

# COMP 472 Artificial Intelligence

## State Space Search *part #3*

### Uninformed Search *video #2*

- Russell & Norvig - Section 3.4

- see also: <https://www.javatpoint.com/ai-uninformed-search-algorithms>

# Today

1. State Space Representation

2. State Space Search

a) Overview

YOU ARE HERE!

b) Uninformed search

1. Breadth-first and Depth-first

2. Depth-limited Search

3. Iterative Deepening

4. Uniform Cost

c) Informed search

1. Intro to Heuristics

2. Hill climbing

3. Best-First


4. Algorithms A & A\*

5. More on Heuristics

d) Summary

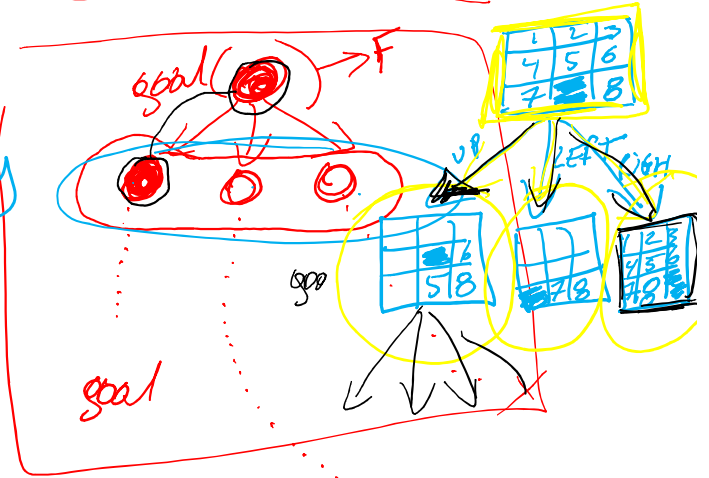
# Uninformed VS Informed Search

## ■ Uninformed search

- all nodes are equally promising, so we explore them systematically
  - aka: systematic/blind/brute force search
  - many algorithms:
- 
- A hand-drawn diagram showing a function  $goal()$  in red ink. A red circle is drawn around the word "goal", and an arrow points from the circle to the letter "F".

1. Breadth-first search
2. Depth-first search
3. Uniform-cost search
4. Depth-limited search
5. Iterative deepening search
6. ...

352  
video today

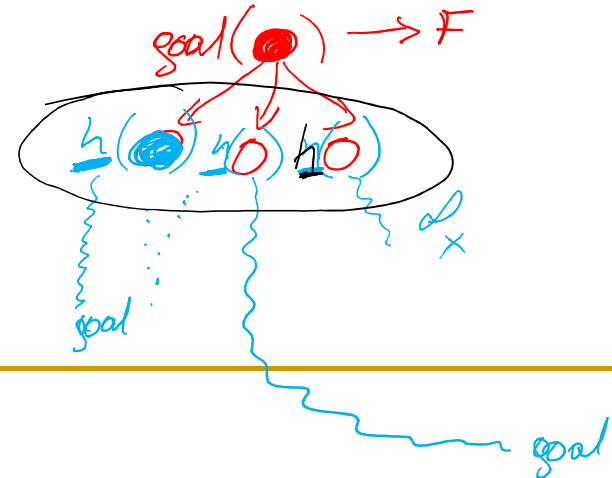


- Informed search (heuristic search)

- we try to identify which nodes seem more promising, and explore these first

- many algorithms:

1. Hill climbing
2. Greedy Best-First search
3. algorithms A and A\*
4. ...

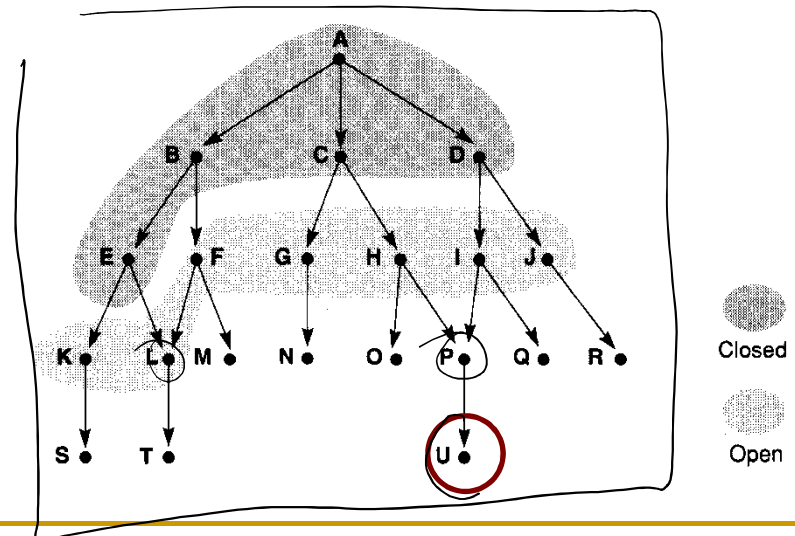


# Data Structures

- Most search strategies require:
  - open list (aka the frontier) *To-DO*
    - lists generated nodes not yet expanded
    - order of nodes controls order of search
  - closed list (aka the explored set)
    - stores all the nodes that have already been visited (to avoid cycles).
- ex:

