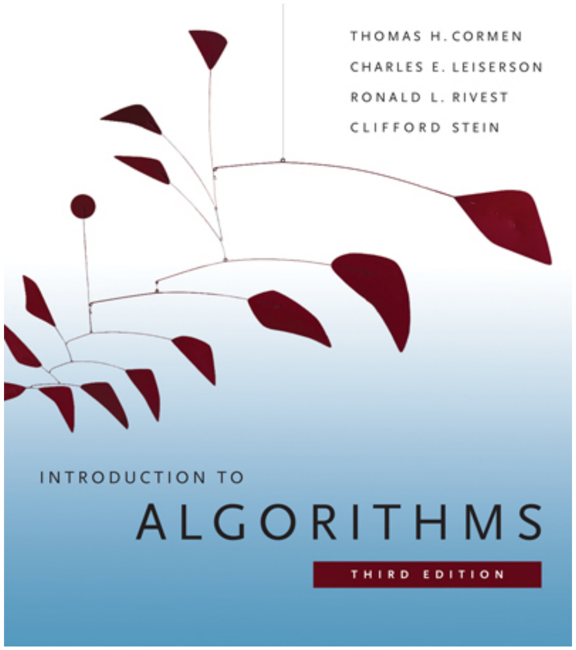
6.006- *Introduction to Algorithms*



Lecture 13

Prof. Constantinos Daskalakis

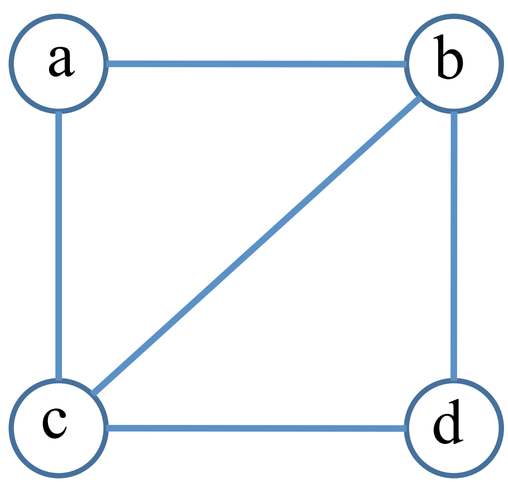
CLRS 22.4-22.5

Graphs

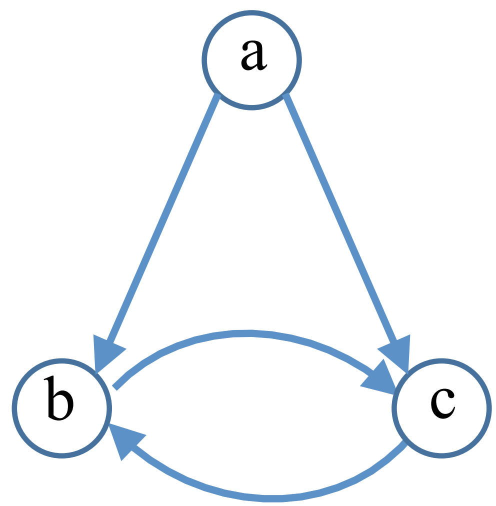
- G=(V,E)
- V a set of vertices
  - Usually number denoted by n
- $E \subseteq V \times V$ a set of edges (pairs of vertices)
  - Usually number denoted by m
- Flavors:
  - Pay attention to order of vertices in edge: ***directed*** graph
  - Ignore order: ***undirected*** graph

Examples

- ***Undirected***
- V={a,b,c,d}
- E={{a,b}, {a,c}, {b,c}, {b,d}, {c,d}}



- ***Directed***
- V = {a,b,c}
- E = {(a,c), (a,b) (b,c), (c,b)}

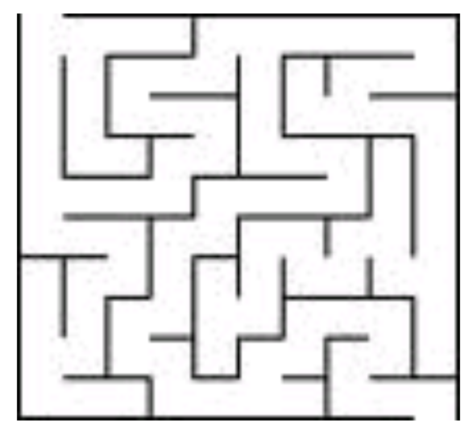


Breadth First Search

- Start with vertex v
- List all its neighbors (distance 1)
- Then all their neighbors (distance 2)
- Etc.

