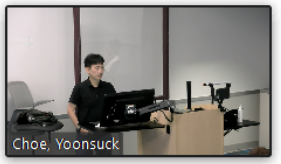


Heuristic Functions: Example

Eight puzzle

5	4	
6	1	8
7	3	2

Goal

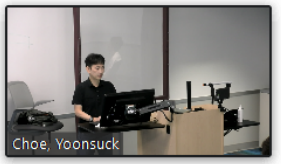
1	2	3
8		4
7	6	5

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total **Manhattan** distance (city block distance)

$h_1(n)$ = 7 (not counting the blank tile)

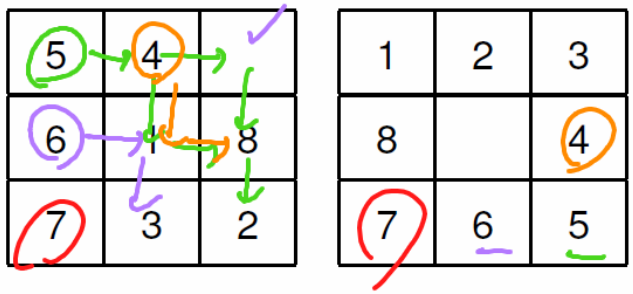
$h_2(n)$ = 2+3+3+2+4+2+0+2 = 18

* Both are admissible heuristic functions.



Heuristic Functions: Example

Eight puzzle

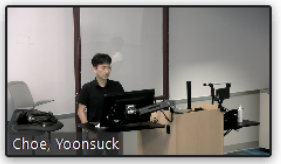


- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total **Manhattan** distance (city block distance)

$h_1(n)$ = 7 (not counting the blank tile)

$h_2(n)$ = 2+3+3+2+4+2+0+2 = 18

* Both are admissible heuristic functions.



Dominance

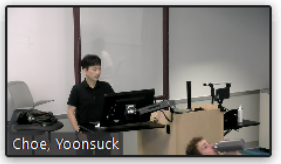
If $h_2(n) \geq h_1(n)$ for all n and both are admissible, then we say that $h_2(n)$ **dominates** $h_1(n)$, and is better for search.

Typical search costs for depth $d = 14$:

- Iterative Deepening : 3,473,941 nodes visited
- $\mathrm{A}^*(h_1)$: 539 nodes
- $\mathrm{A}^*(h_2)$: 113 nodes

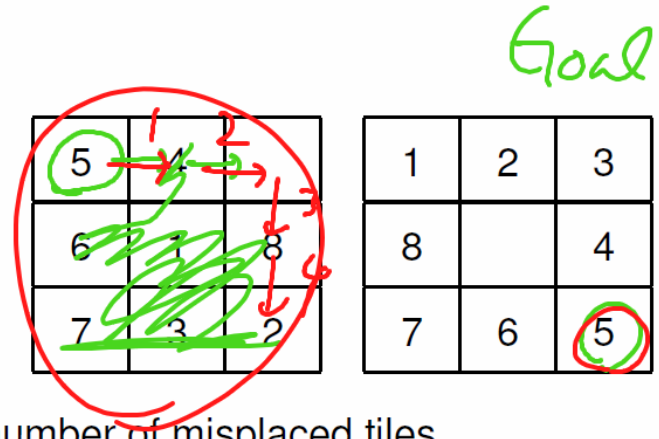
Observe that in A^*, every node with $f < f^*$ is visited . Since $f = g + h$, nodes with $h(n) < f^* - g(n)$ will be visited, so larger h will result in less nodes being visited.

- f^* is the f value for the optimal solution path.



Heuristic Functions: Example

Eight puzzle



- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total **Manhattan** distance (city block distance)

$h_1(n)$ = 7 (not counting the blank tile)

$h_2(n)$ = 2+3+3+2+4+2+0+2 = 18

Tile 1 2 3 4 5

* Both are admissible heuristic functions.

So for each state you can compute the sum of and I think distance which becomes a heuristic function

Heuristic Functions: Example

Eight puzzle

5	4	
6	1	8
7	3	2

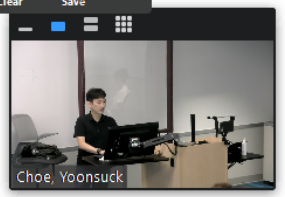
1	2	3
8		4
7	6	5

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total **Manhattan** distance (city block distance)

$h_1(n)$ = 7 (not counting the blank tile)

$h_2(n)$ = 2+3+3+2+4+2+0+2 = 18

* Both are admissible heuristic functions.



Heuristic Functions: Example

Eight puzzle

5	4	
6	1	8
7	3	2

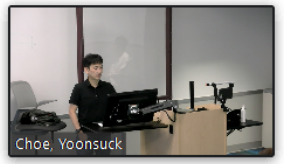
1	2	3
8		4
7	6	5

- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total **Manhattan** distance (city block distance)

$h_1(n)$ = 7 (not counting the blank tile)

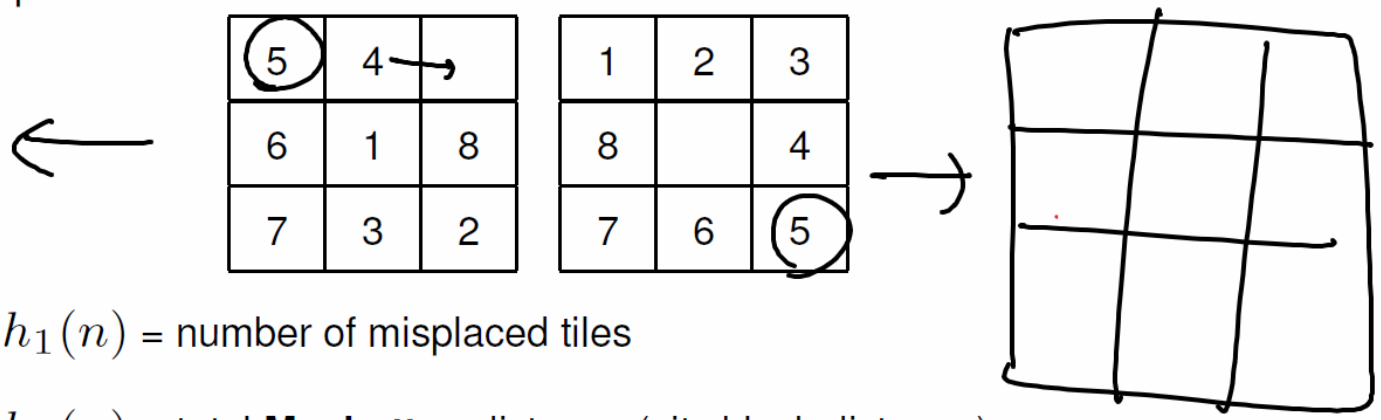
$h_2(n)$ = 2+3+3+2+4+2+0+2 = 18

* Both are admissible heuristic functions.



Heuristic Functions: Example

Eight puzzle

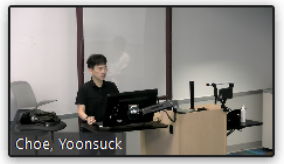


- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total **Manhattan** distance (city block distance)

$h_1(n)$ = 7 (not counting the blank tile)

$h_2(n)$ = 2+3+3+2+4+2+0+2 = 18

* Both are admissible heuristic functions.

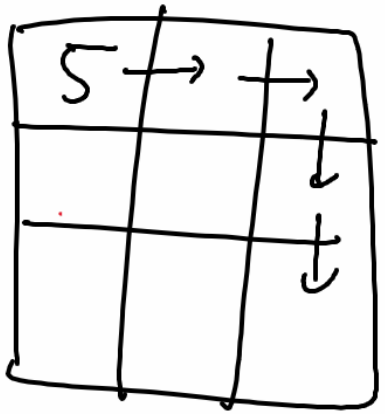


Heuristic Functions: Example

Eight puzzle

5	4	
6	1	8
7	3	2

1	2	3
8		4
7	6	5



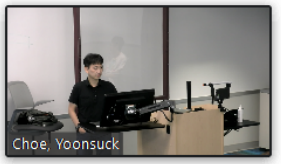
- $h_1(n)$ = number of misplaced tiles
- $h_2(n)$ = total **Manhattan** distance (city block distance)

$h_1(n)$ = 7 (not counting the blank tile)

