Unweighted directed graphs
Announcements

Programming assignment (soon)
- Just homework?

Midterm next Tues
- Covers sorting, selection
Graph

A directed graph $G$ is a set of edges and vertices: $G = (V, E)$

Two common ways to represent a graph:
- Adjacency matrix
- Adjacency list
An adjacency matrix has a 1 in row i and column j if you can go from node i to node j.
Graph

An adjacency list just makes lists out of each row (list of edges out from every vertex)
Graph

Difference between adjacency matrix and adjacency list?
Graph

Difference between adjacency matrix and adjacency list?

Matrix is more memory $O(|V|^2)$, less computation: $O(1)$ lookup

List is less memory $O(E+V)$ if sparse, more computation: $O($branch factor$)$
Graph

Adjaceny matrix, $A = A^1$, represents the number of paths from row node to column node in 1 step.

Prove: $A^n$ is the number of paths from row node to column node in $n$ steps.
Proof: Induction
Base: \( A^0 = I \), 0 steps from \( i \) is \( i \)

Induction: (Assume \( A^n \), show \( A^{n+1} \))

Let \( a^n_{i,j} \) = \( i^{\text{th}} \) row, \( j^{\text{th}} \) column of \( A^n \)

Then \( a^{n+1}_{i,j} = \sum_k a^n_{i,k} a^1_{k,j} \)

This is just matrix multiplication
BFS Overview

Create FIFO queue to explore unvisited nodes

https://www.youtube.com/watch?v=nI0dT288VLs
DFS Overview

Create FILO queue to explore unvisited nodes
BFS and DFS in trees

Solve problems by making a tree of the state space

max
min
max
BFS and DFS in trees

Often times, fully exploring the state space is too costly (takes forever)

Chess: $10^{47}$ states (tree about $10^{123}$)
Go: $10^{171}$ states (tree about $10^{360}$)
At 1 million states per second...

Chess: $10^{109}$ years
Go: $10^{346}$ years
BFS and DFS in trees

BFS prioritizes “exploring”
DFS prioritizes “exploiting”

White to move  Black to move
BFS and DFS in trees

BFS benefits?

DFS benefits?
BFS and DFS in trees

BFS benefits?
- if stopped before full search, can evaluate best found

DFS benefits?
- uses less memory on complete search
BFS and DFS in graphs

BFS: shortest path from origin to any node

DFS: find graph structure

Both running time of $O(V+E)$
**Breadth first search**

BFS(G,s) // to find shortest path from s
for all v in V
    v.color=white, v.d=∞, v.π=NIL
s.color=grey, v.d=0
Enqueue(Q,s)
while(Q not empty)
    u = Dequeue(Q,s)
    for v in G.adj[u]
        if v.color == white
            v.color=grey, v.d=u.d+1, v.π=u
            Enqueue(Q,v)
u.color=black
Breadth first search

Let $\delta(s, v)$ be the shortest path from $s$ to $v$

After running BFS you can find this path as: $v.\pi$ to $(v.\pi).\pi$ to ... $s$

(pseudo code on p. 601, recursion)
BFS correctness

Proof: contradiction
Assume $\delta(s,v) \neq v.d$

$v.d \geq \delta(s,v)$ (Lemma 22.2, induction)

Thus $v.d > \delta(s,v)$

Let $u$ be previous node on $\delta(s,v)$

Thus $\delta(s,v) = \delta(s,u)+1$

and $\delta(s,u) = u.d$

Then $v.d > \delta(s,v) = \delta(s,u)+1 = u.d+1$
BFS correctness

\[ v.d > \delta(s,v) = \delta(s,u)+1 = u.d+1 \]

Cases on color of \( v \) when \( u \) dequeue, all cases invalidate top equation

Case white: alg sets \( v.d = u.d + 1 \)

Case black: already removed thus \( v.d \leq u.d \) (corollary 22.4)

Case grey: exists \( w \) that dequeued \( v \), \( v.d = w.d+1 \leq u.d+1 \) (corollary 22.4)
Depth first search

DFS can be implemented with BFS

We will mark both a start (colored grey) and finish (colored black) times

This helps us quantify properties of graphs
Depth first search

DFS(G)
for all v in V
  v.color=white, v.π=NIL
time=0
for each v in V
  if v.color==white
    DFS-Visit(G,v)
Depth first search

DFS-Visit(G,u)
time=time+1
u.d=time, u.color=grey
for each v in G.adj[u]
    if v.color == white
        v.π=u
        DFS-Visit(G,v)
    u.color=black, time=time+1, u.f=time
Depth first search

Edge markers:

Consider edge $u$ to $v$

$B =$ Edge to grey node ($u.f < v.f$)
$F =$ Edge to black node ($u.f > v.f$)
$C =$ Edge to black node ($u.d > v.f$)
Depth first search

DFS can do topographical sort

Run DFS, sort in decreasing finish time
Depth first search

DFS can find strongly connected components
Depth first search

Let $G^T$ be $G$ with edges reversed

Then to get strongly connected:
1. DFS($G$) to get finish times
2. Compute $G^T$
3. DFS($G^T$) on vertex in decreasing finish time
4. Each tree in forest SC component