Depth First Search

- Exploring a maze
- From current vertex, move to another
- Until you get stuck
- Then backtrack till you find the first new possibility for exploration



BFS/DFS Algorithm Summary

- Maintain "todo list" of vertices to be scanned

- Until list is empty
  - Take a vertex v from front of list
  - Mark it scanned
  - Examine all outgoing edges (v,u)
  - If u not marked, add to the todo list
    - BFS: add to end of todo list (***queue*: FIFO**)
    - DFS: add to front of todo list (***recursion stack*: LIFO**)

Queues and Stacks

- BFS queue is explicit
  - Created in pieces
  - (level 0 vertices) . (level 1 vertices) . (level 2 vert…
  - the frontier at ***iteration i*** is ***piece i*** of vertices in queue
- DFS stack is implicit
  - It's the call stack of the python interpreter
  - From v, recurse on one child at a time
  - But same order if put all children on stack, then pull off (and recurse) one at a time

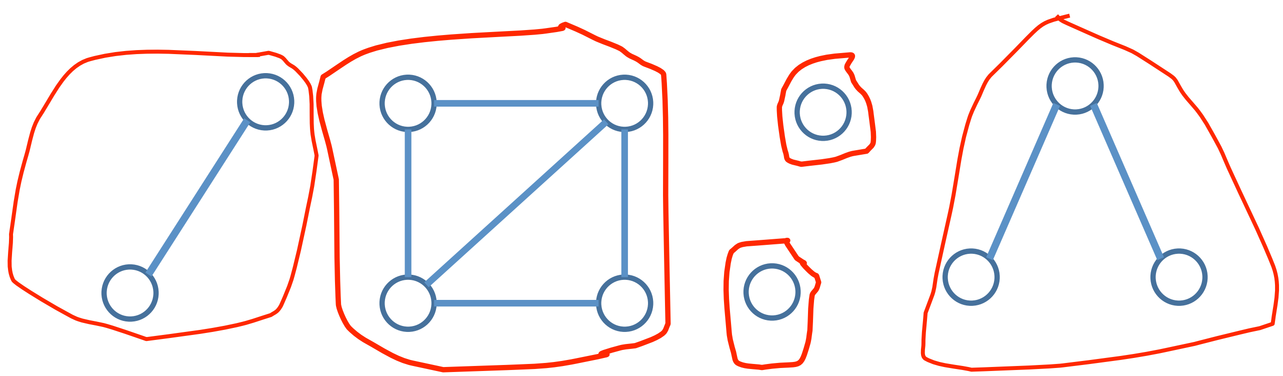
Runtime Summary

- Each vertex scanned once
  - When scanned, marked
  - If marked, not (re)added to todo list
  - Constant work per vertex
    - Removing from queue
    - Marking
  - O(n) total
- Each edge scanned once
  - When tail vertex of edge is scanned
  - Constant work per edge (checking mark on head)
  - O(m) total
- In all, O(n+m)

Connected Components

Connected Components

- Undirected graph G=(V,E)
- Two vertices are connected if there is a path between them
- An equivalence relation
- Equivalence classes are called components
  - A set of vertices all connected to each other



Algorithm

- DFS/BFS reaches all vertices reachable from starting vertex s
- i.e., component of s
- Mark all those vertices as “owned by” s

Algorithm

- DFS-visit (u, ***owner***, o)

```
#mark all nodes reachable from u with owner o
for v in Adj[u]
    if v not in owner          #not yet seen
        owner[v] = o           #instead of parent
        DFS-visit (v, owner, o)
```