$h_2(n)$ = 2+3+3+2+4+2+0+2 = 18

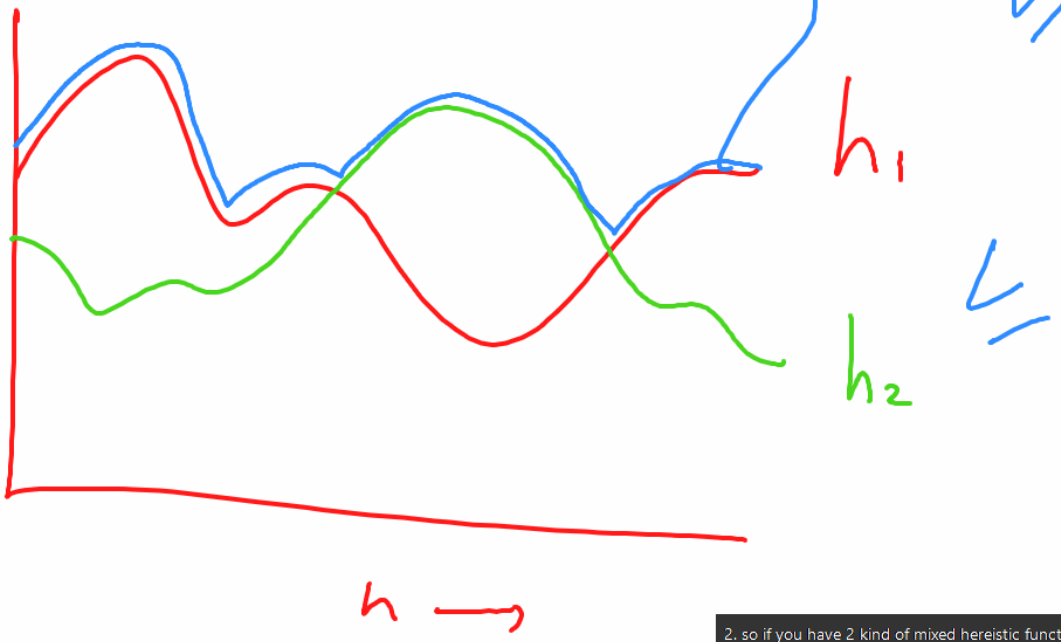
* Both are admissible heuristic functions.

Then that's how you get this the second heuristic. So in general, to get to get get a misunderstanding heuristic, you can always relax the constraint where you can easily

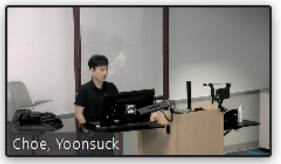


Other Heuristic Design

- Use composite heuristics: $h(n) = max(h_1(n), ..., h_m(n))$
- Use statistical information: random sample h and true cost to reach goal. Find out how often h and true cost is related.



2. so if you have 2 kind of mixed hereistic functions you can make just find the Max of those basically use both to get the strictly domain. And here's

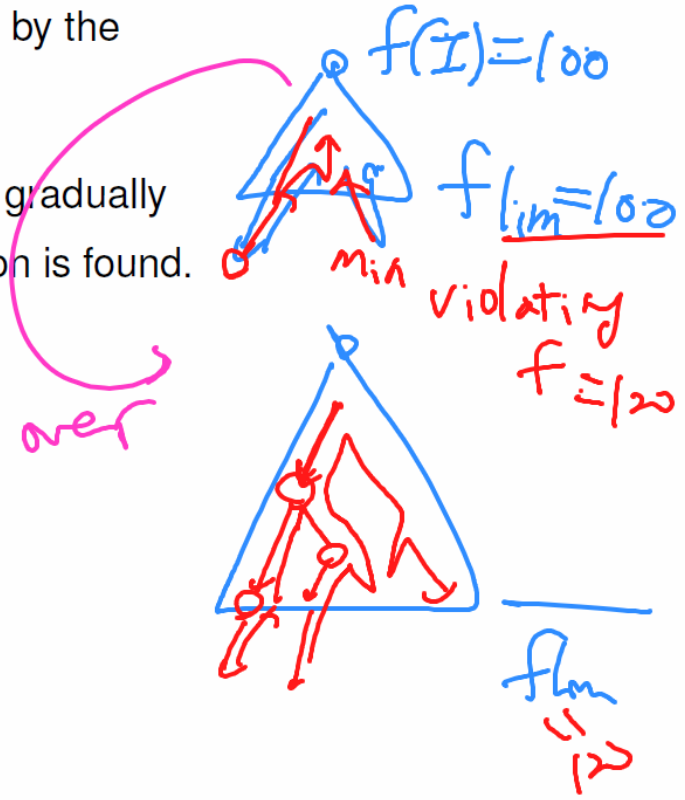


Iterative Deepening A*: IDA^*

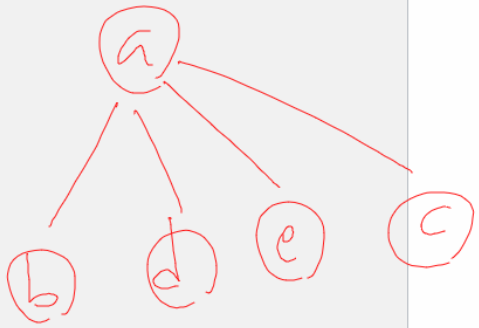
IDS

A^* is complete and optimal, but the performance is limited by the available space.

- Basic idea: only search within a certain f bound, and gradually increase the f bound and restart search until a solution is found.

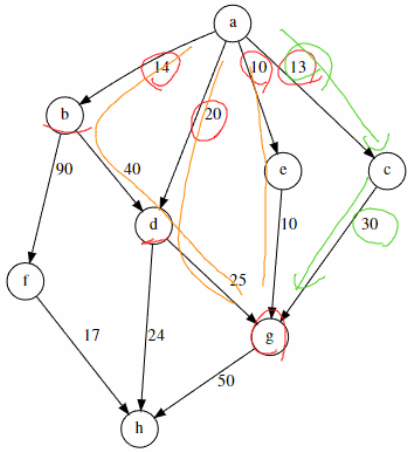


Such methods. Well, here is is



Problem 5 (Written; 10 pts): Consider iterative deepening search. When the goal is (9), how many nodes are visited before reaching that node?

2 Informed Search



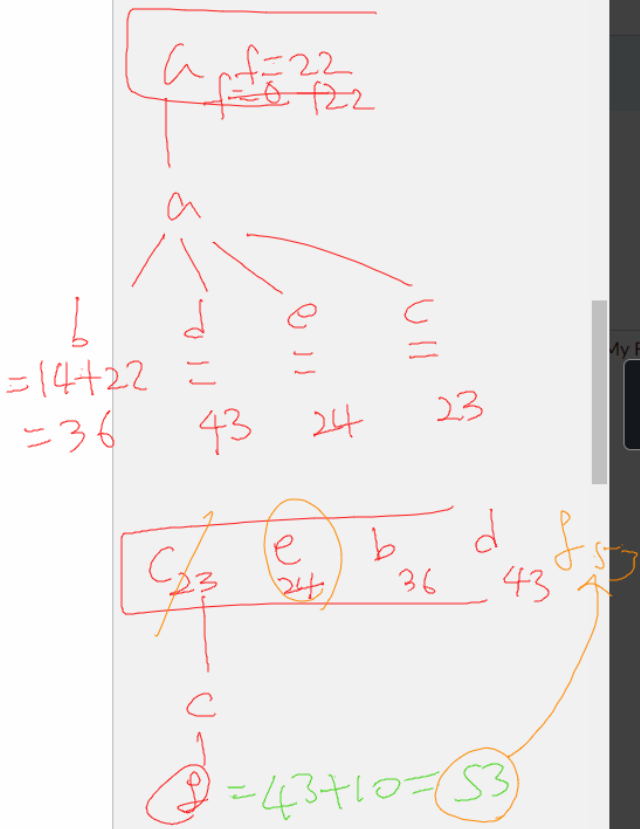
Node	h(n)
a	22
b	22
c	10
d	23
e	14
f	16
g	10
h	0

