6.006 Introduction to Algorithms
Spring 2008

6.006 Recitation

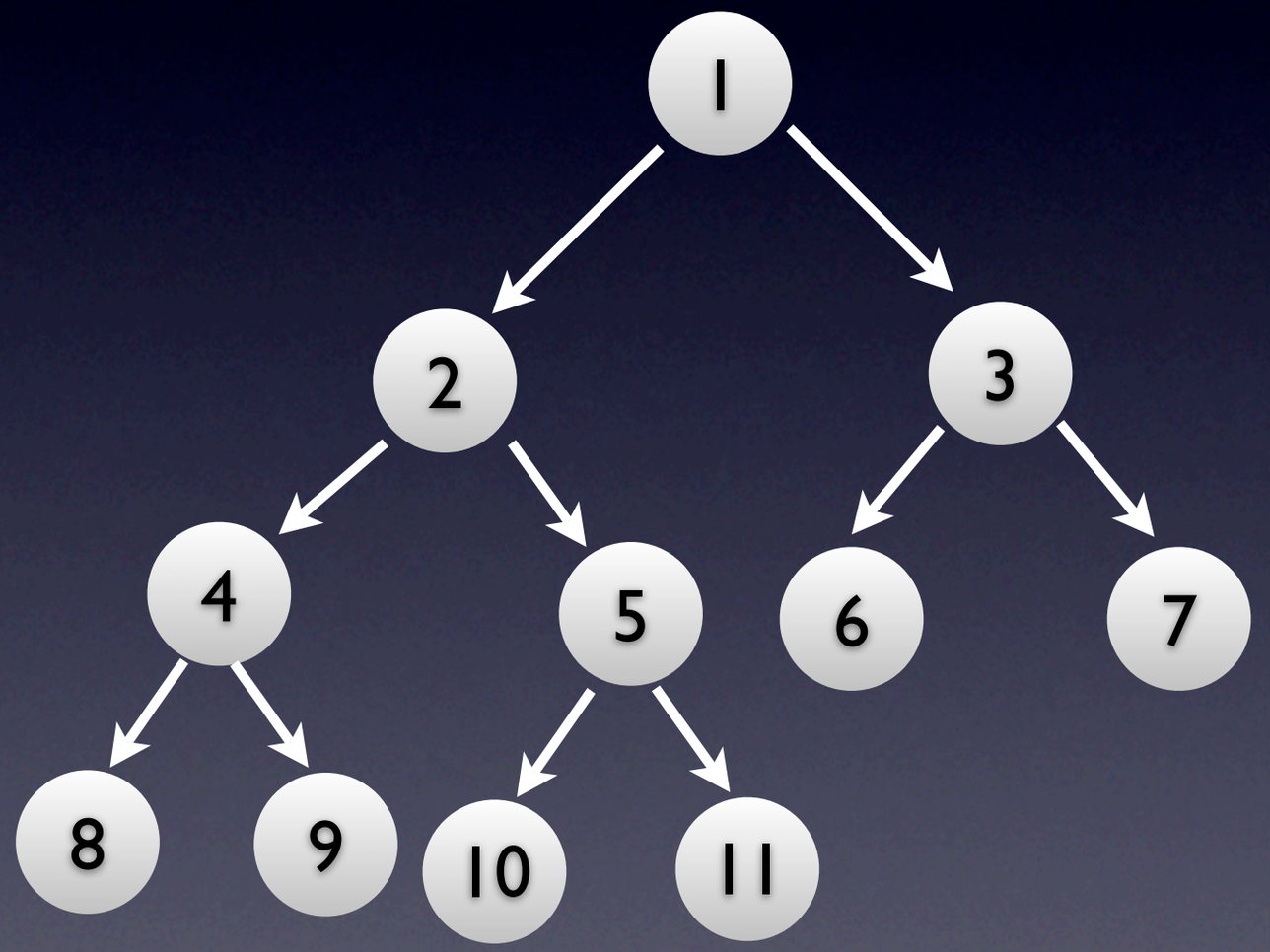
Build 2008.22

6.006 Proudly Presents

- Graph Traversal
  - BFS
  - DFS
- Topological Sorting

Breadth-First Search a.k.a. BFS (not BFG)

- Fix your source
- Visit all the neighbors
- Then visit all the neighbors' neighbors
- Then all the neighbors' neighbors' neighbors'
- ...



