CS5800 – ALGORITHMS

MODULE 5. DATA STRUCTURES & GRAPHS

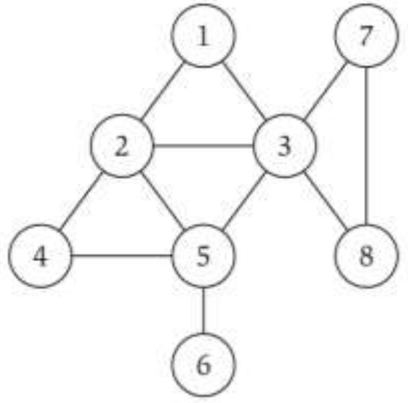
Lesson 2: Graph Traversals

Ravi Sundaram

Topics

- Why graph traversals?
- Breadth-First-Search
  - Shortest Path Tree
- Depth-First-Search
  - Combinatorics of long paths
- Connected component
- Unified approach to BFS & DFS
- Summary

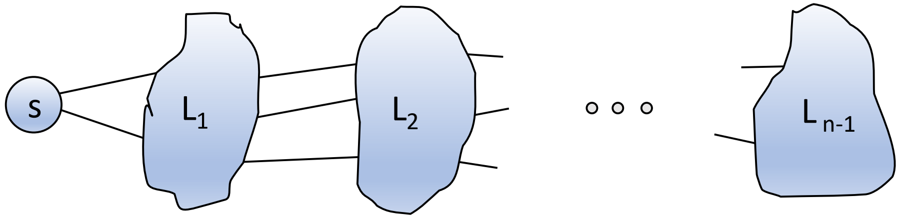
Why graph traversals?



- Variety of applications in exploring and connectivity
  - Facebook/LinkedIn
  - Maze traversal.
  - Kevin Bacon number.
  - Fewest number of hops in a communication network
  - s-t connectivity: given nodes s and t, is there a path between s and t?
  - s-t shortest path: Given nodes s and t, what is the length of the shortest path between s and t?
  - Planarity: can a given graph be drawn in the plane with no crossing edges?

Breadth First Search (BFS)

- BFS intuition: Explore outward from s, adding nodes a "layer" at a time.

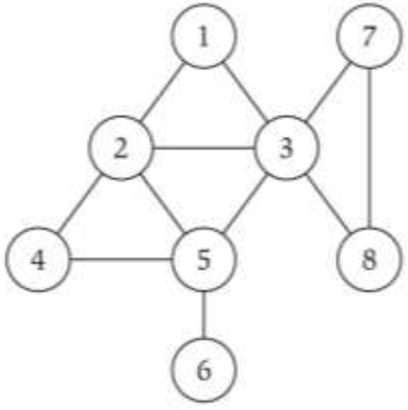


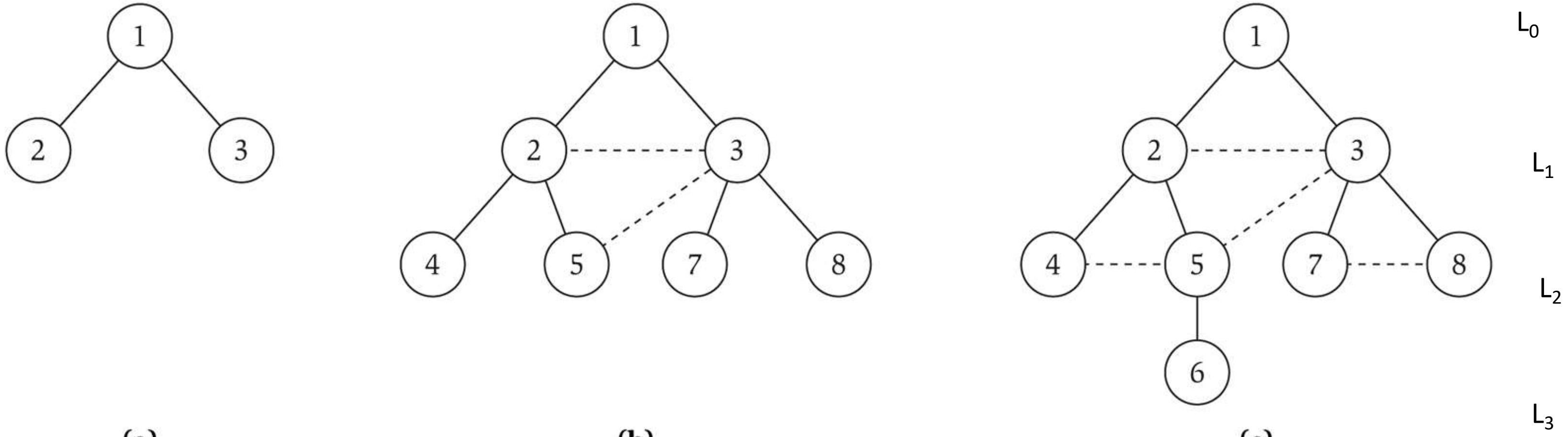
- BFS algorithm.
  - $L_0 = \{ s \}$.
  - L_1 = all neighbors of L_0.
  - L_2 = all nodes that do not belong to L_0 or L_1, and that have an edge to a node in L_1.
  - …
  - L_{i+1} = all nodes that do not belong to an earlier layer, and that have an edge to a node in L_i.