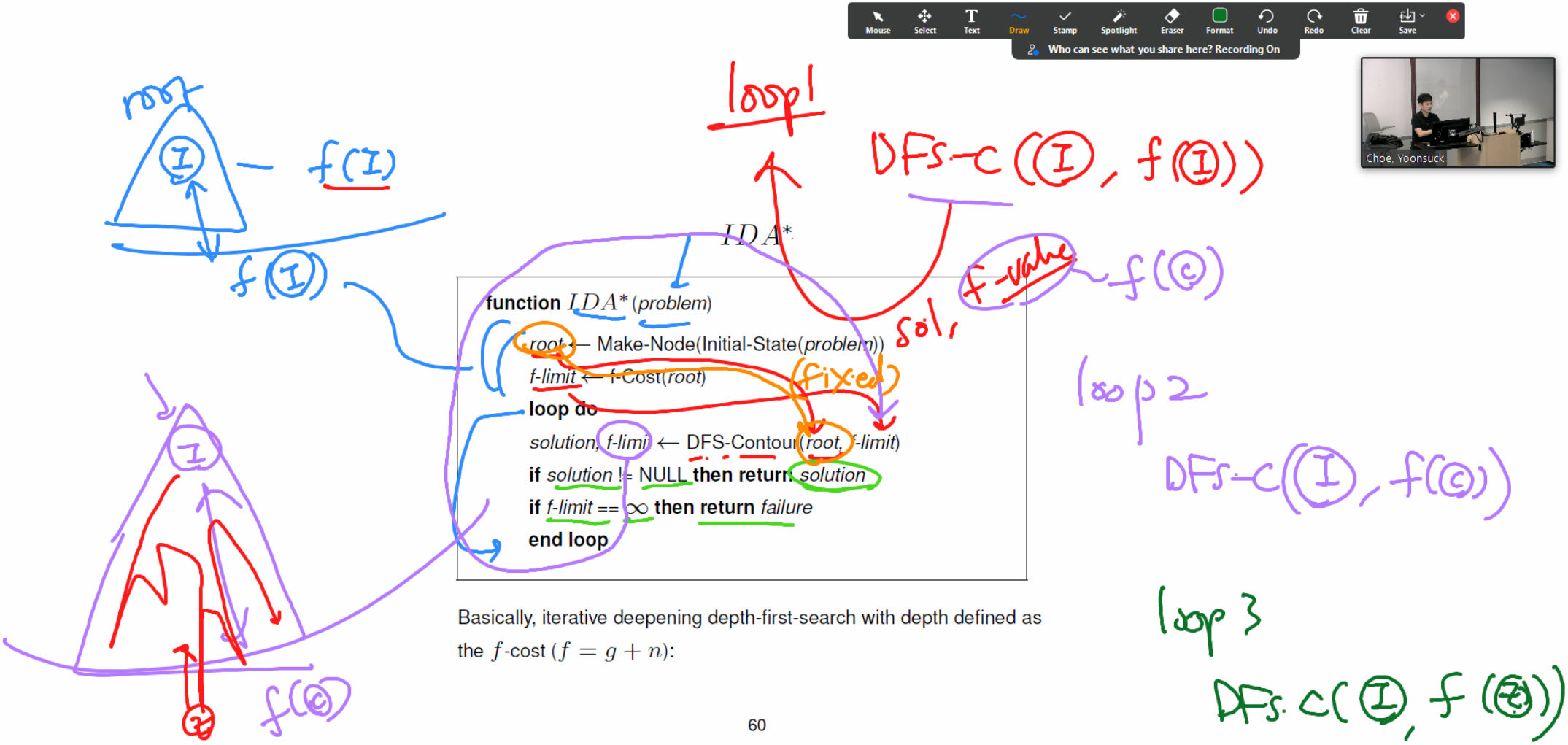
Figure 2: **Informed Search.**

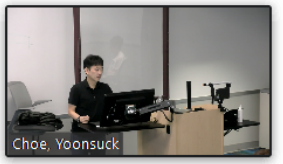
Problem 6 (Written; 10 pts): For the problem shown in Fig. 2, show that the heuristic is admissible ($h(n) \leq h^*(n)$ for all n). Note: You have to compute $h^*(n)$ for each n and compare to the $h(n)$ table.

Hint: It is best to work backwards from the goal, where $h^*(h) = 0$ (already at goal), $h^*(f) = 17$ (true minimum cost from (f) to (h)), $h^*(b) = 40 + 24 = 64$ (true minimum cost from (b) to (h): note that there are multiple paths and this path has the minimum cost!), etc.

Problem 7 (Written; 20 pts): Manually conduct greedy best-first search on the graph below (Fig. 2), with initial node (a) and goal node (h). Actual cost from node to node are shown as edge la




Mouse
Select
Text
Draw
Stamp
Spotlight
Eraser
Format
Undo
Redo
Clear
Save
Who can see what you share here? Recording On
Choe, Yoonsuck
root
I
f(I)
f(I)
loop1
DFS-C(I, f(I))
IDA*
f-value
f(c)
sol,
function IDA*(problem)
root ← Make-Node(Initial-State(problem))
f-limit ← f-Cost(root)
loop do
solution, f-limit ← DFS-Contour(root, f-limit)
if solution != NULL then return solution
if f-limit == ∞ then return failure
end loop
fixed
loop2
DFS-C(I, f(c))
I
f(c)
loop3
DFS-C(I, f(c))
Basically, iterative deepening depth-first-search with depth defined as the f-cost ($f = g + n$):
60
No, no it doesn't change it so
You are screen sharing
Stop Share

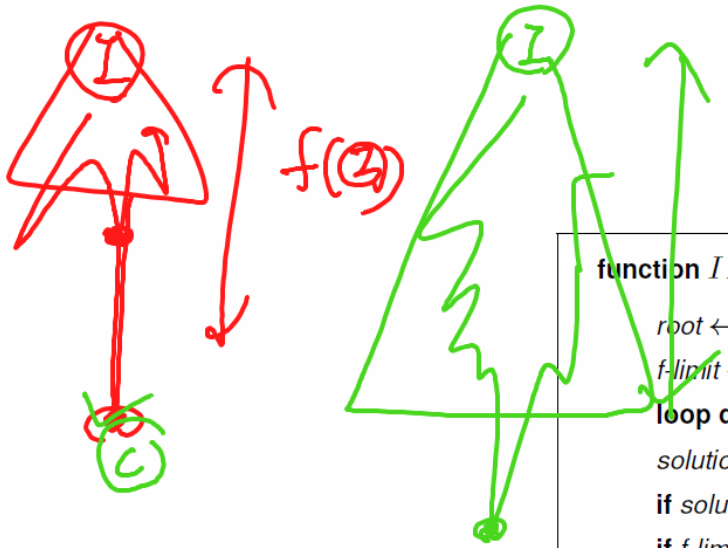


IDA^*

```
function IDA*(problem)
    root ← Make-Node(Initial-State(problem))
    f-limit ← f-Cost(root)
    loop do
    solution, f-limit ← DFS-Contour(root, f-limit)
    if solution != NULL then return solution
    if f-limit == ∞ then return failure
    end loop
```

Basically, iterative deepening depth-first-search with depth defined as the f-cost ($f = g + n$):

And the same applies to the this case



IDA^*

```
function IDA*(problem)
    root ← Make-Node(Initial-State(problem))
    f-limit ← f-Cost(root)
    loop do
    solution, f-limit ← DFS-Contour(root, f-limit)
    if solution != NULL then return solution
    if f-limit == ∞ then return failure
    end loop
```

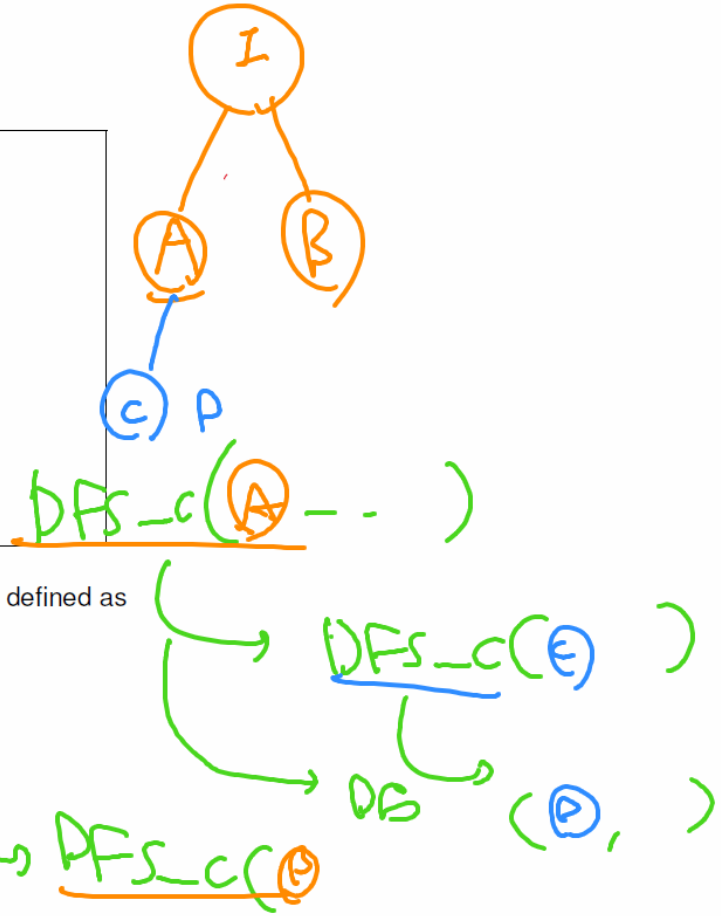
Basically, iterative deepening depth-first-search with depth defined as the f-cost ($f = g + n$):