- DFS-Visit(s, owner, s) will mark owner[v]=s for any vertex reachable from s

Algorithm

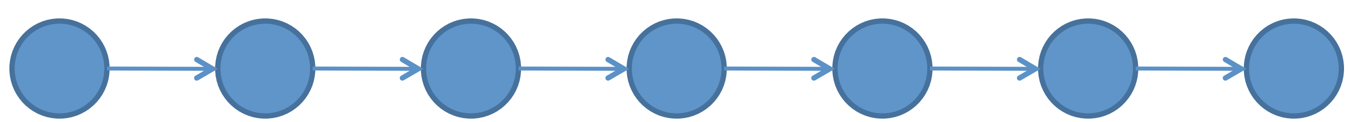
- Find component for s by DFS from s
- So, just search from every vertex to find all components
- Vertices in same component will receive the same ownership labels
- Cost?
  - n times BFS/DFS?
  - ie, O(n(m+n))?

Better Algorithm

- If vertex has already been reached, don't need to search from it!
  - Its connected component already marked with owner
- ***owner*** = {}
  for s in V
      if not(s in ***owner***)
          DFS-Visit(s, ***owner***, s)   #or can use BFS
- Now every vertex examined exactly twice
  - Once in outer loop and once in DFS-Visit
- And every edge examined once
  - In DFS-Visit when its tail vertex is examined
- Total runtime to find components is O(m+n)

Directed Graphs

- In undirected graphs, connected components can be represented in n space
  - One "owner label" per vertex
- Can ask to compute all vertices reachable from each vertex in a directed graph
  - i.e. the "transitive closure" of the graph
  - Answer can be different for each vertex
  - Explicit representation may be bigger than graph
  - E.g. size n graph with size n^2 transitive closure



Topological Sort

Job Scheduling

- Given
  - A set of tasks
  - Precedence constraints
    - saying “u must be done before v”
  - Represented as a directed graph
- Goal:
  - Find an ordering of the tasks that satisfies all precedence constraints

Make bus in seconds flat

Fall out of bed

Drag a comb across my head

Look up (at clock)

