**Mini project2**

To clarify, I assume we can place a block on the table and then we may place another block on top of the block on the table OR can only one block be on the table at a time? Thanks!

our move is valid. You can also put as many blocks on the table as you'd like.

The problem is probably best visualized as an infinitely large table, which holds all of the stacks of blocks on top of it.

K

If we can put an infinite number of blocks on the table, wouldn't you just put all the blocks on the table and simply build the answer? I guess that won't give you the *optimal* path so the test cases must be looking out for that?

W

You get points for solving and points for optimal path.

P

Will is correct, you can certainly put all the blocks on the table and then just restack them, but 99/100 times, that is not the optimal answer. So you'd likely only get 20/40 points on Gradescope.

K

Makes perfect sense. Thanks all! Means-ends analysis here I come 😆

Mini-Project 2 and number of piles[#26816 days ago](https://edstem.org/us/courses/3783/discussion/220699) in **Mini-Projects – Mini-Project 2**

I was assuming that there was a maximum number of piles of 3 when I first started looking at this, but in one of the lectures, it was mentioned that there could have been a fourth pile made. Am I correct now in assuming we can create as many piles as we want?

Theoretically, while not optimal, we could put every block on the table and then arrange it in the solution order. Is that correct?

you can create as many piles as you want, which means as many blocks on the table as you want. Yes, you can put every block on the table but you are graded on how optimal your solution is

New States[#271](https://edstem.org/us/courses/3783/discussion/220783)

**Mini-Projects – Mini-Project 2**

Hey All,

I have the large program flow figured out for mini-project 2. I even know what algorithm I want to use. I'm, however, stuck on picking the best way to generate new states, aka edges, from an existing state. I thought about doing powersets, but that would generate illegal new states and compute way more than is needed. I'm a bit worried about this problem timing out as **permutations** grow at a wild rate. Anyone else overthinking this? Anyone have any resources to check out? Thanks :)

K

Hey Kyle -- I ran into the combinatorial growth rate too. For any node I visit, I do generate all of that node's children. But, I use a means-end analysis approach to decide which of those children are worth visiting. (In fact, my program doesn't just look at those children, but any previously visited children too: if A generates AB, AC, and AD, and my distance heuristic decides to go to AC next, I generate ACA, ACB, ACC, ACD, and then choose between AB, AD, and ACA...ACD for the next visited node.)

The provided distance heuristic on the assignment page is enough to get this approach to *a* solution, and you can improve it a little to get the optimal solution!

A

I have implemented a similar procedure where each node generates its children nodes with each one having some "distance" away from the goal state. These are then inserted into a priority queue based on how close they are, i.e. closer nodes appear earlier than nodes further away from the goal state. I have definitely run into some issues with the number of nodes becoming too large (problem is unsolvable) and or I get a correct, but very non-optimal solution. I'm wondering if maybe how I classify nodes is slightly wrong.

Is it safe to assume that I should always seek out moves that are at least as good as the previous state in terms of proximity to the goal state?

When I implement this approach, I can get some optimal solutions, some non-optimal but correct solutions, and some that take forever to run. Another alternative that I have been thinking about could deal with making a goal state, just a portion of the stack (problem reduction) or potentially changing the heuristic to only include nodes that achieve a higher similarity than the previous state?

Sorry for the long post, but I'm just wondering if anyone has suggestions. Thanks!

If I could suggest one thing, it is to work out a series of problems on paper and track the problem space by creating a semantic network. If you start by first figuring out the best way to optimally solve the problem by hand, then the rest is just putting it in code...which of course is easier said than done, but it's definitely easier than trying to do both at the same time.

100%. I didn't work it out on paper before coding and I really wish I had!

A

Yea, that's what I did to start, but maybe going back and doing it again on some different problems will reveal something that I am missing.

I'm curious, are you putting these options in a priority queue and knowing which to explore first or some other way?

K

That would be the right way to do it, and if I was starting from scratch I would! But I literally just search the entire list every time, that's "efficient enough" to solve these in my experience on the assignment

The natural follow on question though would be how to know you have arrived at an optimal solution. If you are working with a priority queue you are no longer in a BFS situation and not guaranteed optimality. I'm thinking you would still need to continue processing everything in the queue but throwing away anything that was going to generate a longer solution string.

Thoughts?

You all are the best. I wasn't thinking too deeply and used priority queue as I was worried about timing out more than optimal path. That explains why I can complete the challenge but not optimally! I'll be working on making these changes today and update the thread for anyone following or running into the same problem.

K

Hm, after sleeping on it: if the heuristic is strong enough to get you to an optimal solution, it might be strong enough to choose the "right" child at every step (ie, none of the subchildren should make the heuristic worse.) Which would mean you could ditch the queue entirely and just drive straight to the answer. For some reason my gut tells me this isn't always the case but I can't think of a counter example. LMK if your agent implements an approach like this!

K

I think it depends how strong your heuristic is!

## Additional story examples[#274](https://edstem.org/us/courses/3783/discussion/220900)

K

Hello!  
  
I was wondering if the instructors would consider releasing additional public test stories for this project. There are a very high number of potential sentence structures to consider, but we only see two. (This is less of a challenge for questions, since we do see eight of what those look like.) Even just a two to three more would be incredibly helpful to forming an idea of the universe of stories our agent will need to be able to represent.

1

Now I know they haven't responded, but I have just made test cases from the lecture notes and modified them within my code. Below is just the lecture notes sentences without any modifications.

**PLEASE NOTE**, some of these sentences ***won't work*** because the WORDS are ***NOT*** in our 500 word world(*mostcommons.txt*). You can add words from the 500 words and play with these basic sentences to build your own test cases.

Hope this helps everyone

* Bill put wedges on the block
* John took the book from Mary
* John put fertilizer on the field
* John gave Susie the idea to go to the movies
* Anika decided to have a glass of water
* John pushed the cart
* Bill propelled a bullet into Bob
* Marc loved watching TED talks
* Ashok enjoyed eating the frog
* Maria told Ben to throw the ball
* Ashok made pancakes for David with a griddle
* David went to the meeting with Ashok by Car

To give more help on how I am making my extra test cases. I am taking these statements from the lecture notes and adding questions about it.  
**Ashok made pancakes for David with a griddle at 8:00am**

(Make sure to add *Ashok* to 500 words, he is not in it or simply change *Ashok* to a name in the mostcommon.txt)  
 **QUESTIONS**

* What did Ashok make? (pancakes)
* Ashok made what for David? (pancakes)
* What is the griddle for? (pancakes)
* What did Ashok make pancakes with? (griddle)
* At what time were pancakes made? (8:00am)
* How were the pancakes made? (griddle)
* Who did Ashok give pancakes to? (David)
* Who made pancakes? (Ashok)
* Who is with Ashok? (David)
* Who is with David? (Ashok)

Hope this helps you all. Take any of the statements above from the lecture notes and just apply some common questions that can be inferred from that statement

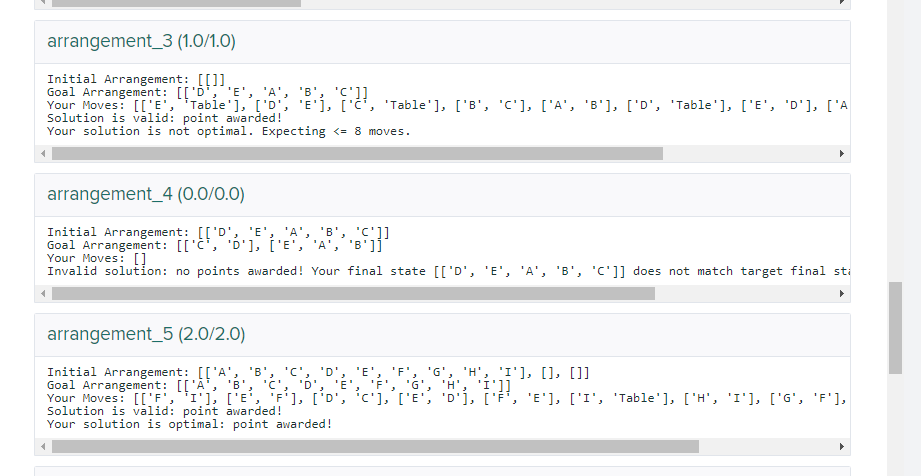
I second this request. The potential state space for possible sentence structures and question pairings is enormous. I'm having trouble just thinking through all the possible primitive action and question combinations. When you add in elements like prepositions that may or may not be present in the initial sentence it starts to feel unmanageable for a project that's only supposed to take 6 working hours.

