Uninformed Search

CMPUT 366: Intelligent Systems

P&M §3.5

Logistics

- TA office hours begin this week
  - See eClass page for times and meeting links
- Assignment #1 released next week

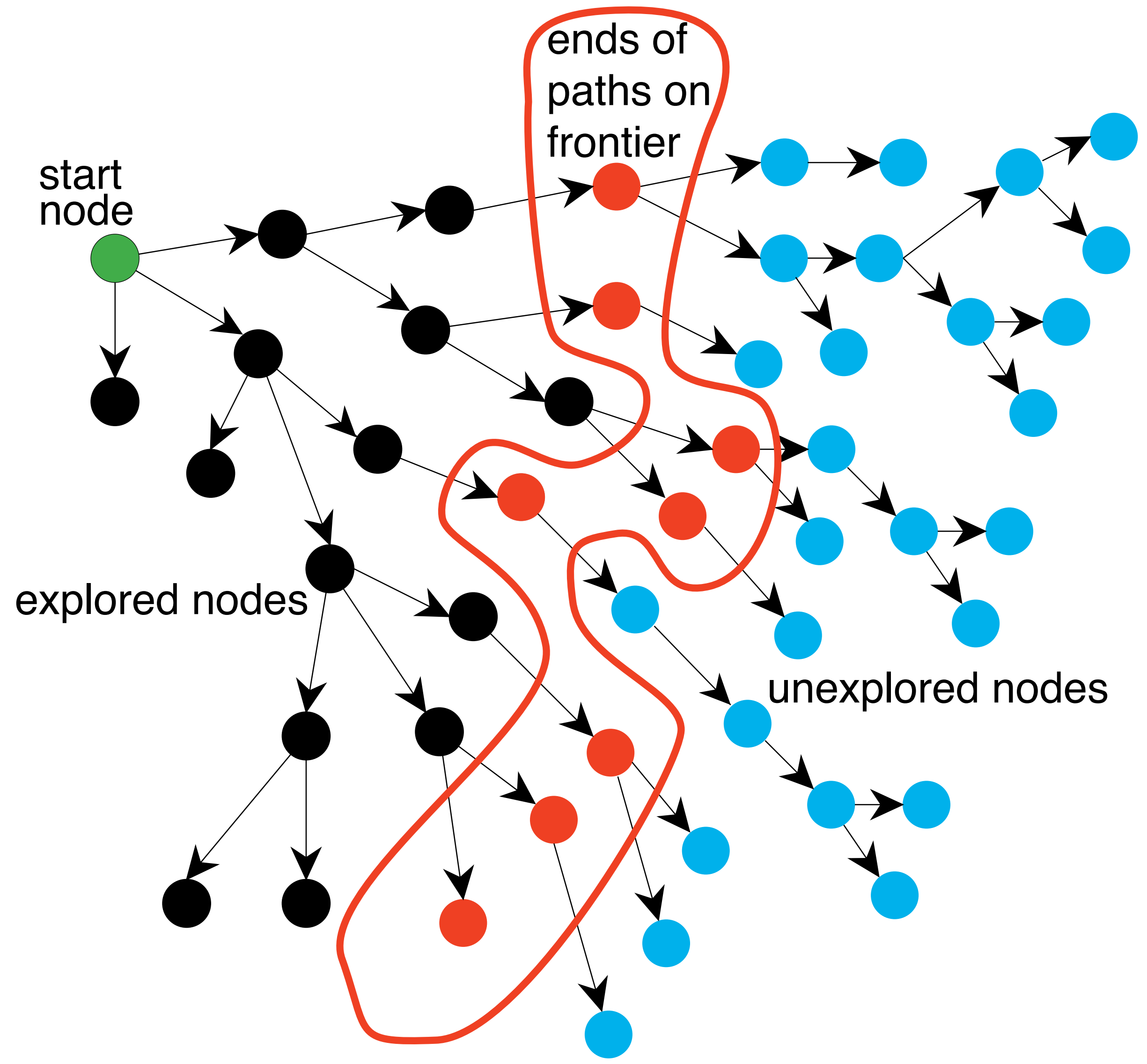
Recap: Graph Search

- Many AI tasks can be represented as **search problems**
  - A single generic **graph search algorithm** can then solve them all!
- A search problem consists of **states**, **actions**, **start states**, a **successor function**, a **goal** function, optionally a **cost** function
- **Solution quality** can be represented by labelling **arcs** of the search graph with **costs**

