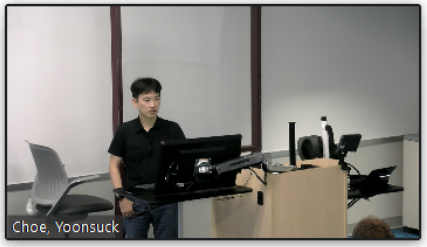
Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On



General Search Algorithm

Pseudo-code:

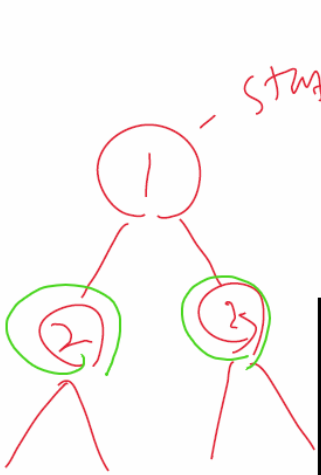
```
function General-Search (problem, Que-Fn)
    node-list ← [ Make-Node(initial-state) ]
    loop begin
        # 1. fail if node-list is empty
        if Empty(node-list) then return FAIL
        # 2. fetch node from node-list: This is a visit
        node ← Get-First-Node(node-list)
        # 3. if picked node is a goal node, success!
        if (node == goal) then return node as solution
        # otherwise, expand node and enqueue
        node-list ← Que-Fn(node-list, Expand(node))
    loop end
```

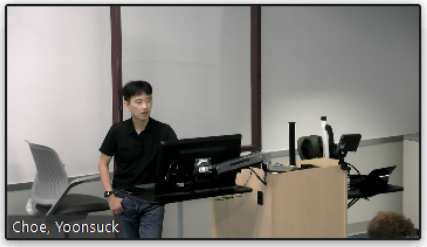
- node: state, plus book-keeping info (current depth, path to current state. etc.)
- Expand(node): returns set of child nodes

Then we turn fail, so that's the one of the the first termination condition

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

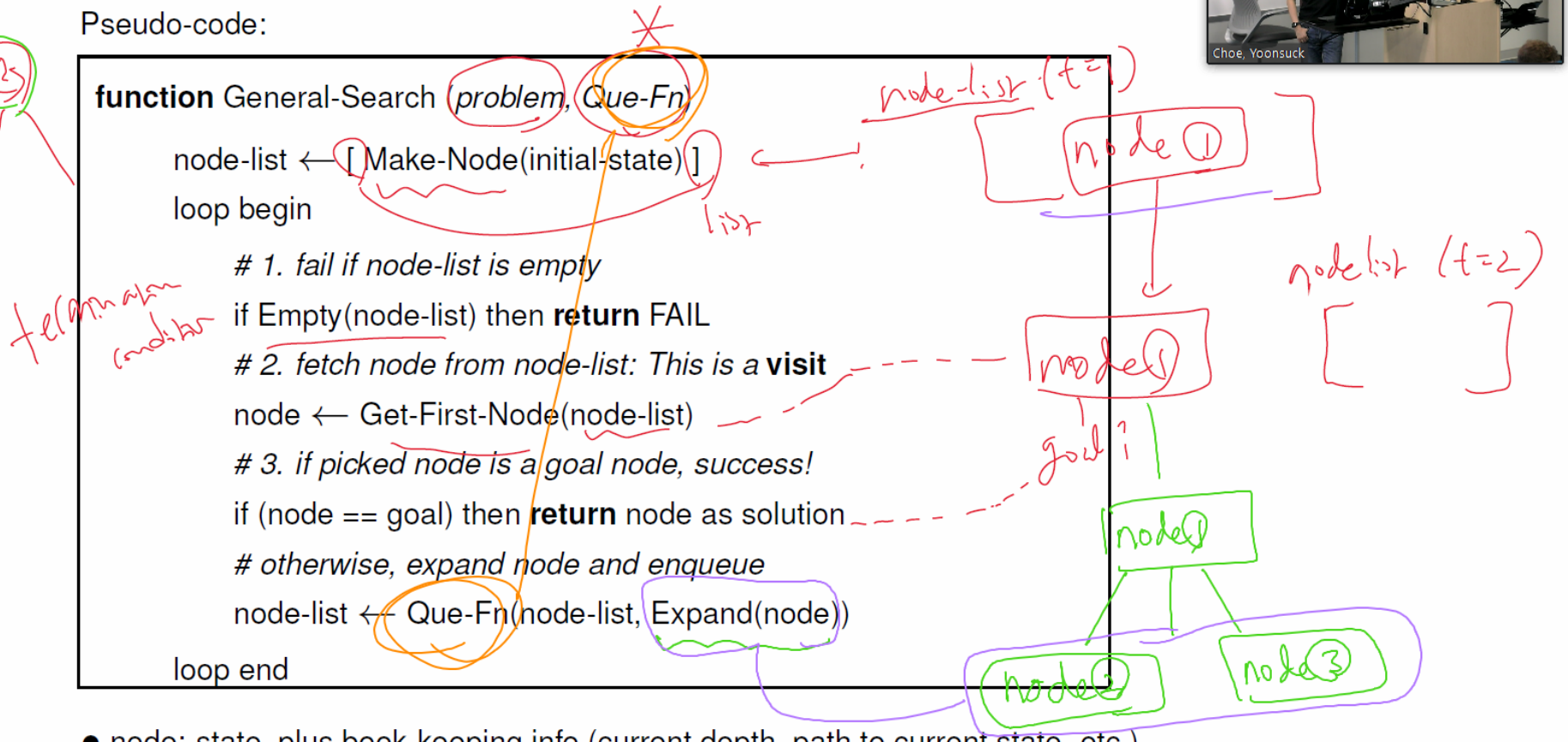




General Search Algorithm

Pseudo-code:

```
function General-Search (problem, Que-Fn)
    node-list ← [ Make-Node(initial-state) ]
    loop begin
        # 1. fail if node-list is empty
        if Empty(node-list) then return FAIL
        # 2. fetch node from node-list: This is a visit
        node ← Get-First-Node(node-list)
        # 3. if picked node is a goal node, success!
        if (node == goal) then return node as solution
        # otherwise, expand node and enqueue
        node-list ← Que-Fn(node-list, Expand(node))
    loop end
```

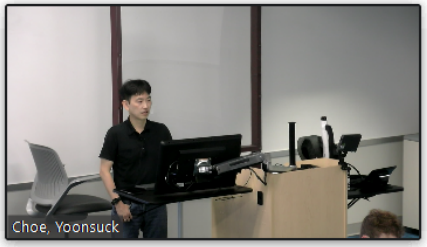


- node: state, plus book-keeping info (current depth, path to current state. etc.)
- Expand(node): returns set of child nodes

So this is where the this comes in

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On



General Search Algorithm

Pseudo-code:

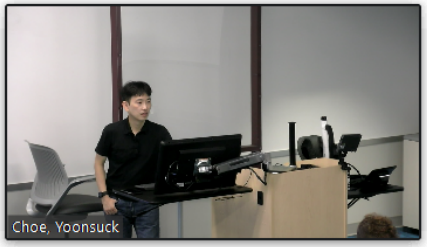
```
function General-Search (problem, Que-Fn)

    node-list ← [ Make-Node(initial-state) ]
    loop begin
        # 1. fail if node-list is empty
        if Empty(node-list) then return FAIL
        # 2. fetch node from node-list: This is a visit
        node ← Get-First-Node(node-list)
        # 3. if picked node is a goal node, success!
        if (node == goal) then return node as solution
        # otherwise, expand node and enqueue
        node-list ← Que-Fn(node-list, Expand(node))
    loop end
```

- node: state, plus book-keeping info (current depth, path to current state, etc.)
- Expand(node): returns set of child nodes

So after you took this out, what happens the nood list is it'll only now contain no 3 in it

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On



General Search Algorithm

Pseudo-code:

```
function General-Search (problem, Que-Fn)

    node-list ← [ Make-Node(initial-state) ]
    loop begin
        # 1. fail if node-list is empty
        if Empty(node-list) then return FAIL
        # 2. fetch node from node-list: This is a visit
        node ← Get-First-Node(node-list)
        # 3. if picked node is a goal node, success!
        if (node == goal) then return node as solution
        # otherwise, expand node and enqueue
        node-list ← Que-Fn(node-list, Expand(node))
    loop end
```

- node: state, plus book-keeping info (current depth, path to current state, etc.)
- Expand(node): returns set of child nodes

Well, do you plug it in in the end
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

General Search Algorithm

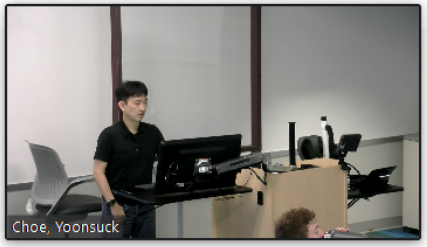
Pseudo-code:

```
function General-Search (problem, Que-Fn)

    node-list ← [ Make-Node(initial-state) ]
    loop begin
        # 1. fail if node-list is empty
        if Empty(node-list) then return FAIL
        # 2. fetch node from node-list: This is a visit
        node ← Get-First-Node(node-list)
        # 3. if picked node is a goal node, success!
        if (node == goal) then return node as solution
        # otherwise, expand node and enqueue
        node-list ← Que-Fn(node-list, Expand(node))
    loop end
```

