u to R
  for each edge (u,v) incident on u {
    if (v is not marked as "explored") {
      DFS(v);
    }
  }
}
```

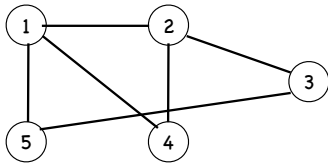
$O()$  ?

2

# Representing Graphs: Adjacency Matrix

Adjacency matrix.  $n$ -by- $n$  matrix with  $A_{uv} = 1$  if  $(u, v)$  is an edge.

- Two representations of each edge.
- How much memory?
- How long does it take to find out if a specific edge exists?
- How long does it take to find all edges incident on a node?



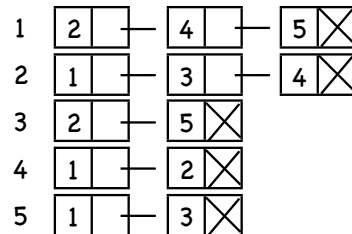
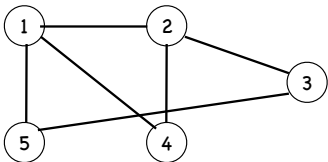
	1	2	3	4	5
1	0	1	0	1	1
2	1	0	1	1	0
3	0	1	0	0	1
4	1	1	0	0	0
5	1	0	1	0	0

3

# 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?



4

```

BFS(G) {
  R = {s}
  layer0 = {s}
  i = 1
  while layeri-1 is not empty {
    for each node u in layeri-1 {
      while there is an edge (u,v) such that v ∉ R{
        add v to layeri and to R
      }
    }
  }
  i = i + 1
}

```

## Order in which we add edges?

```

DFS(u) {
  mark u as "explored"
  add u to R
  for each edge (u,v) incident on u {
    if (v is not marked as "explored") {
      DFS(v);
    }
  }
}

```

5

## Implementing BFS

```

R = {s}
layer0 = {s}
i = 1
for each node u in layeri-1 {
  while there is an edge (u,v) such that v ∉ R{
    add v to layeri and to R
  }
  i = i + 1
}

```

- What functionality do we need?
- How should we represent the graph?
- How should we keep track of the nodes we need to visit?

6

# Runtime Analysis

```
initialize discovered to all false
add s to the queue
R = {s}
discovered[s] = true
while the queue is not empty {
  let n be the first node in the queue
  remove n from the queue
  while there are more nodes in n's adjacency list {
    let n2 be the next node in n's adjacency list
    if (!discovered[n2]) {
      add n2 to the queue
      R = R ∪ {n2}
      discovered[n2] = true
    }
  }
}
```

7

# Implementing DFS

```
DFS(u) {
  mark u as "explored"
  add u to R
  for each edge (u,v) incident on u {
    if (v is not marked as "explored") {
      DFS(v);
    }
  }
}
```

- What functionality do we need?
- How should we represent the graph?
- How should we keep track of the nodes we need to visit?

8

# Runtime Analysis