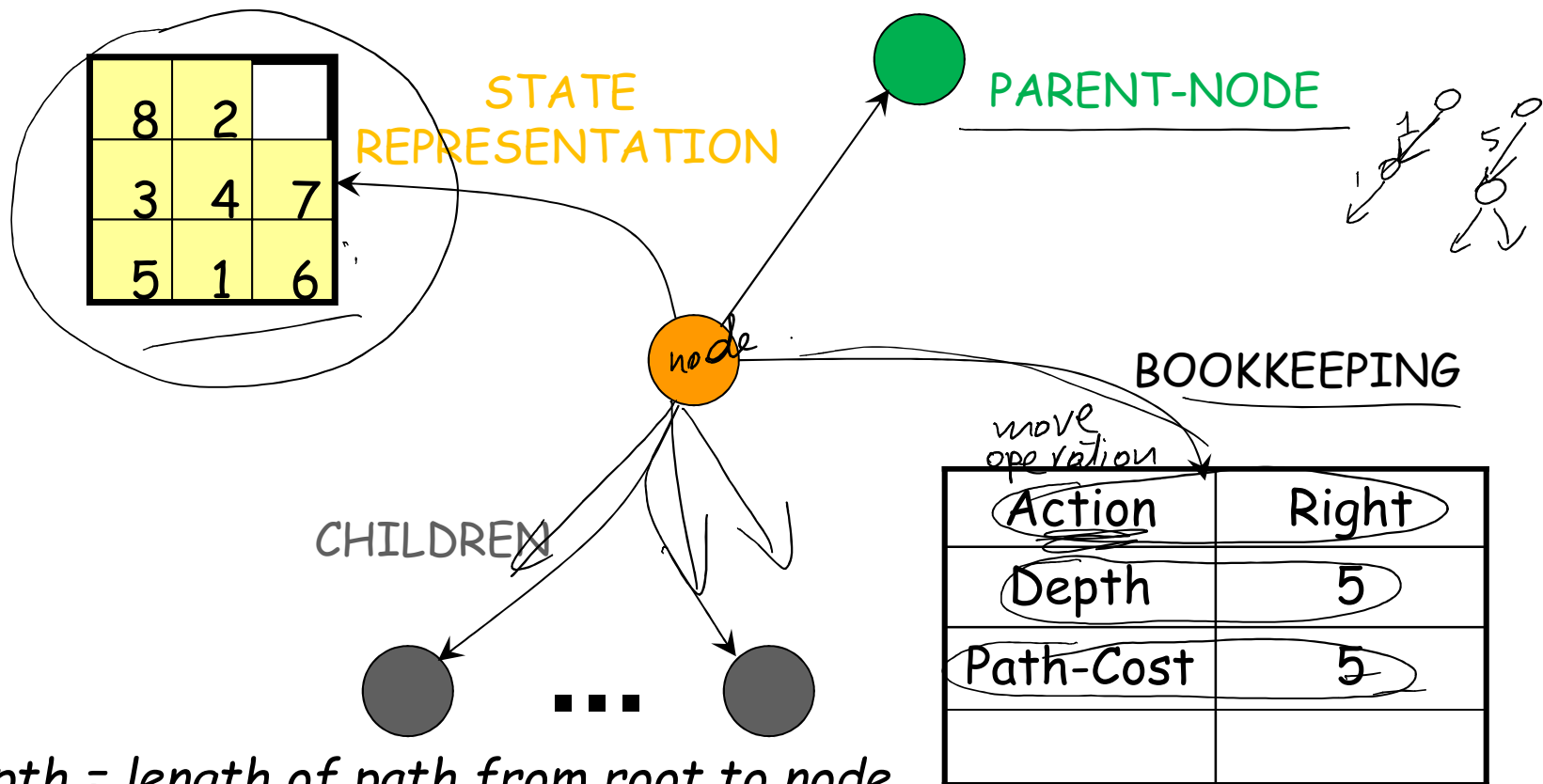
Closed = [A, B, C, D, E]