Find my coat

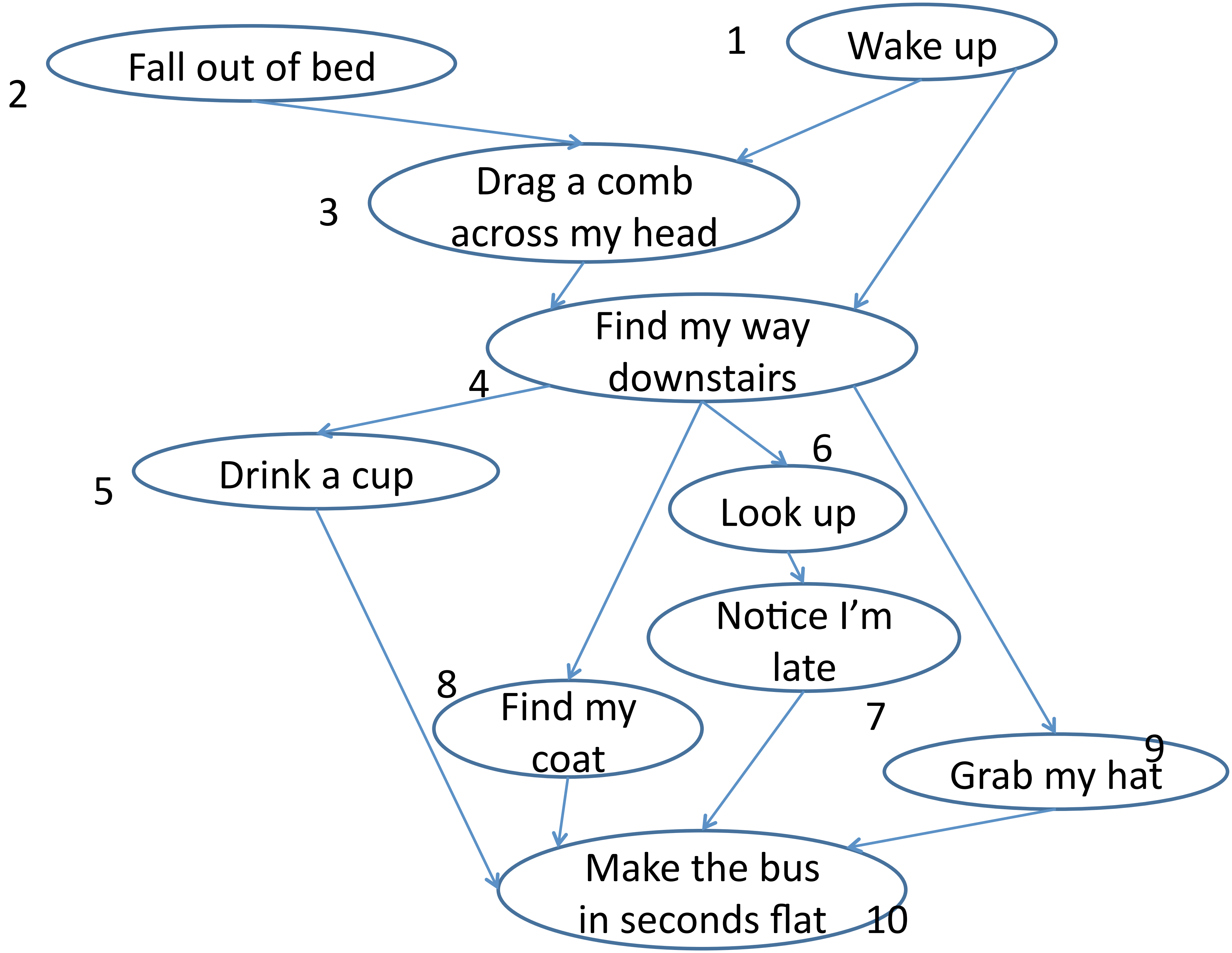
Notice that I'm late

Drink a cup

Wake up

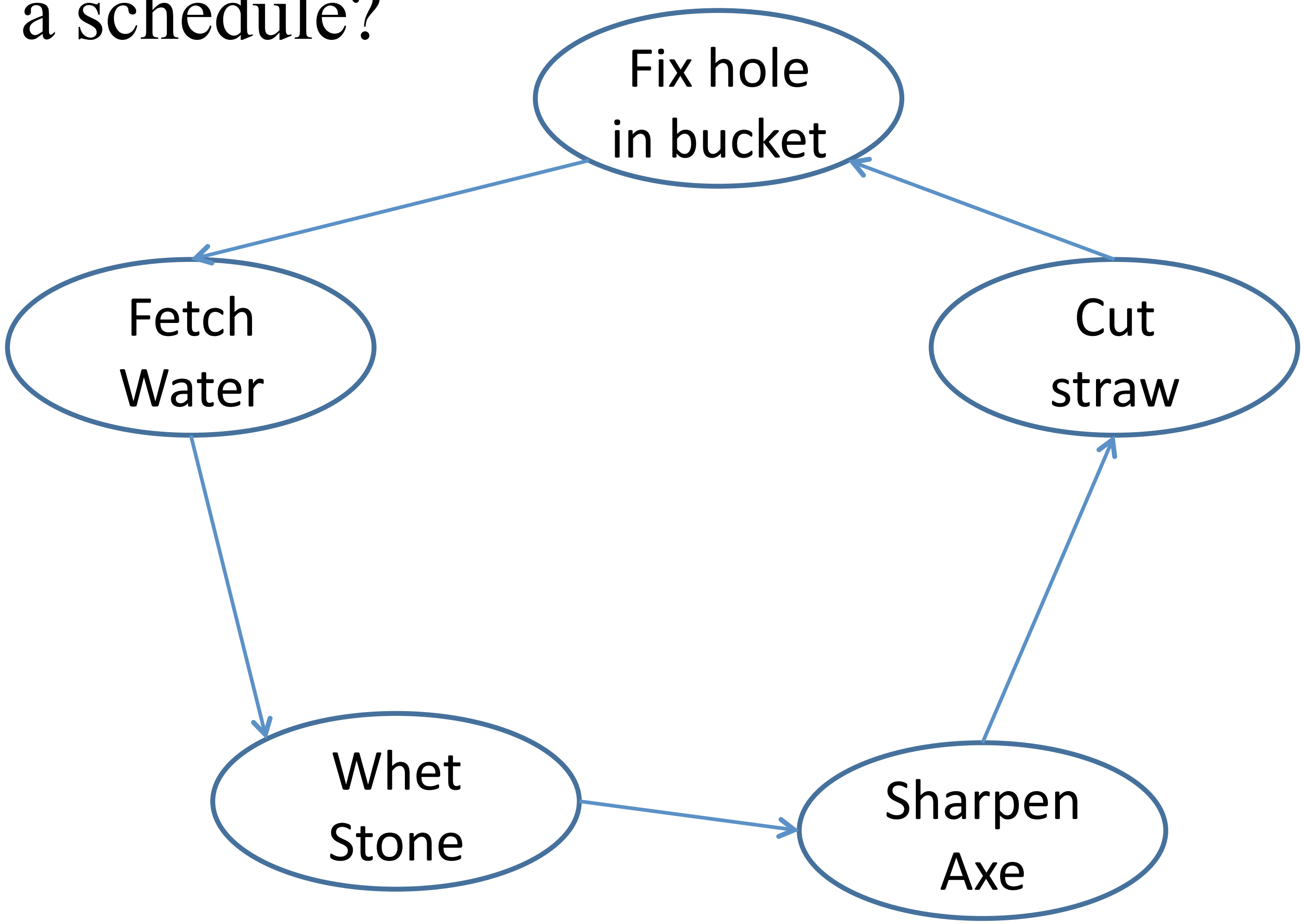
Find my way downstairs

Grab my hat


1
Wake up
2
Fall out of bed
3
Drag a comb across my head
4
Find my way downstairs
5
Drink a cup
6
Look up
Notice I'm late
7
8
Find my coat
9
Grab my hat
Make the bus in seconds flat
10

Existence

- Is there a schedule?



DAG

- Directed Acyclic Graph
  - Graph with no cycles
- Source: vertex with no incoming edges
- Claim: every DAG has a source
  - Start anywhere, follow edges backwards
  - If never get stuck, must repeat vertex
  - So, get stuck at a source
- Conclude: every DAG has a schedule
  - Find a source, it can go first
  - Remove, schedule rest of work recursively

Algorithm I (for DAGs)

- Find a source
  - Scan vertices to find one with no incoming edges
  - Or use DFS on backwards graph
- Remove, recurse
- Time to find one source
  - O(m) with standard adjacency list representation
  - Scan all edges, count occurrence of every vertex as tail
- Total: O(nm)

Algorithm 2 (for DAGs)

- Consider DFS
- Observe that we don't return from recursive call to DFS(v) until all of v's children are finished
- So, "finish time" of v is later than finish time of all children
- Thus, later than finish time of all descendants
  - i.e., vertices reachable from v
  - Descendants well-defined since no cycles
- So, reverse of finish times is valid schedule

Implementation (of Alg 2)