Initial Arrangement: [['A'], ['D', 'E']]

Goal Arrangement: [['A', 'C'], ['D', 'E', 'B']]

Your Moves: [['C', 'Table'], ['B', 'E'], ['C', 'A']]

Invalid solution: no points awarded! Block not found:



## Code running on smaller problem sets, but taking too long on larger sets

I am able to work through logic and get the solutions. But with my limited python knowledge, I am limited to basic data structures. When sheep count exceeds 8, my program takes too long to execute. Are there any TA hours that I can attend to get some guidance on debugging the memory usage or guidance on a better data structure to use? If TA hours are not available, it would be helpful if you can direct me to topics in general direction which I can read up and get ideas on designing it better. I am self taught in python, so googling gives me lot of results and it is confusing

I'm going to assume that your logic is correct because you can get the correct answer for shorter scenarios. What is probably happening is the explosion of states you are visiting. The best thing to try now is to keep a list of all the states you have previously visited and if you happen to find that state again, just drop it on the floor. This little trick made mine run 1000 sheep and 999 wolves in about 1.2 seconds.

Thank you for the tip about keeping track of the states you've visited. It seems obvious in retrospect but adding that was the breakthrough I needed.

Reply

A

I am creating new 'paths' and saving it whenever a valid move can be made. If the previous state exists within the same path, then I drop the path. But when multiple paths end up in same path, I do not drop it. This way , it exploded even when it was 5 sheep, 3 wolf use case. May be I should drop longer paths when multiple paths lead to same state

If you are using a breadth first search, you can drop any state that has been seen at all. You are guaranteed to get the shortest path that way.

Reply

A

I am not fluent in advanced algorithms. So not using DFS/BFS and wrote a logic by myself. For all I know, it is one of the two mentioned ! Dropping paths that have any previous states worked and now everything executed in gradescope with that one modification!!! Thanks Jeff for the feedback !

You are probably using something like BFS. I would strongly suggest you take a look at this. It is really pretty easy. You'll be needing it for upcoming assignments. Also looking at something called a priority queue. You don't need to know how it works, but how to use it would be a great leg up for the next one.

Reply

A

Thank you so much, I will take a look at it and see if I can modify my code. My current code slows down for larger number of animals. When I gave 150 sheep and 100 wolves, it took 42 sec. So it is still not optimized. Whenever you get time, can you comment topics that helps if I read up on them? Much appreciate your guidance!!

Reply

For this assignment, all you need is a basic BFS and a queue. The queue is where you push things to explore. Basic steps...

1. Put the first state into the queue
2. pop a state off the queue
3. did you find the final state? if so then return the path
4. create all the next states to explore and push them onto the queue
5. go back to [#2](https://edstem.org/us/courses/3783/discussion/threads/2)

Let me know if you don't find anything on BFS or queue and I'll take a look. There are tons of videos on youtube for both of them.

Thank you so much! This comment was a breakthrough for me. My BFS is reaching the final state for 5 of the default tests. Now I'm working on return the correct move list

Awesome, glad it helped!

And now I'm done! Woo hoo!

2

I've also had a similar issue, later I saw the patterns in the solutions and tried changing the code a bit. See the moves (solution array) of the problems starting from 3 sheeps till 7 sheeps, you will understand what to do. Refer this thread for the number of optimal solutions you get for any sheeps - [here](https://edstem.org/us/courses/3783/discussion/222338)



A

Peter Vincent Carragher

[14 days ago](https://edstem.org/us/courses/3783/discussion/223747?answer=540663)

I've had the same problem. I thought I was tracking the states that had been 'seen' and ignoring them in the future.

But what I was only adding the current state to the 'seen' list, instead of all states I had generated from the current state. I switched to adding all newly generated 'successor' states to the seen list, and everything worked after that.

This worked for me as I also kept an 'unseen' list that I pick a random state from each iteration to continue generating successor states. The issue was that I might add many identical successor states to the unseen list on separate iterations without checking if they had already been seen.

Comment

Add comment

H

I struggled with performance and identifying the right data structure. I realized, I am thinking in a different programming language and trying to convert it to Python. I feel the data structures in python are really flexible and accommodative. Starting with pseudocode and converting it into python code proved more productive than the earlier approach.

I use basic data structures tuple, list, dictionary, queue and set. If you are not able to get the printed answer for large number of sheep immediately(eg. (16, 3)), maybe you can check if your issue is due to storing too many new states. Do you implement the semantic net instead of traditional bfs/dfs? Do you throw out the unproductive and duplicated states? When you generate new states, do you skip the computation of unproductive movement?

A

I do not have much background in algorithms, so not using traditional bfs/dfs. I am creating new 'paths' and saving it whenever a valid move can be made. If the previous state exists within the same path, then I drop the path. This way I am storing a lot. It is returning answers correct, but takes forever when size increases. Any good resources to read up on DFS/BFS that you know?

A

It sounds like you might be getting into an endless loop. Maybe you're missing a test case or re-visiting states that you've already visited. I'd suggest double checking your code to make sure you've thought through these and implemented them in the right spots! That's what ended up being my problem.

A

Thank you so much for your feedback. I am creating new 'paths' and saving it whenever a valid move can be made. If the previous state exists within the same path, then I drop the path. But when multiple paths end up in same path, I do not drop them. This way, it exploded even when it was 5 sheep, 3 wolf use case and issue was the inefficient data storage method. I modified based on Jeff's suggestion to drop any path that has a previous state instead of dropping only paths where previous state exists within the path itself. Gradescope worked when I made this change as it reduced the size stored in memory considerably. But it still takes about 43 sec to solve for 150 sheep and 100 wolves.

Nice!! I would say 43 seconds to solve 150 sheep and 100 wolves is pretty dang good :)

If you're new to python, some data structures you may find useful for this problem specifically are sets and queues.

A

Thank you Patrick for your suggestion. I am using sets, but not familiar with queues. Let me read up on them. Appreciate

Just realized my error for block world. It has to do with positioning of rows. Example:

current state: D | EAB | C  
goal state: CD | EAB

I've been treating the problem like 2D matrix, however, is it acceptable to have piles sorted internally but not in correct order, e.g.,

Does EAB | CD. == CD | EAB

If not, then not only does order of blocks inside a row matter, but also the order in which you construct the answer matters, e.g, which node to take off a stack first. I've been doing the latter and getting suboptimal answers in gradescope.

Comment

Sort by Newest

A

Alec Ryan Carruthers[14 days ago](https://edstem.org/us/courses/3783/discussion/228406?comment=550570)

As a suggestion, maybe just keep track of the blocks on the table as opposed to specifying a pile number/order. If you have a strong enough heuristic or node selection criteria, your code will put them in the right pile. I started with pile numbers, but later changed it and everything worked out alright.

In my solution I assumed the orders of stacks has to be preserved as well. I guess we will need an instructor to confirm this.

I believe the order of the stacks will be important, however, what was tripping me up was the order of blocks on the table, which I assume does not matter. Can someone confirm this assumption. I thought I was losing my mind by always getting below optimal 🙃

A

Alec Ryan Carruthers[14 days ago](https://edstem.org/us/courses/3783/discussion/228406?comment=551330)

I mentioned it above, but I got all optimal solutions without having to consider a pile or stack order.

Zuliat Owoade[14 days ago](https://edstem.org/us/courses/3783/discussion/228406?comment=551811)

[Replying to Alec Ryan Carruthers](https://edstem.org/us/courses/3783/discussion/228406?comment=551330)

Thanks for the input! I took out all logic that was considering the exact location of each stack and it worked like a charm!

4Reply

Nice. That makes the problem a bit easier.

## Question on starter example[#394](https://edstem.org/us/courses/3783/discussion/228607)

W

Weichen Wang

[14 days ago](https://edstem.org/us/courses/3783/discussion/228607) in **Mini-Projects – Mini-Project 2**

In the project description example, with Initial: [["A", "B", "C"], ["D", "E"]]  
Goal: [["A", "C"], ["D", "E", "B"]], the acceptable move is ("C", "Table")  
("B", "E") ("C", "A"). So when I calculate initial delta, it is 2, since for initial state: [("table", "A"), ("A","B"),("B","C"),("Table","D"),("D","E")] and for goal: [("table","A"),("A","C"),("E","B"),("Table","D"),("D","E")], the difference is [("A","C"),("E","B")] and thus delta is 2. Then for move ("C","Table"), the delta is actually increased to 3, using same approach. Am I calculating in the right way? Or I misunderstood the means-end analysis?

Comment

1 Answer

A

Alec Ryan Carruthers

[14 days ago](https://edstem.org/us/courses/3783/discussion/228607?answer=550980)

Just based on the bottom block, B and C are in incorrect locations in the initial state, so a delta = 2. Moving C to the table is still not the correct state, as it should be on A. Basically, C went from one incorrect state to another, so its delta contribution is 1 in both cases. B did not move, so it still has a delta of 1. Therefore, the delta should be equal to 2.

Comment

Add comment

W

Weichen Wang[14 days ago](https://edstem.org/us/courses/3783/discussion/228607?comment=550996)

Thanks for pointing out. Even the delta is unchanged, based on lecture video, the delta is not closer to the end goal, and therefore means-end analysis should not be applied here. Does this mean we cannot use it in the mini-project 2?

In order for the delta to decrease, subgoals would have to be considered.

For instance, if you have one initial stack [E, A, B, C, D], and the goal is one stack [A, B, C, D, E], you would have to do moves that would increase overall delta (taking D, C, B, and A off the initial stack) because you have to get E from the bottom.