Open = [F, G, H, I, J, K, L]



# Data Structures

- state space representation: To trace back the solution path after the search, each node in the lists contain:



# Generic Search Algorithm

1. Initialize the open list with the initial node  $s_0$  (top node) { TO-DO
2. Initialize the closed list to empty { done / visited
3. Repeat
  - a) If the open list is empty, then exit with failure.
  - b) Else, take the first node  $s$  from the open list.
  - c) If  $s$  is a goal state, exit with success. Extract the solution path from  $s$  to  $s_0$ .
  - d) Else, insert  $s$  in the closed list ( $s$  has been visited / expanded)
  - e) Insert the successors of  $s$  in the open list in a certain order if they are not already in the closed and/or open lists (to avoid cycles)

## Notes:

- The order of the nodes in the open list depends on the search strategy

# Today

1. State Space Representation

2. State Space Search

a) Overview

b) Uninformed search

1. Breadth-first Search and Depth-first Search

2. Depth-limited Search

3. Iterative Deepening

4. Uniform Cost

c) Informed search

1. Intro to Heuristics

2. Hill climbing

3. Best-First

4. Algorithms A & A\*

5. More on Heuristics

d) Summary

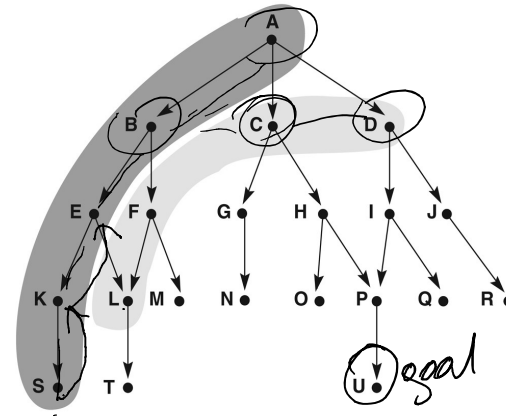


YOU ARE HERE!

# Depth-first vs Breadth-first Search

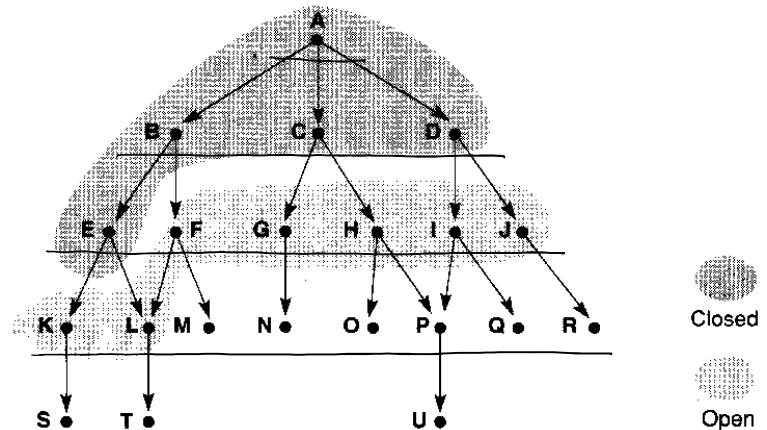
## ■ Depth-first (DFS):

- visit successors before siblings
- Open list is a stack



## ■ Breadth-first (BFS):

- visit siblings before successors
- ie. visit level-by-level
- open list is a queue



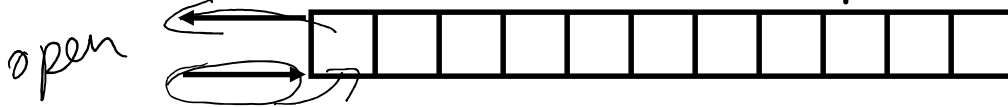


# DFS and BFS

- DFS and BFS differ only in the way they order nodes in the open list:

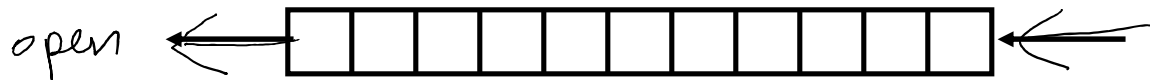
- DFS uses a stack:

- nodes are added on the top of the <sup>open</sup> list.



- BFS uses a queue:

- nodes are added at the end of the list.