Recap: Generic Graph Search Algorithm

Input: a *graph*; a set of *start nodes*; a *goal* function

frontier := { <*s*> | *s* is a start node}
while *frontier* is not empty:
  select a path $<n_1, n_2, ..., n_k>$ from *frontier*
  remove $<n_1, n_2, ..., n_k>$ from *frontier*
  if *goal*(n_k):
    return $<n_1, n_2, ..., n_k>$
  for each neighbour *n* of n_k: (i.e., **expand** node n_k)
    add $<n_1, n_2, ..., n_k, n>$ to *frontier*
end while



https://artint.info/2e/html/ArtInt2e.Ch3.S4.html

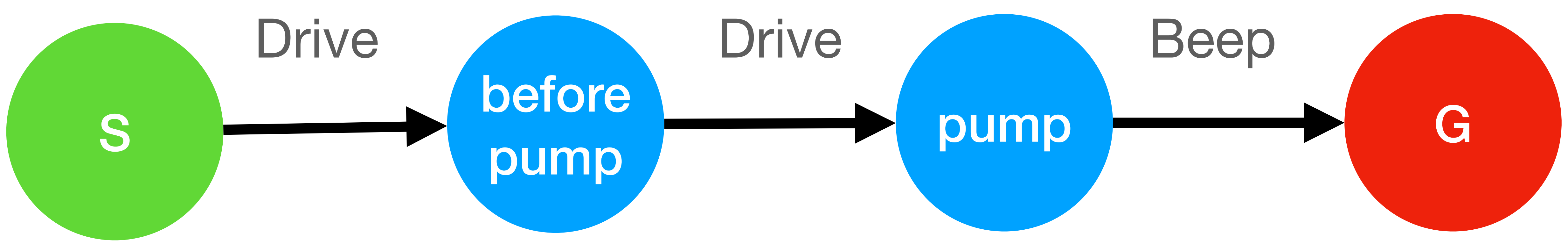
- Which value is **selected** from the frontier defines the **search strategy**

Lecture Outline

1. Logistics & Recap
2. Markov Assumption
3. Properties of Algorithms and Search Graphs
4. Depth First Search
5. Breadth First Search

Markov Assumption: GasBot

The **Markov assumption** is **crucial** to the graph search algorithm



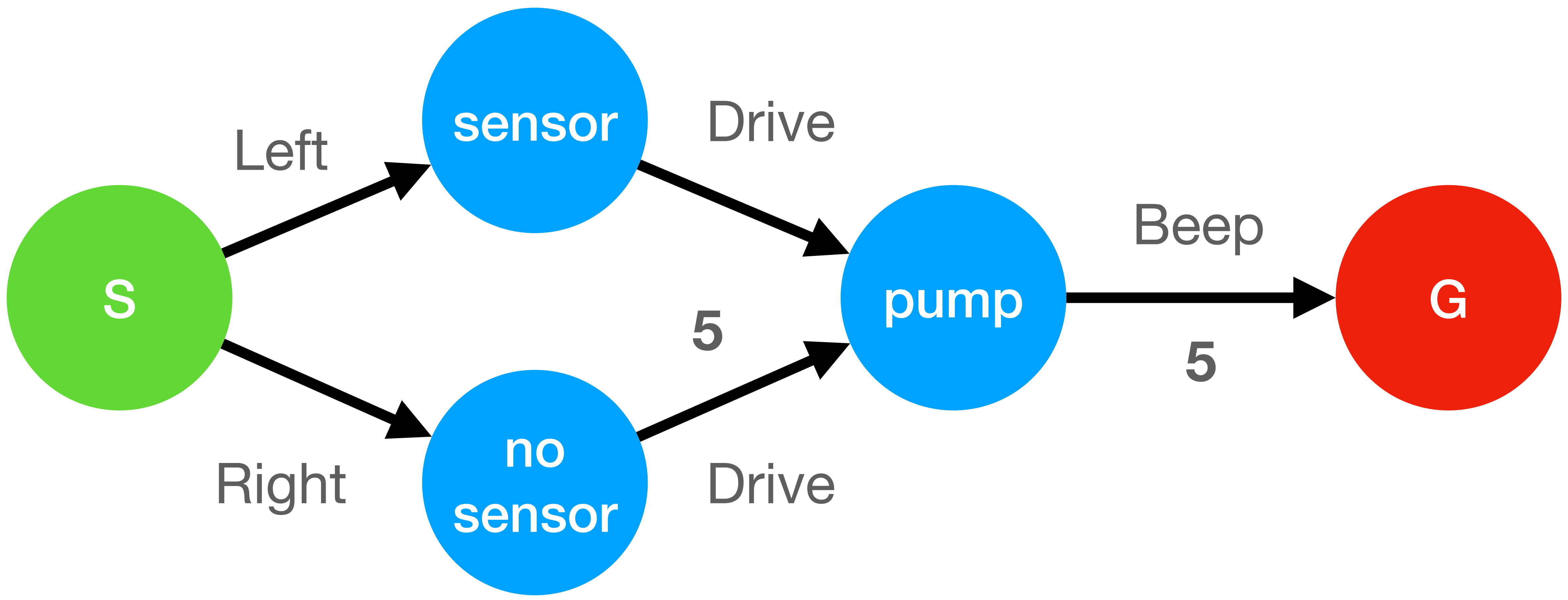
cost(pump, gas):
5 if went through sensor
10 otherwise