But clearly we are getting closer to the goal. So the idea here, I think, is to figure out how to choose/evaluate subgoals so the overall delta is actually decreasing in a scenario like this.

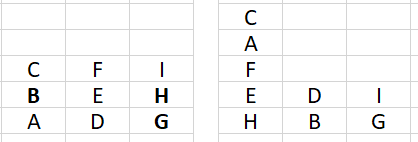
## Does order of final stacks matter?[#470](https://edstem.org/us/courses/3783/discussion/231158)

H

444466

VIEWS

I'm struggling with solving goal\_arrangement\_7 by hand:



The way I'm solving it essentially hinges on getting base blocks first, so my first move would be to put C on table, so then the state would look like [[B,A] , [D,E,F], [I,H,G], [C]] Which is fine. Then I would move (B, Table) and my state would then be: [[A] , [D,E,F], [G,H,I], [C], [B] ].

The issue I have is - B is now in front of G, and ONLY way I can see to change the ordering is to do (G, Table) however that move seems invalid because G is already on the table?   
  
So only way to fix that is the move G on top of a block, then back on to table that way it is after B. Would that be the correct way to solve this? Or does the order not matter at all?

Comment

## Branching factor/Complexity with BFS for Blockworld[#748](https://edstem.org/us/courses/3783/discussion/253648)

Hey all,

y'all mind if I ask for some help with the math here. Before my heuristic worked I'm fairly certain my BFS would of ran forever. I was trying to think about how many nodes there were in the graph but i'm not positive.

if you have N stacks each time you get moves thats N^2 moves per state I believe... not sure where to go from here.

anyone want to do some math 😆

Comment

1

I think the branching factor is a great way to look at this. The calculation for the branching factor would be:

Num piles \* num piles

Essentially what this is saying is you can put the top of any other pile plus the table. The only blocks that can move are the ones in the top so it doesn’t really matter how many blocks there are in the puzzle, just how many piles there are at any given time. The problem becomes the more blocks you have the more piles you can have.

Comment

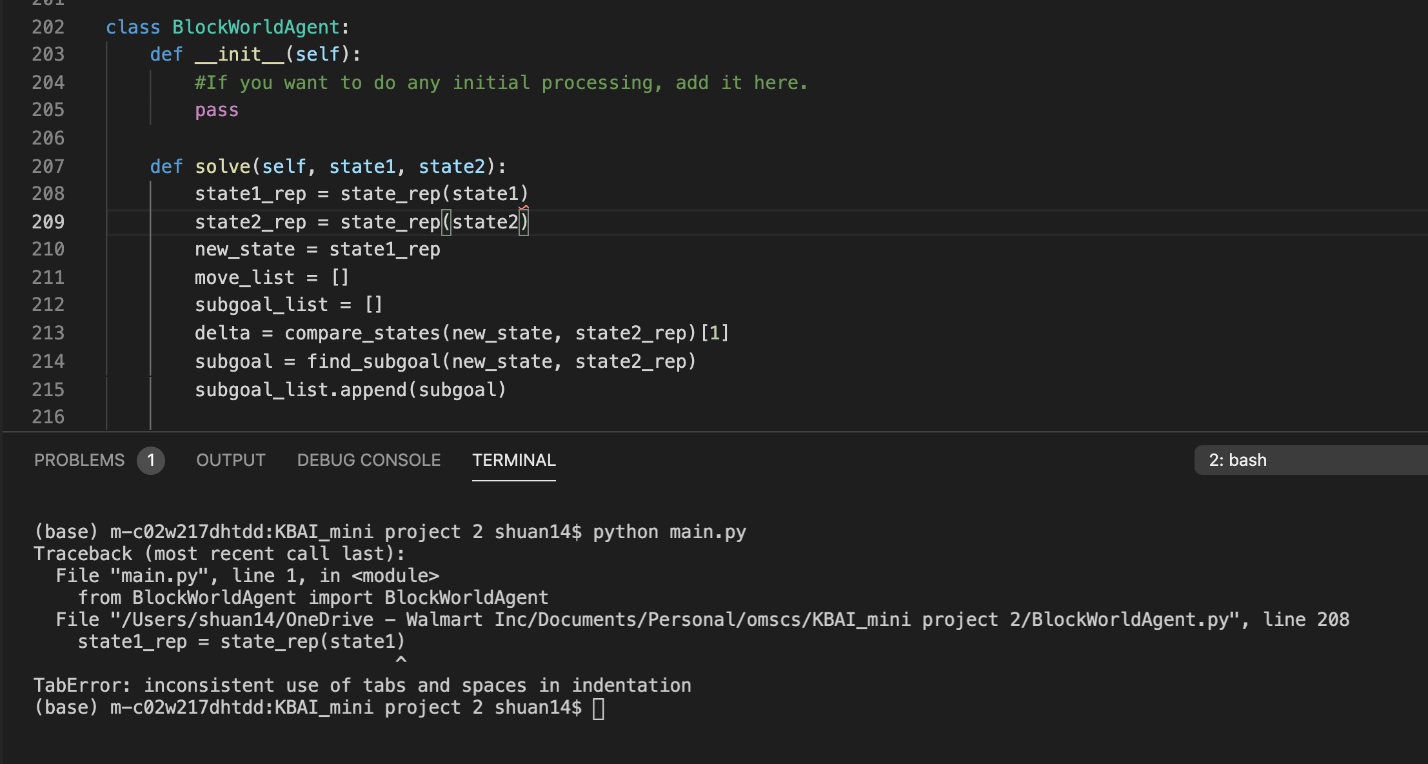
1

Yes. I think N^2 moves per stack. Then 'm' turns would give you growth like (N^2)^m nodes in the network. I just worked through the 2nd start state with Goal [#5](https://edstem.org/us/courses/3783/discussion/threads/5) and it takes 10 moves. You can actually keep the number of stacks at 3 while solving in the optimal number of moves. The highest number of stacks would be 9. So the number of nodes at a minimum would be 3.49e9 and at a maximum 1.22e19. And, since you're expanding the number of possible moves every turn its going to be much closer to the maximum. I naively thought, I can just recycle the code from MiniProject 1. I was wrong.

2

hey! check out Alec's reply in [#389](https://edstem.org/us/courses/3783/discussion/threads/389). I took out the logic in my code that was considering order of stack and then I got points for efficiency.

I went into an indentation code error when I tested the main.py. But actually the same code went well in my jupyter notebook. I really have no idea about the error. Please help!



Comment

4 Answers

E

3

I ran into a similar issue with VSCode. It's a simple fix. Press Ctrl + Shift + p.  
  
Type in "Convert Indentation to Tabs" or "Convert Indentation to Spaces" into the search bar, whichever one you use. The problem should be solved. :)

Any advice before I throw my hands up[#766](https://edstem.org/us/courses/3783/discussion/255014)

Hi all,

I'm currently able to generate states, able to compare states, able to get a delta, checking to see if the blocks are in their final state, but stuck in infinite loops. Rather than stressing about it I'm accepting that I'll probably get a 0 on this and move on with my life, but figured I'd gather any advice people have here. I'm happy I've gotten this far and think it's a silly heuristic I'm missing, so maybe some conversation will get the cogs working!

Comment

Add comment

Sort by Newest

P

Make sure you are minimizing the number of moves you consider. Not moving from the table to another spot on table, to the same stack, to a stack not in a goal state, etc.  
That solved my looping problem in my local tests.

Reply

H

So I used a bfs/priority type algo and had issues with it taking way to long to find the solution. I spent a lot of time trying to make my heuristic smarter and more complex but I was still timing out on gradescope. What ended up working for me was to make my generator a lot smarter, as in make it generate as few states as needed. Pairing that with a really simple heuristic made it work fine. Not sure if that helps

1Reply

I'll look into that!

Reply

R

Ryan Patrick Leonard[42 minutes ago](https://edstem.org/us/courses/3783/discussion/255014?comment=611157)

I think you may be overthinking the solution a bit. Start by drawing it out on paper, then write the steps to solve in plain English. I know it's lame and we all want to jump into coding, but this helped me a lot when things got confusing.

I'd recommend trying to find the simplest solution to the problem, rather than finding the most efficient as well first. Simply solving the problem regardless of how efficient it is, can allow you to worry about the complexities of efficiency later, and will make you understand the core code more.

I know it's a bit ambiguous, but obviously I can't just come out and say the answer, so hope this helps.

Totally understand what you're saying! And good advice! Currently drawing out the problem as it stands. I have states down no problem so that's one less thing to think about, just need to figure out how to get my delta to choose the one closest to the goal.

Reply

A

Alexander Bussmann[44 minutes ago](https://edstem.org/us/courses/3783/discussion/255014?comment=611146)

Hi Brannden, if you are using the distance metric from the lectures, you should end up with a number of possible states (moves) that are all ties. Think about how to perform tie-breaking. If you are stuck in infinite loops, is it possible you are revisiting states that you have already been to? If you are to think about the problem in its most basic form. If you move all blocks to the table first, and then re-arrange into the solution state, you should also never get stuck in loops. Caveat though, that would not necessarily be an optimal approach.