# Breadth-First Search

```
begin
  open := [Start];
  closed := [ ];
  while open ≠ [ ] do
    begin
      remove leftmost state from open, call it X;
      if X is a goal then return SUCCESS
      else begin
        generate children of X;
        put X on closed;
        discard children of X if already on open or closed;
        put remaining children on right end of open
      end
    end
  end
  return FAIL
end.
```

% initialize

% states remain

% goal found

% loop check

% queue

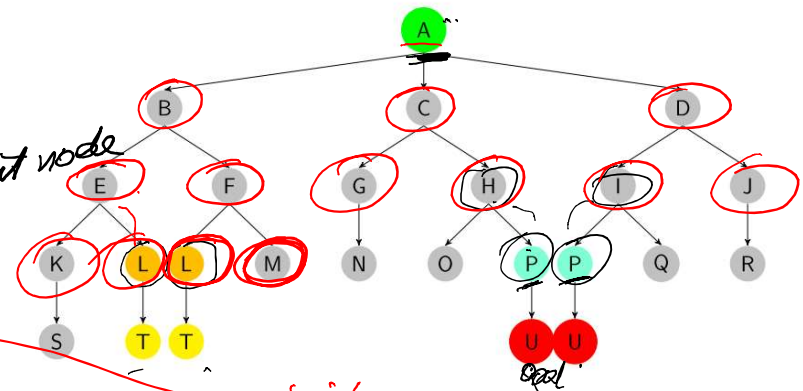
% no states left

# Breadth-First Search Example

## ■ BFS: (open is a queue)

Assume U is goal state

*name of node*  
*name of parent node*



1. open = [A-null] closed = []
2. open = [B-A, C-A, D-A] closed = [A]
3. open = [C-A, D-A, E-B, F-B] closed = [B, A]
4. open = [D-A, E-B, F-B, G-C, H-C] closed = [C, B, A]
5. open = [E-B, F-B, G-C, H-C, I-D, J-D] closed = [D, C, B, A]
6. open = [F-B, G-C, H-C, I-D, J-D, K-E, L-E] closed = [E, D, C, B, A]
7. open = [G-C, H-C, I-D, J-D, K-E, L-E, M-F] as L is already in open closed = [F, E, D, C, B, A]
8. and so on until either U is found or open = []

*visit*  $\Rightarrow$  run goal function

$\rightarrow$  { search path = A B C D E F G H I J K L M N ... U // closed list  
 solution path = A C H P U with a cost of 4 // extract by following the parent pointer

# Depth-First Search

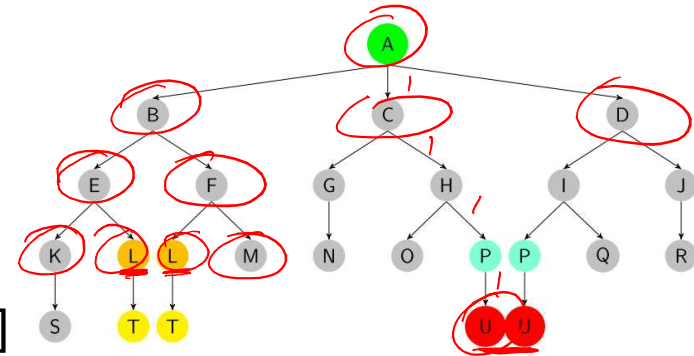
```
begin
  open := [Start];                                % initialize
  closed := [ ];
  while open ≠ [ ] do                             % states remain
    begin
      remove leftmost state from open, call it X;
      if X is a goal then return SUCCESS           % goal found
      else begin
        generate children of X;
        put X on closed;
        discard children of X if already on open or closed;
        put remaining children on left end of open % loop check
      end
    end
  end;
  return FAIL                                     % no states left
end.
```

# Depth-First Search Example

## ■ DFS: (open is a stack)

Assume U is goal state

1. open = [A-null] closed = []
2. open = [B-A C-A D-A] closed [A]
3. open = [E-B F-B C-A D-A] closed = [B A]
4. open = [K-E L-E F-B C-A D-A] closed = [E B A]
5. open = [S-K L-E F-B C-A D-A] closed = [K E B A]
6. open = [L-E F-B C-A D-A] closed = [S K E B A]
7. open = [T-L F-B C-A D-A] closed = [L S K E B A]
8. open = [F-B C-A D-A] closed = [T L S K E B A]
9. open = [M-F C-A D-A] as L is already on closed closed = [F T L S K E B A]
10. open = [C-A D-A] closed = [M F T L S K E B A]
11. open = [G-C H-C D-A] closed = [C M F T L S K E B A]