Limit that is file a value. Does that violate this limit would be used to increase the have contour time. It

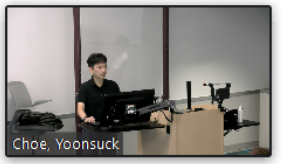
IDA^*

```
function IDA*(problem)
    root ← Make-Node(Initial-State(problem))
    f-limit ← f-Cost(root)
    loop do
      solution, f-limit ← DFS-Contour(root, f-limit)
      if solution != NULL then return solution
      if f-limit == ∞ then return failure
    end loop
```

Basically, iterative deepening depth-first-search with depth defined as the f-cost ($f = g + n$):



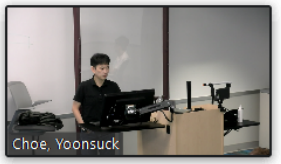
And this is Pd



IDA^*

```
function IDA*(problem)
    root ← Make-Node(Initial-State(problem))
    f-limit ← f-Cost(root)
    loop do
      solution, f-limit ← DFS-Contour(root, f-limit)
      if solution != NULL then return solution
      if f-limit == ∞ then return failure
    end loop
```

Basically, iterative deepening depth-first-search with depth defined as the f-cost ($f = g + n$):

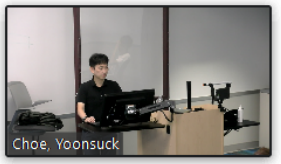


IDA^*

```
function IDA*(problem)
    root ← Make-Node(Initial-State(problem))
    f-limit ← f-Cost(root)
    loop do
      solution, f-limit ← DFS-Contour(root, f-limit)
      if solution != NULL then return solution
      if f-limit == ∞ then return failure
    end loop
```

Basically, iterative deepening depth-first-search with depth defined as the f-cost ($f = g + n$):

So we should keep that in mind.



DFS-Contour(*root, f-limit*)

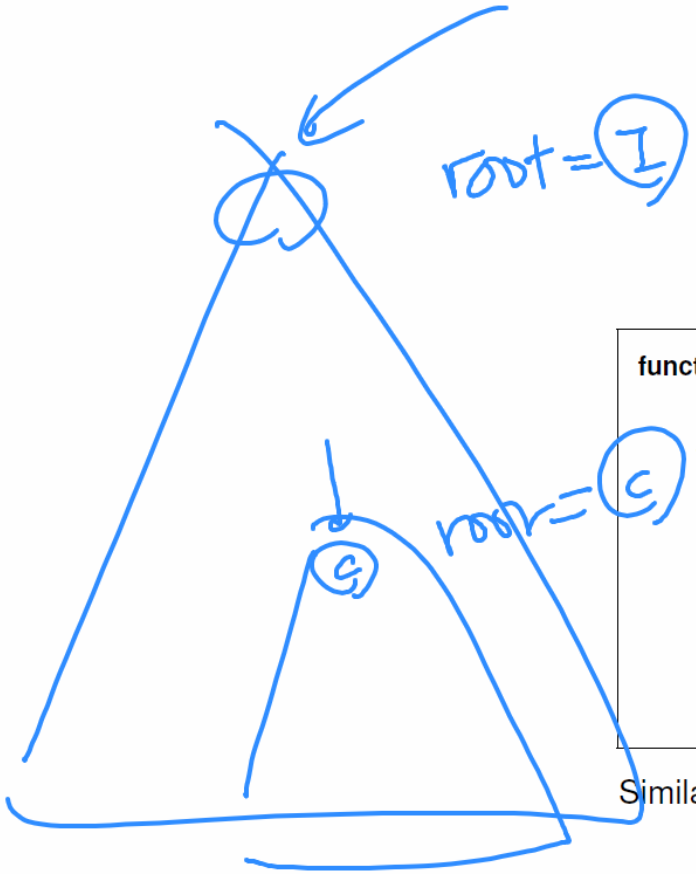
function DFS-Contour (**root, f-limit**): returns **solution, f-limit**

Find solution from node **root**, within the f-cost limit of **f-limit**. DFS-Contour returns **solution sequence** and **new f-cost limit**.

- if f-cost(**root**) $>$ **f-limit**, return **null solution** and the f-cost(**root**).
- if **root** is a goal node, return solution and its f-cost(**root**).
- **recursively call** on all successors, and find successor with minimum f-cost. Return its solution and its f-cost.

Similar to the recursive implementation of DFS.

So where does this come from? So actually it's given from the recursive call



DFS-Contour(*root*, *f-limit*)

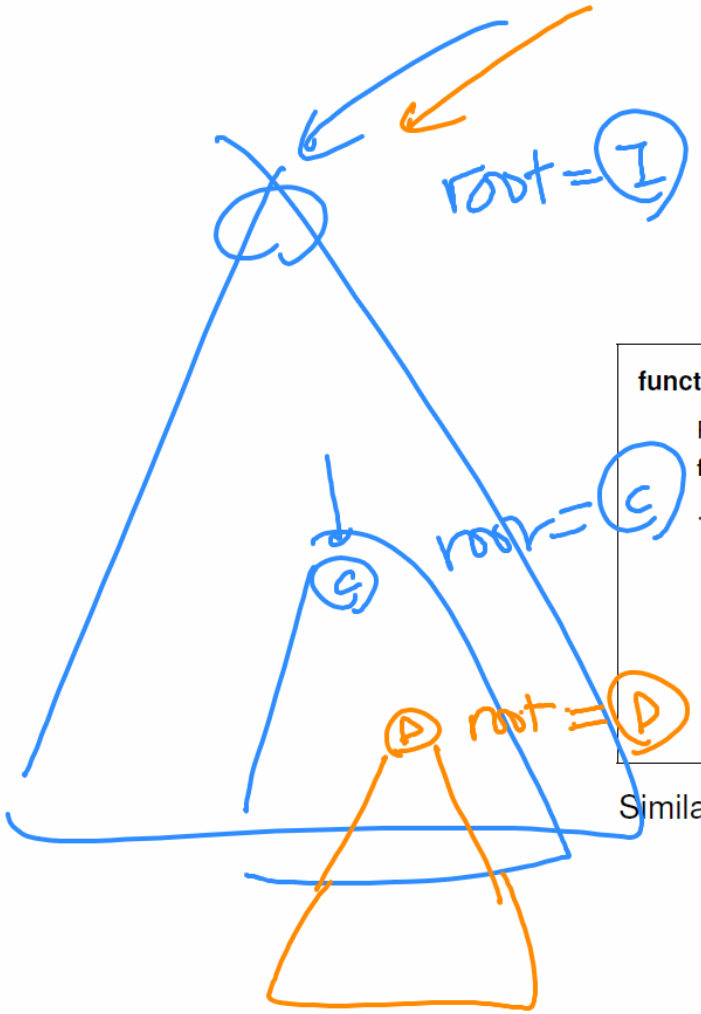
function DFS-Contour (**root, f-limit**): returns **solution, f-limit**

Find solution from node **root**, within the f-cost limit of **f-limit**. DFS-Contour returns **solution sequence** and **new f-cost limit**.

- if f-cost(**root**) $>$ **f-limit**, return **null solution** and the f-cost(**root**).
- if **root** is a goal node, return solution and its f-cost(**root**).
- **recursively call** on all successors, and find successor with minimum f-cost. Return its solution and its f-cost.

Similar to the recursive implementation of DFS.

Then this case root passed on to this, because the call with Pc.