- node: state, plus book-keeping info (current depth, path to current state, etc.)
- Expand(node): returns set of child nodes

4 right so that'd be the case for with this
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On



General Search Algorithm

Pseudo-code:

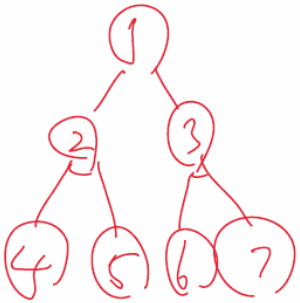
```
function General-Search (problem, Que-Fn)

    node-list ← [ Make-Node(initial-state) ]
    loop begin
        # 1. fail if node-list is empty
        if Empty(node-list) then return FAIL
        # 2. fetch node from node-list: This is a visit
        node ← Get-First-Node(node-list)
        # 3. if picked node is a goal node, success!
        if (node == goal) then return node as solution
        # otherwise, expand node and enqueue
        node-list ← Que-Fn(node-list, Expand(node))
    loop end
```

- node: state, plus book-keeping info (current depth, path to current state, etc.)
- Expand(node): returns set of child nodes

Because this would be like that

You are screen sharing Stop Share

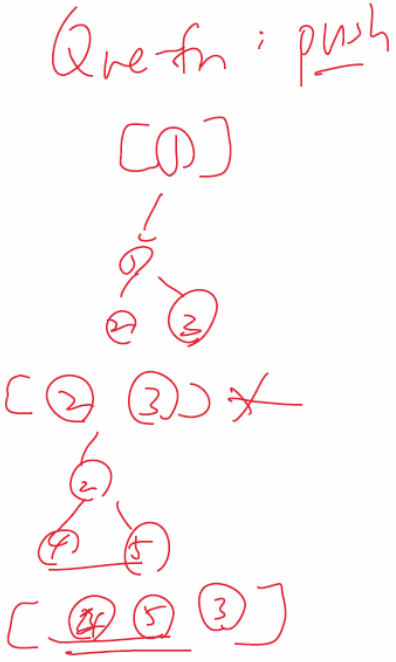
Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

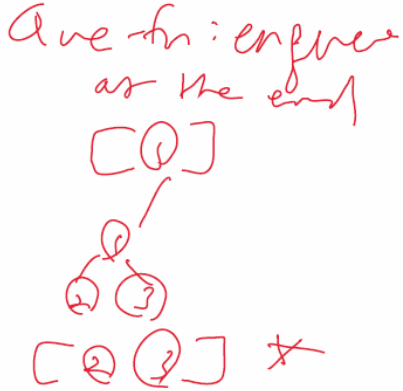


General Search Algorithm

Pseudo-code:

```
function General-Search (problem, Que-Fn)

    node-list ← [ Make-Node(initial-state) ]
    loop begin
        # 1. fail if node-list is empty
        if Empty(node-list) then return FAIL
        # 2. fetch node from node-list: This is a visit
        node ← Get-First-Node(node-list)
        # 3. if picked node is a goal node, success!
        if (node == goal) then return node as solution
        # otherwise, expand node and enqueue
        node-list ← Que-Fn(node-list, Expand(node))
    loop end
```

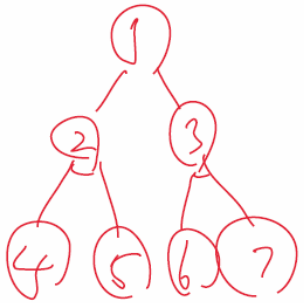




- node: state, plus book-keeping info (current depth, path to current state. etc.)
- Expand(node): returns set of child nodes

Is this empty? So these 2 are the same right
You are screen sharing Stop Share

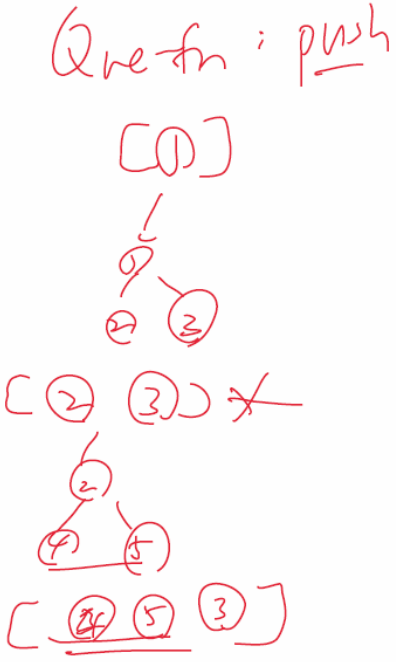
Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

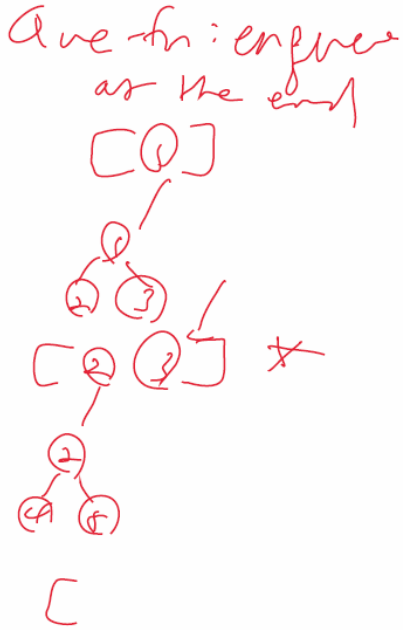


General Search Algorithm

Pseudo-code:

```
function General-Search (problem, Que-Fn)

    node-list ← [ Make-Node(initial-state) ]
    loop begin
        # 1. fail if node-list is empty
        if Empty(node-list) then return FAIL
        # 2. fetch node from node-list: This is a visit
        node ← Get-First-Node(node-list)
        # 3. if picked node is a goal node, success!
        if (node == goal) then return node as solution
        # otherwise, expand node and enqueue
        node-list ← Que-Fn(node-list, Expand(node))
    loop end
```

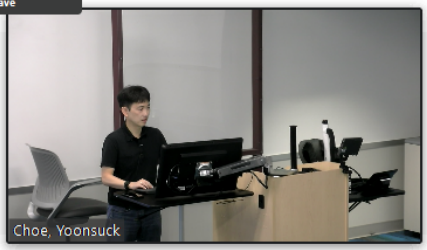




- node: state, plus book-keeping info (current depth, path to current state. etc.)
- Expand(node): returns set of child nodes

Then here, now you have to put it in the back
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On



General Search Algorithm

Pseudo-code:

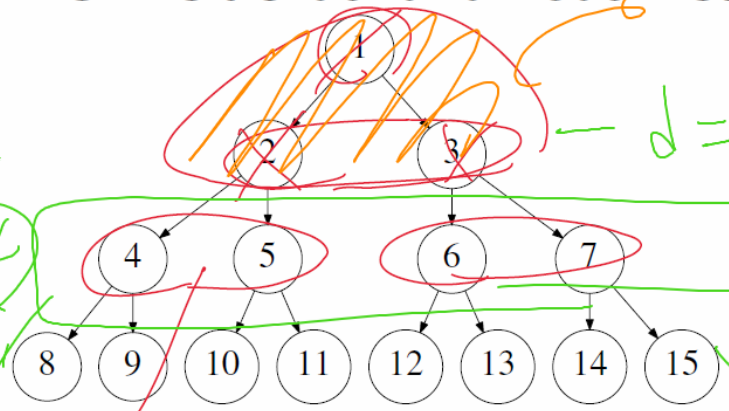
```
function General-Search (problem, Que-Fn)

    node-list ← [ Make-Node(initial-state) ]
    loop begin
        # 1. fail if node-list is empty
        if Empty(node-list) then return FAIL
        # 2. fetch node from node-list: This is a visit
        node ← Get-First-Node(node-list)
        # 3. if picked node is a goal node, success!
        if (node == goal) then return node as solution
        # otherwise, expand node and enqueue
        node-list ← Que-Fn(node-list, Expand(node))
    loop end
```

- node: state, plus book-keeping info (current depth, path to current state. etc.)
- Expand(node): returns set of child nodes

Right, so
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

BFS: Visit Order and Node List



Evolution of the queue (**bold**= expanded and added children):

1. [1] : initial node list contains the initial state
2. [**[2][3]**] : visit 1 (dequeued) and enqueue 2 and 3
3. [[3] **[4][5]**] : visit 2 (dequeued) and enqueue 4 and 5
4. [[4] [5] **[6][7]**] : visit 3 (dequeued) and enqueue 6 and 7. All depth 2 nodes are in the node list.

...

8. [[8] [9] [10] [11] [12] [13] **[14][15]**] : visit 7, ..., All depth 3 nodes are in the node list.