- For each i, L_i consists of all nodes at distance exactly i from s.

BFS Tree

- The BFS tree is grown by adding an edge along which a node is encountered for the first time.
- Orienting edges towards parents we see it forms a tree.
- No edge in the graph can skip a level





(a) (b) (c)

BFS: Shortest Path Tree & Analysis

- Theorem: The BFS tree is a Shortest Path Tree (SPT)
- Pf: By induction on levels. Use the induction hypothesis that the BFS tree upto level i is the SPT on all nodes at most distance i from the root ▪

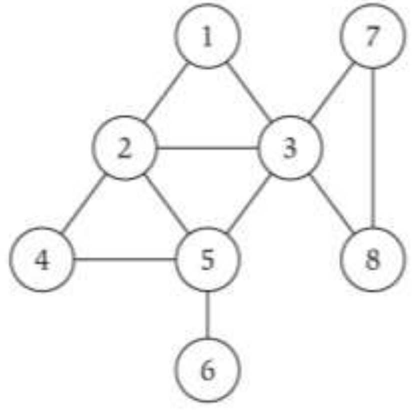
- Theorem: Given an adjacency list representation of the graph BFS can be implemented to run in O(m + n) time.
- Pf: when we consider node u, there are deg(u) incident edges (u, v) Total time processing edges is $\Sigma_{u \in V}$ deg(u) = 2m ▪

Depth First Search (DFS)

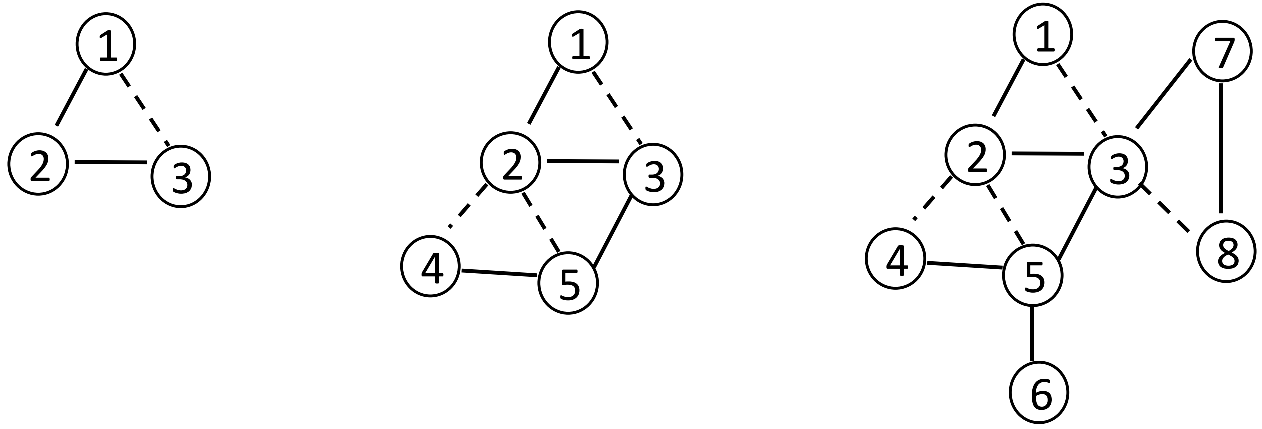
- DFS intuition: Explore in one direction as far as possible before backtracking.
- DFS algorithm – recursive implementation
- <u>DFS(v)</u>

```
Mark v
For each neighbor w of v
    If w is unmarked
    Then DFS(w)
         Add (v, w) to Tree
```

DFS Tree



- The DFS tree grows in long and skinny fashion, compared to the short and bushy BFS tree
- Orienting edges towards parents we see it forms a tree.
- All edges run between ancestor and descendant



