# CSOR4231: Analysis of Algorithms

Fall 2019

Alex Andoni

Lecture 14, 10/22/19

# Last time

- Greedy algorithms
  - Huffman Coding
  - Fractional Knapsack

# Today

- Graphs
  - Defs
  - Basic algorithms: BFS, DFS

# Graphs

- Graph $G = (N, E)$
- set $N$ of vertices (nodes) – represents set of objects
- set $E$ of edges – represents relation between objects
- undirected graph: edge $(u, v)$ same as $(v, u)$
- directed graph: edge $(u, v)$ not same as $(v, u)$

- Fundamental model!

# Modeling by Graphs

- Physical connections
  - communication networks: nodes=switches, computers
  - electric circuits: nodes=gates, edges=wires
  - Chemistry: nodes=molecules, edges=bonds
  - Neural networks: nodes=neurons, edges=synapses
  - geographical maps: nodes=cities, edges= roads, flights, railroads
  - City maps:  nodes=intersections, edges=streets

  …

# Modeling by Graphs

- Logical connections & relations
  - social networks: nodes=people, edges=connections
  - data structures:  nodes and pointers
  - entity relationship diagrams in data modeling
  - dependency relation
  - control/data flow graphs
  - message flow graphs
  - AI: puzzles, mazes
  - Games: nodes=positions, edges=moves

- Structure of other models in mathematics, CS
  - Graphical models,
  - finite automata, state machines
  - Markov chains,
  - partial orders, lattices, …

# Graphs : Terminology

- $G(N, E)$ , Undirected or Directed
- $N$: nodes (or vertices)
- $E$: edges (or arcs for directed)
- Main size parameters: $n = |N|, e = |E|$
  for simple graphs $e$ is between 0 and $n^2$

- Degree of nodes
- Paths, simple paths, cycles
- Forest, Tree : undirected acyclic graphs
- DAG : directed acyclic graph

- Sometimes, weighted graph: $w$: $E \rightarrow R$

- Review Appendix B

# Some Basic Problems

- **Reachability:** Which nodes can reach which other nodes

  - given $s, t$, is there path from $s$ to $t$; find such path

  - given $s$, which nodes can it reach?

- Graph Structure (Properties):
  - Given G, is it connected? Biconnected?
  - Connected components, Cycles?
  - Planar? Bipartite? Eulerian? Hamiltonian?
  - Graph isomorphism…

- Optimization problems:
  - Shortest Paths between nodes, Longest Path
  - Minimum Spanning Tree
  - Maximum Flow, Minimum Cut, …

# Graph Representation : Nodes

- Nodes: Integers $1, \ldots, n$, so can index them (build arrays)
- Generally, nodes belong to some domain $D$ (eg. strings).
- Use a dictionary (symbol table) eg. hash map, trie, BST, etc, to map between $D$ and $[n]$, both ways.

- Example:

EWR 1
JFK 2
LGA 3
...