So once you have gone through this steps, you have 2 to them plus one things in your node list, So that's a huge number

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

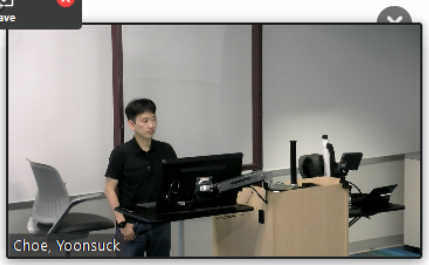
Search and Game Playing

- Uninformed search
- Informed search ← where AI gets involved.
- Iterative improvement, Constraint satisfaction
- Game playing

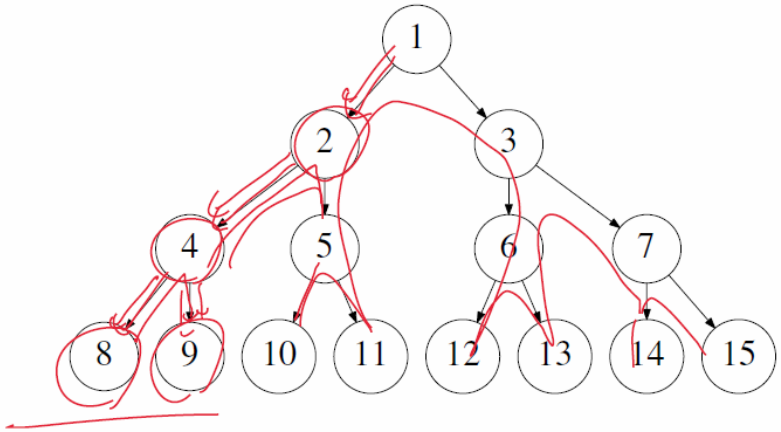
So again, this is where Ai, hey? I gets involved

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On



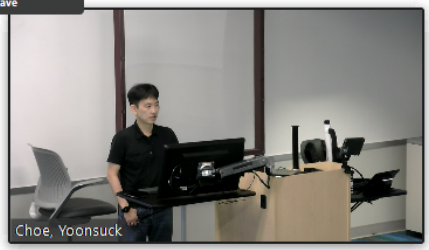
Depth First Search



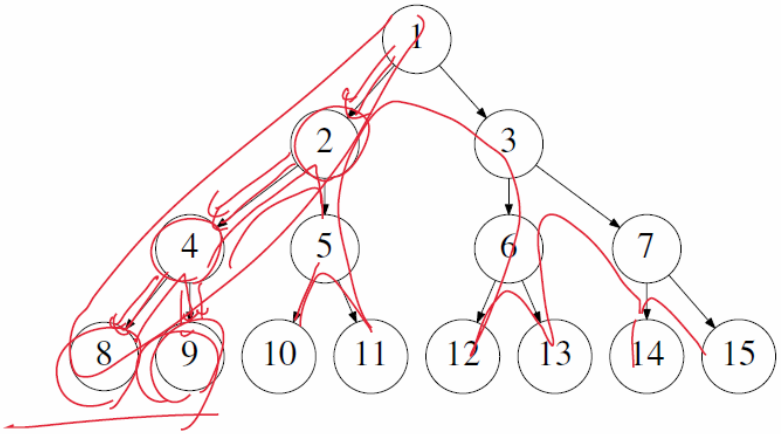
- node visit order (fetch/goal-test): 1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
- queuing function: enqueue at left (stack push; add expanded nodes at the beginning of the list)

And it will basically go like that
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On



Depth First Search

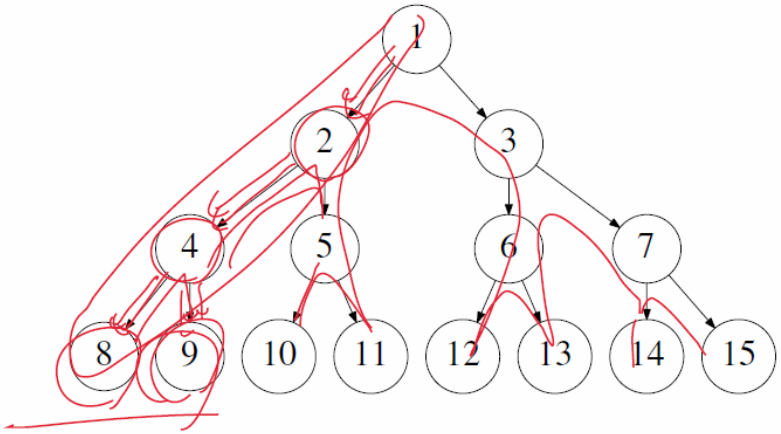


- node visit order (fetch/goal-test): 1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
- queuing function: enqueue at left (stack push; add expanded nodes at the beginning of the list)

The 1, 2, 4, 8, 1, 2, 4, 8, and then 9, and so
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

Depth First Search

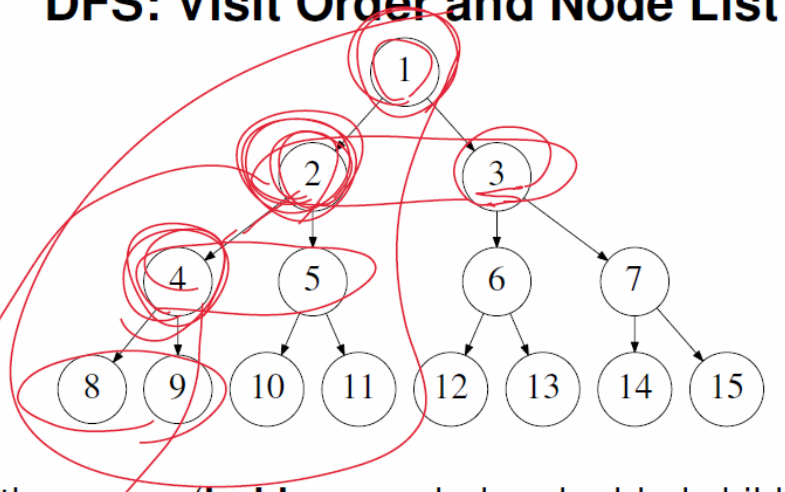


- node visit order (fetch/goal-test): 1 2 4 8 9 5 10 11 3 6 12 13 7 14 15
- queuing function: enqueue at left (stack push; add expanded nodes at the beginning of the list)

At the left, or push in the stack stack, push right
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

DFS: Visit Order and Node List



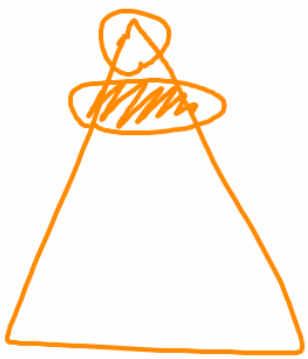
Evolution of the queue (**bold**=expanded and added children):

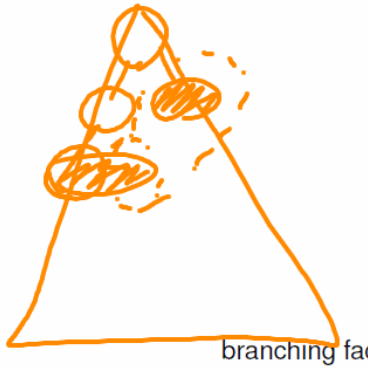
1. [[1]] : initial node list contains the initial state
2. [**[2][3]**] : visit (pop) 1 and push expanded nodes in the front
3. [**[4][5]** [3]] : visit (pop) 2 and push expanded nodes in the front
4. [**[8][9]** [5] [3]] : visit (pop) 4 and push expanded nodes in the front

...

So this 3 that was pushed into the list pretty early on, would keep on remaining in it until all of the Friends here have been external

You are screen sharing Stop Share



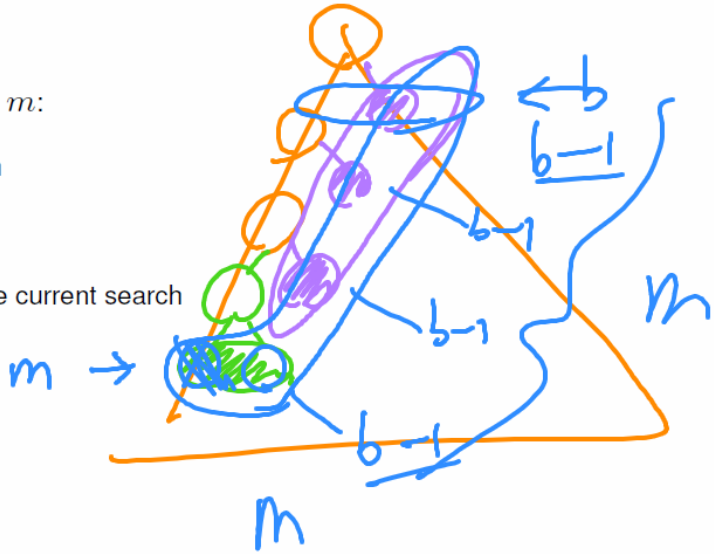


DFS: Evaluation

branching factor b, depth of solutions d, max depth m:

- incomplete: may wander down the wrong path
- time: $O(b^m)$ nodes visited (worst case)
- space: $O(bm)$ (dangling nodes just along the current search path)
- good when there are many shallow goals
- bad for deep or infinite depth state space