Good luck man.

Brannden Kaiel Moss[42 minutes ago](https://edstem.org/us/courses/3783/discussion/255014?comment=611156)

Thanks Alex! I'm currently using the distance metric, but it's getting confused deciding between the states of equal distance. I added some code that tells it when it's settled to improve the distance metric, but still stuck at same distance level.

Reply

A

Alexander Bussmann[38 minutes ago](https://edstem.org/us/courses/3783/discussion/255014?comment=611180)

[Replying to Brannden Kaiel Moss](https://edstem.org/us/courses/3783/discussion/255014?comment=611156)

When you are tie-breaking you need to define some additional metrics. Try something basic:

1. If, I did that before, don't do that again.
2. If, I can't decide, maybe the table is a better choice.

From there, you can think about a few other ***why might this move be better than another***?

One hint for the cycle detection:

What I did was normalize the state so that no matter the move, the state would always be represented by a unique string. ***str(state)***

1Reply

Working on incorporating 1) and 2), it's a headache to do with my current setup but good advice

## Some Search algorithms better than others for Mini-Project 2?[#538](https://edstem.org/us/courses/3783/discussion/236826)

D

4

I built out everything for Mini Project II using techniques from my Mini-Project I, however, for test cases 5-8, it's in an infinite loop. Just so I spend my time wisely, here is my question:

* Can one solve this problem with a breadth first search algorithm with a decent heuristic?

Reason I ask this because, I know that people on the forum have suggested that there should be weights on moves relative to "table" positions. However, I just want to make sure I spend my hours wisely on the heuristic as opposed to spending more time on using another search algorithm. I mean, since we are trying to minimize the number of "edge" traversals, then we can interpret the problem with equally weighted edges.

However, using means analysis and problem reduction, we "could" hypothetically make some decisions better than others, potentially making those edges greater in value.  
  
If anyone can verify that breadth first search can work, or if it is a no go, or any other guidance would be fantastic!! Thank you all for your help on this! :-D

Comment

Sort by Newest

M

Mohammad Nurul Minhaz[a month ago](https://edstem.org/us/courses/3783/discussion/236826?comment=572186)

Try doing by HeapQ on Delta. Heap is basically standard min-priority queue. Though add and get are lg n - it worked very well for me, almost no back-track. You can potentially improve that lg n using some kind of sorted dictionary of list - where key is the delta, you put list of states for that delta, always picking on lowest delta available. But given that heapq already worked for me, I am not going to bother.

1Reply

Using heapq significantly sped up my algorithm compared with sorting a list! Thanks for the tip. I'm amazed how how much faster the log(n) vs n sorting really is, even on these problems that don't have super duper huge lists.

Reply

6 Answers

C

5

I can confirm it’s doable with BFS + priority queue, even without using subgoal, but we need a more efficient heuristic, the one from the lecture was not enough, I spend a lot of time to work out an efficient one, which indeed, is very simple, but effectively helped me to pick up the best one to the top of heap! You have already noted the hint about table in your original post:)

Comment

Add comment

D

Douglas Ian Judice

[a month ago](https://edstem.org/us/courses/3783/discussion/236826?answer=572605)

5

Compared to the first mini project, I felt Mini-Project 2 could be solved optimally without referencing much of the lecture notes.

I went away from thinking of each arrangement of blocks as a graph and instead considered only each separate block's relationship to its goal state: are they in a finished state or not? "Finished" blocks never have to be manipulated again, then you're only moving around unfinished blocks. The possible moves on each unfinished block that are actually productive is also quite small, and each time you move it to a 'finished' state, the problem space is reduced even further. I found it is possible to solve this problem without any actual state generation.

Comment

Add comment

2

David, I used a priority queue on an "otherwise BFS" base, a heuristic similar to what Alec describes and a generator so ruthless (ergo, totally unreadable code if I don't document it soon) it scares me. :)

Solves all 40 gradescope cases without timing out, and only 2 sub-optimal (because of a bug in my heuristic which I'll fix next week).

It's doable!

Comment

Add comment

Kyle James

[a month ago](https://edstem.org/us/courses/3783/discussion/236826?answer=571947)

1

Just to be that guy, I took an approach similar to lesson 7- a forward-chaining production system! I hate "optimizing" parameters. So, this approach was very cut and dry. I just really had to figure out the mental steps of solving the problem, which can be reduced down to two major edges-if you think of the problem space existing with potential combos fanning out.

Comment

Add comment

Jennifer Jordache[25 days ago](https://edstem.org/us/courses/3783/discussion/236826?comment=579149)

This is interesting, I didn't think about that...thanks for posting! And you meant lesson 6 right?

Reply

[a month ago](https://edstem.org/us/courses/3783/discussion/236826?answer=570425)

1

I was not able to do it in BFS, in fact a vanilla DFS hit (almost) infinite loop as well for me. What I have did is to generate child from the node that has more number of subgoals achieved, the more scattered the block on the table , the shortest distance vs end goal

kind of like ordering the list of node to be explored by

"order by SubgoalAchievedCount desc, BlocksOnTableCount desc, DistanceVsGoal asc"

I also spent decent amount of time tweaking the code to count subgoal achieved, how to measure distance vs goal..etc to ensure the agent explore the right node

Another learning I have is that the tester did not help in the way I explore the node (I did code it out anyway, just the check was not invoked by the Agent)

Hope it make sense. Would love to hear others ; )

Comment

Add comment

A

Alec Ryan Carruthers

[a month ago](https://edstem.org/us/courses/3783/discussion/236826?answer=570114)

1

I don't think you would want to consider this graph as having edges with equal weights. For example, if you have [A,B,C] and [D,E] with a goal state of [E,D,A,B,C], moving block E on top of C, which is perfectly legal, should not be considered equal to moving block E to the table. In mini-project 1, you could only generate at most 5 cases at each iteration. With this project, you can generate a whole bunch more.

To answer your question more specifically, in theory, a BFS should work, but unless you have some really smart generator and way of keeping the cases generated per level in the graph low, it will most likely be infeasible to get a solution. Others may have a different experience, but that is the problem I ran into with my first iteration of the code.

I would definitely say having the right heuristic, which their could be more than one, was the key for me figuring out this problem. In general, you need something that can prioritize moves to the table before stacking it on another block, UNLESS, all the blocks in the stack where you intend to place it are correct. Another example would be, if you have a goal state of [A,B,C,D,E] and you currently have [B,C,D],[A,E], you would want your heuristic to send E to the table instead of placing it on D, despite that being the correct block below it in the final state.

Hopefully this helps.

I have a very stupid question: when you say heuristic, what do you mean? Cost function?

Reply

A

heuristic, cost function, penalty, score,.... all one in the same essentially. All of them serve to put a value on a given state, which can later be compared to other states.

Reply

Thank you, Alec. I was completely stuck on this problem even after spending an inordinate amount of time on an A\* algorithm with my own heuristic. Based on your input, I instead made a smart generator and it passed with no problems.

## Heuristic for deciding which sub goal to tackle next[#594](https://edstem.org/us/courses/3783/discussion/241985)

1

I'm planning on building my solution with multiple BFS iterations, one for each required sub goal.

I'll use delta as described in the lecture for picking next state in the BFS.

However, I'm trying to think of an optimal heuristic for which sub goal to tackle next...

In the case that the solution only has a single stack, choosing the next sub goal is trivial, just append the next block up the goal stack to the sub goal state...

but in the case that the solution has multiple stacks, I feel like I need a second heuristic to decide which sub goal is next... My first thought was:

Assume total goal state has 2 or more stacks

if next sub goals available are getting (A onto B, part of stack 1) and (C onto D, part of stack 2) then pick the sub goal such that the (# of blocks in the current state on top of A + on top of B) or (# of blocks in the current state on top of C + on top of D) is minimized.

I think this would be optimal bc number of blocks on top cannot be greater than # of moves to achieve the sub goal + 1...

Just curious if anyone has tried this and failed, if there's something more I'm missing, or if any assumptions I've made aren't valid... What do you all think?

Instructors: Pls lmk if this is not too much detail to share... if so I'll make it public to the class again

Comment

Without a swift cutoff heuristic searching algorithms will introduce combinatorial explosion.

You need planning. ... (I think)

Reply

## Is it BFS + PQ or DFS + PR?[#622](https://edstem.org/us/courses/3783/discussion/243516)

Azmain Amin

[23 days ago](https://edstem.org/us/courses/3783/discussion/243516) in **Mini-Projects – Mini-Project 2**

1

I have seen a lot of people mention that they are using BFS + PQ(priority queue) to solve MP. But doesn't using PR make it into a DFS instead of BFS?

When using a PQ, we are picking 1 state i.e. the one with min delta/cost/heuristic and use it to generate next state and see if the goal is in there. If not, add them to the PQ and repeat. Doesn't this sound more like DFS than BFS? Can someone help me visualize?

