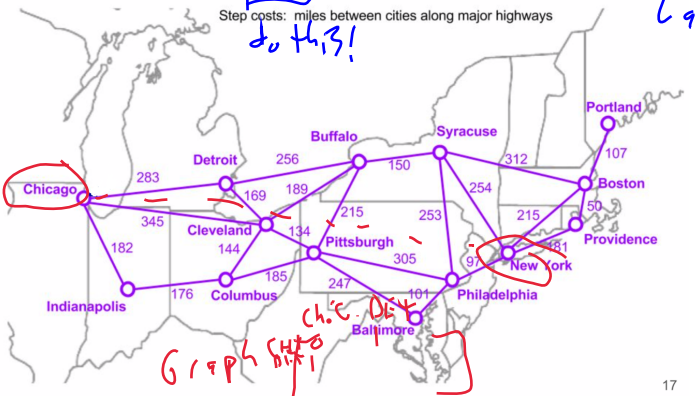
Sept 9: Depth First Searches

Opening Example: Consider map problem below, with the goal of traveling from Chicago to Boston. How does a BFS solve this problem?



Announcements and To-Dos

Announcements:

1. Homework 1 posted, due Friday.
2. Make sure you check Piazza (`piazza.com/colorado/fall2020/csci3202`) for common HW questions and concerns!

Last time we learned:

1. Breadth-first searching.

To do:

1. HW01!

Searching overview

We are still solving *search* problems: algorithms that navigate through the state space to find an ideal **path** to a goal state or states.

As we compare methods, we should ask about:

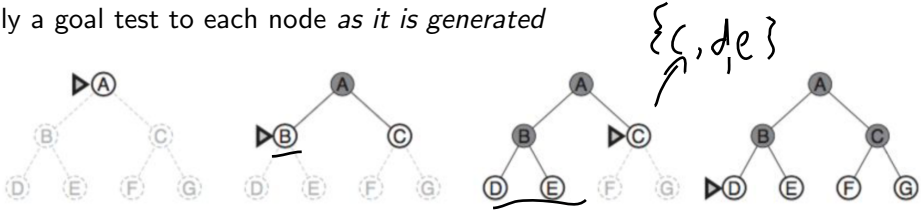
1. **Completeness:** is the algorithm guaranteed to find a solution, when one exists?
2. **Optimality:** is the algorithm guaranteed to find the optimal solution, i.e. that with the lowest path cost?
3. **Time Complexity:** how long does it take - number of operations - to find a solution?
4. **Space complexity:** how much memory is needed to find the solution?

BFS Recap:

Last time we talked about **Breadth-First Searching** (BFS).

Given a tree representing a state space, the BFS search performs the following unfolding of the tree:

1. At each iterative step, expand all nodes at the current depth.
2. Then, proceed to the next layer, in FIFO-style (first in, first out)
3. Apply a goal test to each node *as it is generated*



Explored:

Opening Sol'n: Chicago to Boston

17

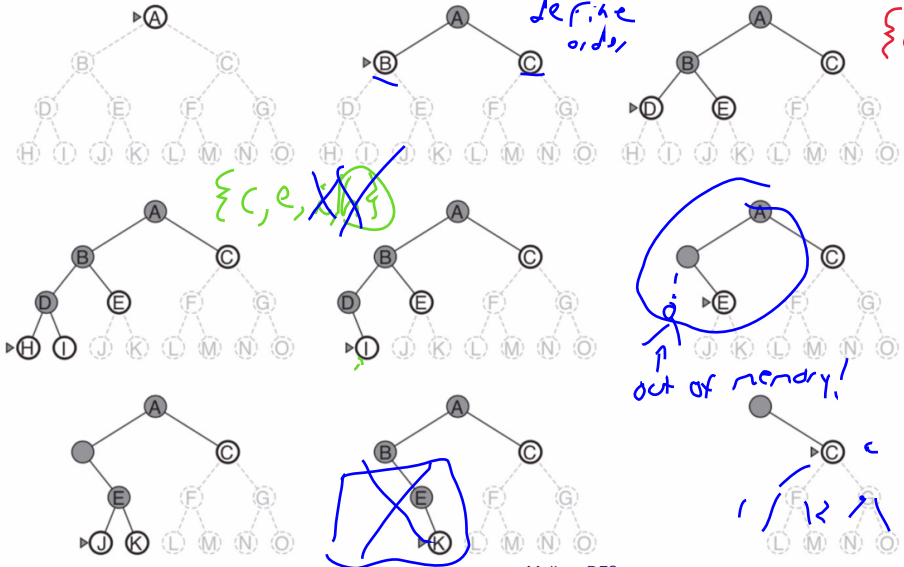
Recall that the BFS solution will find a possibly non-unique shortest path to the goal state.

