Graphs
Representation
Breadth First Search

CS 4231, Fall 2020 Mihalis Yannakakis

Graphs

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

- Simple graph: no self-loops, no parallel (duplicate) edges
- Multigraph: allows multiple edges

- Fundamental model.
- Many applications

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
  - Data structures: nodes and pointers
  - Social relationships, social networks (“knows”, “works for”..)
  - 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
  - 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: $n=|N|$, $e=|E|$

- 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
- Graph Structure:

  Connected components, Cycles

Optimization problems:

- Shortest Paths between nodes
- Minimum Spanning Tree
- Maximum Flow, Minimum Cut, Matching, …..

Graph Representation

- Node representation: Integers 1, …, n, so we can index on them (build arrays)
- Generally, nodes belong to some domain D (eg. strings).
- Use a dictionary (symbol table) eg. hash map, trie, search tree, etc, to map between D and {1,…,n}, both ways.

- Example:

  Newark airport: 1

  Kennedy airport: 2

  La Guardia airport: 3

  ….

Graph Representation - Edges

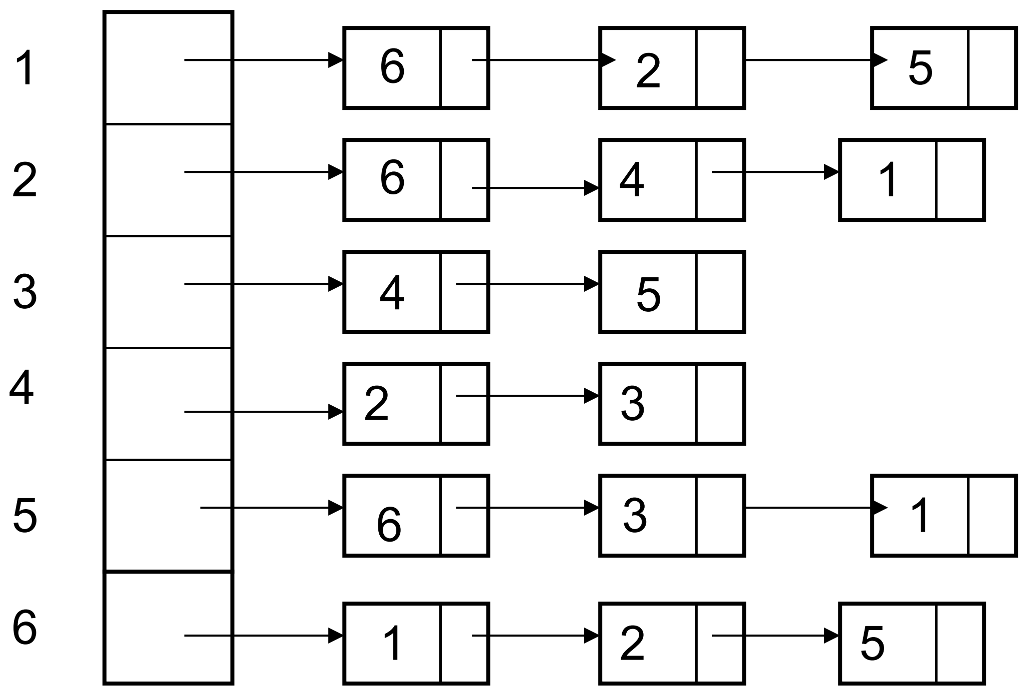
- List of edges, i.e. pairs i,j
- Space = O(e)

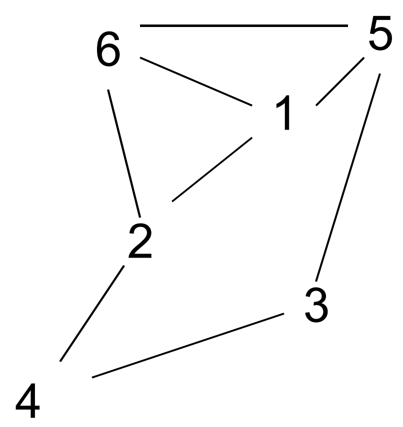
- Adjacency matrix indexed by nodes: 2d boolean array A[i,j] is 1 if edge (i,j) is in E, 0 otherwise
- Undirected graph: matrix is symmetric A[i,j] = A[j,i]
- Directed graph: A maybe asymmetric

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

Adjacency list representation

- Array of lists = adjacency lists of vertices





Space proportional to n+e

Adjacency list representation ctd.

- Undirected Graphs:
- Every edge (i,j) leads to two entries: j in adj[i] and i in ad[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 mate 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 tell if two adj list representations represent same graph in O(n+e) time (HW exercise)

- A graph with vertex names from a general domain has many representations depending how vertex names are mapped to {1,...,n}
- Graph isomorphism problem: Determine whether two representations are isomorphic, i.e. same graph except for the vertex mapping to {1,...,n}
- Is there vertex permutation that preserves the edges?
- Much harder problem. Complexity is open.

