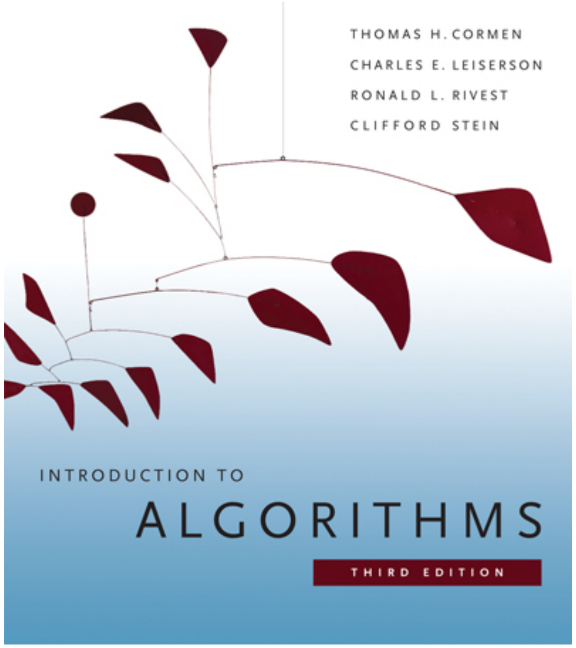
6.006- *Introduction to Algorithms*

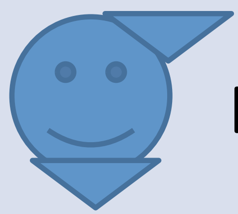
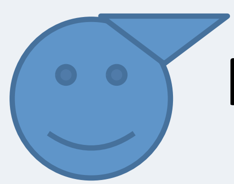


Lecture 12 – Graph Algorithms

Prof. Manolis Kellis

CLRS 22.2-22.3

Combinatorics

	Ponytail	No ponytail
Beard	 Erik	?
No Beard	 Piotr	 Manolis

Unit #4 – Games, Graphs, Searching, Networks

Unit	Pset	Week	Date		Lecture (Tuesdays and Thursdays)		Recitation (Wed and Fri)
Intro	PS1	1	Tue Feb 01	1	Introduction and Document Distance	1	Python and Asymptotic Complexity
Binary	Out: 2/1		Thu Feb 03	2	Peak Finding Problem	2	Peak Finding correctness & analysis
Search	Due: Mon 2/14	2	Tue Feb 08	3	Scheduling and Binary Search Trees	3	Binary Search Tree Operations
Trees	HW lab: Sun 2/13		Thu Feb 10	4	Balanced Binary Search Trees	4	Rotations and AVL tree deletions
Hashing	PS2 Out: 2/15	3	Tue Feb 15	5	Hashing I : Chaining, Hash Functions	5	Hash recipes, collisions, Python dicts
	Due: Mon 2/28		Thu Feb 17	6	Hashing II : Table Doubling, Rolling Hash	6	Probability review, Pattern matching
	HW lab:Sun 2/27	4	Tue Feb 22	-	President's Day - Monday Schedule - No Class	-	No recitation
			Thu Feb 24	7	Hashing III : Open Addressing	7	Universal Hashing, Perfect Hashing
Sorting	PS3. Out: 3/1	5	Tue Mar 01	8	Sorting I : Insertion & Merge Sort, Master Theorem	8	Proof of Master Theorem, Examples
	Due: Mon 3/7		Thu Mar 03	9	Sorting II : Heaps	9	Heap Operations
	HW lab: Sun 3/6	6	Tue Mar 08	10	Sorting III: Lower Bounds, Counting Sort, Radix Sort	10	Models of computation
			Wed Mar 09	Q1	Quiz 1 in class at 7:30pm. Covers L1-R10. Review Session on Tue 3/8 at 7:30pm.		
Graphs	PS4. Out: 3/10		Thu Mar 10	11	Searching I: Graph Representation, Depth-1st Search	11	Strongly connected components
and	Due: Fri 3/18	7	Tue Mar 15	12	Searching II: Breadth-1st Search, Topological Sort	12	Rubik's Cube Solving
Search	HW lab:W 3/16		Thu Mar 17	13	Searching III: Games, Network properties, Motifs	13	Subgraph isomorphism
Shortest	PS5	8	Tue Mar 29	14	Shortest Paths I: Introduction, Bellman-Ford	14	Relaxation algorithms
Paths	Out: 3/29		Thu Mar 31	15	Shortest Paths II: Bellman-Ford, DAGs	15	Shortest Path applications
	Due: Mon 4/11	9	Tue Apr 05	16	Shortest Paths III: Dijkstra	16	Speeding up Dijkstra's algorithm
	HW lab:Sun 4/10		Thu Apr 07	17	Graph applications, Genome Assembly	17	Euler Tours
Dynamic	PS6	10	Tue Apr 12	18	DP I: Memoization, Fibonacci, Crazy Eights	18	Limits of dynamic programming
Program	Out: Tue 4/12		Wed Apr 13	Q2	Quiz 2 in class at 7:30pm. Covers L11-R17. Review Session on Tue 4/13 at 7:30pm.		
ming	Due: Fri 4/29		Thu Apr 14	19	DP II: Shortest Paths, Genome sequence alignment	19	Edit Distance, LCS, cost functions
	HW lab:W 4/27	11	Tue Apr 19	-	Patriot's Day - Monday and Tuesday Off	-	No recitation
			Thu Apr 21	20	DP III: Text Justification, Knapsack	20	Saving Princess Peach
		12	Tue Apr 26	21	DP IV: Piano Fingering, Vertex Cover, Structured DP	21	Phylogeny
Numbers	PS7 out Thu4/28		Thu Apr 28	22	Numerics I - Computing on large numbers	22	Models of computation return!
Pictures	Due: Fri 5/6	13	Tue May 3	23	Numerics II - Iterative algorithms, Newton's method	23	Computing the nth digit of π
(NP)	HW lab: Wed 5/4		Thu May 5	24	Geometry: Line sweep, Convex Hull	24	Closest pair
		14	Tue May 10	25	Complexity classes, and reductions	25	Undecidability of Life
Beyond			Thu May 12	26	Research Directions (15 mins each) + related classes		
		15	Finals week	Q3	Final exam is cumulative L1-L26. Emphasis on L18-L26. Review Session on Fri 5/13 at 3pm		

Unit #4 Overview: Searching

Today: Introduction to Games and Graphs

- Rubik's cube, Pocket cube, Game space
- Graph definitions, representation, searching