Getting to the pump:
from the **left** goes through sensor
from the **right** does not

Question: Does this environment satisfy the Markov assumption? Why or why not?

Markov Assumption: GasBot

The **Markov assumption** is **crucial** to the graph search algorithm



1. Does *this* environment satisfy the Markov assumption? Why or why not?

2. How *else* could we have fixed up the previous example?

Algorithm Properties

What properties of algorithms do we want to analyze?

- A search algorithm is **complete** if it is guaranteed to find a solution within a finite amount of time whenever a solution exists.
- The **time complexity** of a search algorithm is a measure of how much **time** the algorithm will take to run, in the **worst case**.
    - In this section we measure by number of paths **added to the frontier**.
- The **space complexity** of a search algorithm is a measure of how much **space** the algorithm will use, in the **worst case**.
    - We measure by maximum number of paths **in the frontier**.

Search Graph Properties

What properties of the **search graph** do algorithmic properties depend on?

- **Forward branch factor**: Maximum number of neighbours
  Notation: b
- **Maximum path length**. (Could be infinite!)
  Notation: m
- Presence of **cycles**
- Length of the **shortest** path to a **goal** node

Depth First Search

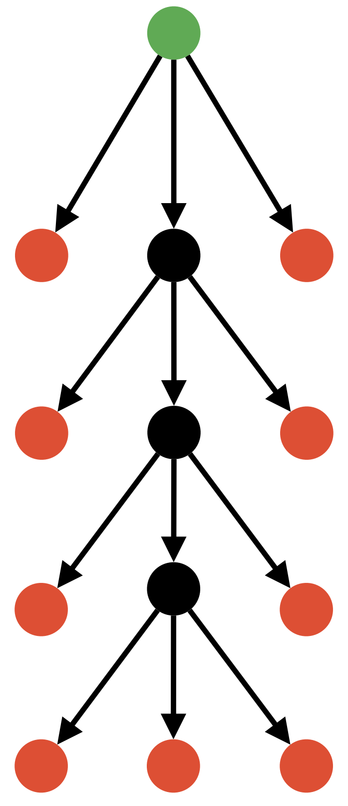
Input: a *graph*; a set of *start nodes*; a *goal* function

frontier := { <*s*> | *s* is a start node}
while *frontier* is not empty:
    select the newest path <n_1, n_2, ..., n_k> from *frontier*
    remove <n_1, n_2, ..., n_k> from *frontier*
    if *goal*(n_k):
        return <n_1, n_2, ..., n_k>
    for each neighbour *n* of n_k:
        add <n_1, n_2, ..., n_k, *n*> to *frontier*
end while

Question:

What **data structure** for the frontier implements this search strategy?

Depth First Search



Depth-first search always removes one of the **longest** paths from the frontier.

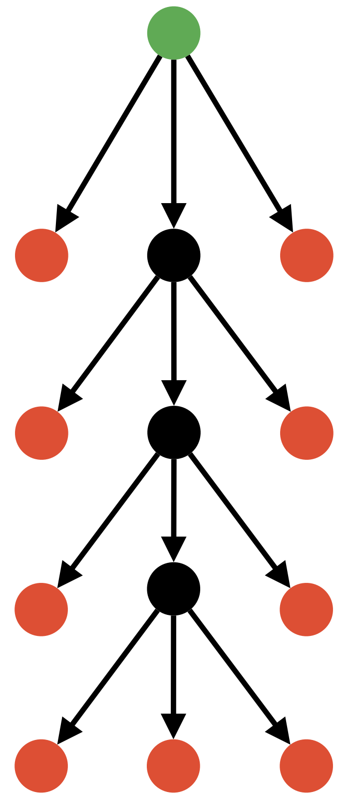
Example:
Frontier: $[p_1, p_2, p_3, p_4]$
successors(p_1) = $\{n_1, n_2, n_3\}$

What happens?