(a) (b) (c)

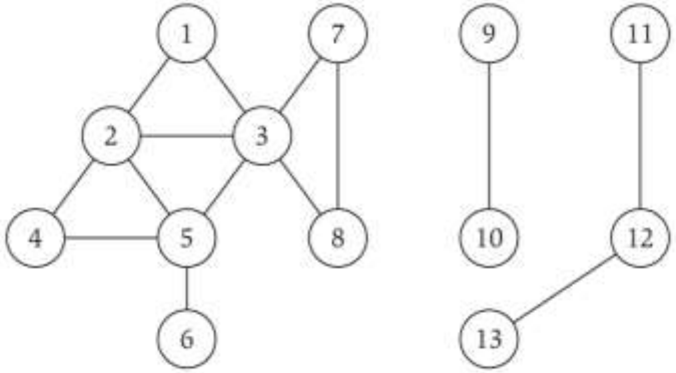
DFS: Combinatorics & Analysis

- Theorem: Any graph on n vertices without a path of length k has O(kn) edges
- Pf: If the graph does not have a path of length k then the height of the DFS tree is at most k. Which means every node has at most k ancestors. But every edge runs between ancestor and descendant and hence the graph has at most kn/2 = O(kn) edges. ▪

- Theorem: Given an adjacency list representation of the graph DFS can be implemented to run in O(m + n) time.
- Pf: when we consider node u, there are deg(u) incident edges (u, v) Total time processing edges is $\Sigma_{u \in V}$ deg(u) = 2m ▪

Connected Component

- Connected component. Find all nodes reachable from s.

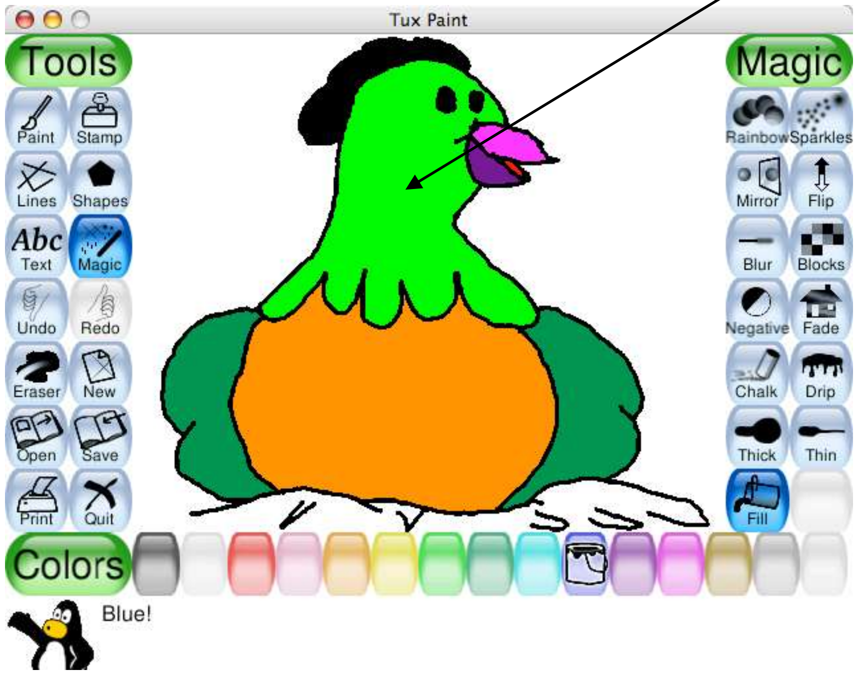


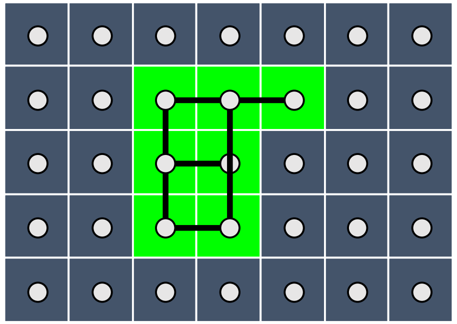
- Connected component containing node 1 = { 1, 2, 3, 4, 5, 6, 7, 8 }.