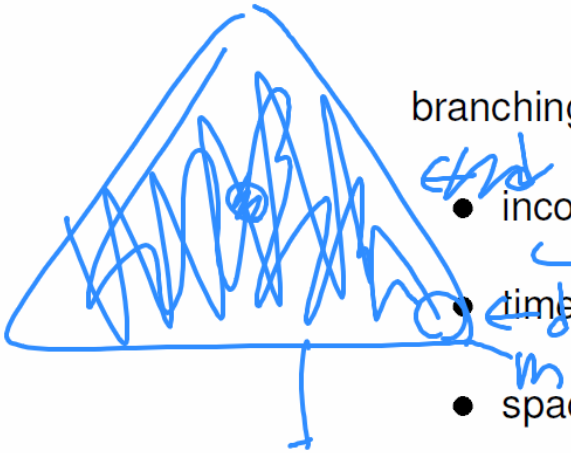
$O(b^{d+1})$



Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On
Choe, Yoonsuck
In this case, if it's supposed to be
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

DFS: Evaluation

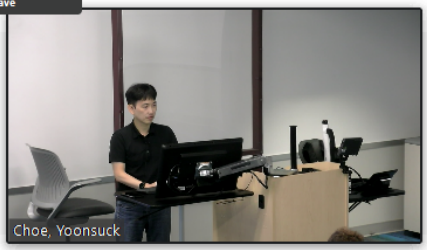


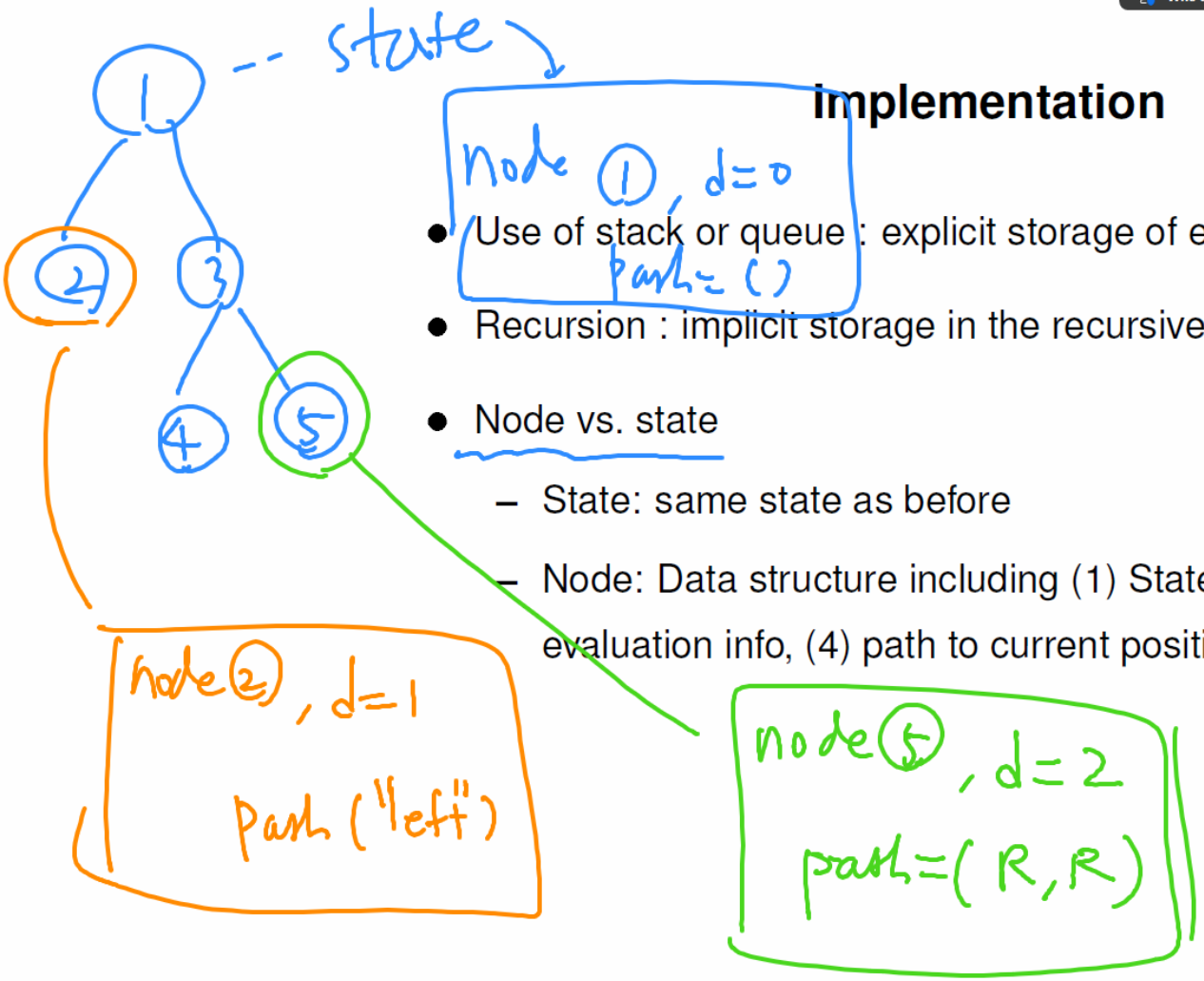
branching factor b, depth of solutions d, max depth m:

- incomplete: may wander down the wrong path
- time: $O(b^m)$ nodes visited (worst case)
- space: $O(bm)$ (dangling nodes just along the current search path)
- good when there are many shallow goals
- bad for deep or infinite depth state space

So it's good when there are many shallow goals but that for deep or very, very deep or infinite
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On





Implementation

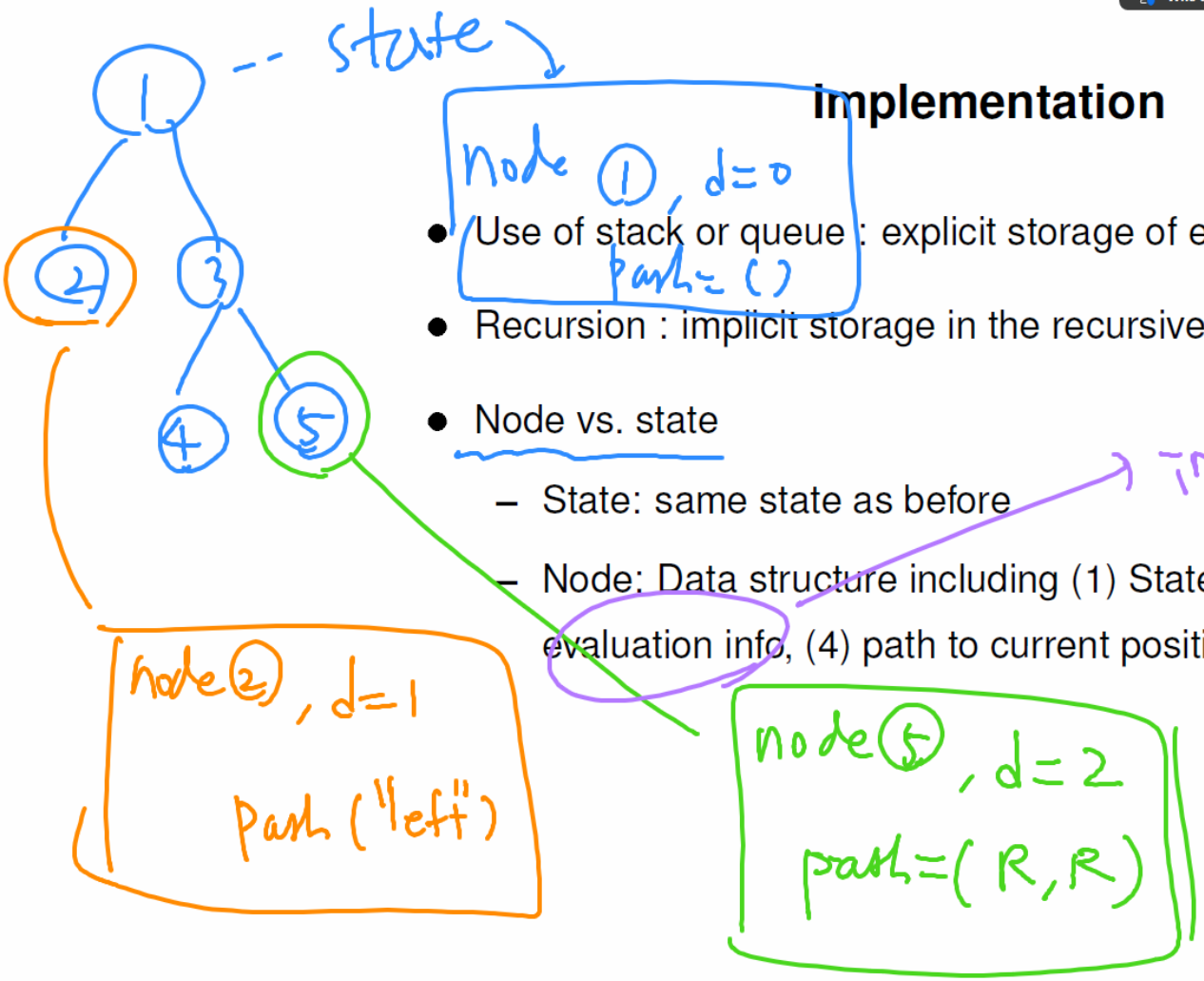
- Use of stack or queue : explicit storage of expanded nodes
- Recursion : implicit storage in the recursive call stack
- Node vs. state
  - State: same state as before
  - Node: Data structure including (1) State, (2) current depth, (3) evaluation info, (4) path to current position.