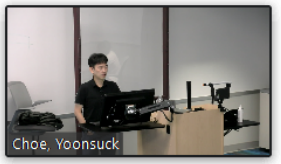
DFS-Contour(*root*, *f-limit*)

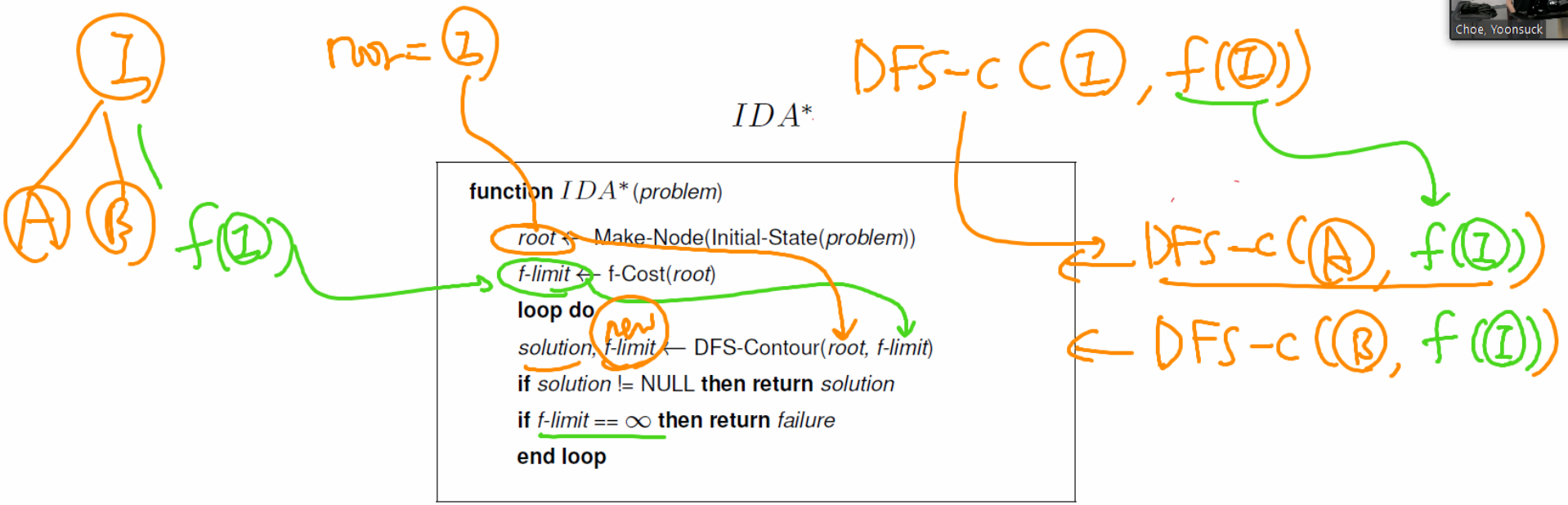
function DFS-Contour (**root, f-limit**): returns **solution, f-limit**

Find solution from node **root**, within the f-cost limit of **f-limit**. DFS-Contour returns **solution sequence** and **new f-cost limit**.

- if f-cost(**root**) $>$ **f-limit**, return **null solution** and the f-cost(**root**).
- if **root** is a goal node, return solution and its f-cost(**root**).
- **recursively call** on all successors, and find successor with minimum f-cost. Return its solution and its f-cost.

Similar to the recursive implementation of DFS.





IDA^*

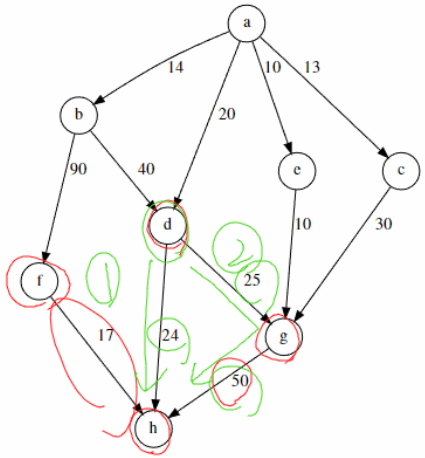
```
function IDA*(problem)
    root ← Make-Node(Initial-State(problem))
    f-limit ← f-Cost(root)
    loop do
    solution, f-limit ← DFS-Contour(root, f-limit)
    if solution != NULL then return solution
    if f-limit == ∞ then return failure
    end loop
```

Basically, iterative deepening depth-first-search with depth defined as the f-cost ($f = g + n$):

So think of this as new f limits. Okay.

Problem 5 (Written; 10 pts): Consider iterative deepening search. When the goal is (9), how many nodes are visited before reaching that node?