1. Remove p_1; test p_1 for goal
2. Add $\{<p_1,n_1>, <p_1,n_2>, <p_1,n_3>\}$ to **front** of frontier
3. New frontier: $[<p_1,n_1>, <p_1,n_2>, <p_1,n_3>, p_2, p_3, p_4]$
4. $p2$ is selected only after **all paths starting with p_1** have been explored

Question: When is $<p_1,n_3>$ selected?

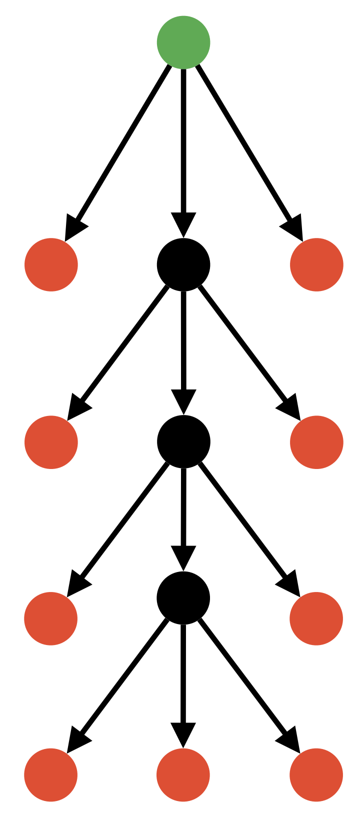
Depth First Search Analysis



For a search graph with maximum branch factor *b* and maximum path length *m*...

1. What is the worst-case **time complexity**?
   - [A: $O(m)$] [B: $O(mb)$] [C: $O(b^m)$] [D: it depends]
2. When is depth-first search **complete**?
3. What is the worst-case **space complexity**?
   - [A: $O(m)$] [B: $O(mb)$] [C: $O(b^m)$] [D: it depends]

When to Use Depth First Search



- When is depth-first search **appropriate**?
  - Memory is restricted
  - All solutions at same approximate depth
  - Order in which neighbours are searched can be tuned to find solution quickly
- When is depth-first search **inappropriate**?
  - Infinite paths exist
  - When there are likely to be shallow solutions
    - Especially if some other solutions are very deep

Breadth First Search

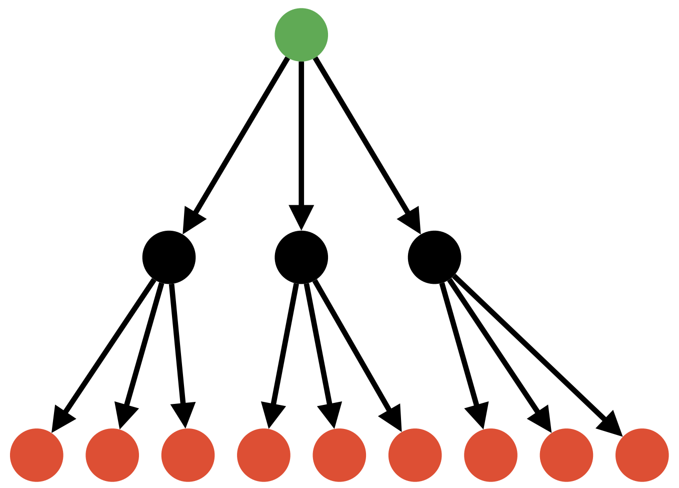
Input: a *graph*; a set of *start nodes*; a *goal* function

frontier := { <*s*> | *s* is a start node}
while *frontier* is not empty:
  select the oldest path <n_1, n_2, ..., n_k> from *frontier*
  remove <n_1, n_2, ..., n_k> from *frontier*
  if *goal*(n_k):
    return <n_1, n_2, ..., n_k>
  for each neighbour *n* of n_k:
    add <n_1, n_2, ..., n_k, *n*> to *frontier*
end while

Question:

What **data structure** for the frontier implements this search strategy?