Was this kind of a node node data structure

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

Implementation



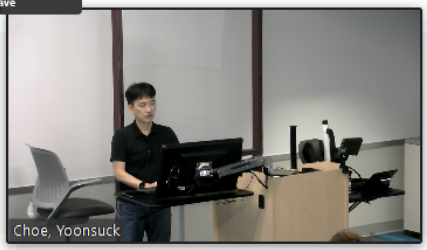
- Use of stack or queue : explicit storage of expanded nodes
- Recursion : implicit storage in the recursive call stack
- Node vs. state
  - State: same state as before
  - Node: Data structure including (1) State, (2) current depth, (3) evaluation info, (4) path to current position.

informed search

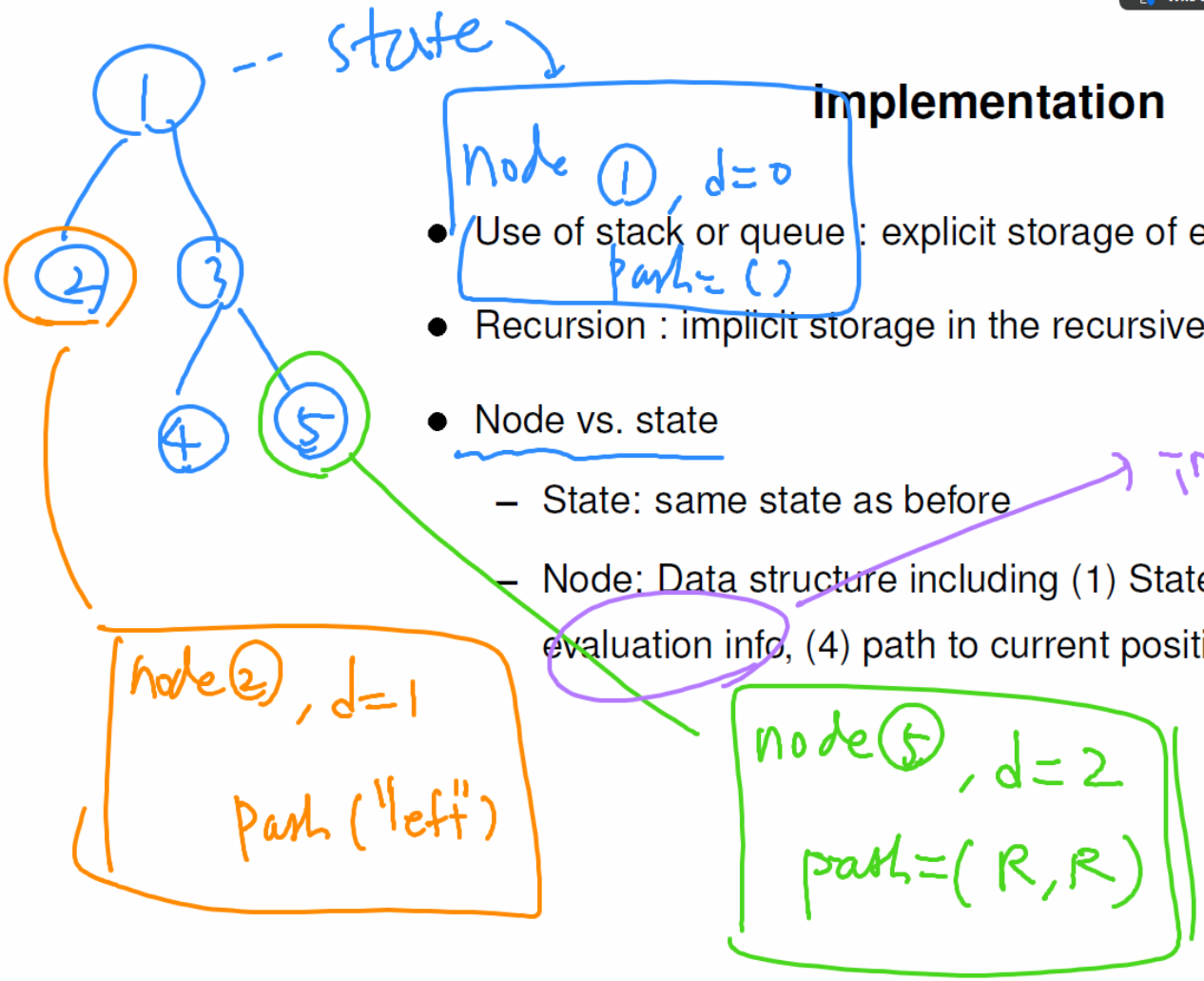
But this is for informed such

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On



Implementation

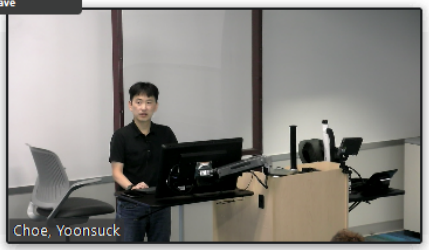


- Use of stack or queue : explicit storage of expanded nodes
- Recursion : implicit storage in the recursive call stack
- Node vs. state
  - State: same state as before
  - Node: Data structure including (1) State, (2) current depth, (3) evaluation info, (4) path to current position.

informed search : how good is ⑤

Okay, So how good is this Node 5
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On



Key Points

- Description of a search problem: initial state, goals, operators, etc.
- Considerations in designing a representation for a state
- Evaluation criteria — space, time, completeness, optimality
- BFS, UCS, DFS: time and space complexity, completeness
- Differences and similarities between BFS and UCS
- When to use one vs. another
- Node visit orders for each strategy
- Tracking the stack or queue at any moment

There is, and then optimality, would you find the shortest path to the

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

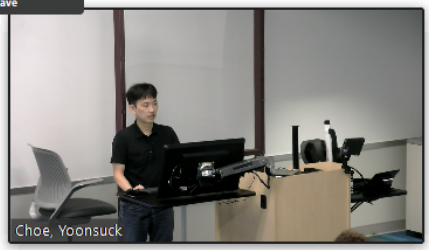
Search and Game Playing

- Uninformed search
- Informed search ← where AI gets involved
- Iterative improvement, Constraint satisfaction
- Game playing

You can consider that as an extension of search so we'll look at that

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On



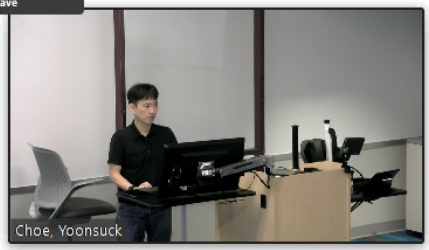
Key Points

- Description of a search problem: initial state, goals, operators, etc.
- Considerations in designing a representation for a state
- Evaluation criteria — space, time, completeness, optimality
- BFS, UCS, DFS: time and space complexity, completeness
- Differences and similarities between BFS and UCS
- When to use one vs. another
- Node visit orders for each strategy
- Tracking the stack or queue at any moment — node list

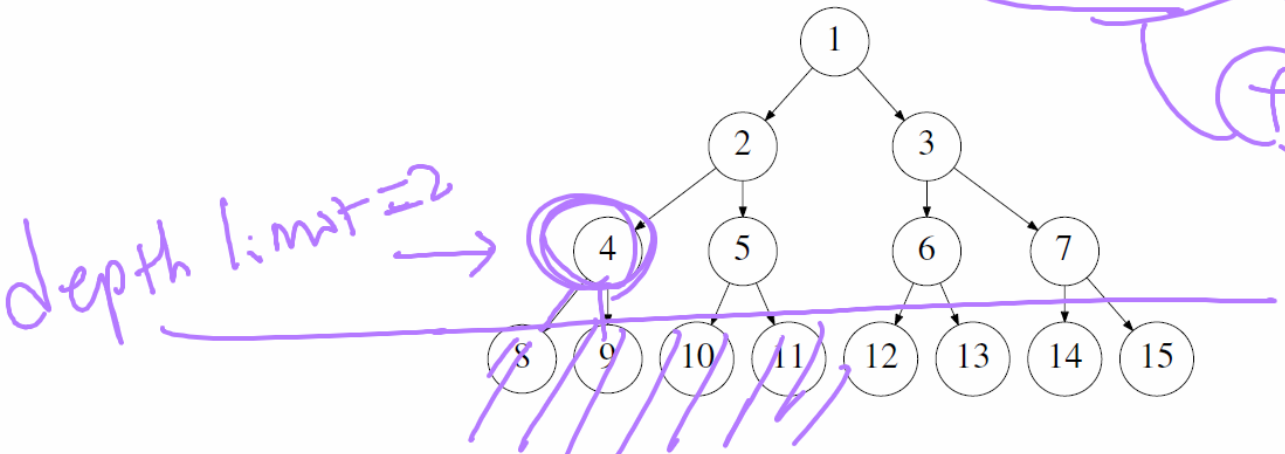
I mean I went to use one versus the other and then the node visitor, and and so on, cracking tracking the stack order queue at any moment that's the noodle