Weighted Graphs

- Weights or other info associated with edges and/or vertices
- For example, if graph = connections/roads between cities, lengths of edges
- Weighted adjacency matrix: A[i,j]= weight(i,j)
- Adjacency list: node contains also weight field

- Same for “labels” on nodes (for example, chemical compound has nodes labeled by molecules C, Fe etc)

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
- Sometimes use doubly linked for easy deletion
- Also, for undirected graphs may have pointers connecting the two occurrences of an edge

Simple Tasks

- Assume Adj lists representation
- Print all edges

  For Directed Graphs:

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

  For Undirected Graphs:

  for i=1 to n { for each 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 or undirected graph G=(N,E), “source” node s

Which nodes are reachable from node s?

SEARCH(G,s)

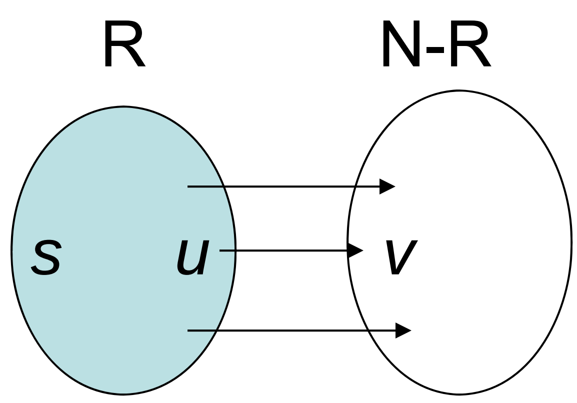
  R = {s}

  while ∃ edge from R to N-R

    { (u,v)=such an edge

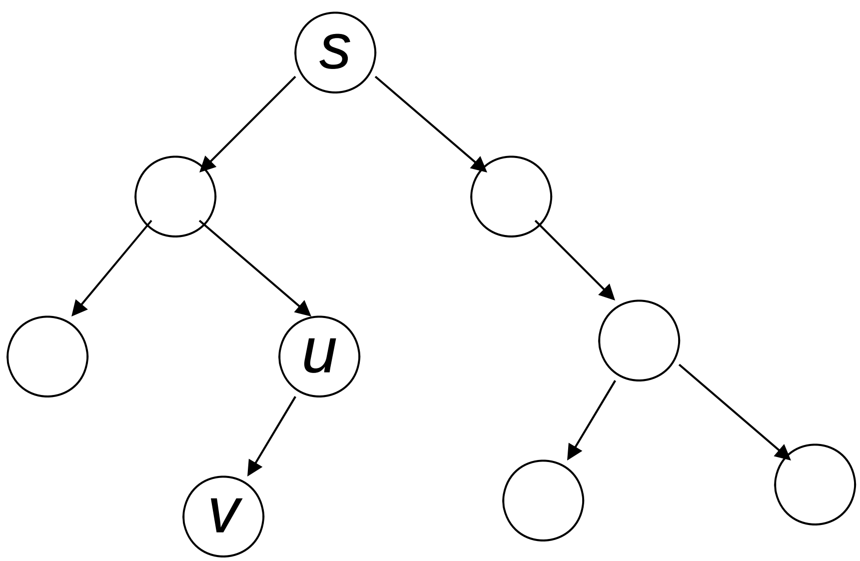
      R = R∪{v}

    }



Reachability (Search) Tree

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



Correctness

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

Proof:

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

  By induction on time that v was added to R

  -> s-v path in Search tree

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

  By induction on length of s-v path

R can be implemented by a bitvector *mark:* N$\rightarrow${0,1}

Selection of edge from R to N-R depends on policy

Search Strategies

- In general, many edges from R to N-R
- Different policies for choosing node u in R and the edge (u,v) to N-R

  →

- 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: Choose edge (u,v) where u is earliest node added to R

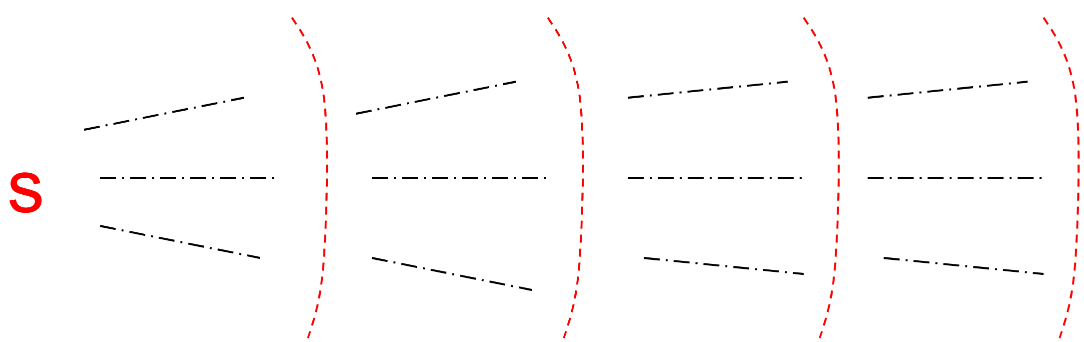
Queue Q keeps reached, unprocessed nodes