```
initialize explored to all false
push s onto the stack
while the stack is not empty {
  pop node u from the stack
  if (!explored[u]) {
    explored[u] = true
    R = R ∪ {u}
    for each edge (u,v) incident on u {
      push v onto the stack
    }
  }
}
```

9

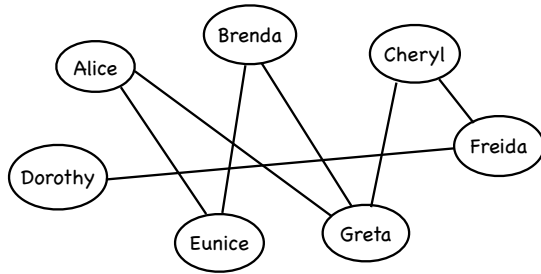
# Party Problem

- 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?

10

# The party problem

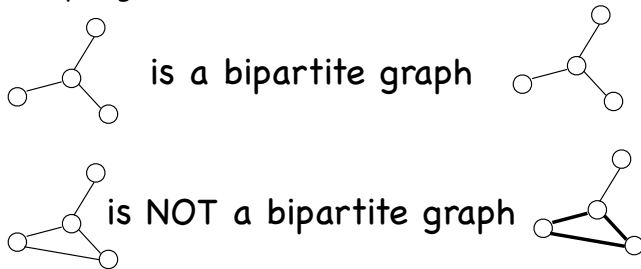
- Represent each guest as a node
- Draw an edge between guests who do not get along
- Find the largest set of nodes where there is no edge between any pair of nodes in the set



11

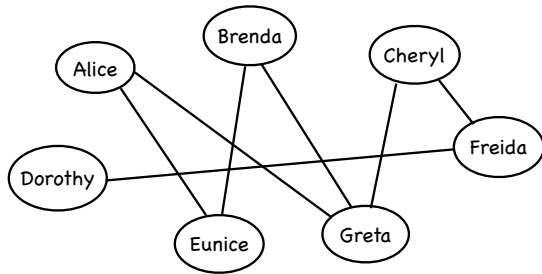
# Bipartite Graphs

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.



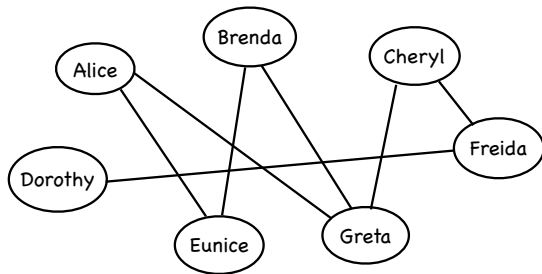
12

# A bipartite solution



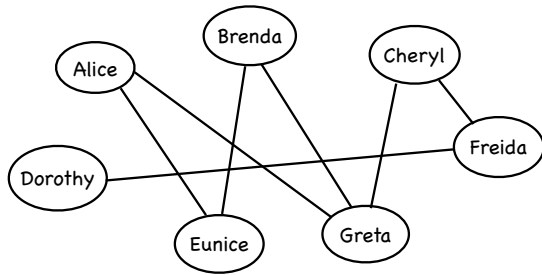
13

# A bipartite solution



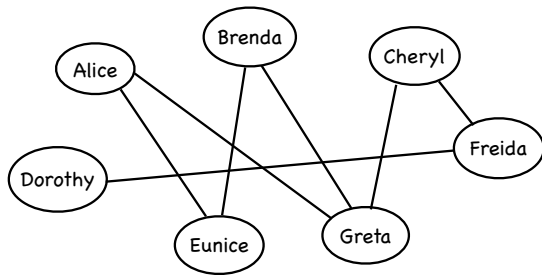
14

# A bipartite solution



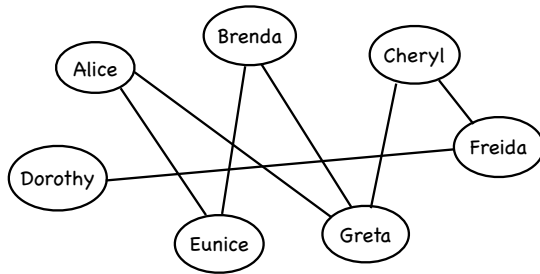
15

# A bipartite solution



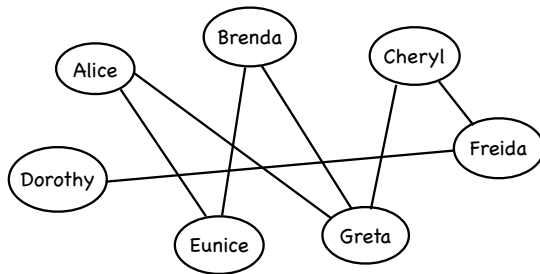
16

# A bipartite solution



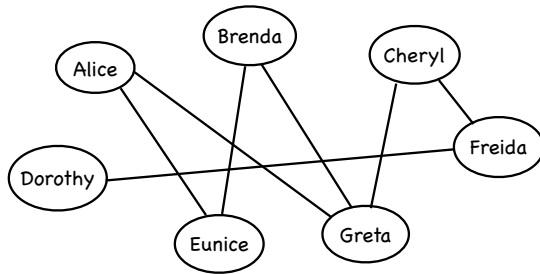
17

# A bipartite solution



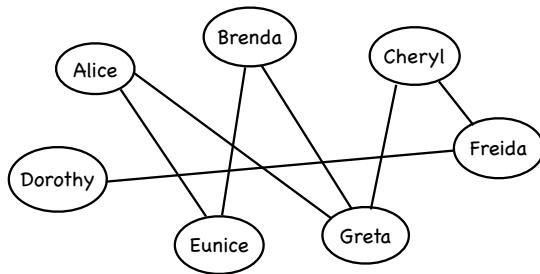
18

# A bipartite solution



19

# A bipartite solution



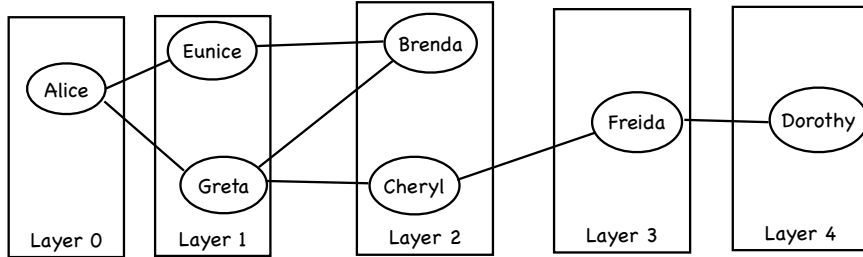
Who should you invite?

20