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On



Depth Limited Search (DLS): Limited Depth DFS



- nodes at depth=l cannot be expanded and further.
- node visit order for each depth limit l:
  1 ($l = 0$); 1 2 3 ($l = 1$); 1 2 4 5 3 6 7 ($l = 2$);
- queuing function: enqueue at front (i.e. stack push)
- **push the depth of the node as well**:
  (<**depth**> <**node**>)

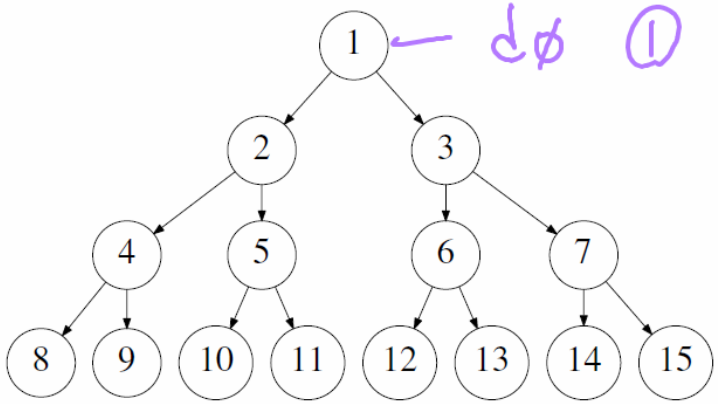
state

Actually, this is a little bit redundant. So the node, and, as we have shown you, includes the the

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

DLS: Visit Order and Node List



Evolution of the queue (**bold**=expanded and then added):

(<depth>,<node>); Depth limit = 2

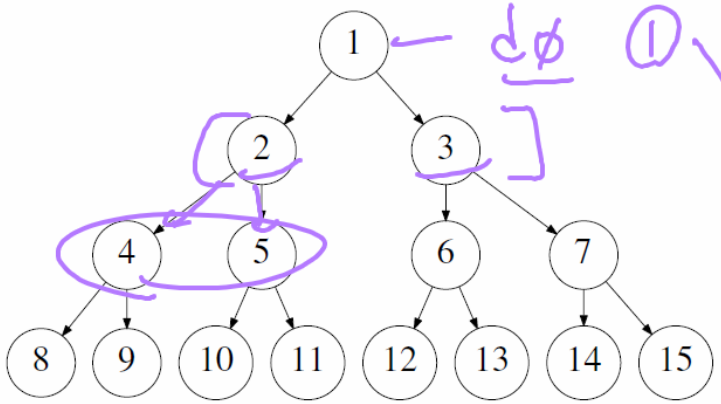
1. [[(d0,1)]] : initial state
2. [**[(d1,2)][(d1,3)]**] : pop 1 and push 2 and 3
3. [**[(d2,4)][(d2,5)]**[(d1,3)]] : pop 2 and push 4 and 5
4. [[(d2,5)][(d1,3)]]: pop 4, cannot expand it further
5. [[(d1,3)]]: pop 5, cannot expand it further
6. [**[(d2,6)][(d2,7)]**]: pop 3, and push 6, 7

...

So you have a no one that looks like this. So this is
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

DLS: Visit Order and Node List



Evolution of the queue (**bold**=expanded and then added):

(<depth>,<node>); Depth limit = 2

1. [[(d0,1)]] : initial state
2. [**[(d1,2)][(d1,3)]**] : pop 1 and push 2 and 3
3. [**[(d2,4)][(d2,5)]**[(d1,3)]] : pop 2 and push 4 and 5
4. [[(d2,5)][(d1,3)]]: pop 4, cannot expand it further
5. [[(d1,3)]]: pop 5, cannot expand it further
6. [**[(d2,6)][(d2,7)]**]

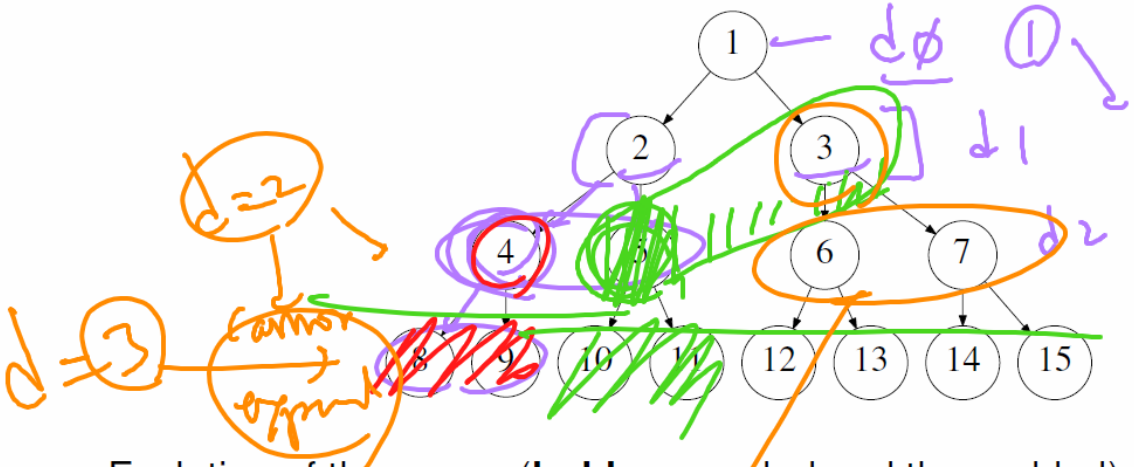
...

These are the wall, so you can expand 2 you're allowed to expand, too, because it's below the that's limit to expand them into 4 and 5

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

DLS: Visit Order and Node List



Evolution of the queue (**bold**=expanded and then added):

(<depth>,<node>); Depth limit = 2

1. [[(d0, 1)]] : initial state
2. [**(d1,2)**][**(d1,3)**] : pop 1 and push 2 and 3
3. [**(d2,4)**][**(d2,5)**][(d1, 3)]] : pop 2 and push 4 and 5
4. [[(d2, 5)] [(d1, 3)]] : pop 4, cannot expand it further
5. [[(d1, 3)]] : pop 5, cannot expand it further
6. [**(d2,6)**][**(d2,7)**]] : pop 3, and push 6, 7

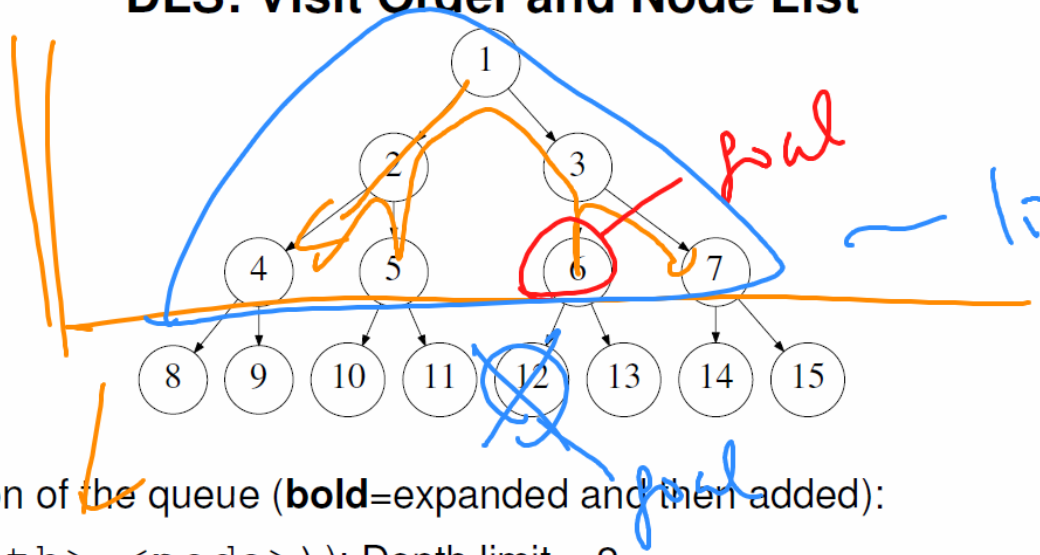
...

We showed that that's goes 3 red this the nodes at depth equals 2 cannot expand right. So that's where this deaf limi

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

DLS: Visit Order and Node List



Evolution of the queue (**bold**=expanded and then added):

(<depth>,<node>); Depth limit = 2

1. [[(d0,1)]] : initial state
2. [**[(d1,2)][(d1,3)]**] : pop 1 and push 2 and 3
3. [**[(d2,4)][(d2,5)]**[(d1,3)]] : pop 2 and push 4 and 5
4. [[(d2,5)][(d1,3)]]: pop 4, cannot expand it further
5. [[(d1,3)]]: pop 5, cannot expand it further
6. [**[(d2,6)][(d2,7)]**]: pop 3, and push 6, 7

...