Tuesday: Graph algorithms and analysis

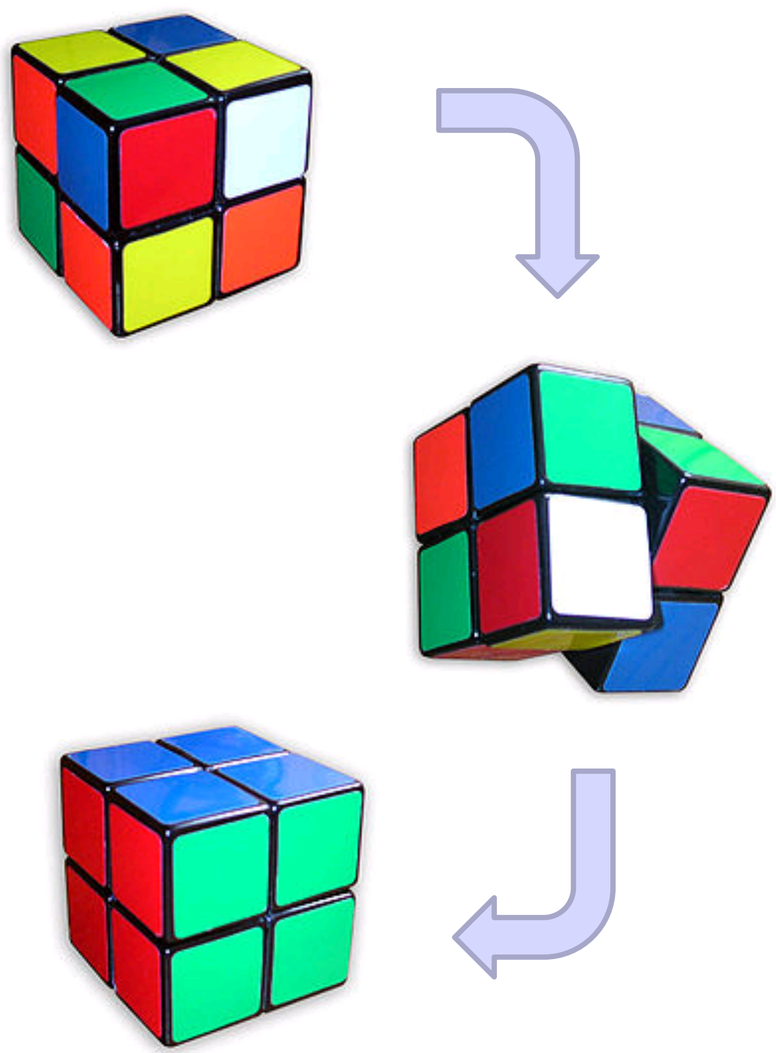
- Breadth First Search, Depth First Search
- Queues, Stacks, Augmentation, Topological sort

Thursday: Networks in biology and real world

- Network/node properties, metrics, motifs, clusters
- Dynamic processes, epidemics, growth, resilience

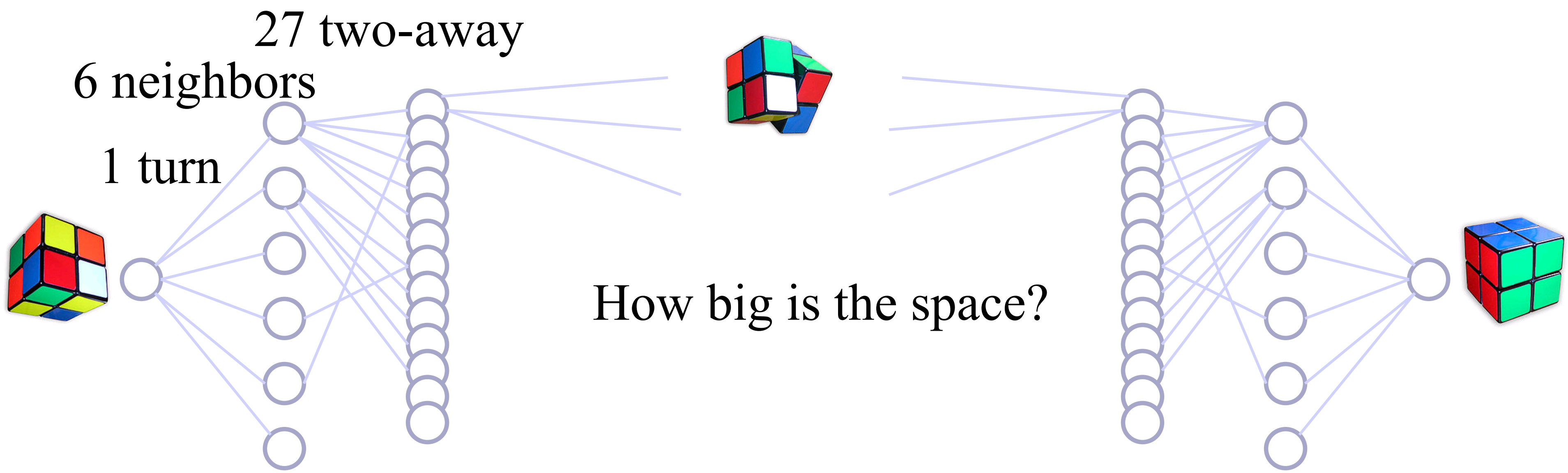
Last time: Games and Graphs

Pocket Cube



- $2 \times 2 \times 2$ Rubik's cube
- Start with any colors
- Moves are quarter turns of any face
- "Solve" by making each side one color

Searching for a solution path

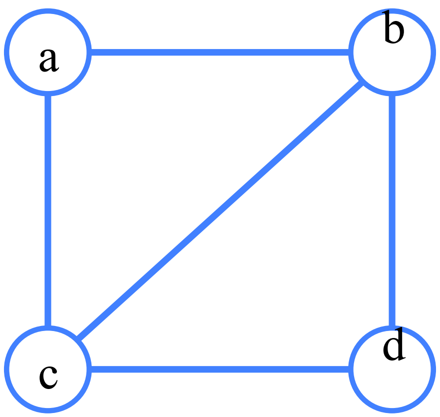


- Graph algorithms allow us explore space
  - Nodes: configurations
  - Edges: moves between them
  - Paths to ‘solved’ configuration: solutions

Graphs

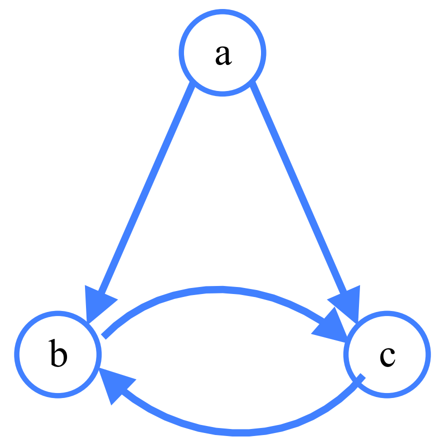
- G=(V,E)
- V a set of vertices
  - Usually number denoted by n
- $E \subseteq V \times V$ a set of edges (pairs of vertices)
  - Usually number denoted by m
  - Note $m \leq n(n-1) = O(n^2)$

Undirected example



- V={a,b,c,d}
- E={{a,b}, {a,c}, {b,c}, {b,d}, {c,d}}

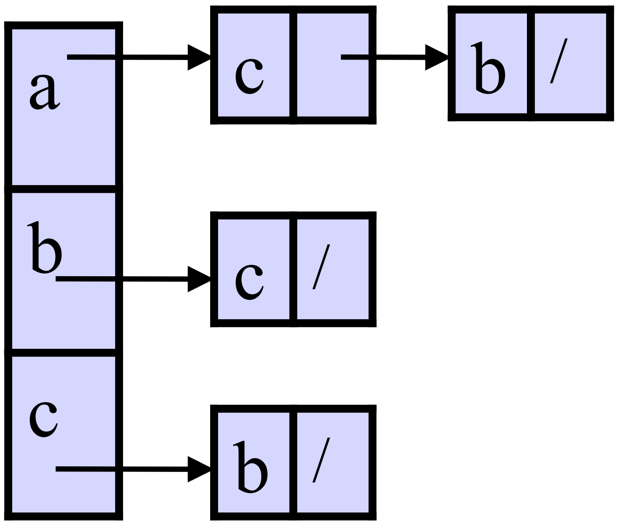
Directed example



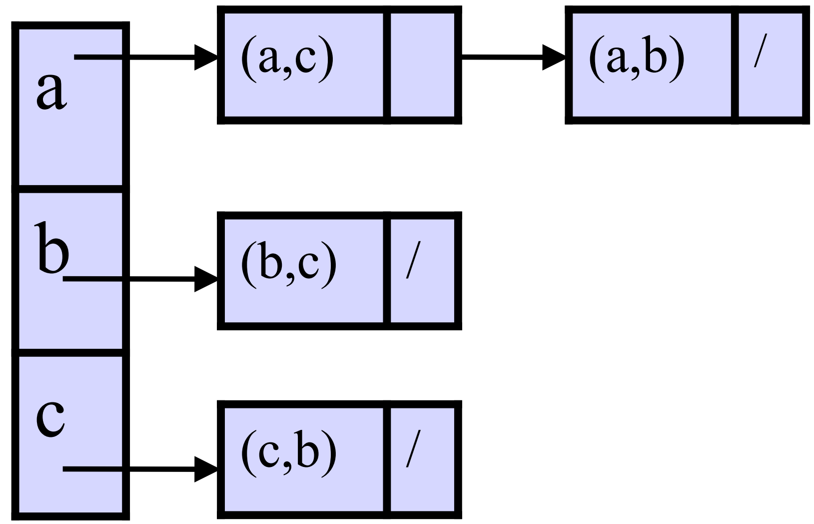
- V = {a,b,c}
- E = {(a,c), (a,b) (b,c), (c,b)}

