CS3000
6/1 – Thurs :)

Admin
- Long HW4 out, due Tue. 6/6
- today: fun optional recitation
- Head's up: next exam 6/15 (future us to worry about it)

Agenda
1. Breadth First Search overview
2. BFS Implementation
3. BFS proof

Recap

Graph G	Func(G, ...)
G.V	vertices (rep. by label, or by ordering)
G.Adj[v]	adjacency list for v

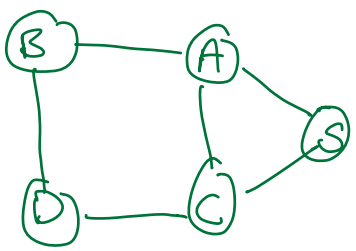
today: simple graph
undirected
unweighted

1. BFS Overview

BFS is algorithm for searching a graph
- Graph G
- source vertex s
- not searching for a key, though!
- discover all vertices reachable from s

Today:
- undirected, unweighted, adj list

- connected graph
- all vertices reachable from s — trivial



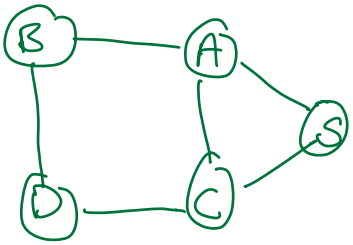
- BFS also gives us how to reach them

Side effect: path from s to every other vertex

Path from S to B: $\langle S, A, C, D, B \rangle$
valid? ✓
ideal? no

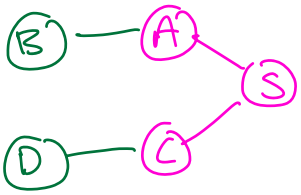
Another path from S to B: $\langle S, A, B \rangle$
valid? ✓
ideal? ✓

BFS is an optimization problem!
↳ want shortest path from s to every other vertex



→ what does BFS do?

- visit S's neighbors (A, C)
  - ↳ reachable in one step
- visit A's neighbors, C's neighbors
  - ↳ reachable in 2 steps



→ BFS tree
(shows all shortest paths)

distance to
A = 1 C = 1
B = 2 D = 2

2. BFS Implementation

Conceptual → pseudocode

- Function needs G, and source vertex S
- keep track of nodes we've visited
- keep track of distances from S
- which node to visit next
- for each vertex, its predecessor
  - ↳ how did I get here in the shortest path?

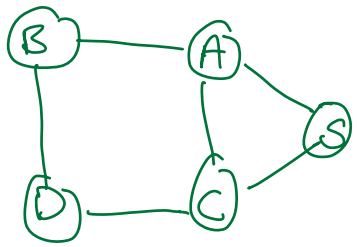
- How long is shortest path from s to any v?
- What is the path from s to any v?

Distances:
- have an attribute on each vertex
- $v.d$ ⇝ distance from s

Predecessors:
- have attribute on vertex
- $v.\pi$ ⇝ predecessor node

Keep track of visiting:
- once we've computed d and π on a vertex, never need to visit it again
- add attributes to vertex v, $v.\underline{color}$
  - ↳ white = unvisited
    gray = in process
    black = done
- use a queue for vertices to visit
  - ↳ enqueue (Q, e) adds element to back
    dequeue (Q) remove + return element from the front

```
BFS(G, s)
 1  for each vertex in u ∈ G.V − {s}
 2        u.color = WHITE
 3        u.d = ∞
 4        u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11        u = DEQUEUE(Q)
12        for each v ∈ G.Adj[u]
13              if v.color == WHITE
14                    v.color = GRAY
15                    v.d = u.d + 1
16                    v.π = u
17                    ENQUEUE(Q, v)
18        u.color = BLACK
```



	S	A	B	C	D
d	0	∞	∞	∞	∞
π	nil	nil	nil	nil	nil
color	g	w	w	w	w

Q = <S>

- When is a vertex gray?
- What are values after S is done, A is done, C is done?

① After S is done

	S	A	B	C	D
d	0	1		1	
π	nil	S		S	
color	b	g		g	

Q = <C, A> front
back

→ gray → black

(2) After A is done. A has C, S

	S	A	B	C	D
d	0	1	2	1	
π	nil	S	A	S	
color	b	b	g	g	

Q = <B, C>

(3) After C is done. C has A, S

	S	A	B	C	D
d	0	1	2	1	2
π	nil	S	A	S	C
color	b	b	g	b	g

Q = <D, B>

(4) dequeue B. nothing happens. B. color = black Q = <D>

(5) dequeue D. nothing happens. D. color = black

- when is vertex gray?
  - ↳ when it's being processed or in the queue

	S	A	B	C	D
d	0	1	2	1	2
π	nil	S	A	S	C
color	b	b	b	b	b

- given this table, we have the distance from s to v
  (s we can construct the shortest path from s to v
  (work backwards from v)

10:54

3. BFS Proof

- runtime
- correctness

runtime of BFS

```
BFS(G, s)
 1  for each vertex in u ∈ G.V − {s}
 2        u.color = WHITE
 3        u.d = ∞
 4        u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11        u = DEQUEUE(Q)
12        for each v ∈ G.Adj[u]
13              if v.color == WHITE
14                    v.color = GRAY
15                    v.d = u.d + 1
16                    v.π = u
17                    ENQUEUE(Q, v)
18              u.color = BLACK
```

→ traditional W.C.

- visiting every edge
  - U : A → B → C → D ....
- visits every ~~node~~ edge, every time we run the loop

Amortized analysis

- vertex gets enqueued if it was painted white
- the loop never paints a vertex white (only init)
- every vertex gets enqueued once
- and therefore can only be dequeued once

enqueue/dequeue are both $\Theta(1)$

↳ whole sequence $\Theta(V)$

- loop at line 12 sees every edge once

↳ whole sequence $\Theta(E)$

total: $\Theta(V+E)$ Linear

Correctness of BFS: Finds shortest path from s to every vertex v

Observation:

- We use a queue to store (order vertices)
- In example, if we never dequeue

back $\langle D, B, C, A, s \rangle$ front

$2 \quad 1 \quad 0$ d values

- if v is enqueued before u, then $v.d \leq u.d$

Define $\delta(s,v)$ = distance of shortest path from s to v

Want to show $v.d = \delta(s,v)$ for every vertex v

Proof by induction on value of $v.d$

↳ $0 \ldots n$

Base case: $v.d = 0$

only happens for s, which is correct ☑

IH all vertices v such that $v.d \leq k$,

$v.d = \delta(s,v)$

$s \sim \cdot \sim \cdot \sim v$ $\quad v.d = \delta(s,v)$

$\leq k$

Consider vertex w, with $w.d = k+1$

Algo at any points adds 1 for existing d value

↳ some vertex v, | w was discovered from v
$v.d = k$ |

$s \rightsquigarrow \cdot \rightsquigarrow \cdot \rightsquigarrow v - w$ (we want this to be a SP to w)

$v.d$ (k) $w.d$ (k+1)

• $v.d = \delta(s,v)$ by IH

Assume not!

- Some other path exists with length $\leq k$ that goes from s to w
- v_0 precedes w in this path

$s \rightsquigarrow \cdot \rightsquigarrow \cdot \rightsquigarrow v_0 - w$

$v_0.d = \delta(s,v_0) \leq k-1$

$\delta(s,w) \leq k$

$v_0.d \leq k-1$

$v.d = k$

↳ implies v_0 was enqueued before v

But, that would mean we dequeued v_0 before v, and it would be w's predecessor. $w.d$ would have been $v_0.d+1$ instead of $v.d+1$

↳ Contradiction!