# Graph Representation : Edges

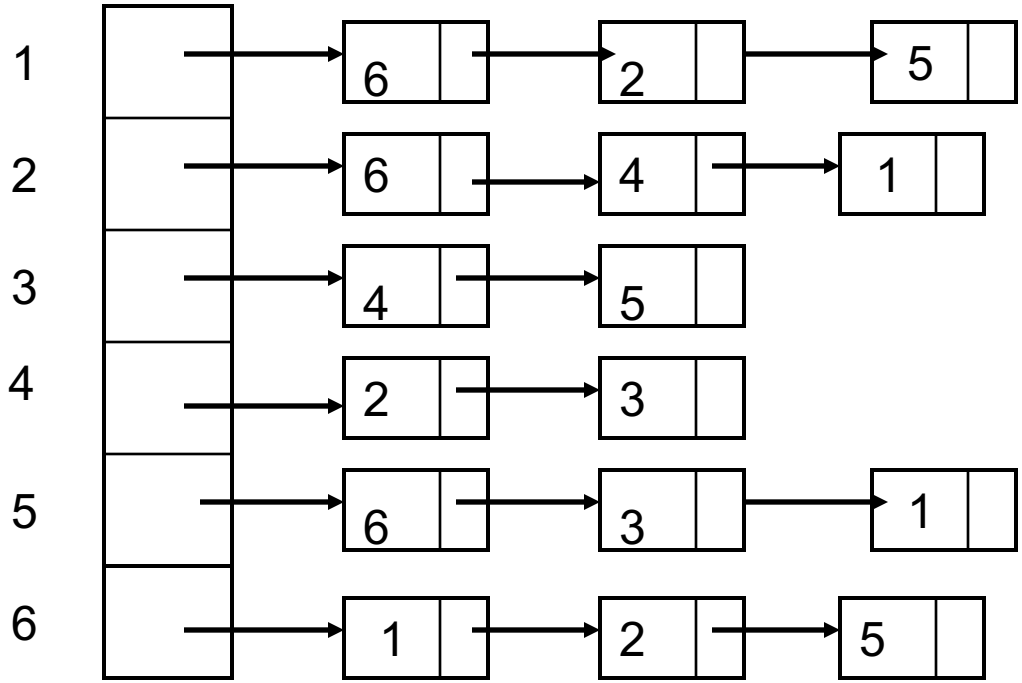
- #1: List of edges, i.e. pairs $(i, j)$
- Space = $O(e)$

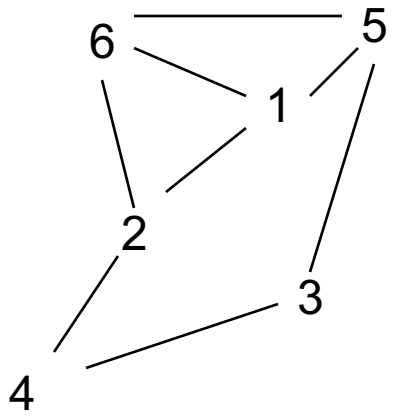
- #2: Adjacency matrix indexed by nodes: 2D boolean array A[i,j] is 1 if edge (i,j) is in E, 0 otherwise
- Undirected graph: matrix is symmetric A[i,j] = A[j,i]
- Directed graph: A maybe asymmetric

- Space=$O(n^2)$
- Good for dense graphs: #edges close to (# nodes)²

# #3: Adjacency list representation

- Array of lists =adjacency lists of nodes




Space proportional to $n + e$

# Adjacency list representation (2)

- Undirected Graphs:
- Every edge $(i, j)$ leads to two entries:

  $j$ in adj[$i$] and $i$ in adj[$j$]

- Sometimes it is helpful to connect the two entries so that we can access easily one from the other. That is, node in adj list contains in addition a link to the other node
- Also, sometimes may use doubly linked lists

- Directed Graphs: every edge appears once

# Nonuniqueness of representation

- A graph on node set N={1,...,n} has one adjacency matrix representation but many different adjacency list representations (lists with different orderings)
  - Can determine if two adj list representations represent same graph in O(n+e) time
    - just sort each list
- A graph with vertex names from a general domain has many representations depending how vertex names are mapped to [n]={1,...,n}
- Graph isomorphism problem: Determine whether two representations are isomorphic, i.e. same graph except for the vertex mapping to [n]
  - Is there vertex permutation that preserves the edges?
  - Much harder problem. Complexity is open!

# Comparison between representations

| Rep.\Task | Space | Edge (i,j) ? | List edges(i) |
|---|---|---|---|
| List of edges | e | e | e |
| Adj matrix | $n^2$ | 1 | n |
| Adj lists | n+e | deg(i) | deg(i) |

- Often in practice, sparse graphs -> use Adj lists

- Can augment Adj lists with a dictionary of edges to answer quickly edge queries with space O(e)
  - eg. hash table: (i,j)-> hash value

# Simple Problems

- Assume Adj lists representation
- Print all edges

  For Directed Graphs:

  **for** $i = 1$ **to** $n$ { **foreach** $j$ in $Adj[i]$ print $(i, j)$ }

  For Undirected Graphs:

  **for** $i = 1$ **to** $n$ { **foreach** $j$ in $Adj[i]$: **if** $(i < j)$ print $(i, j)$ }

  *// so that every edge is printed only once*
- Construct Adjacency matrix
  - Initialize matrix to 0
  - Traverse every Adj list and change entry to 1
- Construct reverse of a directed graph
- Compute degree of nodes
- …..