Graph Representation

- *Adjacency lists*



- *Incidence lists*



- *Adjacency matrix*

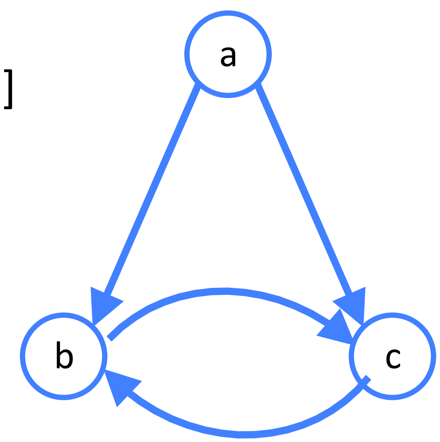
a (1)	b (2)	c (3)	
0	1	1	a (1)
0	0	1	b (2)
0	1	0	c (3)

- *Implicit representation*

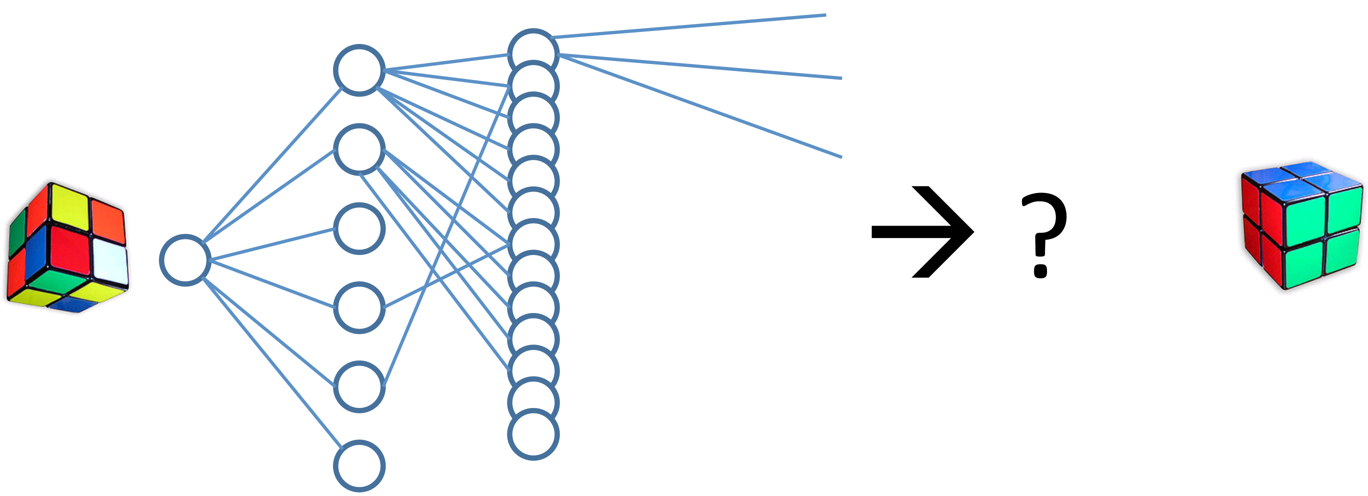
Neighbors(a) → [c,b]

Neighbors(b) → [b]

Neighbors(c) → [b]



Today: Searching graphs

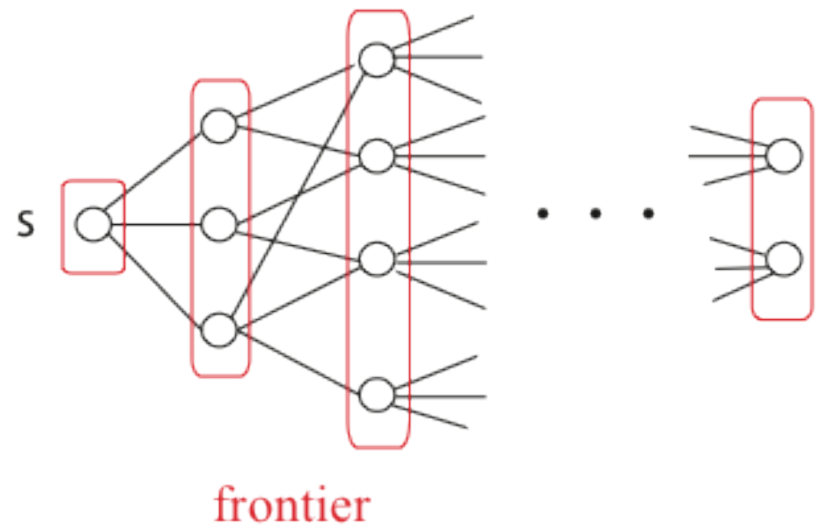


Searching Graph

- We want to get from current Rubik state to "solved" state
- How do we explore?

Breadth First Search

- start with vertex v
- list all its neighbors (distance 1)
- then all their neighbors (distance 2)
- etc.

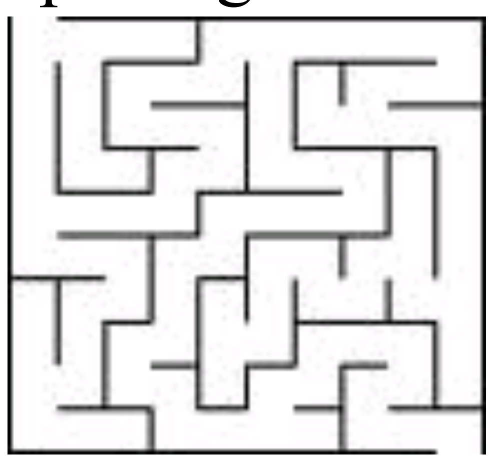


- algorithm starting at s:
  - define frontier F
  - initially F={s}
  - repeat F=all neighbors of vertices in F
  - until all vertices found

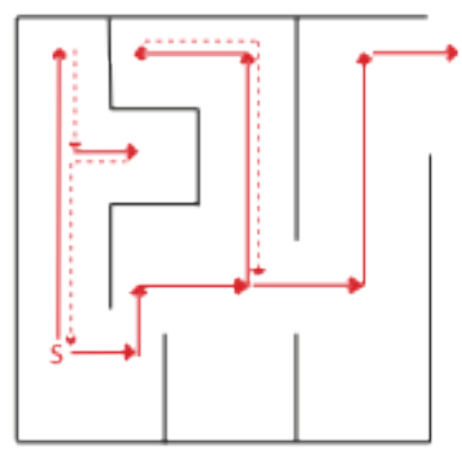
Depth First Search

- Like exploring a maze
- From current vertex, move to another
- Until you get stuck
- Then backtrack till you find a new place to explore