Comment

5

A better way to think about this is to consider the evaluation function used in the algorithm. They breakout like this:

* BFS (Breadth-first search): uses shortest distance first from left to right.
* UCS (Uniform cost search/Cheapest-first search): cheapest cost first
* DFS (Depth-first search): longest node first (left-to-right)

The queue essentially helps you manage which of these are your priority, so DFS uses last-in-first-out approach to the queue, whereas BFS uses first-in-first-out approach to the queue.   
  
Theoretically, in Dijkstra, the algorithm would behave exactly the same as BFS if you have an unweighted graph. The priority queue becomes more important when you find that you need to sort the queue based on the cost and heuristic evaluation function.   
  
For example, in *A\**, you evaluate:  
*f*(*n*)=*g*+*h* where *g* is the path cost and *h* is the distance heuristic (e.g. euclidean distance on a Cartesian plane).   
  
Among other things valuable about the priority queue, if you look at the algorithm pseudocode, you will see that the priority might change if a node that has previously been visited is visited again with a lower cost.

Comment

Add comment

3

Yea, to my knowledge a BFS with a priority queue somewhat contradict each other. Let's say you visit 5 nodes and their costs are 10, 9, 8, 7, 6, with the lower numbers being better than higher. Adding this to a priority queue will have you visit the node with a cost of 6 first. You now generate potential states for it and let's say their costs are 4,6,7,9. Adding these to the priority queue would result in four being explored and this process repeats until you converge to a solution (hopefully).

A true BFS explores all nodes at a given level of a graph before exploring the next. A BFS + Priority Queue will explore all possible next states of the node with the most optimal cost, i.e. you could be exploring the fifth level before exploring the children of sub-optimal nodes in layer one.

In summary a BFS + priority queue: pick the node with the optimal cost, fully expand it's next level, pick the optimal node with optimal cost, expand it's next layer ....

Plain old BFS: Visit all nodes at a given level of the graph and generate their children, add all children to a queue, process all children nodes, generate the next level of nodes, add to queue....

This is to my understanding though.

Comment

Add comment

Azmain Amin[23 days ago](https://edstem.org/us/courses/3783/discussion/243516?comment=586789)

That's exactly my understanding as well! Might be wrong to label this as "BFS"

2Reply

J

2

To my understanding, that is neither BFS (FIFO queue) nor DFS (recursion / LIFO stack) as those have a defined order of traversal which this one doesn't follow as it can potentially jump around between states at different levels (move counts) and paths. IIRC I think this is called best-first search (priority queue / heap), or perhaps also the A\* algorithm. So while they may start with BFS, changing the data structure to PQ makes the search a different kind entirely.

Comment

Add comment

Azmain Amin[23 days ago](https://edstem.org/us/courses/3783/discussion/243516?comment=587398)

I think I have seen best-first-search mentioned somewhere. Also, isn't Djikstra very similar? You always take the route that has the lowest cost?

Reply



Jas L[22 days ago](https://edstem.org/us/courses/3783/discussion/243516?comment=593282)

BFS finds the shortest path and it's usually used when the location of the goal is unknown. It's only optimal when all cost of edges are 0 or the same. Dijkstra visit the node with shortest distance but the idea of Dijkstra is taking the whole concrete graph as an input and calculating the cost from start to every node. A\*search is used when we have the information of the goal. A heuristic is an estimation of the cost from node N to goal.

More information about these search algorithms:

<https://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html>

Reply

J

Jeanette Tan[23 days ago](https://edstem.org/us/courses/3783/discussion/243516?comment=587576)

Hmm I'm not sure, I completely forgot what Djikstra is about. I just mentioned A\* because the key to it is using the heuristic + the normal move count to decide what states to prioritise. Anyway I quickly googled Djikstra, and if I understand it correctly, it only considers raw cost and not any heuristic, and it's an algorithm to solve a different class of problem, like how to find shortest paths in an already defined graph. Whereas A\* is more of a search where you're generating new states as you go. Anyway, this is just my understanding, I don't know enough about algorithms to be sure.

1Reply

Prashanth Aditya Susarla

[23 days ago](https://edstem.org/us/courses/3783/discussion/243516?answer=586791)

1

A simple way to think about this is that in the next level, you don't dequeue states in the order in which they were enqueued. So if your generator emits 5 child states (s1, s2, s3, s4, s5) from a parent state p1, then you would analyze these child states in that order.

With a PQ, it's possible you may analyze them in the order s4, s2, s1, s5, s3, which is determined by your heuristic. To this extent, the behaviour is still like in BFS.

However when you do this, it's entirely possible that a child of s4 emits a state (say s41) that in priority makes it available before s3 (the last node in your current level according to priority). That doesn't mean, however, that you're *exploring* nodes in depth-first manner all the time, either.

Comment

Add comment

Azmain Amin[23 days ago](https://edstem.org/us/courses/3783/discussion/243516?comment=586821)

In BFS, you are guaranteed to explore all nodes in a layer first before going to the next. When that rule is broken, I am having a hard time calling it a BFS. PQ almost makes it a hybrid BFS + DFS.

1Reply

Prashanth Aditya Susarla[22 days ago](https://edstem.org/us/courses/3783/discussion/243516?comment=590186)

Correct, and by that token, it isn't entirely DFS either (apropos your question), was the point I was trying to make. :)

1Reply

Jeff Richley

[22 days ago](https://edstem.org/us/courses/3783/discussion/243516?answer=591379)

If you simply use a priority queue and a heuristic it is a Greedy First Search. It goes with the best bet it has at the time. You can also implement the A\* algorithm by using the score with how many moves you have taken plus the number of approximate moves needed to solve. Be careful with this because your approximation needs to be admissible in order to guarantee optimality.. In other words, it can't over estimate. It is fine to under estimate but not over estimate.

## Looking for more resources for Mini-Project 2[#626](https://edstem.org/us/courses/3783/discussion/243709)

2

I have watched Module 5 - Means-End Analysis and I still am not solving the problem correctly.

My approach is incredibly slow, nest For-Loops everywhere. I have looked at other posts such as [#419](https://edstem.org/us/courses/3783/discussion/threads/419) [#568](https://edstem.org/us/courses/3783/discussion/threads/568) [#594](https://edstem.org/us/courses/3783/discussion/threads/594) and I still am not making progress.

How I have my problem setup:

class Block:

Block Name such as "A", "B", ... "Z"

Block Location such as (row,column)

Blocks Goal Location (row, column)

If Block is Moveable. This is just a boolean which checks if the block can/should be moved

In the block class the 'is\_movable' boolean checks if the current block is in its goal location and if it is on top of a block, if that block is also in its correct position.

*Goal*      *CBA*​*ED*​*GF*​​

Looking at block "C"

If in setup

*CEG*​*AB*​*DF*​​

"C" would be in its goal position of (0, 0) but would still be moveable because "E" is not in its correct location.

*CBG*​*AE*​*DF*​​

In this "C" would still be moveable even though it is in its correct position and "B" is also in its correct position BUT "B" is still in a moveable state.

class State:

Current State

Goal State

Goal Locations of Blocks

Total Delta for State

Pseudocode

Create priority queue with initial state

Create Visited States

While priority queue is not empty and problem is not solved:

Get next item from priority queue

Find next states using that item, which also returns the delta for each state returned

Add states to visited

Check if any of those states are goal state

If goal state:

problem is solved

else:

add those next states to priority queue based on the delta calculated

The problem I am getting into is not finding an optimal solution.

Also my delta is calculated by checking if the block is in the correct index as well its neighbors are correct. 0 for correct + 1 for not correct, total.

So in this state

*CBG*​*AE*​*DF*​​

C would have a delta of 0

B would have a delta of 2

A would have a delta of 5, wrong index and wrong neighbors

Comment

3

Hi Josh,

A lot of great discussion in forum on how to solve and seems like have many different approaches..; )

I actually use a combination of 3 informations to prioritise which state to explore.

1. number of sub goals achieved

2. number of blocks on table

3. difference vs goal\_arrangement (means-ends analysis)

The state has the most sub goals achieved, has the most number of blocks on table and finally the smaller difference vs goal\_arrangement will get explored first.

I manage to get optimal solution for all test cases in Gradescope and finish pretty quickly (no timeout in Gradescope)

With the approach above, I actually modify the difference-vs- goal calculation to a much simpler way. I just concatenate them into single string and find element difference

i.e ["A","B","C"] vs ["A","C","B"] have 2 difference.

Hope it helps.

I did quite some trial and error in counting sub\_goals achieved, means-ends analysis and how to prioritise the state for exploration. This is what I got that works..

Comment

Add comment

J

John Seungwoo Kook[22 days ago](https://edstem.org/us/courses/3783/discussion/243709?comment=594542)

what is the difference between the "subgoal" and the differences?

Josh Adams[22 days ago](https://edstem.org/us/courses/3783/discussion/243709?comment=591551)

Thank you George. I will try this because I have solved my speed issue, and I am fairly certain my issue now is the delta calculation. I was doing a 2 step look ahead to try and find the shortest path which quickly blew up in number of computations.