# BFS and Bipartite Graphs

Lemma. Let  $G$  be a connected graph, and let  $L_0, \dots, L_k$  be the layers produced by BFS starting at node  $s$ . Exactly one of the following holds.

- (i) No edge of  $G$  joins two nodes of the same layer, and  $G$  is bipartite.
- (ii) An edge of  $G$  joins two nodes of the same layer, and  $G$  contains an odd-length cycle (and hence is not bipartite).

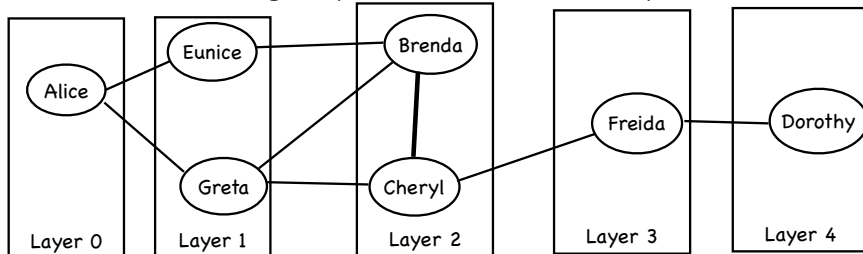


21

# BFS and Bipartite Graphs

Lemma. Let  $G$  be a connected graph, and let  $L_0, \dots, L_k$  be the layers produced by BFS starting at node  $s$ . Exactly one of the following holds.

- (i) No edge of  $G$  joins two nodes of the same layer, and  $G$  is bipartite.
- (ii) An edge of  $G$  joins two nodes of the same layer, and  $G$  contains an odd-length cycle (and hence is not bipartite).



22

## Bipartite Algorithm - Color the Graph

```
initialize discovered to all false
add s to the queue
layer[s] = 0
color s red
discovered[s] = true
while the queue is not empty {
  let n be the first node in the queue
  remove n from the queue
  while there are more nodes in n's adjacency list {
    let n2 be the next node in n's adjacency list
    if (!discovered[n2]) {
      add n2 to the queue
      layer[n2] = layer[n] + 1
      if (layer[n2] is even) color n2 red, else color n2 blue
      discovered[n2] = true
    }
  }
}
```

23