- Exploring a maze



- “left-hand” rule



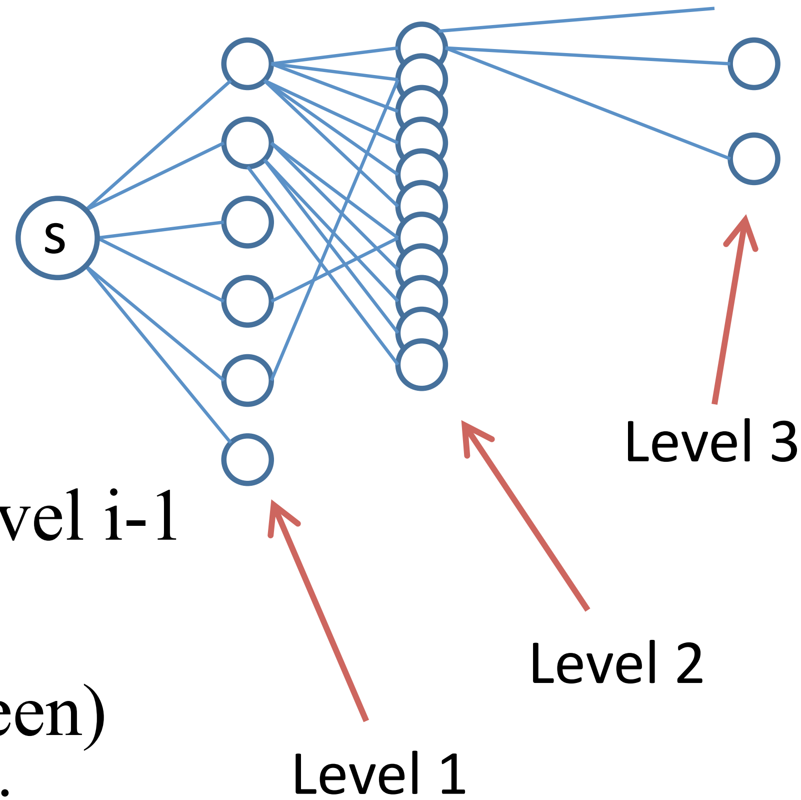
How to handle cycles: BFS/DFS

- What happens if unknowingly revisit a vertex?
  - Will eventually happen if graph contains a cycle
- BFS: get wrong notion of distance
- DFS: may get in circles
- Solution: mark vertices
  - BFS: if you’ve seen it before, ignore
  - DFS: if you’ve seen it before, back up

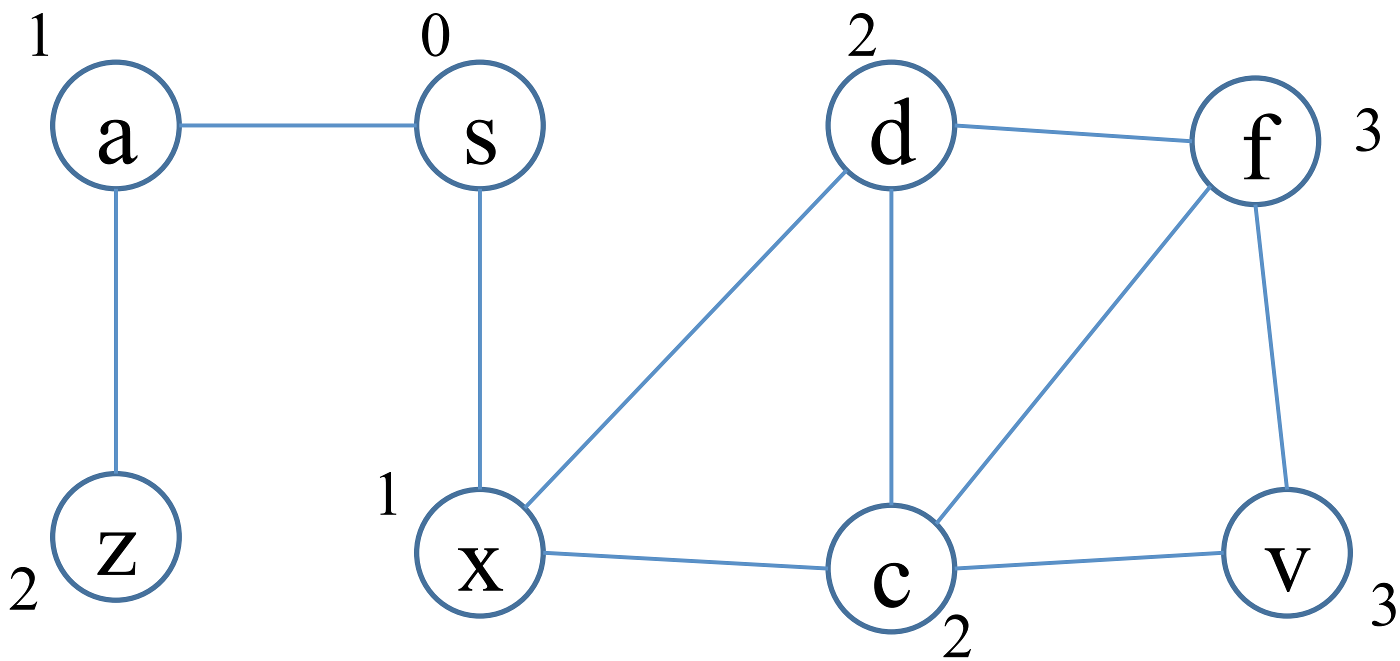
Breadth First Search (BFS)

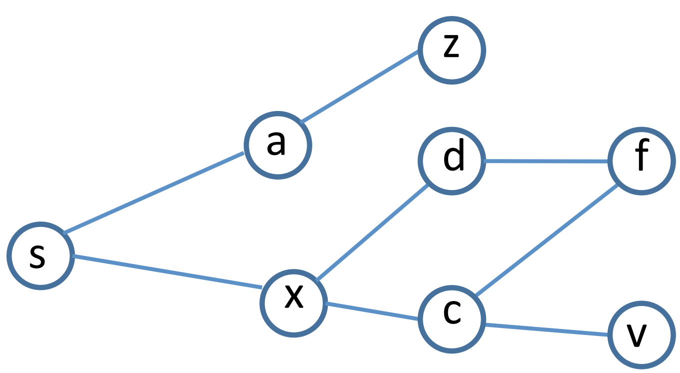
BFS algorithm outline

- Initial vertex s
  - Level 0
- For i=1,…
  grow level i
  - Find all neighbors of level i-1 vertices
  - (except those already seen)
  - i.e. level i contains vertices reachable via a path of i edges and no fewer