2 Informed Search



Node	h(n)
a	22
b	22
c	10
d	23
e	14
f	16
g	10
h	0

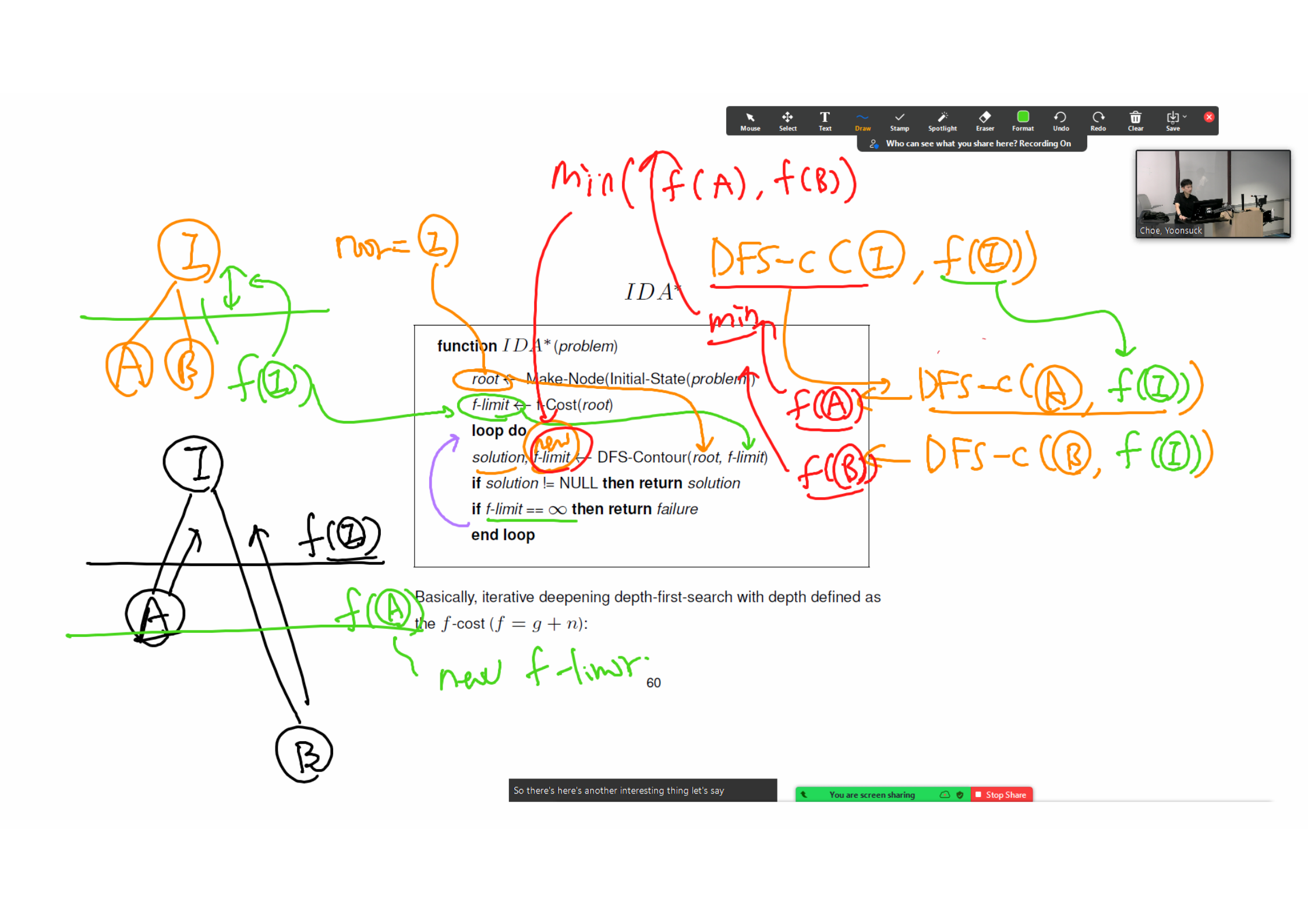
Figure 2: **Informed Search.**

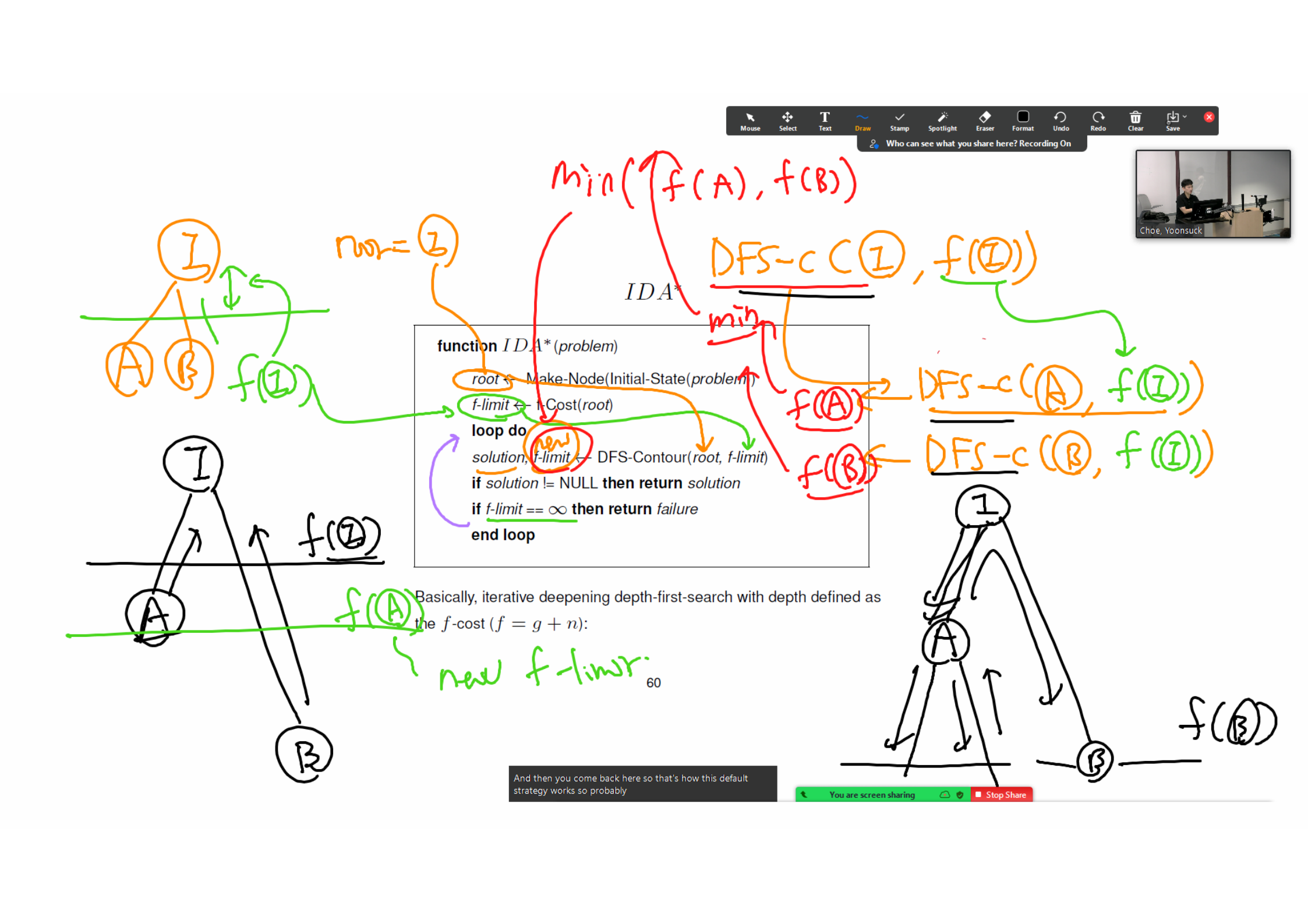
Problem 6 (Written; 10 pts): For the problem shown in Fig. 2, show that the heuristic is admissible ($h(n) \leq h^*(n)$ for all n). Note: You have to compute $h^*(n)$ for each n and compare to the $h(n)$ table.

Hint: It is best to work backwards from the goal, where $h^*(h) = 0$ (already at goal), $h^*(f) = 17$ (true minimum cost from (f) to (h)), $h^*(b) = 40 + 24 = 64$ (true minimum cost from (b) to (h): note that there are multiple paths and t

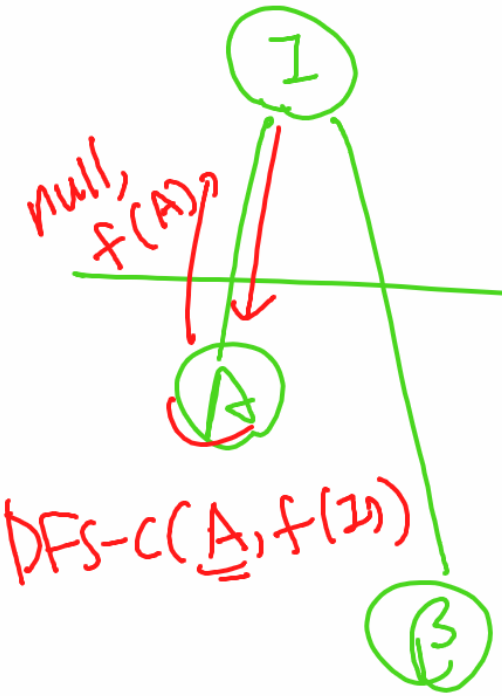
Problem 7 (Written; 2 graph below (Fig. 2), with initial node (a) and goa from node to node are shown as edge la

What are it? This is 24. So what? So now, then, you need to answer whether this all of this, compared to the true value, whether this is admissible


Mouse
Select
Text
Draw
Stamp
Spotlight
Eraser
Format
Undo
Redo
Clear
Save
Who can see what you share here? Recording On
Choe, Yoonsuck
IDA*
function IDA*(problem)
root ← Make-Node(Initial-State(problem))
f-limit ← f-Cost(root)
loop do
solution, f-limit ← DFS-Contour(root, f-limit)
if solution != NULL then return solution
if f-limit == ∞ then return failure
end loop
Basically, iterative deepening depth-first-search with depth defined as the f-cost (f = g + n):
60
So there's here's another interesting thing let's say
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
Choe, Yoonsuck
min(f(A), f(B))
DFS-c (I, f(I))
min
IDA*
function IDA*(problem)
root ← Make-Node(Initial-State(problem))
f-limit ← f-Cost(root)
loop do
solution, f-limit ← DFS-Contour(root, f-limit)
if solution != NULL then return solution
if f-limit == ∞ then return failure
end loop
DFS-c (A, f(I))
DFS-c (B, f(I))
f(A)
f(B)
Basically, iterative deepening depth-first-search with depth defined as the f-cost ($f = g + n$):
new f-limit
60
And then you come back here so that's how this default strategy works so probably
You are screen sharing
Stop Share

So that answer would be 120, because you want to return the minimum cost



DFS(I, ...)

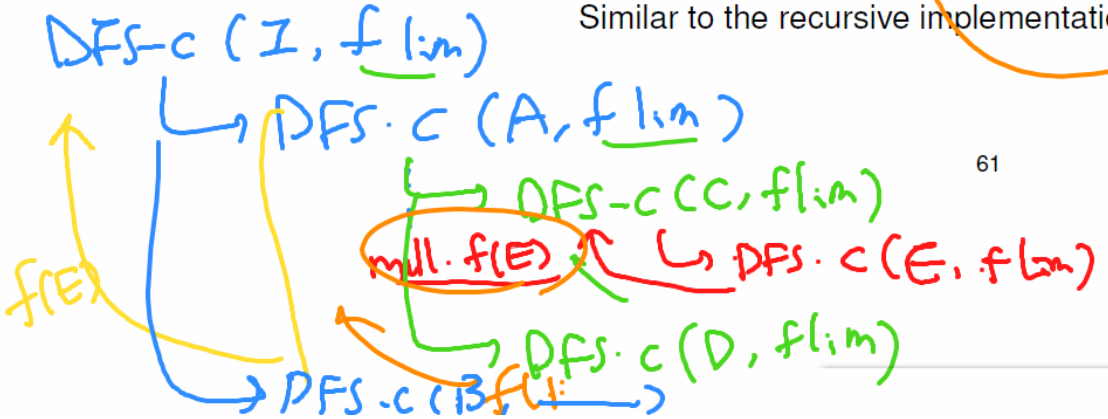
DFS-Contour(*root*, *f-limit*)