Now we explore **Depth-First Searching** (DFS).

Given a tree representing a state space, the BFS search performs the following *uninformed* unfolding of the tree:

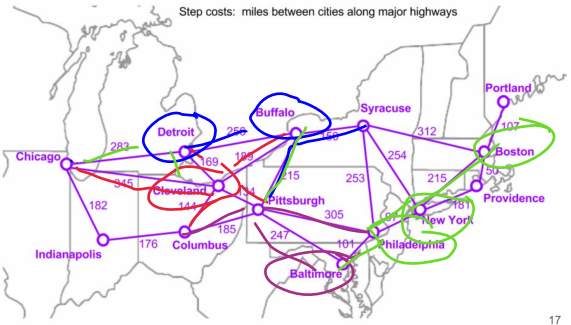
1. At each iterative step, expand the deepest node first (last in, first out).
2. If we reach a redundant state or dead end, "back up" to the next-deepest node with unresolved successors.

DFS Example: Depth 2 Full Binary Tree



DFS Example:

Consider our map problem, with the goal of traveling from Chicago to Boston. How does DFS solve this problem, if states are sorted alphabetically?



DFS Example:

Consider our map problem, with the goal of traveling from Chicago to Boston. How does DFS solve this problem, if states are sorted alphabetically?

0 The root.
- 0a Unfold Chicago, discovering: {Cleveland, Detroit, Indianapolis}.
- 0b Explored set is: {Chicago}.
- 0c Check if those 3 are goal states.

1 Level 1.
- 1a Unfold first state at deepest depth (1), which is again Cleveland, discovering {Buffalo, Pittsburgh, Columbus, Detroit}.
- 1b Explored set is now: {Chicago, Cleveland}.

2 Move to depth 2.
- 2a Unfold first state at deepest depth (2), which is Buffalo, discovering {Syracuse, Detroit, Pittsburgh}.
- 2b Explored set is now: {Chicago, Cleveland, Buffalo}.

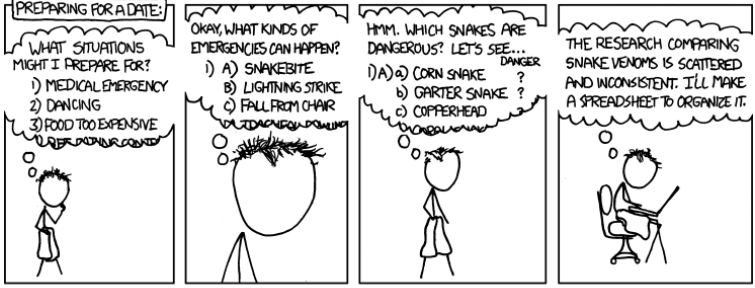
3a Move to depth 3: explore Detroit. All successors are already explored!

3b Back up to depth 3: explore another option from Buffalo: Pittsburgh

4 Move to depth 5: explore Baltimore

5 ...

k Final route: Chicago, Cleveland, Buffalo, Pittsburgh, Baltimore, Philadelphia, New York, Boston.
Only dead end was the trip to Detroit.