BFS in Python: Design

- Use the **graph** module shown before, and Python's **deque**

- Encapsulate traversal data in a class, return at the end of the traversal

- Implement traversal as stand-alone function

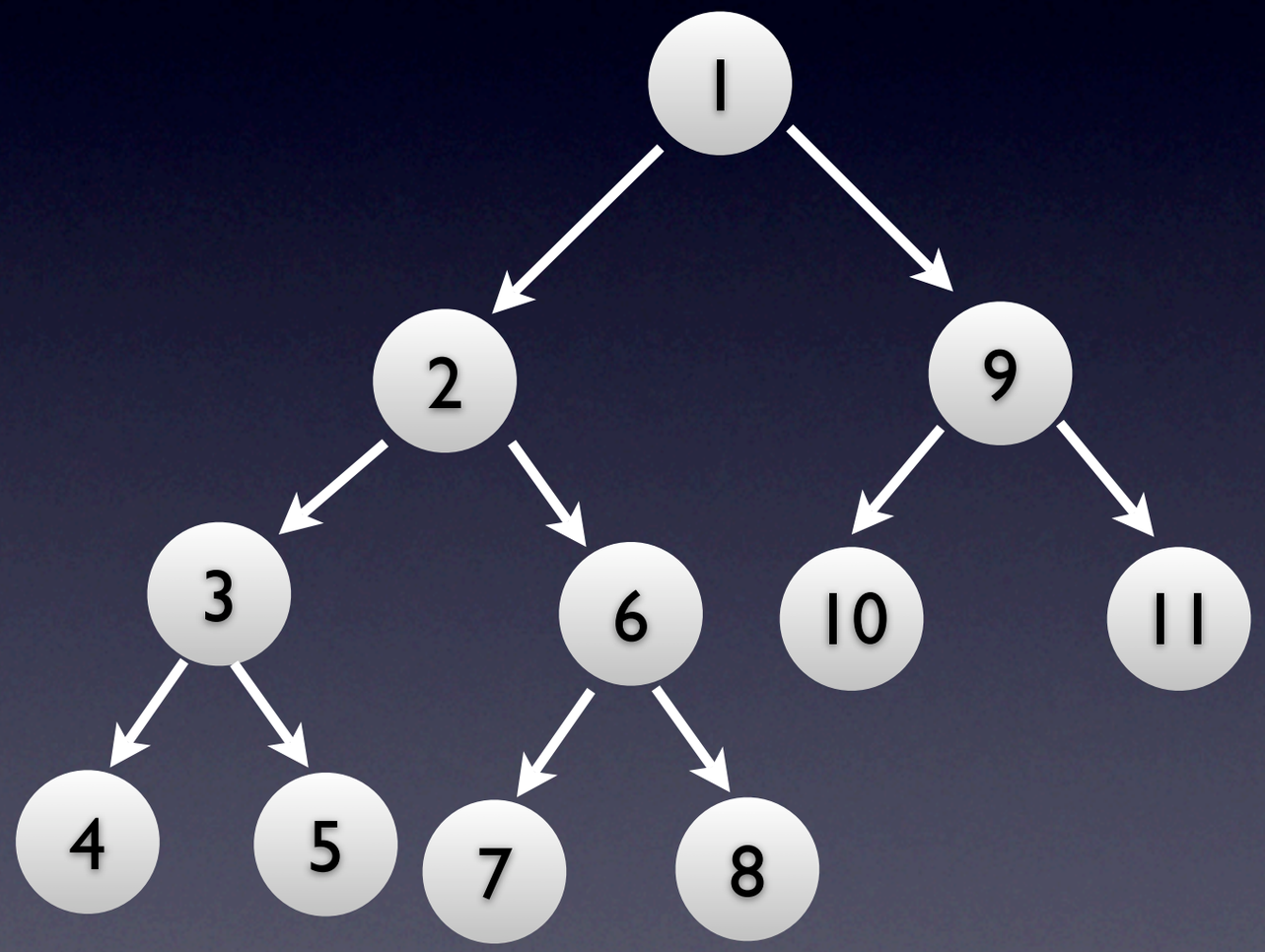
```
1 from graph import *
2 from collections import deque
3
4 class BFSResults:
5     def __init__(self):
6         self.level = dict()
7         self.parent = dict()
```