Breadth First Search



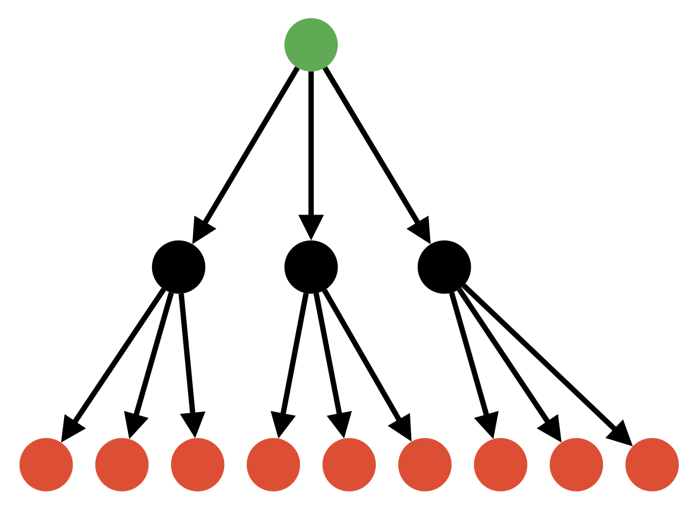
Breadth-first search always removes one of the **shortest** paths from the frontier.

Example:
Frontier: $[p_1, p_2, p_3, p_4]$
successors(p_1) = $\{n_1, n_2, n_3\}$

What happens?

1. Remove p_1; test p_1 for goal
2. Add $\{<p_1,n_1>, <p_1,n_2>, <p_1,n_3>\}$ to **end** of frontier:
3. New frontier: $[p_2, p_3, p_4, <p_1,n_1>, <p_1,n_2>, <p_1,n_3>,]$
4. p_2 is selected **next**

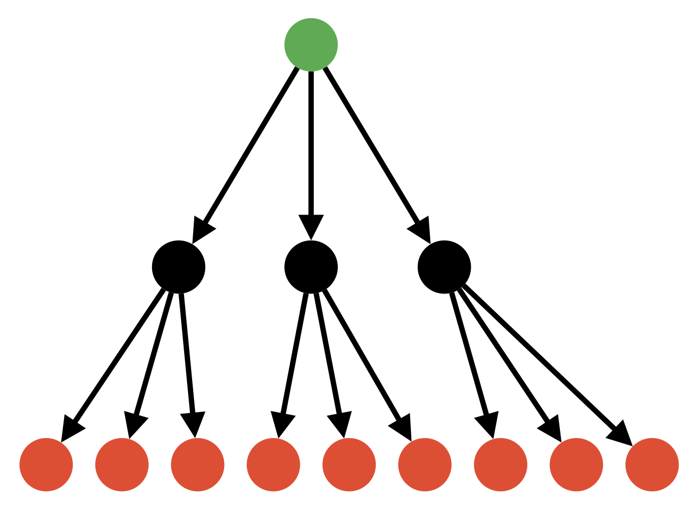
Breadth First Search Analysis



For a search graph with maximum branch factor *b* and maximum path length *m*...

1. What is the worst-case **time complexity**?
   - [A: $O(m)$] [B: $O(mb)$] [C: $O(b^m)$] [D: it depends]
2. When is breadth-first search **complete**?
3. What is the worst-case **space complexity**?
   - [A: $O(m)$] [B: $O(mb)$] [C: $O(b^m)$] [D: it depends]

When to Use Breadth First Search



- When is breadth-first search **appropriate?**
  - When there might be infinite paths
  - When there are likely to be shallow solutions, *or*
  - When we want to guarantee a solution with fewest arcs
- When is breadth-first search **inappropriate?**
  - Large branching factor
  - All solutions located deep in the tree
  - Memory is restricted

Comparing DFS vs. BFS

	Depth-first	Breadth-first
Complete?	Only for finite graphs	Complete
Space complexity	$O(mb)$	$O(b^m)$
Time complexity	$O(b^m)$	$O(b^m)$

- Can we get the space benefits of depth-first search without giving up completeness?
- Run depth-first search to a maximum depth
  - then try again with a larger maximum
  - until either goal found or graph completely searched

Iterative Deepening Search

Input: a *graph*; a set of *start nodes*; a *goal* function

for *max_depth* from 1 to ∞:
    Perform **depth-first search** to a maximum depth *max_depth*
end for