I REALLY NEED TO STOP USING DEPTH-FIRST SEARCHES.

tutorials

w: k:

Chain

BFS: a, b, d, h, c, e, i, l, f

DFS practice

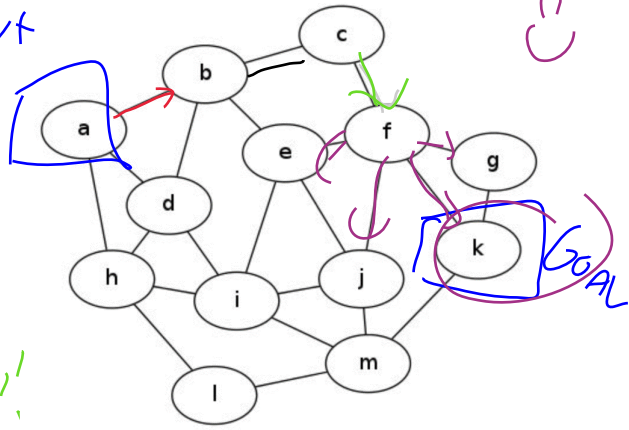
Example: Number the nodes in the search graph according to the order in which they would be expanded using DFS to find a path from a to k. Assume that nodes within a layer are expanded in alphabetical order.

{h, d, b}

{h, ~~b~~, e, d, c}

{h, d, e, d, f}

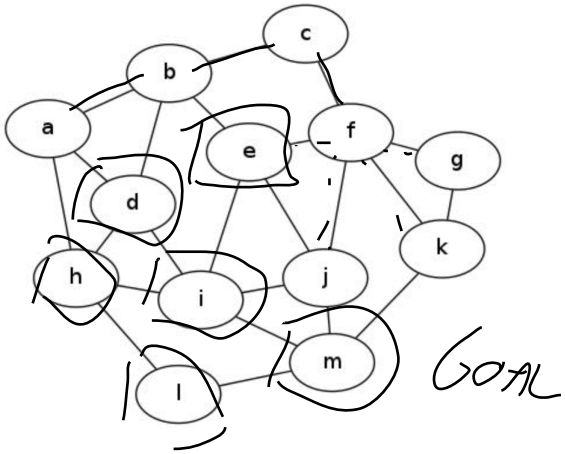
→find goal!



DFS practice

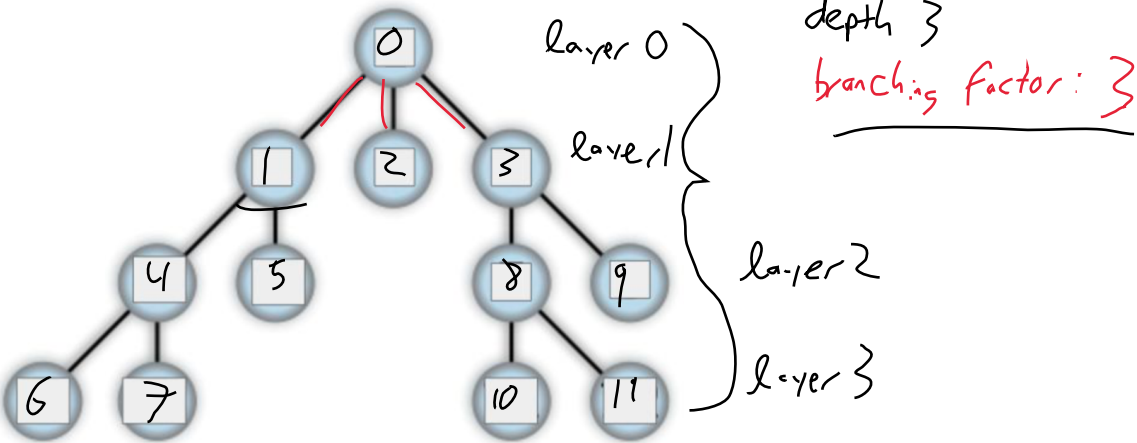
Example: Number the nodes in the search graph according to the order in which they would be expanded using DFS to find a path from a to k. Assume that nodes within a layer are expanded in alphabetical order.