# Graph Searching:
# Single Source Reachability

Input: directed (undirected) graph $G = (N, E)$, "source" node $s$

Which nodes are reachable from node $s$ ?

SEARCH($G, s$)

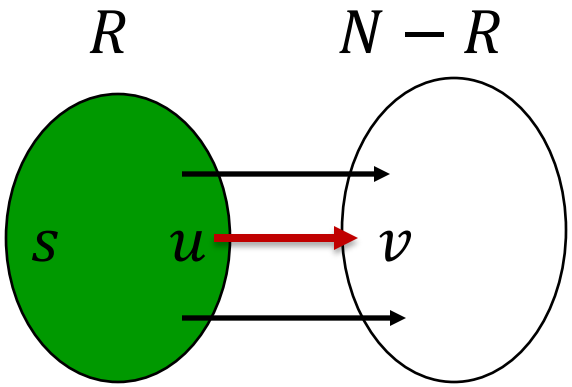
$R = \{s\}$

**while** $\exists$ edge from $R$ to $N - R$

{ $(u, v)$=such an edge

$R = R \cup \{v\}$

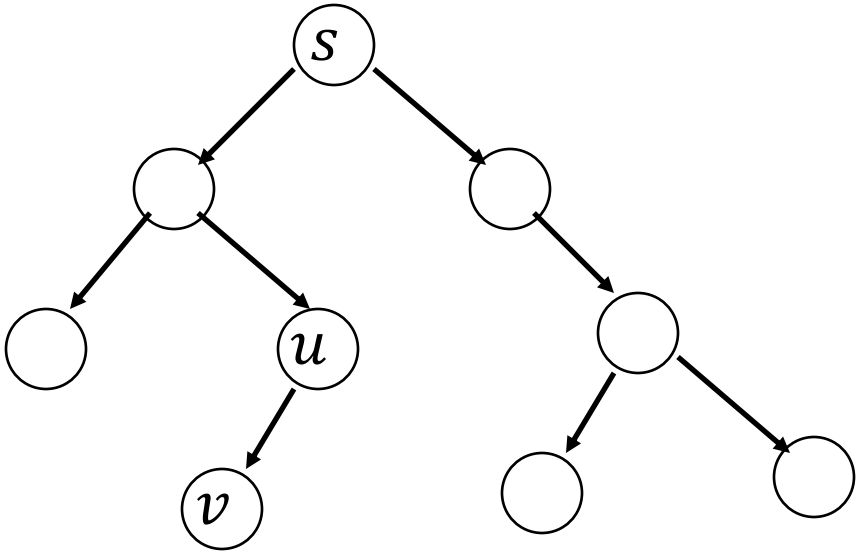
}

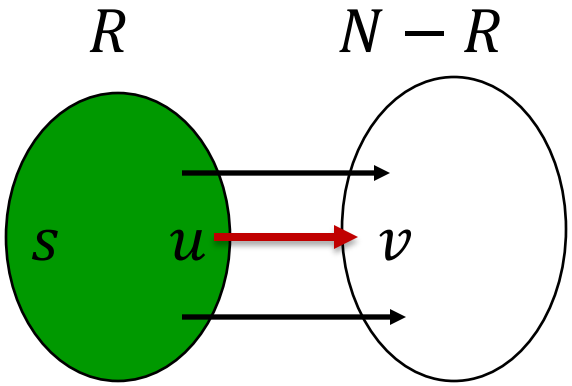



At the end, *R* = set of reachable nodes

# SEARCH => Reachability/Search Tree

- Rooted tree with root $s$
- Includes all nodes of $R$
- parent $p[v]$ = node $u$ that added $v$ to $R$



# Correctness proof

Theorem: At the end, $R$ = set of nodes that are reachable from $s$

Proof:

- $v \in R \Rightarrow v$ reachable

  in particular, exists $s \rightarrow v$ path in Search tree

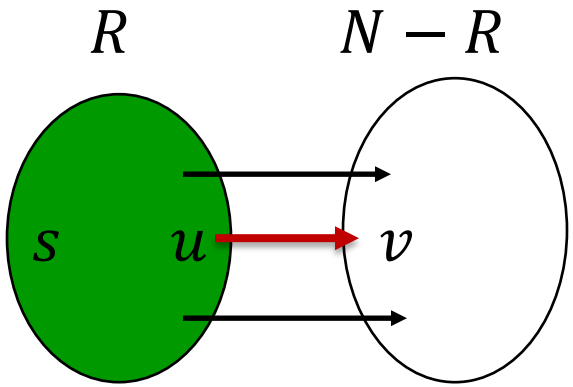
- $v$ reachable $\Rightarrow v \in R$

  by induction on length of $s \rightarrow v$ path

  IH($i$): all nodes at distance $\leq i$ are in $R$

# Search Strategies

- In general, many $(u, v)$ edges from $R$ to $N - R$




- Which to choose?

- Different policies…
- Different algorithms useful in different contexts:
  - Breadth-First Search (BFS) → BFS tree
  - Depth-First Search (DFS) → DFS tree
  - Dijkstra's algorithm → shortest path tree
  - Prim's algorithm → minimum spanning tree
  - …

# Breadth First Search

Policy: $(u, v)$ where $u$ is earliest node added to $R$ (F-I-F-O)

Queue $Q$ keeps reached, unprocessed nodes

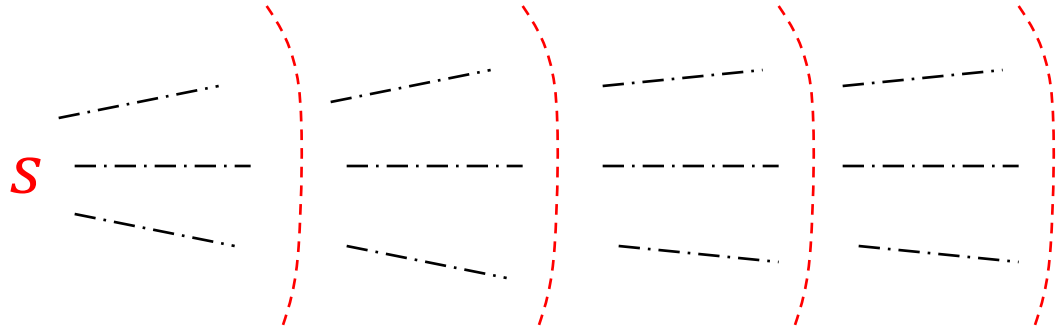
- white: $\in N - R$: not reached
- gray: $R \cap Q$: reached, active (did not explore its neighbors)
- black: $R \setminus Q$: done

Search = Wave, out of $s$ traveling along the graph's edges

$d[v]$: time the wave reaches node $v$

= distance from the source $s$

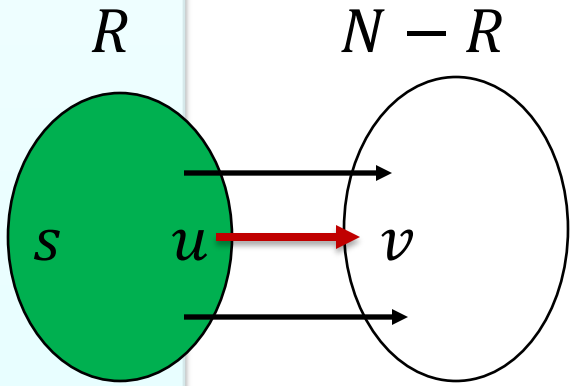
= length of shortest path from $s$ to $v$