search path = A B E K S L ... U // closed list

solution path = A C H P U with a cost of 4

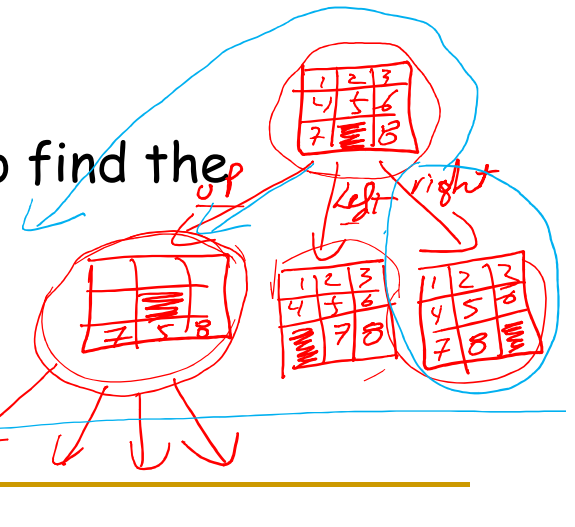
# Depth-first vs. Breadth-first

- Breadth-first:

- advantage: optimal, i.e. will always find shortest path <sup>solution</sup>
- disadvantage:
  - high memory requirement as we need to keep all states of a level before expanding to the next level
  - exponential space for states required  $B^n$  // B=branching factor, n=level <sub>average # of successors</sub>

- Depth-first:

- advantage: Requires less memory
- disadvantage: Not optimal (no guarantee to find the shortest path)



# Today

1. State Space Representation
2. State Space Search
  - a) Overview
  - b) Uninformed search
    1. Breadth-first Search and Depth-first Search ✓
    2. Depth-limited Search
    3. Iterative Deepening
    4. Uniform Cost
  - c) Informed search
    1. Intro to Heuristics
    2. Hill climbing
    3. Best-First
    4. Algorithms A & A\*
    5. More on Heuristics
  - d) Summary



YOU ARE HERE!

# Depth-Limited Search

- Compromise for DFS :
  - Do depth-first but
  - with depth cutoff  $k$  (depth at which nodes are not expanded)
- Three possible outcomes:
  - Solution *within your  $k$  limit :)*
  - Failure (no solution)
  - Cutoff (no solution found within cutoff) ✗
- advantage: memory efficient - it's a DFS
- disadvantage: may not find a solution if it is below the cutoff



# Today

1. State Space Representation
2. State Space Search
  - a) Overview
  - b) Uninformed search
    1. Breadth-first search and Depth-first search
    2. Depth-limited Search
    3. Iterative Deepening
    4. Uniform Cost
  - c) Informed search
    1. Intro to Heuristics
    2. Hill climbing
    3. Best-First
    4. Algorithms A & A\*
    5. More on Heuristics
  - d) Summary

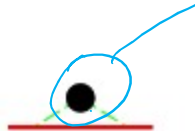


# Iterative Deepening

- Combination of BFS and DFS:
  - do depth-first search, but
  - with a maximum depth before going to next level
  - i.e. Repeats depth first search with gradually increasing depth limits
- advantage:
  - Requires little memory (fundamentally, it's a depth first)
  - optimal: will find the shortest path (limited depth)
- disadvantage:
  - repeated traversal of the tree top