0 Explored: {}. Frontier: {a}.
1 Explored: {a}. Frontier: {d,h,b}.
2 Explored: {a,b}. Frontier: {d,h,e,c}.
3 Explored: {a,b,c}. Frontier: {d,h,e,f}.
4 Explored: {a,b,c,f}. Frontier: {d,h,e,g,j,k}.
5 Goal found! But we would do•e next if it wasn't.



DFS practice

Example: Number the nodes in the search tree according to the order in which they would be expanded using DFS. Assume that the goal is never found, and nodes within each layer expand from left to right.



DFS properties (on all search problems)

path:

CHI-DET-CHI-DET-CHI-DET

weather? time?

Is DFS **complete?**

Is DFS **optimal?**

DFS properties

17

Is DFS **complete?** Maybe! This depends on how well we can avoid redundant paths (what if there are timestamps!?) and whether there is an infinite state space.

Is DFS **optimal?** Definitely not: see prior example on Chicago to Boston. We're also not using edge weights!

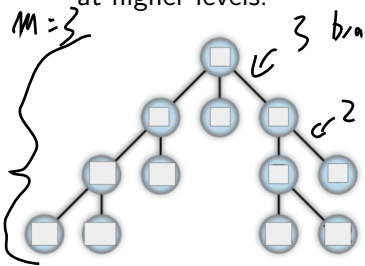
DFS complexity

Time Complexity Suppose as we did for BFS that our DFS problem has *branching factor* b. Suppose that the deepest any tree goes is m, and the shallowest of all solutions is in layer d.

0 Might need to generate all (up to) b^m states, as before.

1 This might substantially slower than just finding the shallowest goal state.

2 Worst-case: $O(b^m)$. The exponential growth rate of a full tree dominates the time spent at higher levels.



DFS complexity

more than half of nodes: on bottom layer

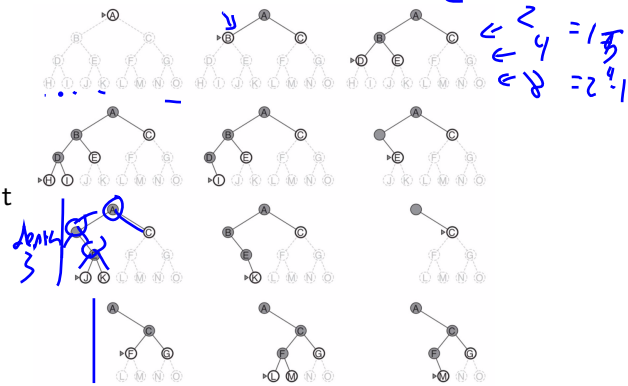
Space Complexity:

A big payoff of DFS here! We only need one branch open at a time.

We open one branch at a time for b nodes, but may have up to m layers open at once: this includes the entire frontier!

DFS is $\boldsymbol{O(mb)}$

both currently exploring & frontier set.



BFS vs DFS

2^{10} vs 10^2 DFS

1024 vs 100

So when do we choose one versus the other?

BFS:

1. **Space:** $O(b^d)$ ← exponential :(
2. **Time:** $O(b^d)$.
3. Definitely best when solution is relatively shallow in the tree small value of d.
4. Avoids pitfalls of possible infinite state spaces
5. Optimal for shortest path, always complete

DFS:

multiplying :)

1. **Space:** $O(mb)$
2. **Time:** $O(b^m)$. m is always greater than or equal to d!
3. Great for problems with large branching factors but less depth.
4. Requires careful definition of redundancy and infinite state spaces to be complete.

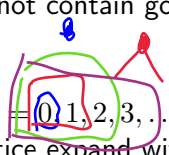
BFS AND DFS

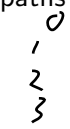
Two "best of both worlds"

Depth-limited DFS:

1. Expand nodes in the typical DFS order, until...
2. Once you expand nodes at fixed maximal depth l stop expanding and move to deepest unexplored branch of depth $l-1$ or less.
3. Purpose: memory savings of DFS with guaranteed solution of BFS when "deep" paths are inefficient or known to not contain good solutions.