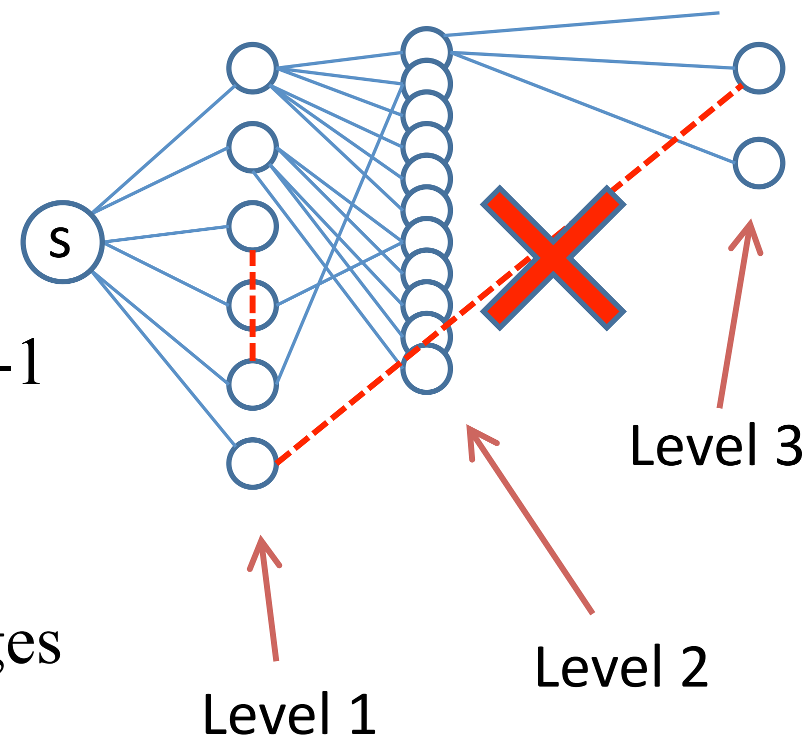
BFS example





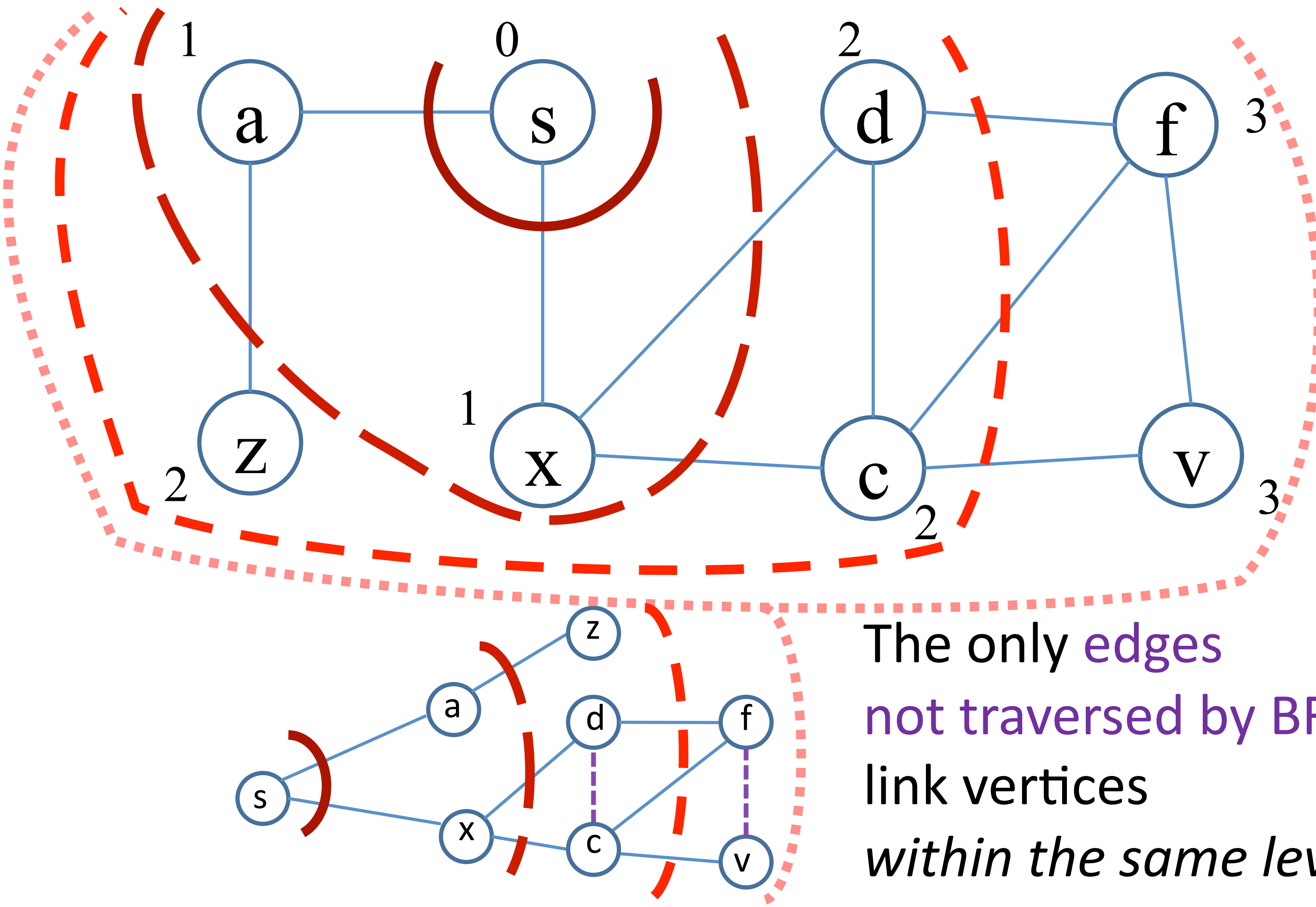
BFS algorithm outline

- Initial vertex s
  - Level 0
- For i=1,…
  grow level i
  - Find all neighbors of level i-1
  - (except those already seen)
  - i.e. level i contains vertices reachable via a path of i edges and no fewer



- Where can the other edges of the graph be?
  - They cannot jump a layer (otherwise *v* would be in Level 2)
  - But they can be between nodes in same or adjacent levels

The ‘frontier’ of BFS exploration



The only edges
not traversed by BFS
link vertices
within the same level

BFS Algorithm

- BFS(V,Adj,s)

```
level={s: 0}; parent = {s: None}; i=1
frontier=[s]                        #previous level, i-1
while frontier
    next=[]                         #next level, i
    for u in frontier
        for v in Adj[u]
            if v not in level       #not yet seen
                level[v] = i        #level of u+1
                parent[v] = u
                next.append(v)
    frontier = next
    i += 1
```

BFS Analysis: Runtime

- Naïve analysis: outer loop |V| * inner loop |V|
- Vertex v appears at the *frontier* at most once
  - Since then it has a level
  - And nodes with a level aren't added again
  - Total time spent adding nodes to *frontier* O(n)
- Adj[v] only scanned once
  - Just when v is in *frontier*
  - Total time $\sum_v |Adj[v]|$
    - This sum counts each "outgoing" edge
    - So O(m) time spend scanning adjacency lists
- Total: O(m+n) time --- "Linear time"
  - For sparse graphs |V|+|E| is much better than $|V|^2$

BFS Analysis: Correctness

i.e. why are all nodes reachable from s explored?

(we'll actually prove a stronger claim)

- **Claim**: If there is a path of L edges from s to v, then v is added to *next* when i=L or before
- **Proof**: induction
  - **Base case:** s is added before setting i=1
  - **Inductive step when i=L:**
    - Consider path of length L from s to v
    - This must contain: (1) a path of length L-1 from s to u
    - (2) and an edge (u,v) from u to v
  - By inductive hypothesis, u was added to *next* when i=L-1 or before
    - If v has not already been inserted in *next* before i=L, then it gets added during the scan of Adj[u] at i=L
  - So it happens when i=L or before. QED

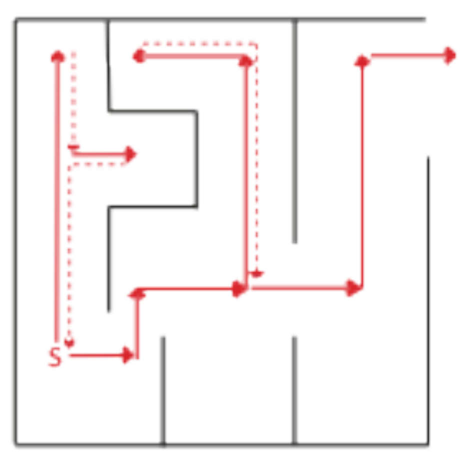
Corrollary: BFS→Shortest Paths

- From correctness analysis, conclude more:
  - Level[v] is length of shortest s→v path
- Parent pointers form a shortest paths tree
  - i.e. the union of shortest paths to all vertices
- To find shortest path from s to v
  - Follow parent pointers from v backwards
  - Will end up at s

Depth First Search (DFS)

DFS Algorithm Outline

- Explore a maze
  - Follow path until you get stuck
  - Backtrack along breadcrumbs till find new exit
  - i.e. recursively explore