# Iterative Deepening: Example

Limit = 0

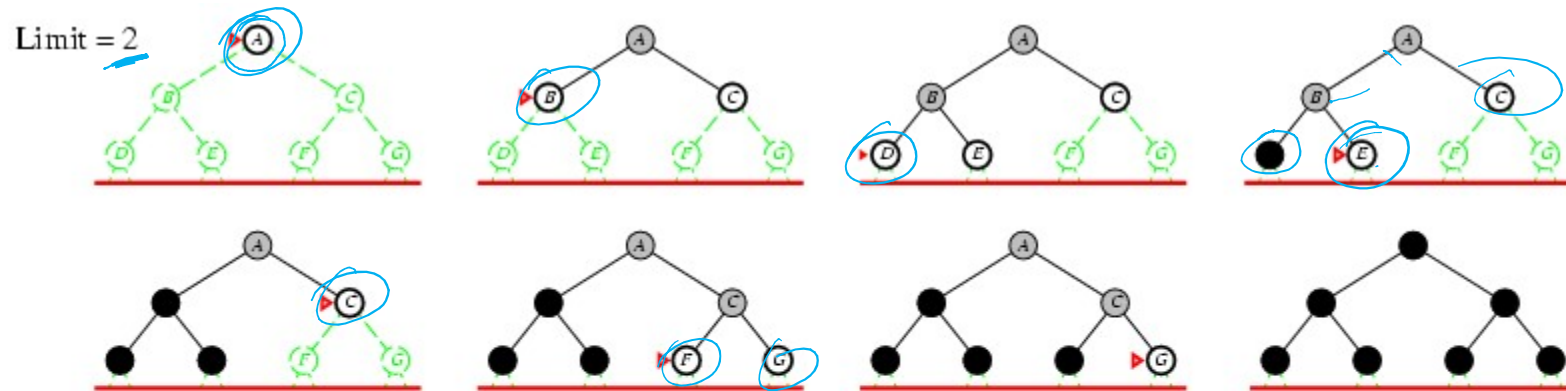


black node  
 $\Rightarrow$  node has been  
visited

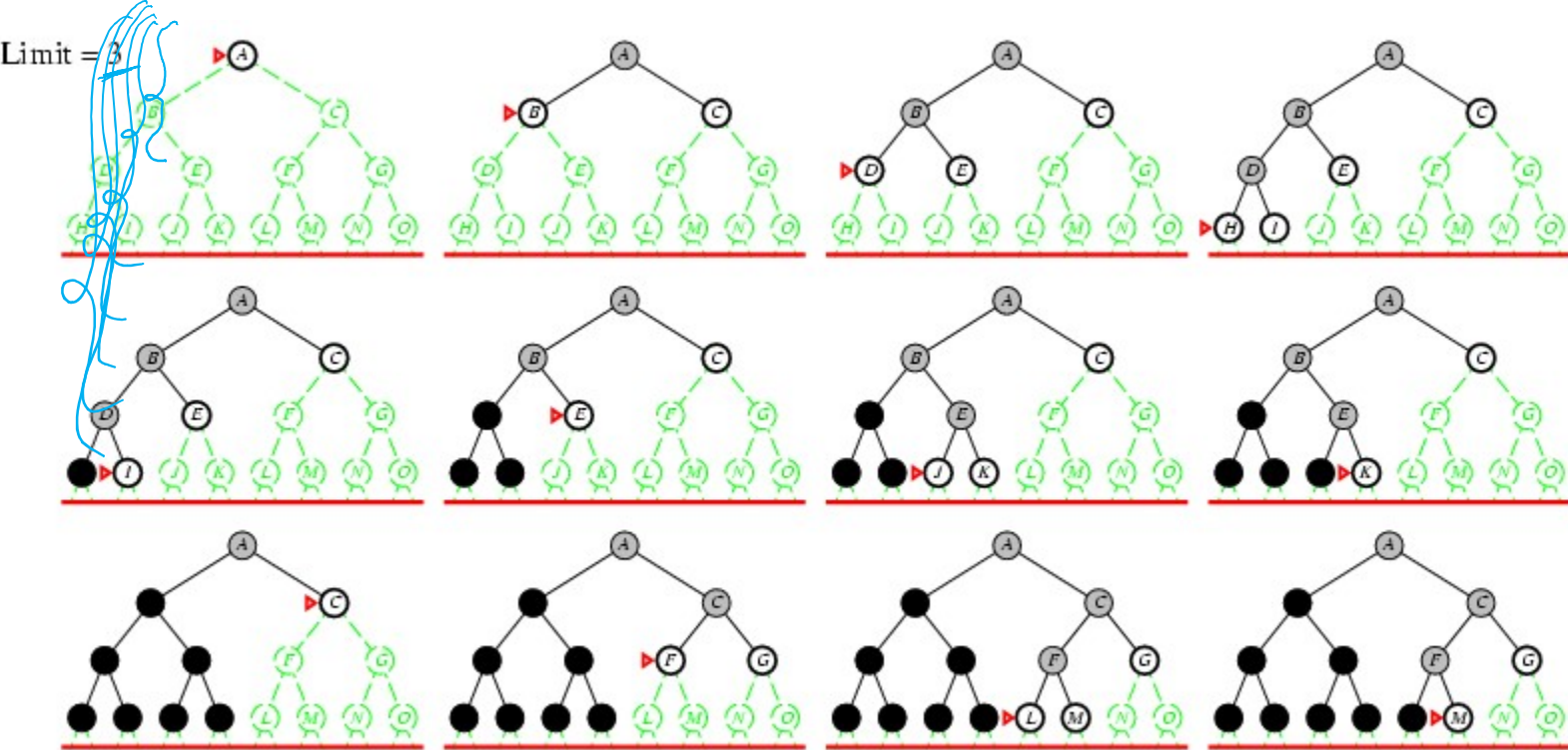
# Iterative Deepening: Example



# Iterative Deepening: Example



## Iterative Deepening: Example



# Today

1. State Space Representation
2. State Space Search
  - a) Overview
  - b) Uninformed search
    1. Breadth-first and Depth-first
    2. Depth-limited Search
    3. Iterative Deepening
    4. Uniform Cost
  - c) Informed search
    1. Intro to Heuristics
    2. Hill climbing
    3. Best-First
    4. Algorithms A & A\*
    5. More on Heuristics
  - d) Summary



YOU ARE HERE!