# Breadth First Search: Algorithm

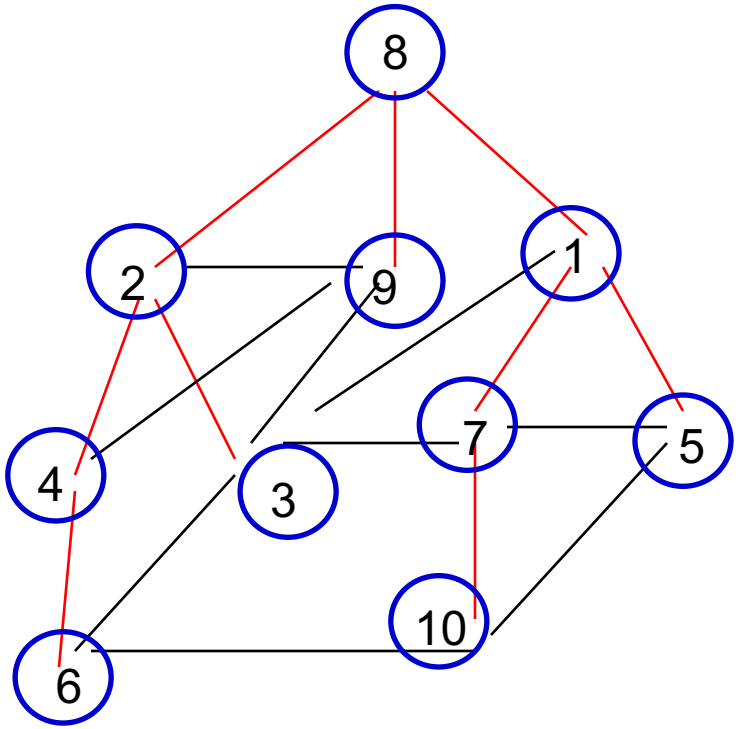
BFS$(G, s)$

**for each** $v \in N \setminus \{s\}$ **do** $\{d[v] = \infty;\ p[v] = \perp\}$
$d[s] = 0$
$p[s] = \perp$
$Q = \{s\}$
**while** $Q \neq \emptyset$ **do**
  { $u$=Dequeue$(Q)$
    **for each** $v \in Adj[u]$ **do**
      **if** $d[v] = \infty$ **then**
        { $d[v] = d[u] + 1;\ p[v] = u;$
          Enqueue$(Q, v)$}
  }




Time Complexity: $O(n + e)$

# Example



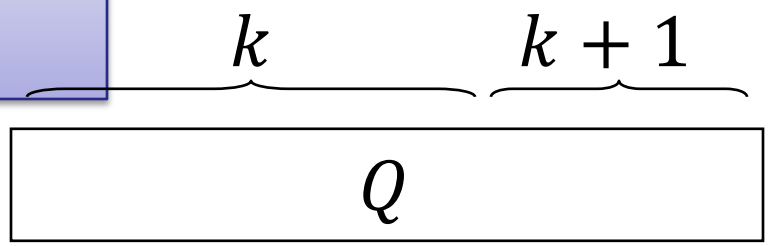

In queue $Q$

# BFS Invariants

Reached nodes $R = \{ v \mid d[v] < \infty \}$

$d[u]$: $\leq k$ if $u \in$ Done

$= k$ or $k+1$ *if* $u \in Q$:




Theorem: $\forall v$: $d[v]$ =length of shortest $s \rightsquigarrow v$ path

Proof:

- $\geq$: length of $s \rightsquigarrow v$ path in BFS tree = $d[v]$

  By induction on the time that $v$ was reached (on $d[v]$)

- $\leq$: By induction on length of shortest $s \rightsquigarrow v$ path

  Consider shortest path $s \rightsquigarrow u - v$

  by IH, $d[u] \leq$ length of $s \rightsquigarrow u$ path

  after $u$ processed, $d[v] \leq d[u] + 1 \leq$ length of $s \rightsquigarrow v$ path

# BFS Tree => Partition into Layers

- Layer 0: $L_0 = \{s\}$
- Layer $i$: $L_i = \{ v \mid d[v] = i \}$
- Undirected graphs: edges connect nodes in same layer or adjacent layers
- Directed graphs: edges can go only to next layer, to same layer or to previous layers (any number back)