BFS in Python: Code

```
1 def bfs(g, s):
2     r = BFSResults()
3     actives = deque()
4     actives.append(s)
5     r.parent[s] = None
6     r.level[s] = 0
7
8     while len(actives):
9         v = actives.popleft()
10        for n in g.neighbors(v):
11            if n not in r.parent:
12                r.parent[n] = v
13                r.level[n] = r.level[v] + 1
14                actives.append(n)
15    return r
```

Depth-First Search a.k.a. Backtracking

- Fix your source
- Move to its first neighbor
- Then to that guy's first neighbor
- ...
- When stuck, backtrack and visit next neighbor



DFS in Python: Design

- Use the graph module shown before
- Encapsulate traversal data in a class, return at the end of the traversal
- Implement traversal as stand-alone function

```
1 from graph import *
2
3 class DFSResults:
4     def __init__(self):
5         self.parent = dict()
6         self.time = dict()
7         self.vertices = list()
8         self.t = 0
```

DFS in Python: Code

```
1 def dfs(g):
2     results = DFSResults()
3     for vertex in g.itervertices():
4         if vertex not in results.parent:
5             dfs_visit(g, vertex, results)
6     return results
7
8 def dfs_visit(g, v, results, parent = None):
9     results.vertices.append(v)
10    results.parent[v] = parent
11
12    for n in g.neighbors(v):
13        if n not in results.parent:
14            dfs_visit(g, n, results, v)
15
16    results.t += 1
17    results.time[v] = results.t
```

DFS and CLRS Colors

Color	Meaning
White (not visited)	vertex not in **parents**
Gray (visiting)	vertex in **parents** and vertex not in **time**
Black (visited)	vertex in **time**

Application:

Porting BFS and DFS to a New Platform

Disclaimers (Please Don't Sue Me!)

- You may close your eyes and cover your ears if you find this material offensive
- If you are under 13 and your mommy doesn't allow you on the Internet: please close your eyes
- Under 18: please don't use this knowledge to do something inappropriate for your age

Stalking Hotties on Facebook

- Our Platform: Firefox 3.0b4
  - any browser **with tabs** would do
- **Profiles + Friendship = Graph**
- Our mission:
  - apply DFS and BFS to the fine art of stalking hot boys/babes on Facebook

Hueihan's Heuristic

- **"Hot boys have hot friends"**

- Heuristics are useful in huge graphs, with multiple solutions
  - Goal: avoid visiting most of the graph
- So we'll only follow paths of hot* people

Facebook as Graph

- Traversal: go to ‘Friends’ to display all your friends (like g.neighbors)
- BFS: the tabs are a queue - open all friends profiles in new tabs, then close current tab and go to the next one
- DFS: the history is a stack - open the first hot friend profile in the same window; when hitting a dead end, use back button