Then this won't be found right but as you're exploring this, you get the same benefit of linear space complexity

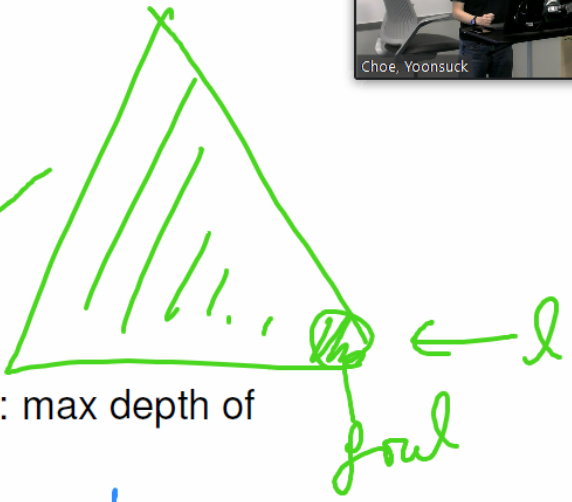
You are screen sharing Stop Share

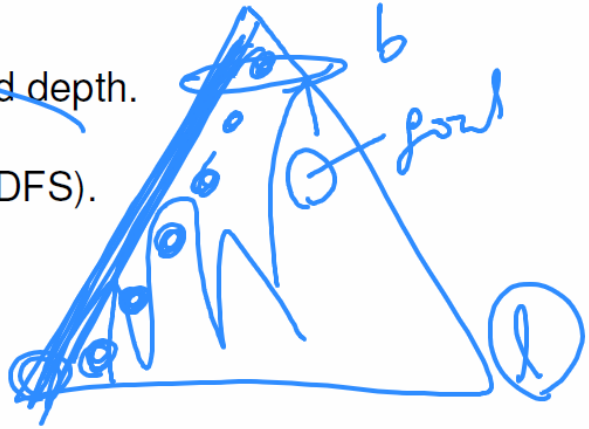
Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

DLS: Evaluation

branching factor b, depth limit l, depth of solution d:

- complete: if $l \geq d$
- time: $O(b^l)$ nodes visited (worst case)
- space: $O(bl)$ (same as DFS, where $l = m$ (m: max depth of tree in DFS)
- good if solution is within the limited depth.
- non-optimal (same problem as in DFS).





And if you know that depths, what is non optimal
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

DLS: Evaluation

branching factor b, depth limit l, depth of solution d:

- complete: if $l \geq d$
- time: $O(b^l)$ nodes visited (worst case)
- space: $O(bl)$ (same as DFS, where $l = m$ (m: max depth of tree in DFS)
- good if solution is within the limited depth.
- non-optimal (same problem as in DFS).

As the solution. right? So this this steps is greater than this steps right So that's why it's not optimal
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

what we desire

DLS: Evaluation

branching factor b, depth limit l, depth of solution d:

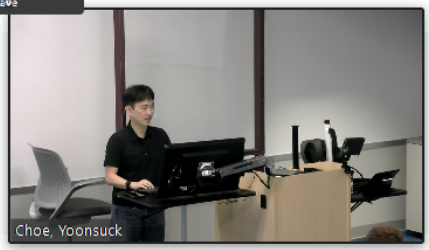
- complete: if $l \geq d$ ← BFS
- time: $O(b^l)$ nodes visited (worst case) ← same as others
- space: $O(bl)$ (same as DFS, where $l = m$ (m: max depth of tree in DFS) ← DFS
- good if solution is within the limited depth.
- non-optimal (same problem as in DFS).

bad. ⟹ want to be like BFS

So this is what we this desire

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

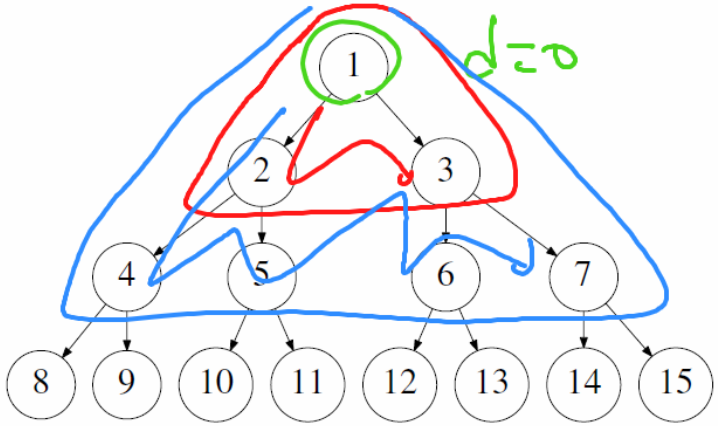


Iterative Deepening Search: DLS by Increasing Limit

DLS with $l=0$

DLS with $l=1$

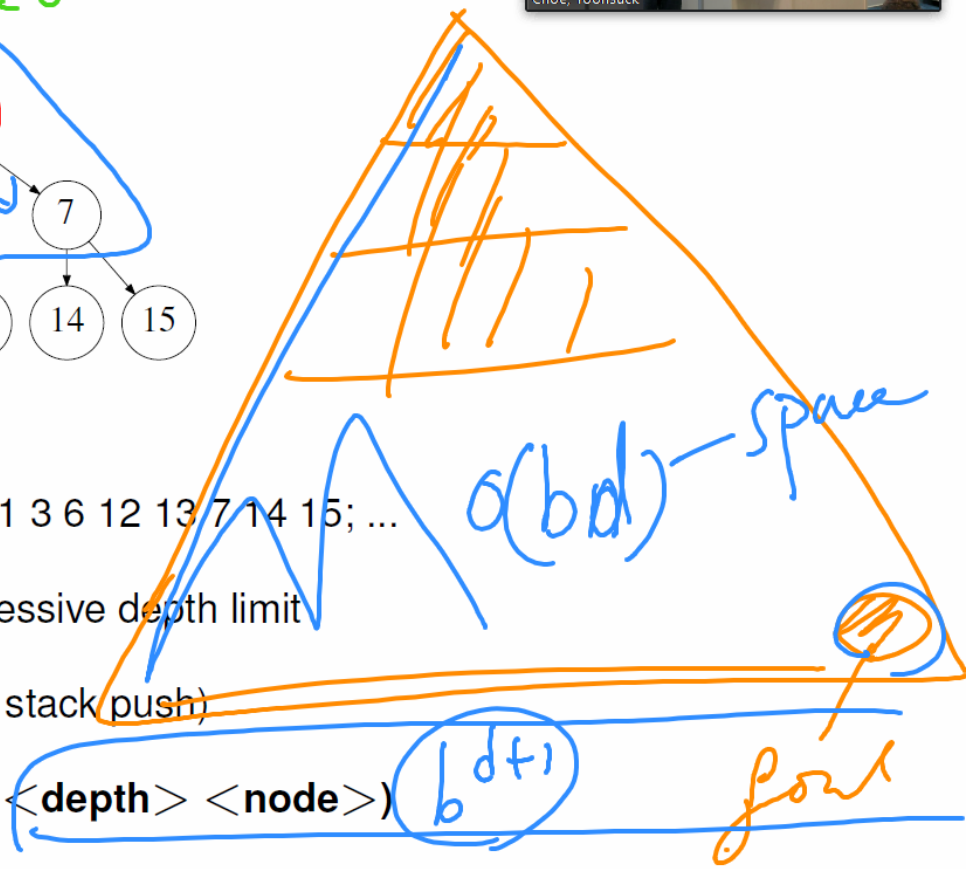
DLS w/ $l=2$



- node visit order:

  1 ; 1 2 3; 1 2 4 5 3 6 7; 1 2 4 8 9 5 10 11 3 6 12 13 7 14 15; ...

- revisits already explored nodes at successive depth limit
- queuing function: enqueue at front (i.e. stack push)
- **push the depth of the node as well**: (**<depth>** **<node>**)



Complexity is exponential, even though, in fact, this is a slower

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

Overview: Uninformed search

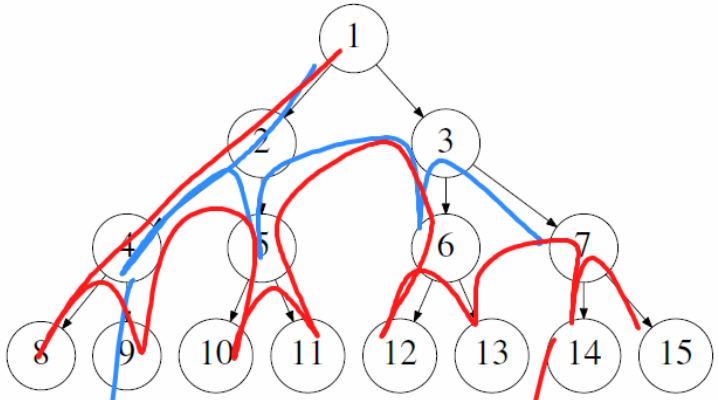
- Search problems: definition
- Example: 8-puzzle
- General search
- Evaluation of search strategies
- Strategies: breadth-first, uniform-cost, depth-first
- More uninformed search: depth-limited, iterative deepening, bidirectional search

First search, the first search, and also things like in from cool search, and so on, and then more advanced versions of it.

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

Iterative Deepening Search: DLS by Increasing Limit



- node visit order:

  1; 1 2 3; 1 2 4 5 3 6 7; 1 2 4 8 9 5 10 11 3 6 12 13 7 14 15; ...