Reply

A

Alec Ryan Carruthers

[23 days ago](https://edstem.org/us/courses/3783/discussion/243709?answer=587776)

1

If I understand how you calculate your delta correctly, you may want to consider changing it. For example,

State = [[Z,Y,X,W,V,U,T,S,R,Q],[E]], Goal State: [[E,Y,X,W,V,U,T,S,R,Q],[Z]]

Would seem like a somewhat decent state because only two nodes are out of index and two mismatches for what should be on top of them. However, I would this state is very far where you want to be.

Comment

1

1. Make sure you implement the \_\_eq\_\_ and \_\_lt\_\_ method of your classes properly.  
2. You might need a better heuristic than just the row,col difference of a state vs goal row and col. If there is a mismatch, are there block on top of it that you might need to move so that you can move that particular block?

3. For visited, might be better to use a set() instead of []. Sets don't allow duplicates and have O(1) search time. But you might need to implement \_\_hash\_\_ for your custom class.

Josh Adams[23 days ago](https://edstem.org/us/courses/3783/discussion/243709?comment=587015)

I will look into the equals and less than suggestions.

The delta is calculated on more than that. It checks if the block is in the correct index as well as its neighbors. So "C" may be at the correct index of (0,0) but has incorrect neighbors, such as whats under/below or beside and it would have a higher delta than "C" that is in the correct neighborhood.

As for visited, I am using a set.

1Reply

Azmain Amin[23 days ago](https://edstem.org/us/courses/3783/discussion/243709?comment=587023)

I don't think it matters what's beside it. So, [['A', 'B'], ['C]] is the same as [['C], ['A', 'B']]. What matters is what blocks are on top or below it.

Reply

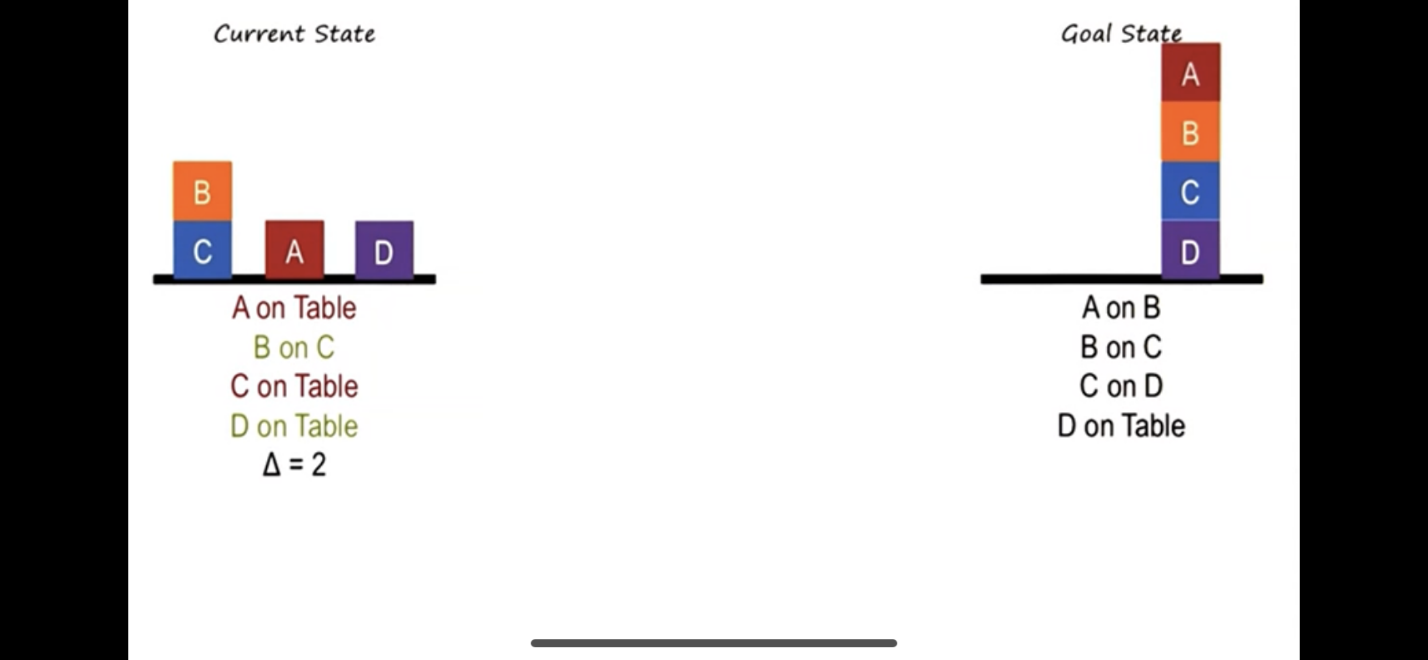
[23 days ago](https://edstem.org/us/courses/3783/discussion/243709?answer=589039)

Josh, the biggest problem I see here is your delta calculation. I think you're going to want a slightly more advanced heuristic. You already know whether a block is moveable... so in your examples above C is in the right location, but you'd actually have to make a number of moves for the stack C is in to be correct (including moving C). Maybe think about how you could use your is\_moveable awareness to make a more informed decision about the quality of a state?

28

In case you're running into issues, here's a hint that might help

Keep track of which blocks are "settled" in a state. For example:



The only block settled in the "current state" is D.

Why?

Because D is on top of it's target ("Table" in this case), and all blocks under D are settled (there are no blocks under D, so settled by default)

You may be asking yourself, "Isn't B settled too?" The answer is no. Even though B is on top of it's target block, that target block (C) is not settled.

So settled is this:

Am I on my target block? Is that target block also settled?

This helps in a few ways, you don't ever want to move settled blocks to make new states, and you never want to move a block on top of a settled bock UNLESS the newly moved block ALSO becomes settled.

## How to use Delta to throw away invalid states?[#641](https://edstem.org/us/courses/3783/discussion/245122)

WATCH

335511

VIEWS

I calculate the delta in my project and then I do the following:

1.) I make sure the next state I generate has the delta less than the previous state - Case 1& 2 passes but other fails

2.) Now I remove the delta condition completely and pass through every case (with only one condition in place which is not to repeat the states) - Case 1-4 pass and for the rest, there is an endless loop.

Now, I am using BFS and I understand the states are getting exploded due to the delta not being set in step 2.

The confusion is that Delta can be less than the previous state, it can be more than the previous state and it can even be equal so how am I suppose to use delta as the means of comparison between the current state and previous state if the delta can be anything.

Has anyone any comments on how we are defining delta to avoid states that are not needed?

Comment

1 Answer

K

Kevin Y Lin

[22 days ago](https://edstem.org/us/courses/3783/discussion/245122?answer=589969)

1

You might not need to throw away states completely - I put states and their deltas in a priority queue (as opposed to a regular queue in BFS), so that the agent will "prefer" the state that has the minimal delta among all generated-but-unvisited states. This allowed my agent to get to the goal faster without exhausting the entire search space. The key however was testing and refining a delta that suited the problem - the example given from the lectures was a good starting point but wasn't good enough for my agent to solve the problem efficiently.

## Calculating Delta[#650](https://edstem.org/us/courses/3783/discussion/246320)

I am not sure what I am doing on this because I have tried so many approaches for calculating Delta and they are not giving me all optimal solutions.

Approach 1 ( did not work because stack location does not matter ):

total\_delta = Euclidean\_distance of point to point + Count of Incorrect index for blokc

Approach 2 (Non Moveable being determined by correct placement based on goal):

Such as "A" being a table block, located on the table in both goal and current state

total\_delta = total number of block - number of non moveable blocks

Approach 3:

total\_delta = total\_differences \* current number of moves to arrive at that state

Approach 4:

total\_delta = total\_differences - current number of non moveable blocks

Approach 5:

total\_delta = total\_differences

Where total differences were calculated as

if current block is a table block in goal and not currently on the table:

total\_differences += 1

if current block should have a block above it and does not:

total\_differences += 1

if current block has something above it and it is the wrong block:

total\_differences += 1

if current block should have a block below it and does not (such as a block on the table that should not be):

total\_differences += 1

if current block has a block below it but is the wrong block:

total\_differences += 1

I have tried many approaches which weight different values, such as not being on the correct row and weighing the differences between goal row and current row.

I am able to solve the gradescope 31/40 because I am not able to get the optimal solutions.

I have read many posts about the various ways to calculate but I do not seem to be able to replicate.

Example being

If my Initial state is

*CBA*​*FED*​*IHG*​

and my goal state is

*IHGFEDCBA*​

What should be the delta of the initial state?

A is on the table and B correctly located above +0

B is correctly located on A and below C +0

C is correctly located on B but incorrectly below D, +1

D is incorrectly located on C but correctly under E, +1

E is correctly located on D and under F +0

F is correctly located on E but incorrectly under G, +1

G is incorrectly located on F but correctly under H, +1

H is correctly located between I and G, +0

I is correctly located on H and nothing is onto of it. +0

Total\_delta = 4

Comment

4 Answers

1

I think Prashanth and I agree on one of the points:

I don't know if you need to use Delta as a metric to solve this problem on its own. It may provide some help, but I don't think it will get you all the way to the solution. What worked for me and some of other students is using a search and a priority queue as the foundation.