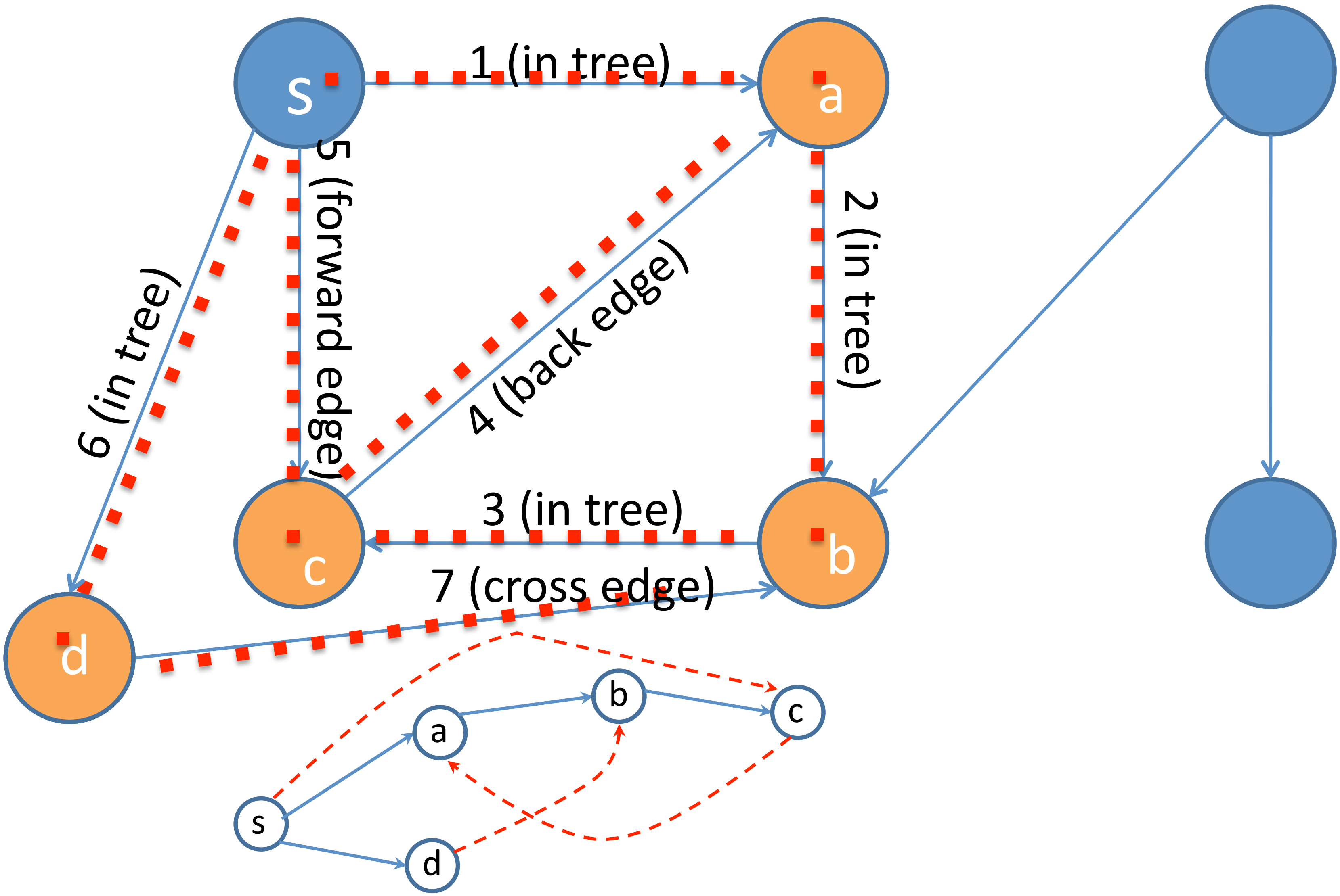


DFS Algorithm

- ***parent*** = {s: None}
- call ***DFS-visit*** (V, Adj, s)

```
def DFS-visit (V, Adj, u)
    for v in Adj[u]
        if v not in parent                    #not yet seen
            parent[v] = u
            DFS-visit (V, Adj, v)             #recurse!
```

DFS example run (starting from s)



DFS Runtime Analysis

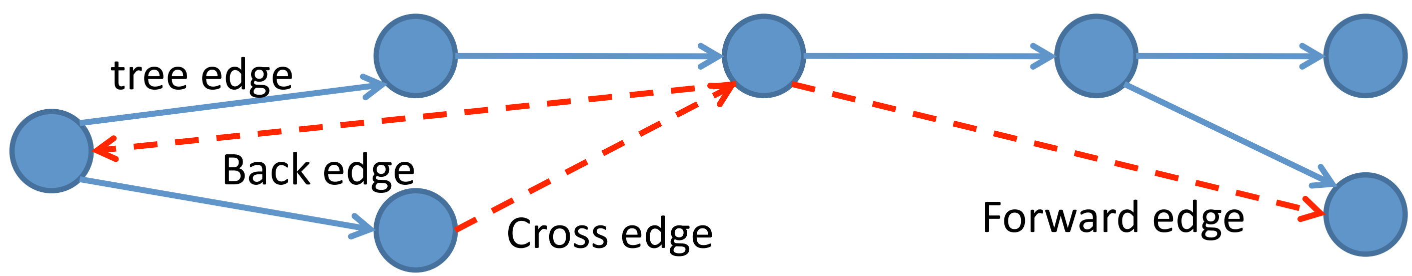
- Quite similar to BFS
- DFS-visit only called once per vertex v
  - Since next time v is in ***parent*** set
- Edge list of v scanned only once (in that call)
- So time in DFS-visit is:
  - 1 per vertex + 1 per edge
- So time is O(n+m)

DFS Correctness?

- Trickier than BFS
- Can use induction on length of ***shortest*** path from starting vertex
  - Inductive Hypothesis:
    "each vertex at distance k is visited (eventually)"
  - Induction Step:
    - Suppose vertex v at distance k.
      - Then some u at ***shortest*** distance k-1 with edge (u,v)
      - Can decompose into s→u at ***shortest*** distance k-1, and (u,v)
    - By inductive hypothesis: u is visited (eventually)
    - By algorithm: every edge out of u is checked
      - If v wasn't previously visited, it gets visited from u (eventually)