Iterative Deepening Search:





1. Do depth-limited DFS for $l = 0, 1, 2, 3, \ldots$. Increasing the depth limit each step of the way means that we in practice expand with breadth for higher levels...
2. while saving memory on ever holding that breadth in memory

capped by b^l memory

3. Cost: redundancy! If the solution is at depth 5, we open the root of the tree 5 times (and level 1 4 times, and so forth)

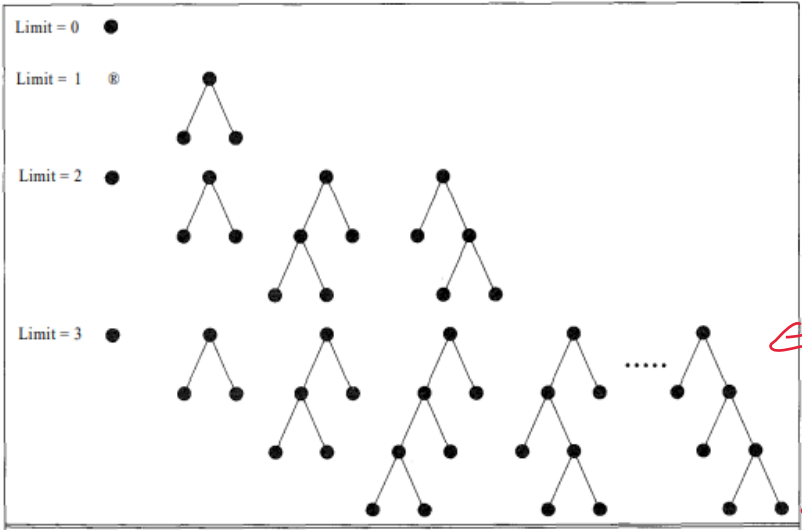
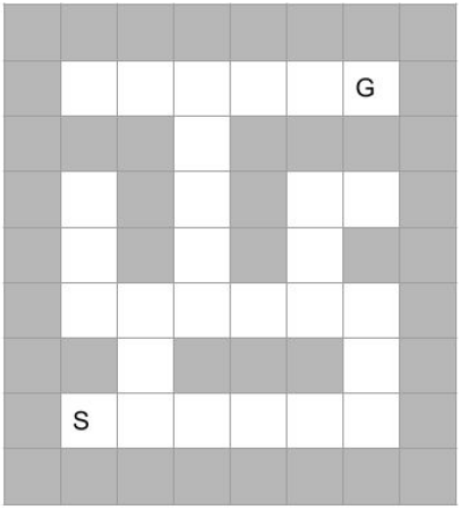


Figure 3.16 Four iterations of iterative deepening search on a binary tree.

# of nodes
1 ← repeat 3x
2 ← repeat 2x
4 ← repeat 1x
8
8 + 4(1) + 2(2) + 1(3) = 19

Redundancy not always that bad in terms of algorithmic complexity: the areas with most

Search is a-maze-ingly useful!



Imagine telling an agent (Roomba?) to navigate this maze. We have either BFS or DFS, and we decide to "explore" actions in a clockwise order (N, E, S, W).

(or 4+bi;d)

$b=2 \quad \ell=20$

$b=4$

$2^{20} + 2(2^{19}) + 3(2^{18}) + 4(2^{17}) \ldots$

$2(2^{20}) \qquad 10^{20} + 2(10)^{19} + 3(10)^{18}$

Moving Forward

- This week:
  1. HW1 Due Friday: main issue seems to be challenges with typing of your getChildren() function. Check the couple of relevant Piazza posts.
  2. HW 2 for probable release around the weekend.
- Next time: Friday lecture (not a nb day this week!) on **uniform-cost** searching!