- CLRS colors nodes: white ($\in$ N-R: not reached),
  gray (R$\cap$Q: reached active), black (R-Q: done)

Queue $\Rightarrow$ nodes reached earlier are processed earlier

d[v]: length of path of BFS tree from the source s to node v

Theorem: d[v] = distance from s to v in the graph =length of shortest path from s to v



Breadth First Search

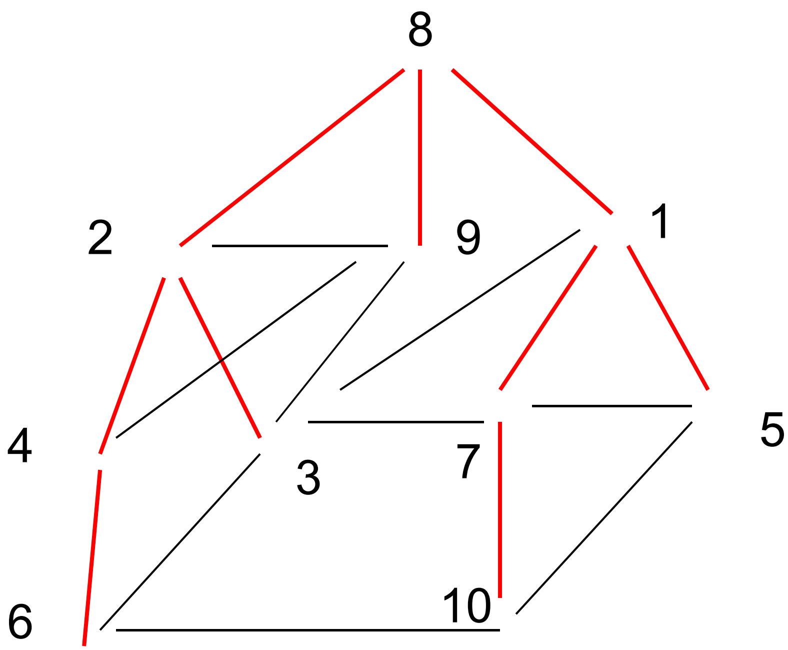
BFS(G,s)

```
for each v∈N-{s} do {d[v]=∞; p[v]=⊥}
d[s]=0; p[s]= ⊥ ;
Q = {s}
while Q ≠∅ do
  {  u=Dequeue(Q)
     for each v ∈ Adj[u] do
        if d[v]= ∞ then
         {d[v]=d[u]+1; p[v]=u; Enqueue(Q,v)}
  }
```

Reachable nodes: d[v]=finite.   Unreachable nodes: d[v]= ∞

Time Complexity: $O(n+e)$

Example



Adjacency lists:

1: 3, 7, 5, 8

2: 4, 3, 8, 9

3: 2, 9, 6, 7, 1

4: 6, 2, 9

5: 1, 7, 10

6: 4, 3, 10

7: 3, 1, 10, 5

8: 2, 9, 1

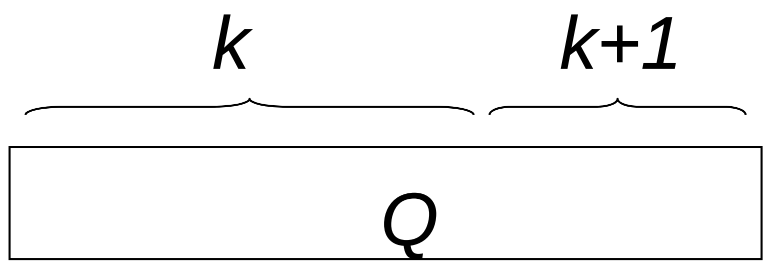
9: 8, 2, 4, 3

10: 6, 7, 5

BFS from node 8

BFS Invariants

- Reached nodes R = { u | d[u] < ∞ }
- d[u]: u $\in$Done $\leq k$ u $\in$Q: k or $k+1$



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

Proof:

- $\geq$: length of s-v path in BFS tree = d[v]
- $\leq$: By induction on length of shortest s-v path

  Consider shortest path s - - - u – v .

  By i.h. d[u] $\leq$ length of s---u path

  After u is processed, d[v]$\leq$d[u]+1 $\leq$ length of s---v path

BFS Tree & Partitioning Graph 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