Flood Fill

- Flood fill. Given lime green pixel in an image, change color of entire blob of neighboring lime pixels to blue.
  - Node: pixel.
  - Edge: two neighboring lime pixels.
  - Blob: connected component of lime pixels.

recolor lime green blob to blue

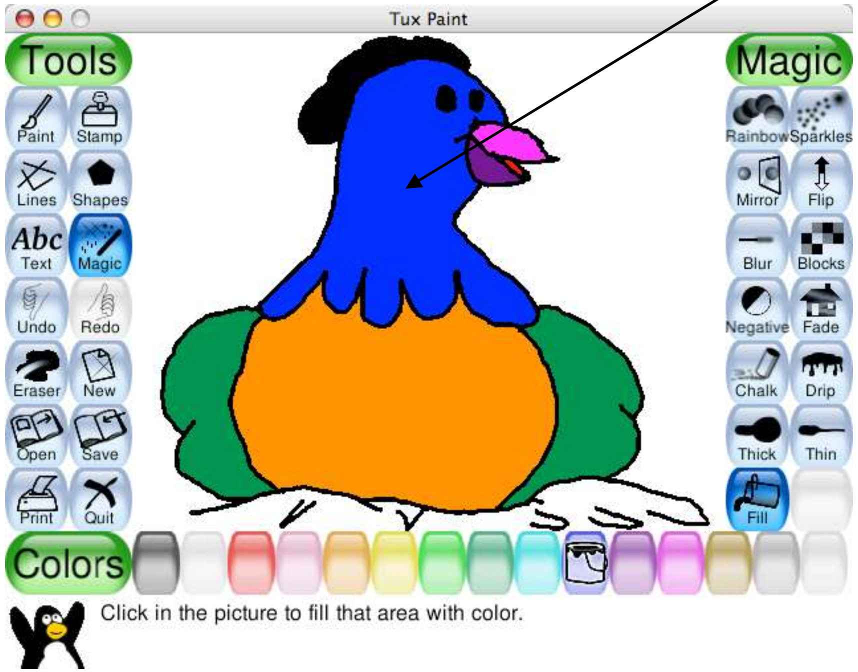


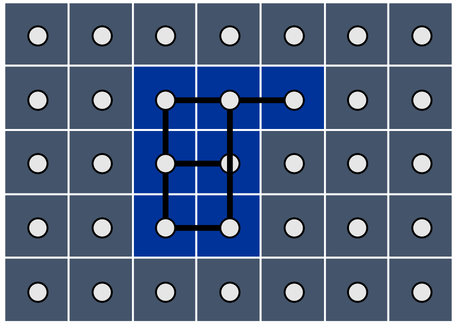


Flood Fill

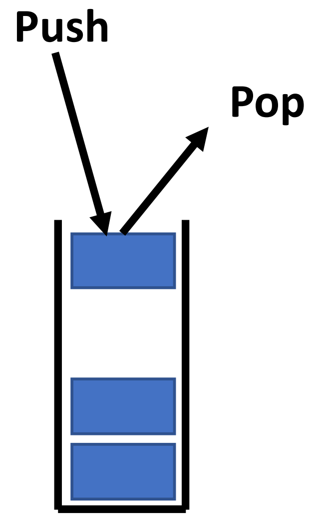
- Flood fill. Given lime green pixel in an image, change color of entire blob of neighboring lime pixels to blue.
  - Node: pixel.
  - Edge: two neighboring lime pixels.
  - Blob: connected component of lime pixels.

recolor lime green blob to blue

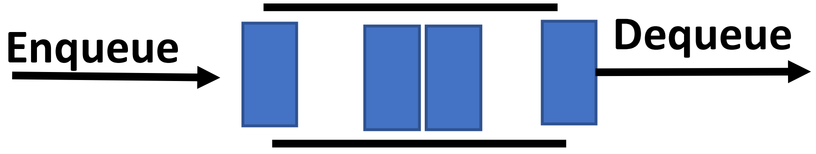




Unified iterative approach to DFS and BFS



- Use Data-Structure = Queue for BFS, Stack for DFS




```
Add (r, ϕ = parent(r)) to Data-Structure
While not empty Data-Structure
    Remove (v, u)
    If not marked v
    then add (v, u = parent(v)) to Tree
         Mark v
         For each neighbor w of v
             Add (w, v) to Data-Structure
```

Summary

- Graph traversals – several applications including exploration and connectivity
- Many kinds of traversals of which two are used frequently – BFS and DFS
  - BFS – generates a bushy tree, a shortest path tree
  - DFS – generates a narrow spindly tree
  - Both can be implemented in linear time, using queue for BFS and stack for DFS
- Main Takeaway: Graphs can be traversed in linear time to solve connectivity problems