Topological Sorting

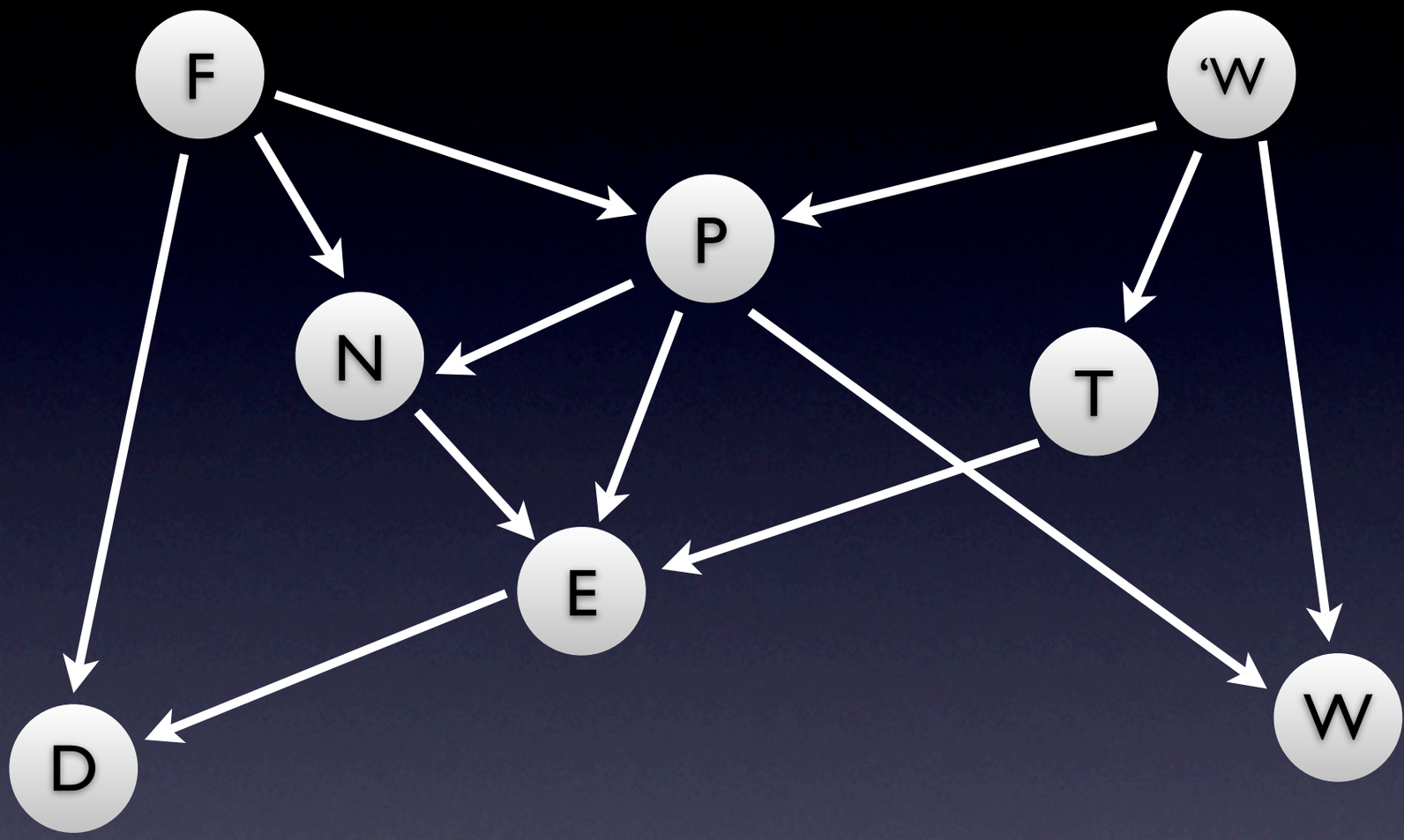
even your Course 15 friends know it

Topological Sorting

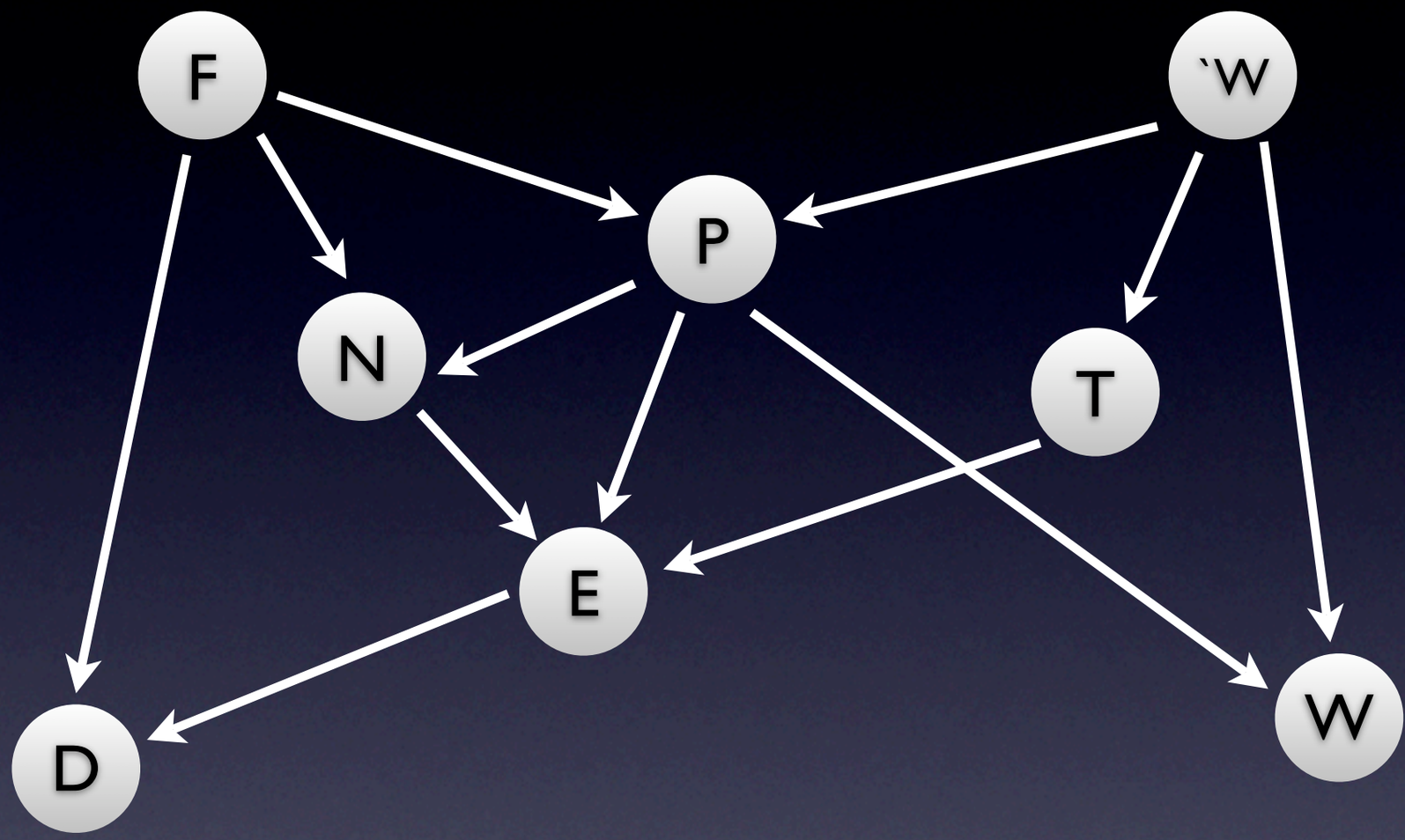
- Do a DFS on the graph, record exiting times for the nodes
- Sort the nodes in the inverse order of the exit times (just draw it!)
  - A node is never exited before a node it points to is exited

```
1 def topological_sort(graph):
2     dfs_result = dfs(graph)
3     top = [None for i in
dfs_result.vertices]
4     count = len(dfs_result.vertices)
5     for vertex in dfs_result.time:
6         top[count -
dfs_result.time[vertex]] = vertex
7     return top
```

Topological Sorting



Topological Sorting



F	D	N	E	P	W	`W	T
6	1	3	2	5	4	8	7

`W	T	F	P	W	N	E	D

Two-Way BFS

Discussion on Implementation