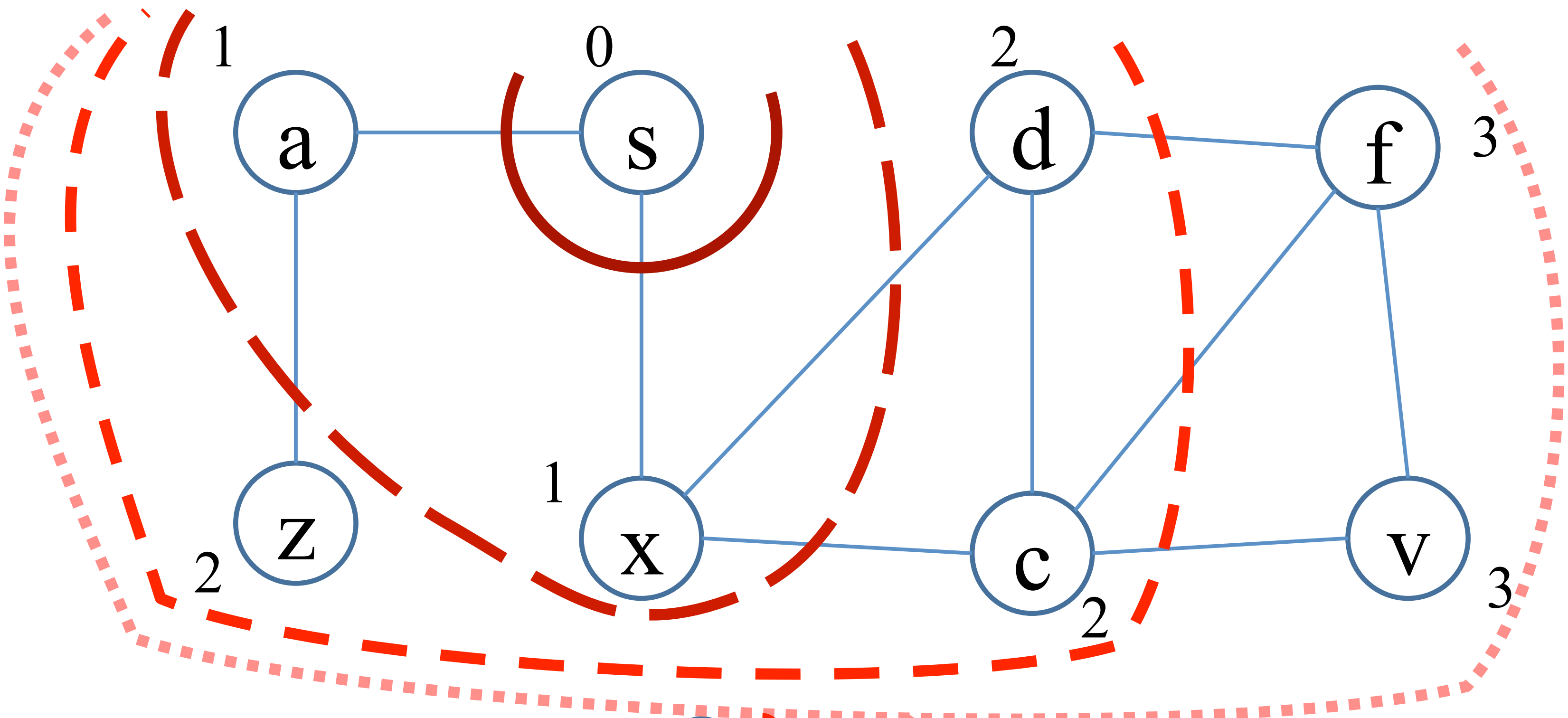
Edge Classification

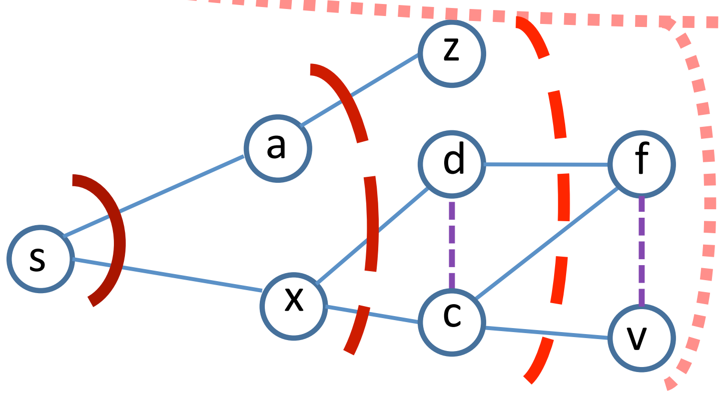
- Tree edge used to get to new child
- Back edge leads from node to ancestor in tree
- Forward edge leads to descendant in tree
- Cross edge leads to a different subtree
- To label what edge is of what type, keep global time counter and store interval during which vertex is on recursion stack



BFS vs. DFS

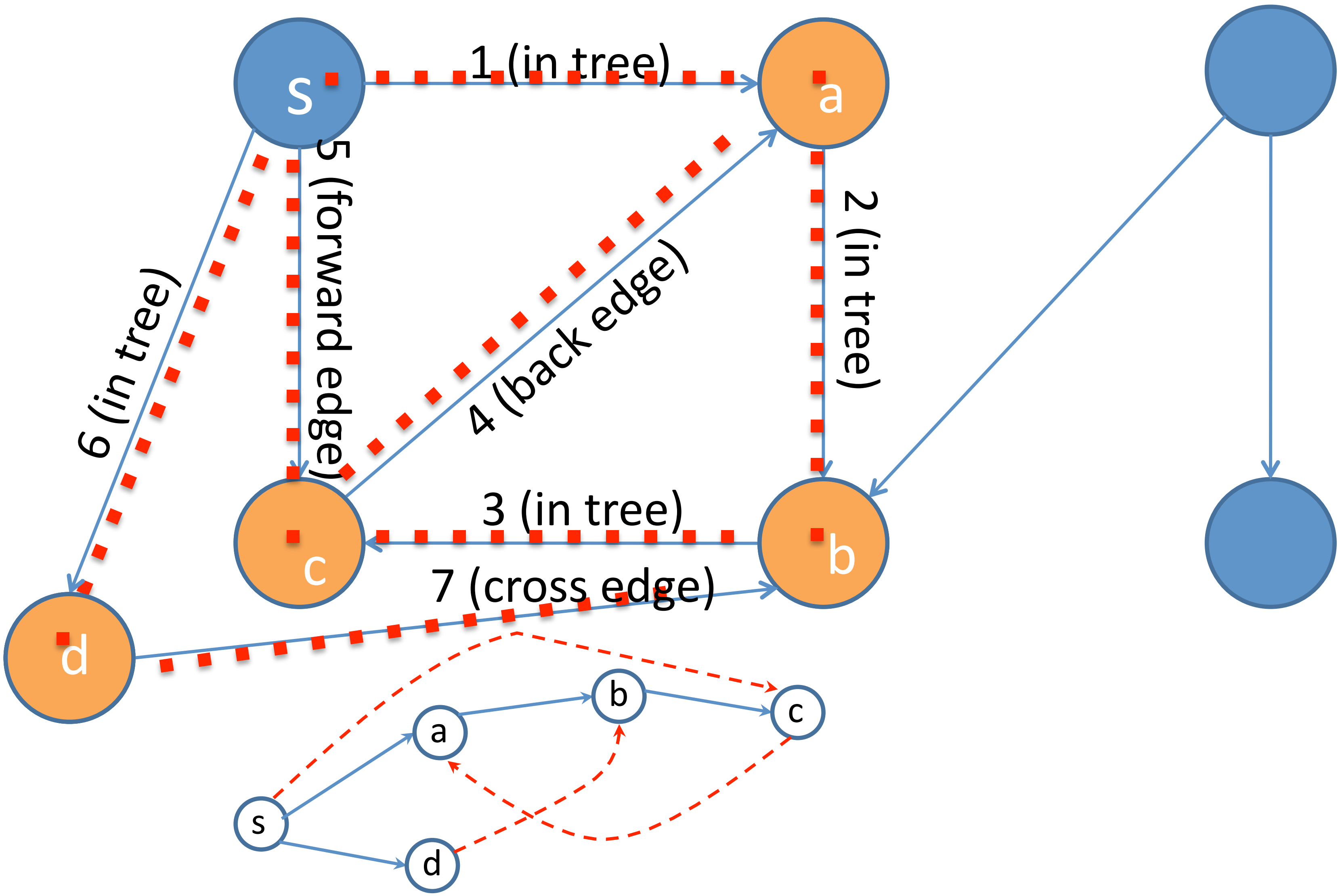
The ‘frontier’ of BFS exploration





The only edges
not traversed by BFS
link vertices
within the same level

The tree of DFS exploration



BFS/DFS Algorithm Summary

- Maintain "todo list" of vertices to be scanned

- Until list is empty
  - Take a vertex v from front of list
  - Mark it scanned
  - Examine all outgoing edges (v,u)
  - If u not marked, add to the todo list
    - BFS: add to end of todo list (***queue*: FIFO**)
    - DFS: add to front of todo list (***recursion stack*: LIFO**)

Data structures: Queues and Stacks

- BFS queue is explicit
  - Created in pieces
  - (level 0 vertices) . (level 1 vertices) . (level 2 vert…
  - the frontier at ***iteration i*** is ***piece i*** of vertices in queue
- DFS stack is implicit
  - It's the call stack of the python interpreter
  - From v, recurse on one child at a time
  - But same order if put all children on stack, then pull off (and recurse) one at a time