- ***seen*** = {}; ***finishes*** = {}; ***time*** = 0

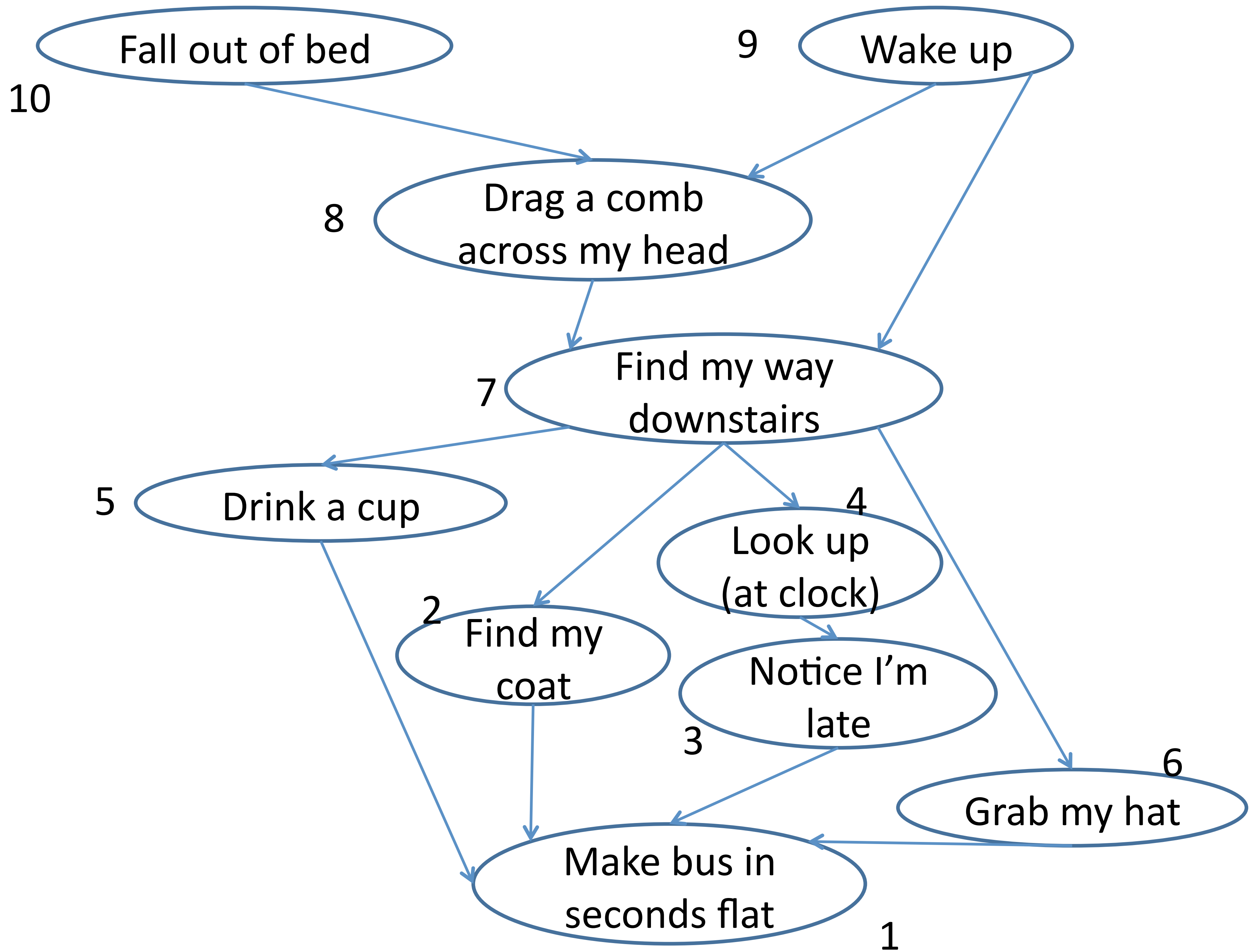
```
DFS-visit (s)
    for v in Adj[s]
        if v not in seen
            seen[v] = 1
            DFS-visit (v)
            time = time+1
            finishes[v] = time
```

*only set **finishes** if done processing all edges leaving v*

- TopologicalSort

```
for s in V
    DFS-visit(s)
```

- Sort vertices by ***finishes***[] key


Fall out of bed
10
9
Wake up
8
Drag a comb across my head
7
Find my way downstairs
5
Drink a cup
4
Look up (at clock)
2
Find my coat
Notice I'm late
3
6
Grab my hat
Make bus in seconds flat
1

Analysis

- Just like connected components DFS
  - Time to DFS-Visit from all vertices is O(m+n)
  - Because we do nothing with already seen vertices
- Might DFS-visit a vertex v before its ancestor u
  - i.e., start in middle of graph
  - Does this matter?
  - No, because finish[v] < finish[u] in that case

Handling Cycles

- If two jobs can reach each other, we must do them at same time
- Two vertices are strongly connected if each can reach the other
- Strongly connected is an equivalence relation
  - So graph has strongly connected components
- Can we find them?
  - Yes, another nice application of DFS
  - But tricky (see CLRS)
  - You should understand algorithm, not proof