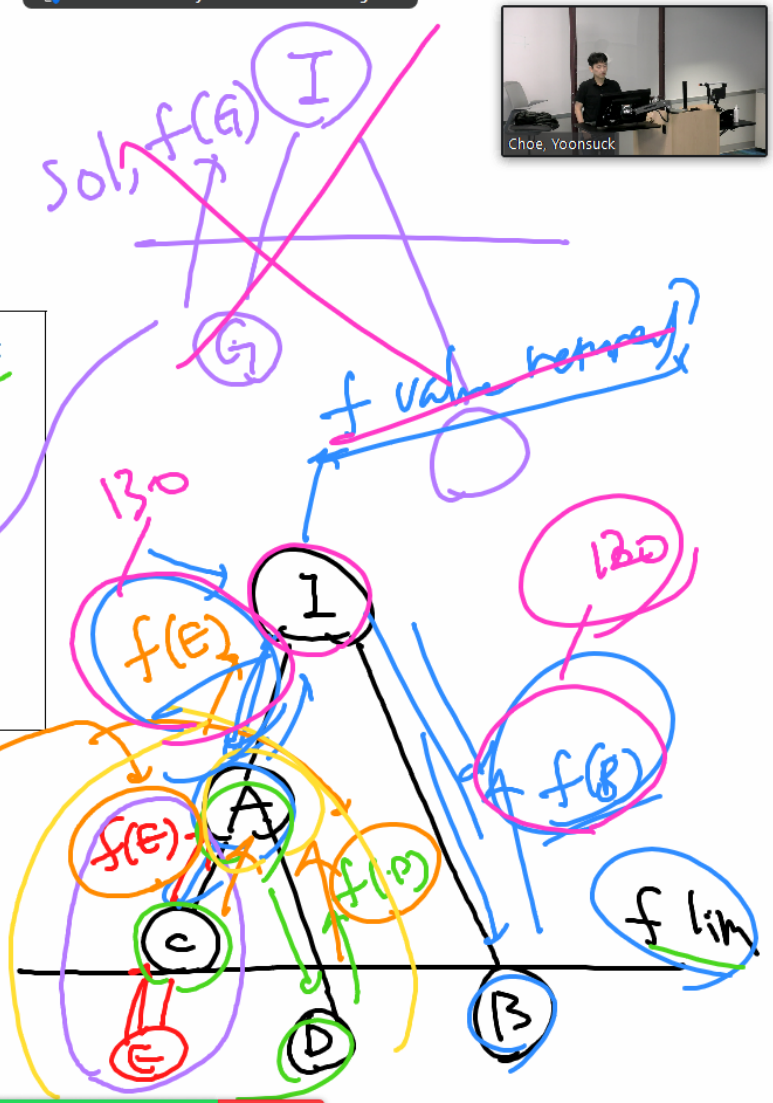
function DFS-Contour (**root**, **f-limit**): returns **solution**, **f-limit**

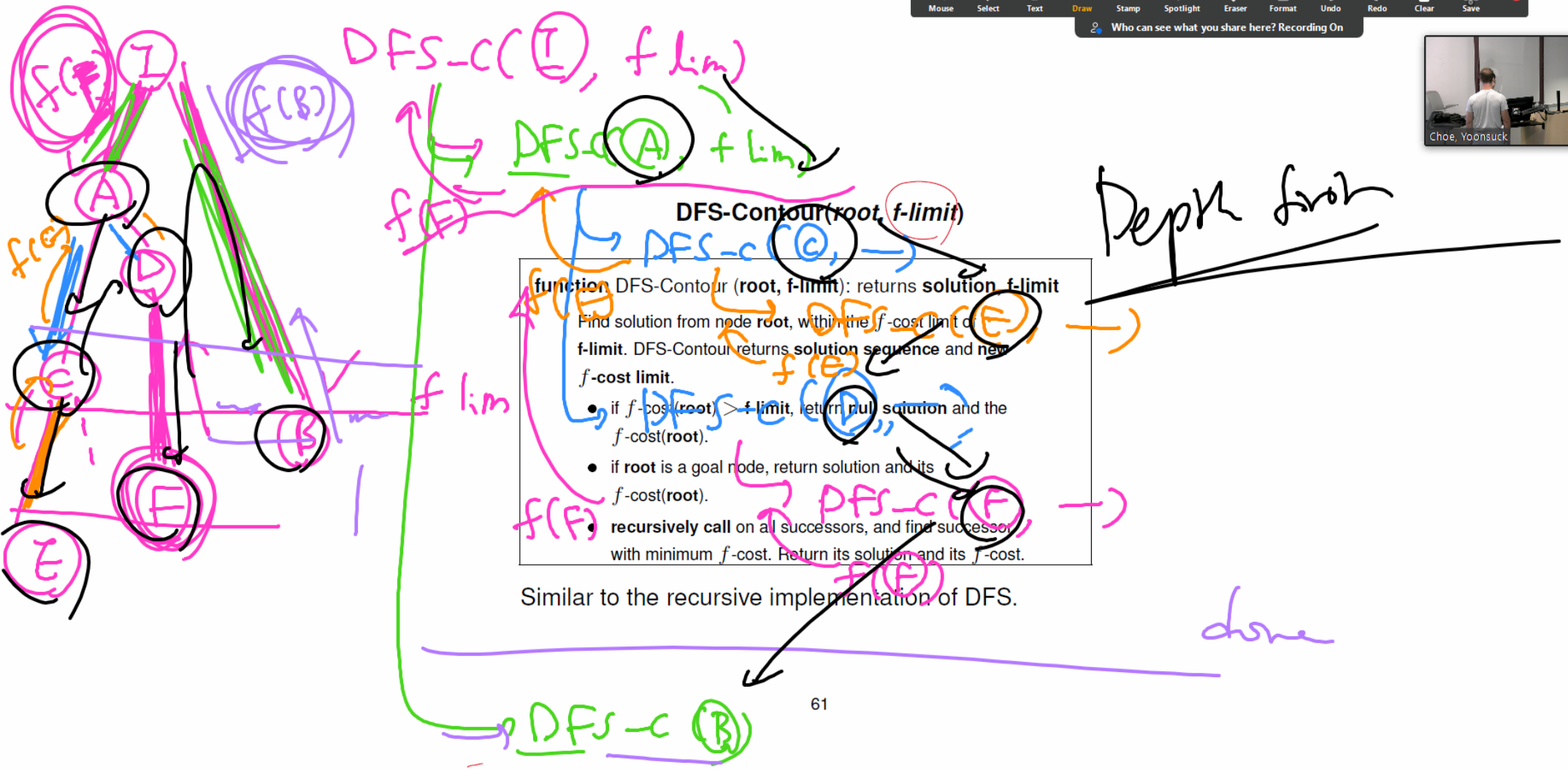
Find solution from node **root**, within the f-cost limit of **f-limit**. DFS-Contour returns **solution sequence** and **new f-cost limit**.

- if f-cost(**root**) $>$ **f-limit**, return **null solution** and the f-cost(**root**).
- if **root** is a goal node, return solution and its f-cost(**root**).
- **recursively call** on all successors, and find successor with minimum f-cost. Return its solution and its f-cost.

Similar to the recursive implementation of DFS.







DFS-Contour(*root*, *f-limit*)

function DFS-Contour (**root, f-limit**): returns **solution**, **f-limit**

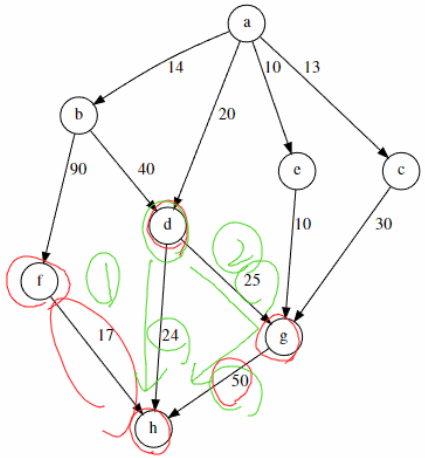
Find solution from node **root**, within the f-cost limit of **f-limit**. DFS-Contour returns **solution sequence** and **new f-cost limit**.

- if f-cost(**root**) $>$ **f-limit**, return **null solution** and the f-cost(**root**).
- if **root** is a goal node, return solution and its f-cost(**root**).
- **recursively call** on all successors, and find successor with minimum f-cost. Return its solution and its f-cost.

Similar to the recursive implementation of DFS.

Problem 5 (Written; 10 pts): Consider iterative deepening search. When the goal is (9), how many nodes are visited before reaching that node?