A couple of things that proved helpful for me were to make sure I didn't add onto a stack that wasn't a part of the goal stack. For example, one of the first possible moves is moving I onto F. However, that is not a part of the A,B,C,..., H,I goal ordering. So, you would want a way to make your agent ignore this as a possible move and prefer other, more useful moves.

As you check the various possible moves, maybe use the delta as a starting point and then try to use some heuristics, which depend on your own architecture, in order to modify the priorities of each potential move and dequeue the move that looks like the best option.

Comment

Josh Adams[22 days ago](https://edstem.org/us/courses/3783/discussion/246320?comment=593381)

Thank you for the response Adam, I currently am using a priority queue and a one step look ahead to get my results. I just have not been able to solve things optimally. I will see if I can make any progress.

Reply

Adam Walls[22 days ago](https://edstem.org/us/courses/3783/discussion/246320?comment=593536)

Good luck man! For what it's worth, I think you're on the right rack. I had to grind out what wasn't working using my debugger and manipulate my agent to do what I wanted it to. It was more challenging than I expected it to be.

Reply

J

Jeremy Christopher Lay

[22 days ago](https://edstem.org/us/courses/3783/discussion/246320?answer=593449)

1

I found it easier just to think of "distance" to the goal, which in this case is number of moves you have to make to get to goal state. Think about the easiest non-optimal case to program: move all blocks to the table, then stack them up correctly. Consider the number of moves required to do that as a baseline. But, you don't need to move blocks that are already stacked correctly (as mentioned see [#634](https://edstem.org/us/courses/3783/discussion/threads/634) on that), so you can subtract any moves you would have made for blocks already correctly stacked in terms of the goal state. This would be only A, B, and C here. Does it really matter that E is correctly positioned between D and F? D is on the table and there aren't any stacks with D in the table in the goal state - so at a minimum you're going to have to move E somewhere to then move D in the right place and then move E back on top of D - 3 moves. Just say the entire stack is wrong and count up the worst case moves.

Comment

Add comment

Sreedevi Viswambaran

[21 days ago](https://edstem.org/us/courses/3783/discussion/246320?answer=595551)

Here in this context, for me, the Delta is anything that is measurable and would get you closer to the goal state. Initially, I tried the points approach and ended up with a very confusing code. My algorithm base is also a DFS - priority queue (a\*) algorithm. Delta for my approach is the number of "settled" elements, which move will give me more number of "settled" elements will be the one which I pick for processing.

Comment

Add comment

E

Eugene Kwak[21 days ago](https://edstem.org/us/courses/3783/discussion/246320?comment=596193)

I was also thinking of doing this. Glad to hear I am not the only one going the utility function rather than cost function route! Fingers crossed it works! :)

1Reply

[22 days ago](https://edstem.org/us/courses/3783/discussion/246320?answer=593256)

Couple of things -

1. I would advise keeping the "delta" and the check on the stacking of the final block as separate checks. [#634](https://edstem.org/us/courses/3783/discussion/threads/634) for more!
2. In your example above, you've double-counted the delta. The total delta should be 2 (contribution of +1 each from D and G only). You don't need to count the delta bi-directionally. Just the 'on' relation is enough. 'Under' or 'below' isn't necessary.

Comment

Add comment

Thank you for your response Prashanth, I will look at that post as well as modify my delta calculation to not double count.

As for your first point, I am not sure what you mean by keeping the delta and the check on the stacking of final block as separate checks.

## Confused on how to represent problem state[#651](https://edstem.org/us/courses/3783/discussion/246569)

[22 days ago](https://edstem.org/us/courses/3783/discussion/246569) in **Mini-Projects – Mini-Project 2**

As far as representing the state of the problem, I'm having some trouble figuring out how to represent the blocks and their position in a way that makes it easier to generate new states based on a delta. Keeping the state as just the exact config that the initial arrangement comes in doesn't seem to make much sense to me, so does anyone have any suggestions as to data structures of custom classes that they used to store their state?

Comment

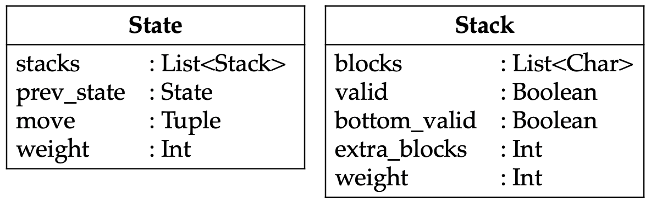
4 Answers

Thu Anh Hoang

[21 days ago](https://edstem.org/us/courses/3783/discussion/246569?answer=597621)

6

I ended up creating two classes and that made solving the problem easier for me. Hopes this helps someone.



Comment

Add comment

A

Andrew Aignarong Vathanakamsang[18 days ago](https://edstem.org/us/courses/3783/discussion/246569?comment=608795)

Thanks for sharing! Did you have to equate or compare different States (pruning states that have occurred in the past)?   
  
I was planning on adding all past States in a set and only generating new States if they are new. The problem I am having is that because the stacks are of type list it becomes unhashable TypeError: unhashable type: 'list' for comparison sake.

Reply

Thu Anh Hoang[18 days ago](https://edstem.org/us/courses/3783/discussion/246569?comment=609590)

Yep, I kept a list of all past States in a set too. I had to overrode the \_\_hash\_\_ and \_\_eq\_\_ functions for my needs.

1Reply

This is interesting. Can you explain your Stack class a little more: I get the list of characters for the blocks but not really understanding what you are representing with each of those other fields like valid, valid bottom, etc.

So the valid and bottom\_valid booleans come into play with my generator.

A stack is valid if it completely matches a stack in the goal state. The generator should not touch it then since it's already correct.

A stack is bottom\_valid if it is partially correct from the bottom up. The generator can take blocks off or put on blocks to try and reach the goal. The generator should only pop blocks and not add blocks on stacks where bottom\_valid is false.

The extra\_blocks keeps tracks of how many unwanted blocks are on top of a bottom\_valid stack.

I use the combination of valid, bottom\_valid, and extra\_blocks to assign a weight to a stack. The weights of the stacks is used to pick which state to pursue.

Reply

[21 days ago](https://edstem.org/us/courses/3783/discussion/246569?answer=598783)

I have a state class with two instance variables :

1. A dictionary of state. eg. {A:B, C:"TABLE}

2. A list of the blocks that are on top of stacks

You can put action and cost into the same class too.

Python dictionaries are easy to use. How to compare two python dictionaries:

<https://www.geeksforgeeks.org/python-intersect-two-dictionaries-through-keys/>

<https://www.geeksforgeeks.org/python-difference-in-keys-of-two-dictionaries/>

Or you can use set to store your state like {(A,B), (B,TABLE)}.

Comment

Sreedevi Viswambaran

[21 days ago](https://edstem.org/us/courses/3783/discussion/246569?answer=595513)

I kept the given format for the state. Initially, I tried out the approach of creating a custom class. Eventually, I realized it may not be necessary. I had trouble with deep copy and equating. So I took the input as given and process it as and when necessary. It worked for me.

Comment

Add comment

E

Eugene Kwak[21 days ago](https://edstem.org/us/courses/3783/discussion/246569?comment=598875)

If you don't mind my asking, what was the trouble you had with deep copy and equating? I am going down the deep copy path at the moment.

Reply

Alejandro (Alex) Diaz[20 days ago](https://edstem.org/us/courses/3783/discussion/246569?comment=603138)

deepcopy is really inefficient and will slow down the whole process by minutes

Reply

E

Eugene Kwak[20 days ago](https://edstem.org/us/courses/3783/discussion/246569?comment=603352)

[Replying to Alejandro (Alex) Diaz](https://edstem.org/us/courses/3783/discussion/246569?comment=603138)

I see! thank you! My other thought was to just use tuples and rely on their immutability. I guess I'll play around and learn as I go! I am as of now, relying on strong regularizing heuristics in my state generator to try and vastly limit the growth of the tree. so hopefully i can squeak by with the deepcopy, but good to know others got through this without creating classes!

I'm relatively new to all this so I really appreciate this sort of feedback!

Reply

Alejandro (Alex) Diaz[20 days ago](https://edstem.org/us/courses/3783/discussion/246569?comment=603363)

[Replying to Eugene Kwak](https://edstem.org/us/courses/3783/discussion/246569?comment=603352)

I did something similar but with lists and found that my agent halted around 25 steps in. Let me know if you have better luck with tuples, I am interested in finding out!

Reply

E

Eugene Kwak

[22 days ago](https://edstem.org/us/courses/3783/discussion/246569?answer=593571)

I followed what was presented in the lectures, mostly.

A on B

C on Table

...

I'm approaching this by instantiating block classes and providing attributes to more or less mirror that. So, object A is block A, it is on top of B, etc. I'm just storing all the classes in a list and hacking my way through the logic. Hopefully this helps. It may or may not work for me as I am still in the middle of this assignment.