- revisits already explored nodes at successive depth limit
- queuing function: enqueue at front (i.e. stack push)
- **push the depth of the node as well**: (<**depth**> <**node**>)

Very soon the slide press for search
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

IDS: Evaluation

branching factor b, depth of solution d:

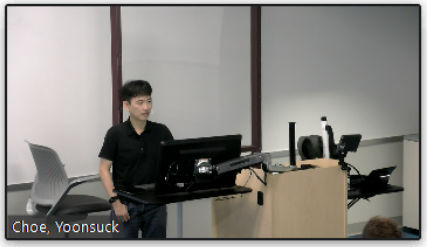
- complete: cf. DLS, which is conditionally complete
- time: $O(b^d)$ nodes visited (worst case)
- space: $O(bd)$ (cf. DFS and DLS)
- **optimal**: unlike DFS or DLS
- good when search space is huge and the depth of the solution is not known (*)

$1 + (1+b) + (1+b+b^2) + \cdots$

and the uninformed search arena it treated, being such, is practically the best approach

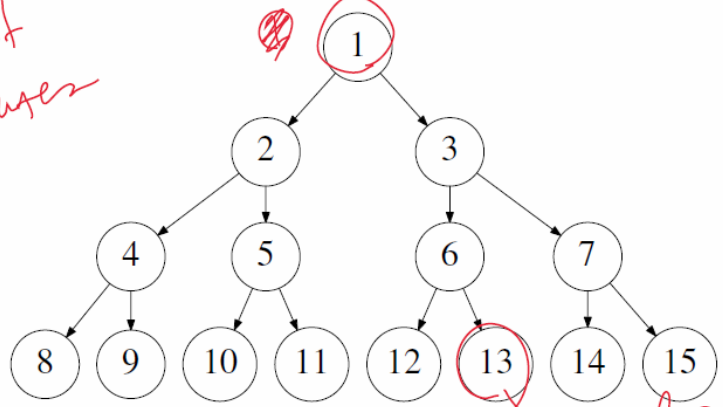
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On



Search Problems: Definition

Op : State → Set of States



goal

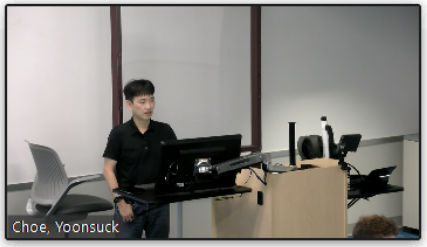
Search = < initial state, operators, goal states >

- Initial State: description of the current situation as given in a problem
- Operators: functions from any state to a set of successor (or neighbor) states
- Goal: subset of states, or test rule

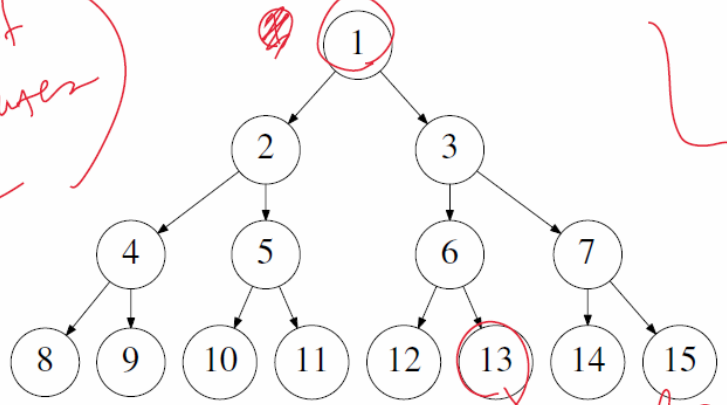
So let me confuse me. K. I wrote this let's just ignore it, for now

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On



Search Problems: Definition



Search = < initial state, operators, goal states >

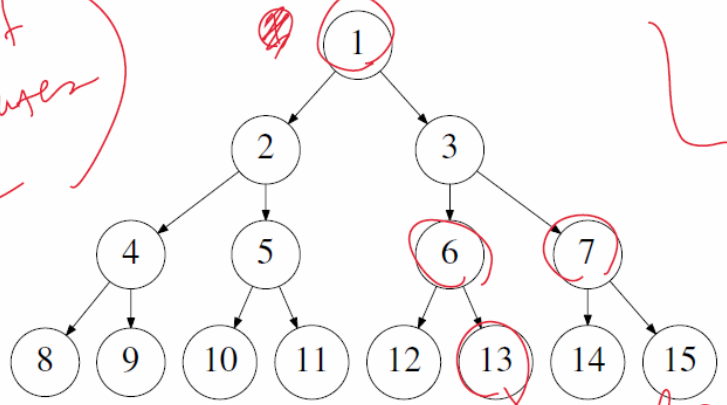
- Initial State: description of the current situation as given in a problem
- Operators: functions from any state to a set of successor (or neighbor) states
- Goal: subset of states, or test rule

Okay, Then, the set of States would include what for given this 3 so it's 3 actually be 2 and 3 right

You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On

Search Problems: Definition

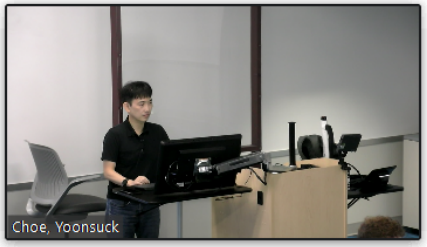


Search = < initial state, operators, goal states >

- Initial State: description of the current situation as given in a problem
- Operators: functions from any state to a set of successor (or neighbor) states
- Goal: subset of states, or test rule

Then operator with operating on this day 3 would give you 6 and 7, as the out output
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On



Variants of Search Problems

Search = < state space, initial state, operators, goal states >

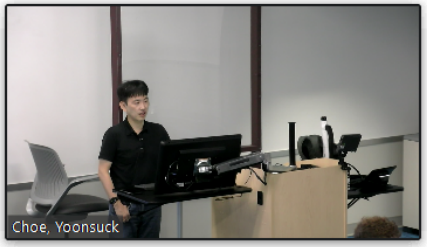
- State space: set of all possible states reachable from the current initial state through repeated application of the operators (i.e. path).

Search = < initial state, operators, goal states, path cost >

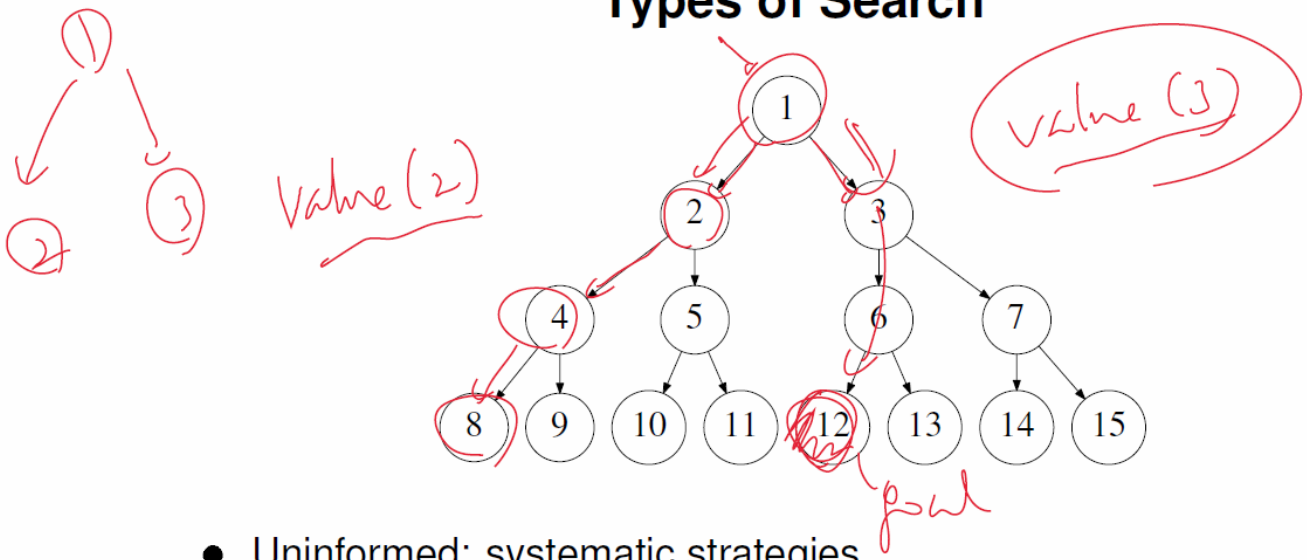
- Path cost: find **the best** solution, not just **a** solution. Cost can be many different things.

To reach your goal. But on top of that you want to ask what is how good it is, or how bad it is
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On



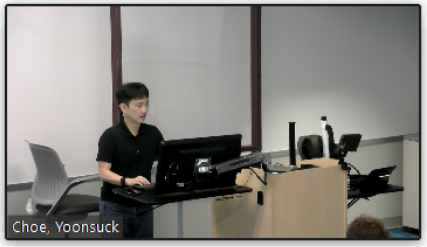
Types of Search



- Uninformed: systematic strategies
- Informed: Use domain knowledge to narrow down scope of search
- Game playing as search: minimax, state pruning, probabilistic games

And suppose this was the actual goal, then you can just go
You are screen sharing Stop Share

Mouse Select Text Draw Stamp Spotlight Eraser Format Undo Redo Clear Save
Who can see what you share here? Recording On



Goal Test

Simply check if the current state is the same as the goal state.

Implementation may vary.

But if you do it like this, since they attend then use keep on repeating the State. Alright, so you'll never get to the ball

You are screen sharing Stop Share