2 Informed Search



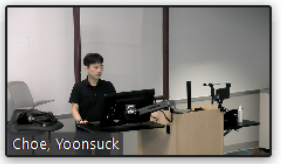
Node	h(n)
a	22
b	22
c	10
d	23
e	14
f	16
g	10
h	0

Figure 2: **Informed Search.**

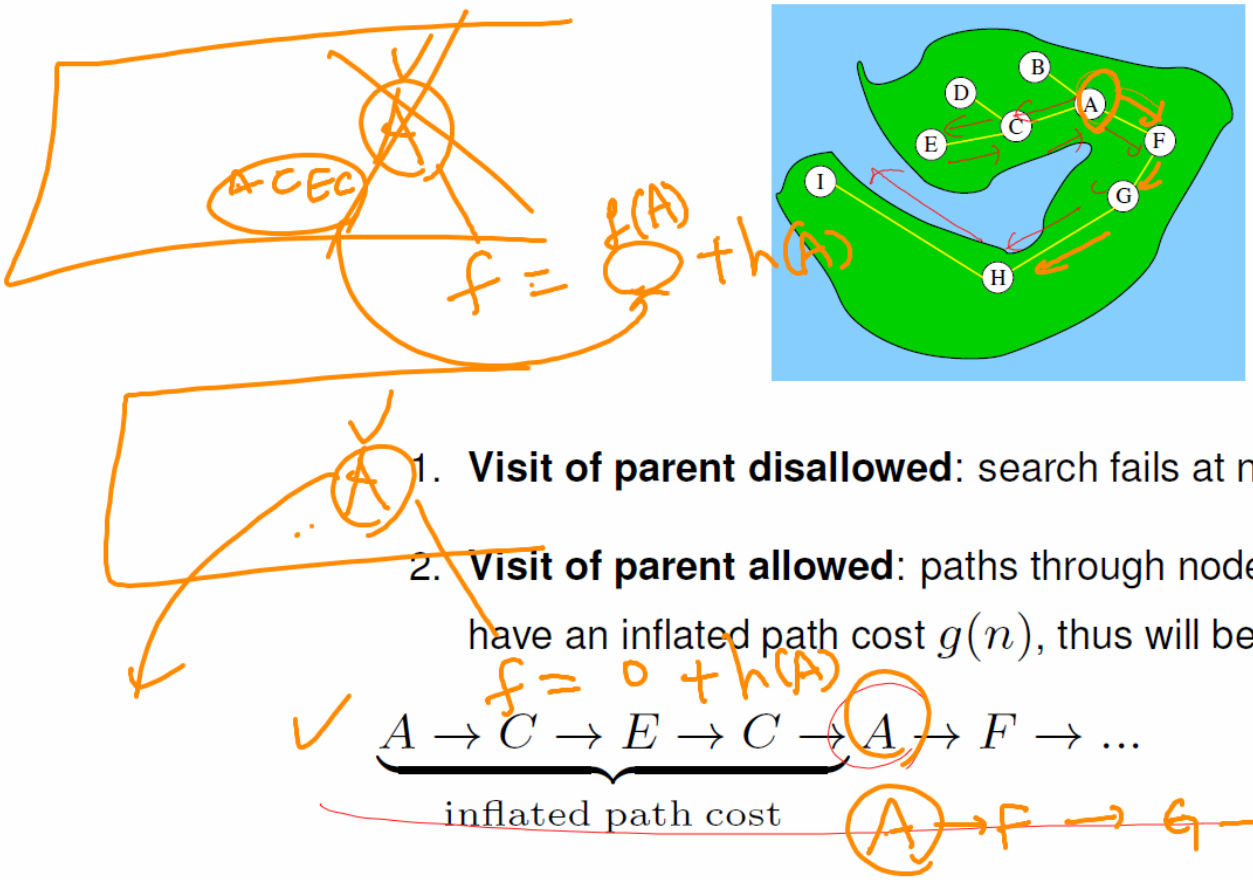
Problem 6 (Written; 10 pts): For the problem shown in Fig. 2, show that the heuristic is admissible ($h(n) \leq h^*(n)$ for all n). Note: You have to compute $h^*(n)$ for each n and compare to the $h(n)$ table.

Hint: It is best to work backwards from the goal, where $h^*(h) = 0$ (already at goal), $h^*(f) = 17$ (true minimum cost from (f) to (h)), $h^*(b) = 40 + 24 = 64$ (true minimum cost from (b) to (h): note that there are multiple paths and this path has the minimum cost!), etc.

Problem 7 (Written; 20 pts): Manually conduct greedy best-first search on the graph below (Fig. 2), with initial node (a) and goa[illegible] actual cost from node to node are shown as edge la[illegible]



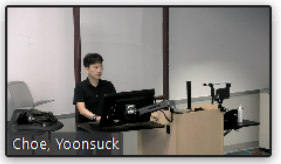
Optimality of A^*: Example



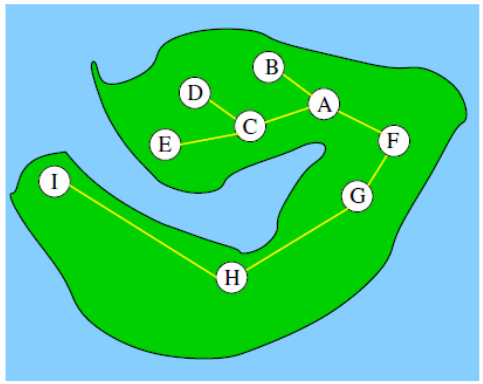
1. **Visit of parent disallowed**: search fails at nodes **B**, **D**, and **E**.
2. **Visit of parent allowed**: paths through nodes **B**, **D**, and **E** will have an inflated path cost $g(n)$, thus will become nonoptimal.

$$A \rightarrow C \rightarrow E \rightarrow C \rightarrow A \rightarrow F \rightarrow \ldots$$

inflated path cost



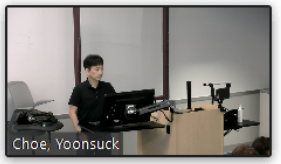
Optimality of A^*: Example



1. **Visit of parent disallowed**: search fails at nodes **B**, **D**, and **E**.
2. **Visit of parent allowed**: paths through nodes **B**, **D**, and **E** will have an inflated path cost $g(n)$, thus will become nonoptimal.

$$\underbrace{A \rightarrow C \rightarrow E \rightarrow C \rightarrow}_{\text{inflated path cost}} A \rightarrow F \rightarrow \ldots$$

So this won't be selected at all so this is called the inflated pass coast

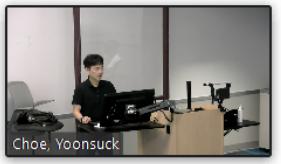


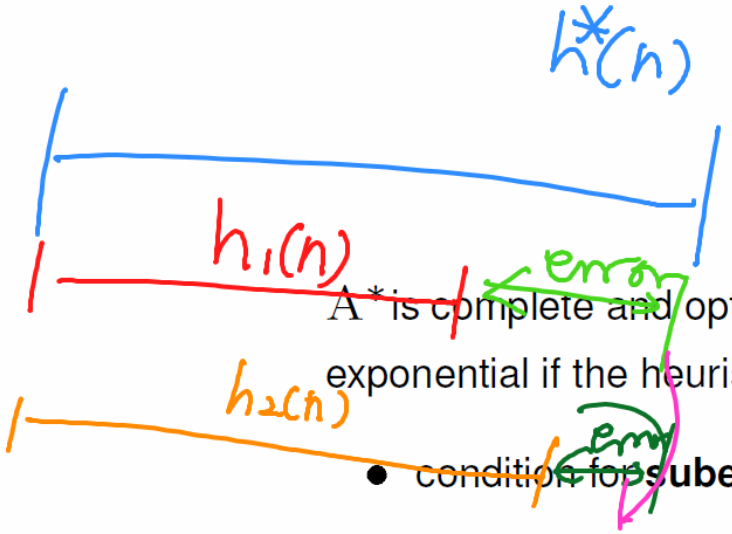
Lemma to the Optimality of A^*

Lemma: A^* visits nodes in the order of increasing $f(n)$ value.

- Gradually adds **f-contours** of nodes (cf. BFS adds depth layers).
- The goal state may have an f value: let's call it f^*
- This means that all nodes with $f < f^*$ will be visited!

So you'll have you find that the number of these nodes that are tangling outside of you have control will become exponential





Complexity of A*

A* is complete and optimal, but space complexity can become exponential if the heuristic is not good enough.

- condition for **subexponential** growth:

$$|h(n) - h^*(n)| \leq O(\log h^*(n)),$$

where $h^*(n)$ is the **true** cost from n to the goal.

- that is, error in the estimated cost to reach the goal should be less than even linear, i.e. $< O(h^*(n))$.

Unfortunately, with most heuristics, error is at least proportional with the true cost, i.e. $\geq O(h^*(n)) > O(\log h^*(n))$.

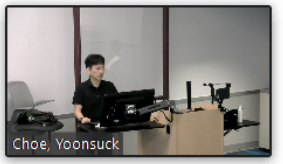
Q) Which heuristic is better?

① h_1

② h_2

$h^*(n) = 100$ (98) $\sim \leq 2$

$h^*(n) = 1000$ ≤ 3 (997)

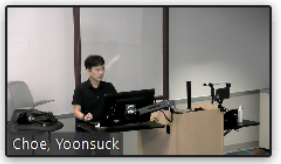


Problem with A^*

Space complexity is usually **exponential**!

- we need a memory bounded version
- one solution is: Iterative Deepening A^*, or IDA^*

Then you may



Problem with A^*

Space complexity is usually **exponential**!

- we need a memory bounded version
- one solution is: Iterative Deepening A^*, or IDA^*

≈ IDS for A*?