# Uniform Cost Search

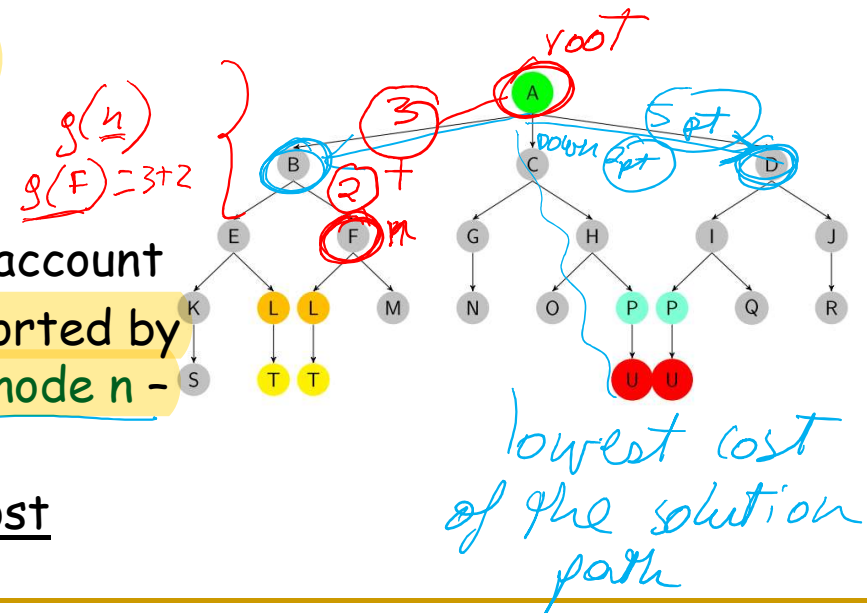
- all algorithms so far assume that all edges have the same cost
- but what if they have different costs?
  - eg: move UP  $\rightarrow$  2pts but move DOWN  $\rightarrow$  1 pt
  - eg: cost(residential road)  $>$  cost(commercial road)

## Breadth First Search

- uses OPEN as a priority queue sorted by the depth of nodes
- guarantees to find the shortest solution path

## Uniform Cost Search

- takes the cost of the edge into account
- uses OPEN as a priority queue sorted by the total cost from the root to node  $n$  - later we will call this  $g(n)$
- guarantees to find the lowest cost solution path





Example

~~open~~

closed

$$\{P_{51} \quad E_{59} \quad D_{53}\}$$
$$\{P_{S1}, D_{S3}, E_{S9}\}$$
 $\{Q_{p/6} \quad D_{s3} \quad E_{s4}$ 