Comment

Taylor James Windsor[22 days ago](https://edstem.org/us/courses/3783/discussion/246569?comment=594069)

Thanks I will try the class based approach, how are you handling comparing the overall state with the goal state? Creating two lists of classes and comparing or does each class keep track of if it is in its goal position?

Reply

D

Daniel Patrick Ofarrell[21 days ago](https://edstem.org/us/courses/3783/discussion/246569?comment=596776)

At one point I was stuck because I kept trying to approach this problem in 2 dimensions. I kept getting stuck in infinite loops because my "goal state" and the real "goal state" didn't look the same. The problem laid in me assuming the block towers would also be left->right. like [Figure A]. But there's no reason you couldn't just walk to the other side of the table, where now the blocks are like [Figure B]

Figure A:

[C]   
[B] [E]  
\_\_[A] [D]\_\_

Figure B:

[C]  
[E] [B]  
\_\_[D] [A]\_\_

Reply

E

Eugene Kwak[22 days ago](https://edstem.org/us/courses/3783/discussion/246569?comment=594535)

Yes - I am representing states in an ordered list, ordering the block classes alphabetically. Basically i take advantage of the fact that the goal and initial states will have the exact same blocks, just in different positions, I can simply check the two ordered lists element by element to see which blocks differ. I also plan to factor in which blocks have "settled" ([#634](https://edstem.org/us/courses/3783/discussion/threads/634)) so i prevent those from any manipulation down stream.

Reply

A

Alec Ryan Carruthers[22 days ago](https://edstem.org/us/courses/3783/discussion/246569?comment=594817)

[Replying to Eugene Kwak](https://edstem.org/us/courses/3783/discussion/246569?comment=594535)

If your heuristic is working as it should, I don't see why that would be necessary because a move of "settled" blocks would inevitably lead to a worse state.

2Reply

E

Eugene Kwak[22 days ago](https://edstem.org/us/courses/3783/discussion/246569?comment=594874)

[Replying to Alec Ryan Carruthers](https://edstem.org/us/courses/3783/discussion/246569?comment=594817)

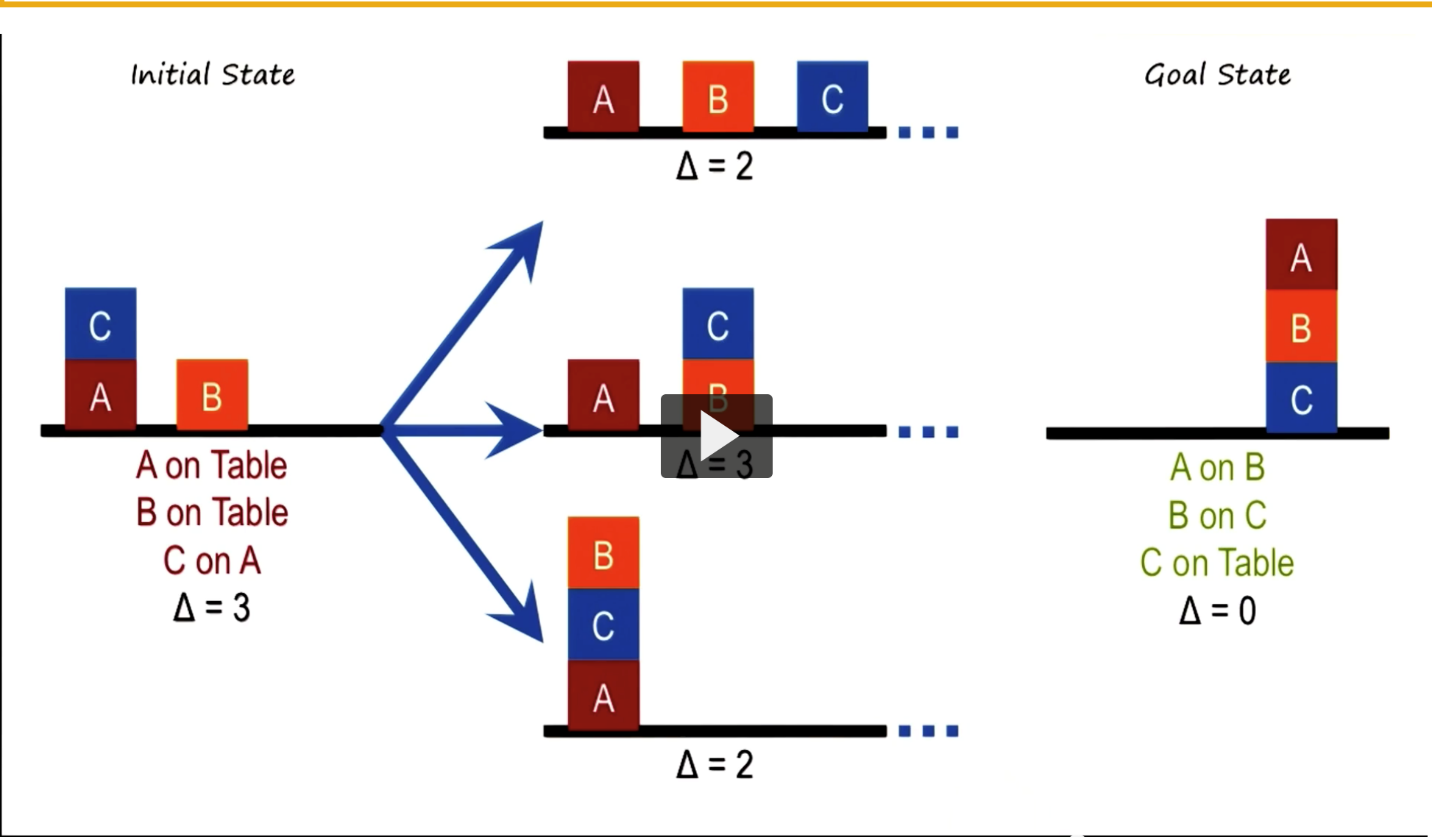
Ah ok, Thanks! That's a cleaner way to phrase it. Thanks again for the hint, btw! Thoughts and words are not aligning after a few days of staring at the GDPR law 😵

## Explain Mean Ends Analysis[#655](https://edstem.org/us/courses/3783/discussion/246923)

T

2

Hey Y'all I have been struggling to understand how the Delta between the bottom state and the goal state is 2. Can someone explain



Comment

1 Answer

P

2

On the top C is on the table in it and the goal state (A and B are different from goal - 2).  
In the middle none of the blocks are in the goal state position.  
On the bottom B is on C in both it and the goal state (A and C are different from goal - 2).  
  
The bottom isn't productive towards the goal because if C is not in its goal state on the table then B on top of C means it is not in its final state and does not help towards the goal so there is no reason to accept it. The only valid move after that in this example is to move it back to where it came from.

## Print function for block stacks[#711](https://edstem.org/us/courses/3783/discussion/250794)

13

In case it's helpful to others in debugging their Mini-Project 2, here's a function to pretty print a list of lists as stacks of blocks, per the project spec.

TEXT

1

2

3

def print\_stacks(stacks):

    max\_len = max(len(s) for s in stacks)

    for i in reversed(range(max\_len)):

        for stack in stacks:

            if i < len(stack):

                print(stack[i], end = ' ')

            else:

                print(' ', end = ' ')

        print()

    for i in range(len(stacks)):

        print(u'\u203E', end = ' ')

    print()





Ex.: print\_stacks([['A', 'B'], ['C', 'D', 'E'], ['F'], ['G', 'H'], ['I', 'J', 'K'], ['L']]) results in:

E K

B D H J

A C F G I L

‾ ‾ ‾ ‾ ‾ ‾

I totally get you, I also underestimated the assignment at first and struggled with measuring how far I was from the goal state for a long time. At some point I realized I was going in circles trying to somehow score and compare hypothetical states that were one move away from the current state (which had worked fine for wolves and sheep).

So I reformulated my approach to extracting the next move directly from the goal state, thinking like this:

1) Which blocks on the current state are already on the same place they will be on the goal state? (Only considering a block is in place when no block under it is misplaced).

2) What move (or moves if there's more than one final stack) would put another block on its final destination? (No matter if that move is legal in the current state).

3) If there's only one final stack, execute that move. If it is not legal, remove one by one all the blocks that are an obstacle for that move and then execute it. If there's more than one final stack, pick the one move that requires getting out of the way less blocks with intermediate moves.

I had no way to be sure that this (rather greedy) approach would be optimal, but it was. And it's quite efficient since it only creates one path.

chose to create a "block key" using a dictionary/hash that could be used for O(1) lookups of what blocks were below another block. For example, if the goal system looked like:

[['A', 'B', 'C'], ['D', 'E']]

... then the dictionary for that would look like:

{'C': 'B', 'B': 'A', 'A': 'Table', 'E': 'D', 'D': Table'}

I could then use that to quickly check what block is below "B", etc. Part of my heuristic involved knowing which blocks were "in their final place", and having this kind of key was useful for that check!