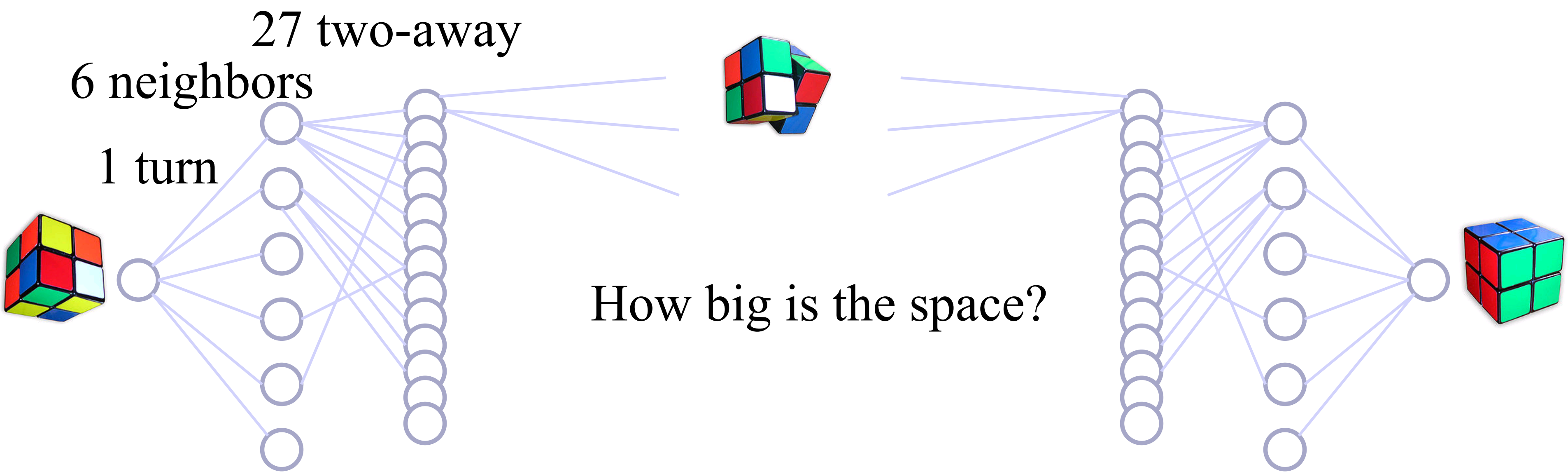
Runtime Summary

- Each vertex scanned once
  - When scanned, marked
  - If marked, not (re)added to todo list
  - Constant work per vertex
    - Removing from queue
    - Marking
  - O(n) total
- Each edge scanned once
  - When tail vertex of edge is scanned
  - Constant work per edge (checking mark on head)
  - O(m) total
- In all, O(n+m), linear in the ‘size’ of the graph

Back to our game graphs

So, how do we solve
the 2x2 Rubik's cube?

Searching for a solution path



- Graph algorithms allow us explore space
  - Nodes: configurations
  - Edges: moves between them
  - Paths to ‘solved’ configuration: solutions

Tradeoffs and Applications

- BFS:
  - Solving Rubik's cube?
  - BFS gives shortest solution
- DFS:
  - Robot exploring a building?
  - Robot can trace out the exploration path
  - Just drops markers behind

Unit #4 Overview: Searching

Today: Introduction to Games and Graphs

- Rubik's cube, Pocket cube, Game space
- Graph definitions, representation, searching

Tuesday: Graph algorithms and analysis

- Breadth First Search, Depth First Search
- Queues, Stacks, Augmentation, Topological sort

Thursday: Networks in biology and real world

- Network/node properties, metrics, motifs, clusters
- Dynamic processes, epidemics, growth, resilience

Unit #4 – Games, Graphs, Searching, Networks

Unit	Pset	Week	Date		Lecture (Tuesdays and Thursdays)		Recitation (Wed and Fri)
Intro	PS1	1	Tue Feb 01	1	Introduction and Document Distance	1	Python and Asymptotic Complexity
Binary	Out: 2/1		Thu Feb 03	2	Peak Finding Problem	2	Peak Finding correctness & analysis
Search	Due: Mon 2/14	2	Tue Feb 08	3	Scheduling and Binary Search Trees	3	Binary Search Tree Operations
Trees	HW lab: Sun 2/13		Thu Feb 10	4	Balanced Binary Search Trees	4	Rotations and AVL tree deletions
Hashing	PS2 Out: 2/15	3	Tue Feb 15	5	Hashing I : Chaining, Hash Functions	5	Hash recipes, collisions, Python dicts
	Due: Mon 2/28		Thu Feb 17	6	Hashing II : Table Doubling, Rolling Hash	6	Probability review, Pattern matching
	HW lab:Sun 2/27	4	Tue Feb 22	-	President's Day - Monday Schedule - No Class	-	No recitation
			Thu Feb 24	7	Hashing III : Open Addressing	7	Universal Hashing, Perfect Hashing
Sorting	PS3. Out: 3/1	5	Tue Mar 01	8	Sorting I : Insertion & Merge Sort, Master Theorem	8	Proof of Master Theorem, Examples
	Due: Mon 3/7		Thu Mar 03	9	Sorting II : Heaps	9	Heap Operations
	HW lab: Sun 3/6	6	Tue Mar 08	10	Sorting III: Lower Bounds, Counting Sort, Radix Sort	10	Models of computation
			Wed Mar 09	Q1	Quiz 1 in class at 7:30pm. Covers L1-R10. Review Session on Tue 3/8 at 7:30pm.		
Graphs	PS4. Out: 3/10		Thu Mar 10	11	Searching I: Graph Representation, Depth-1st Search	11	Strongly connected components
and	Due: Fri 3/18	7	Tue Mar 15	12	Searching II: Breadth-1st Search, Topological Sort	12	Rubik's Cube Solving
Search	HW lab:W 3/16		Thu Mar 17	13	Searching III: Games, Network properties, Motifs	13	Subgraph isomorphism
Shortest	PS5	8	Tue Mar 29	14	Shortest Paths I: Introduction, Bellman-Ford	14	Relaxation algorithms
Paths	Out: 3/29		Thu Mar 31	15	Shortest Paths II: Bellman-Ford, DAGs	15	Shortest Path applications
	Due: Mon 4/11	9	Tue Apr 05	16	Shortest Paths III: Dijkstra	16	Speeding up Dijkstra's algorithm
	HW lab:Sun 4/10		Thu Apr 07	17	Graph applications, Genome Assembly	17	Euler Tours
Dynamic	PS6	10	Tue Apr 12	18	DP I: Memoization, Fibonacci, Crazy Eights	18	Limits of dynamic programming
Program	Out: Tue 4/12		Wed Apr 13	Q2	Quiz 2 in class at 7:30pm. Covers L11-R17. Review Session on Tue 4/13 at 7:30pm.		
ming	Due: Fri 4/29		Thu Apr 14	19	DP II: Shortest Paths, Genome sequence alignment	19	Edit Distance, LCS, cost functions
	HW lab:W 4/27	11	Tue Apr 19	-	Patriot's Day - Monday and Tuesday Off	-	No recitation
			Thu Apr 21	20	DP III: Text Justification, Knapsack	20	Saving Princess Peach
		12	Tue Apr 26	21	DP IV: Piano Fingering, Vertex Cover, Structured DP	21	Phylogeny
Numbers	PS7 out Thu4/28		Thu Apr 28	22	Numerics I - Computing on large numbers	22	Models of computation return!
Pictures	Due: Fri 5/6	13	Tue May 3	23	Numerics II - Iterative algorithms, Newton's method	23	Computing the nth digit of π
(NP)	HW lab: Wed 5/4		Thu May 5	24	Geometry: Line sweep, Convex Hull	24	Closest pair
		14	Tue May 10	25	Complexity classes, and reductions	25	Undecidability of Life
Beyond			Thu May 12	26	Research Directions (15 mins each) + related classes		
		15	Finals week	Q3	Final exam is cumulative L1-L26. Emphasis on L18-L26. Review Session on Fri 5/13 at 3pm		