$D_{53}$   ~~$E_{89}$~~   $Q_{P16}$

$\{B_{D4}, C_{D11}, E_{D5}, Q_{D16}\}$

B<sub>D4</sub> E<sub>D5</sub> C<sub>D11</sub> Q<sub>P16</sub>

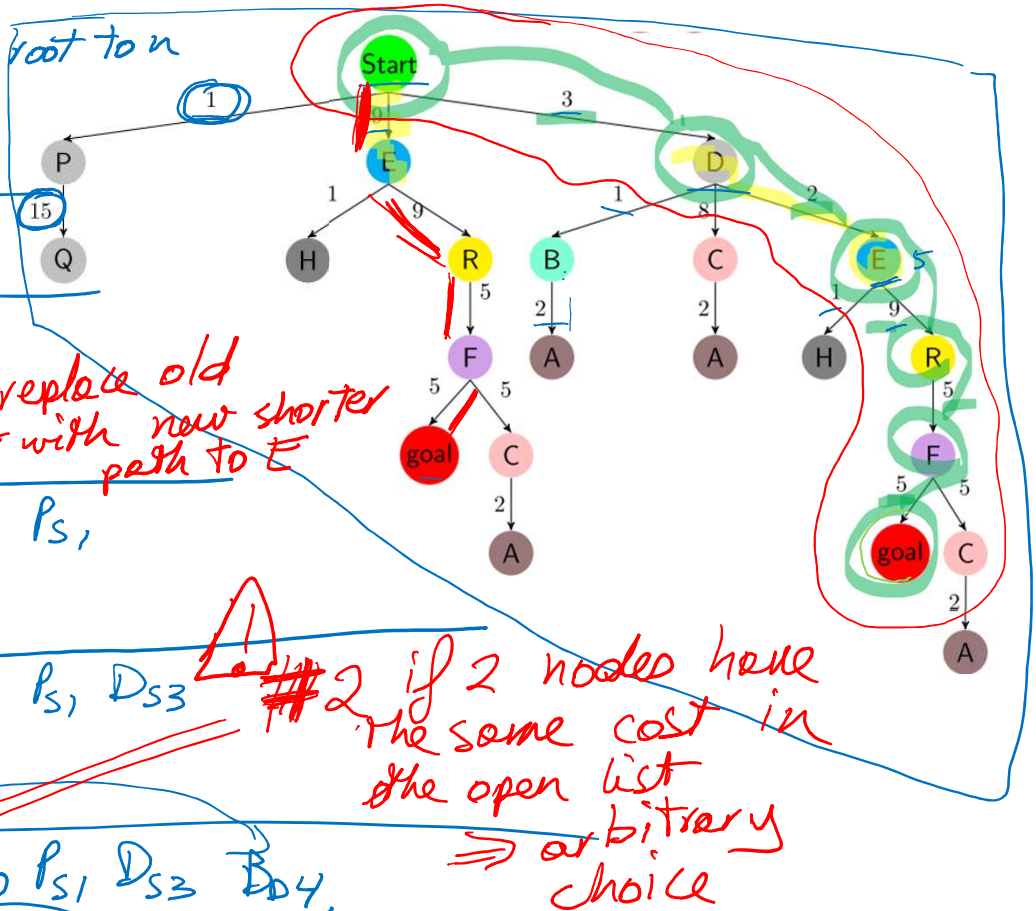
$\{ A_{B6} E_{D5} C_{D11} Q_{P16}$

$\{E_{D5}, AB_6, CD_{11}, QP_{16}\}$

$\{H_{EE}, R_{E,4}, AB_6, CD_{11}, QP_{16}\}$

$HE_6 \quad AB_6 \quad CD_{11} \quad RE_{14} \quad \underline{QP_{16}}$

Solution on path : Goal F R E D S cost of 24



I skipped  
the last  
steps

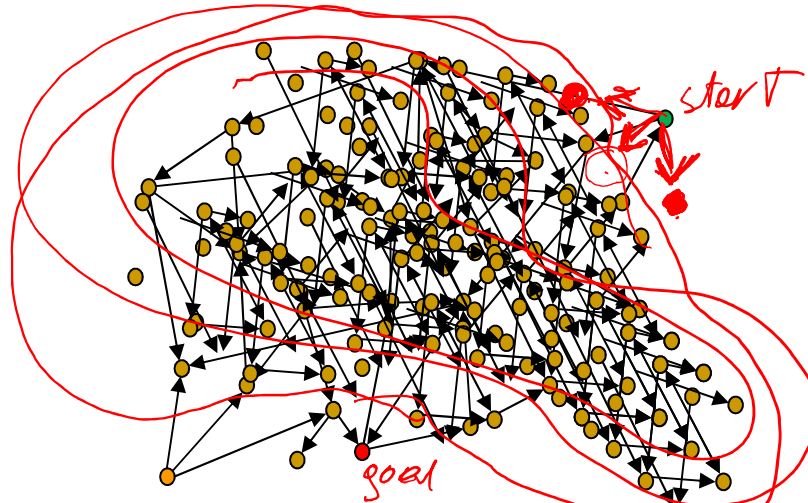
# Today

1. State Space Representation ✓
2. State Space Search ✓
  - a) Overview ✓
  - b) Uninformed search ✓
    1. Breadth-first and Depth-first ✓
    2. Depth-limited Search ✓
    3. Iterative Deepening ✓
    4. Uniform Cost ✓
  - c) Informed search
    1. Intro to Heuristics
    2. Hill climbing
    3. Best-First
    4. Algorithms A & A\*
    5. More on Heuristics
  - d) Summary

352

# Problem with Uninformed Search

- inefficient for most AI problems, the state space is too large!
  - e.g. state space of all possible moves in chess =  $10^{120}$
  - $10^{75}$  = nb of molecules in the universe
  - $10^{26}$  = nb of nanoseconds since the "big bang"



- we need a way to try the most promising nodes first

# Up Next

1. State Space Representation
2. State Space Search
  - a) Overview
  - b) Uninformed search
    1. Breadth-first and Depth-first
    2. Depth-limited Search
    3. Iterative Deepening
    4. Uniform Cost
  - c) **Informed search**
    1. Intro to Heuristics  $h(n)$
    2. Hill climbing
    3. Best-First
    4. Algorithms A & A\*
    5. More on Heuristics
  - d) Summary

