ColumbiaX: CSMM.101x Artificial Intelligence (AI) Help sandipan_dey

Home Course Discussion Progress

Support

All Topics Add a Post Search all posts Search

filter topics

All Discussions

★ Posts I'm Following

General

Guidelines and FAQs

Week 1

Lecture

Topic-Level Student-Visible Label

DQ1

DQ2

Quiz

Week 2

Project

Lecture

DQ1

DQ2

Quiz

Week 3

Lecture

[A-Star Search] Notes on Implementation

discussion posted about a month ago by **dkc2122** (Staff)

Pinned

Hi all,

There is a lot of discussion over the implementation details of A-Star Search, which is great. Here are a few clarifications regarding the nstructions on maintaining "**UDLR**" ordering:

Support

s per the instructions, UDLR ordering must be followed **where an rbitrary choice must be made**. For DFS, this arbitrary choice is when child nodes of the same parent are being popped. For BFS, this arbitrary choice is when child nodes of the same layer are being dequeued. For A-Star, this arbitrary choice is... when?

A priority queue works by maintaining the entire queue in the order of some **key**. An arbitrary choice, then, must be made when you want to dequeue something, but there are multiple objects in the queue with the same key. In this case, as per the instructions, we want to pick the "Up" choice first, then "Down", etc.

Now you might ask, what if there are multiple "Up" choices with the exact same key? We don't want to give away too much, so we leave this case for you to think about. First off, does this happen with the 8-puzzle? Why or why not? Does this depend on your choice of heuristic? Of course, if don't want to think too hard about it, you can simply enforce the classical idea of a priority "**queue**" as nothing more than a "smarter" version of the original queue data structure. To this end, you may consider adding a **time** value as part of the tuple that constitutes the key when you insert into the priority queue.

As a general issue, however, if there are choices you have to make in your programming projects that **not specified** in the instructions (check carefully!), but you sincerely believe that they will make a material difference in your output, you can bet that our test cases will happily accommodate correct answers :)

Related to: Week 2 / Project
This post is visible to everyone.

Add a Response

7 responses

pisymbol
about a month ago

Support

Wait a minute...
The lecture slides (video) clearly shows an algorithm that DOES NOT MAINTAIN duplicates but rather updates the existing state's cost and then heapify's (decreaseKey).
Are you saying we should maintain duplicates in our project's assignments? This was asked in the context of A* search.

No, here we are referring to duplicate **keys**. In this case, that refers to two distinct **states** (i.e. objects that you store in the priority queue) that turn out to have the same resulting **cost** (i.e. the keys that the objects are indexed with). Here the **state** is the object (a.k.a. "value"), the **cost** is the key. We certainly do not want to visit the same node twice!

posted about a month ago by dkc2122 (Staff)

@pisymbol: Be careful to distinguish between state, node and priority. State is the representation of the problem instance. Node refers to the node of the search tree. Priority here refers to the f(n) value of a node in the general A* search algorithm. As a priority queue is used, each node must have a priority associated with it.

The algorithm does not maintain nodes with duplicate states in the fringe (priority queue). However, different nodes may have the same priorities. This is what this thread is talking about.

In the lecture, the function name decreaseKey may be misleading. In the reference book, it said "replace that frontier node with the neighbor". Also, the lecture seems to mix up state and node some times. This may also confuse you or others. My advice is to read the reference book to clear any misunderstanding and/or confusion you may have.

posted about a month ago by fitcat

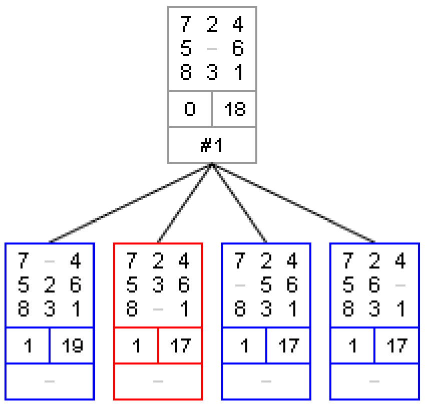
I used state as node.

posted about a month ago by antonio84a

Add a comment

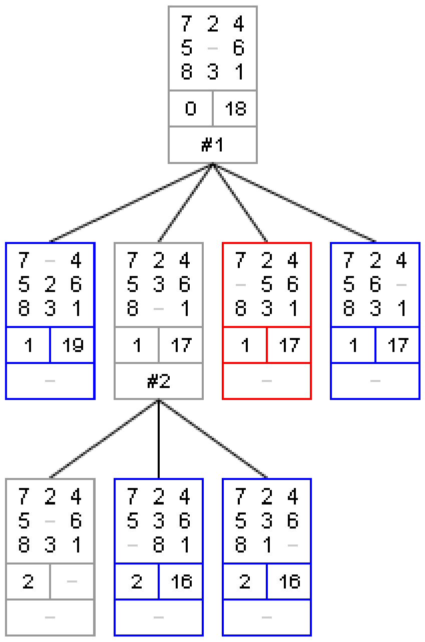
NataliaRequejo
about a month ago

Ok, I have a lot of questions, suppose we have the following board We expand the first node and we obtain 4 neighbors f() = g()+h(), if we follow in UDLR mode, we expand the node in red.



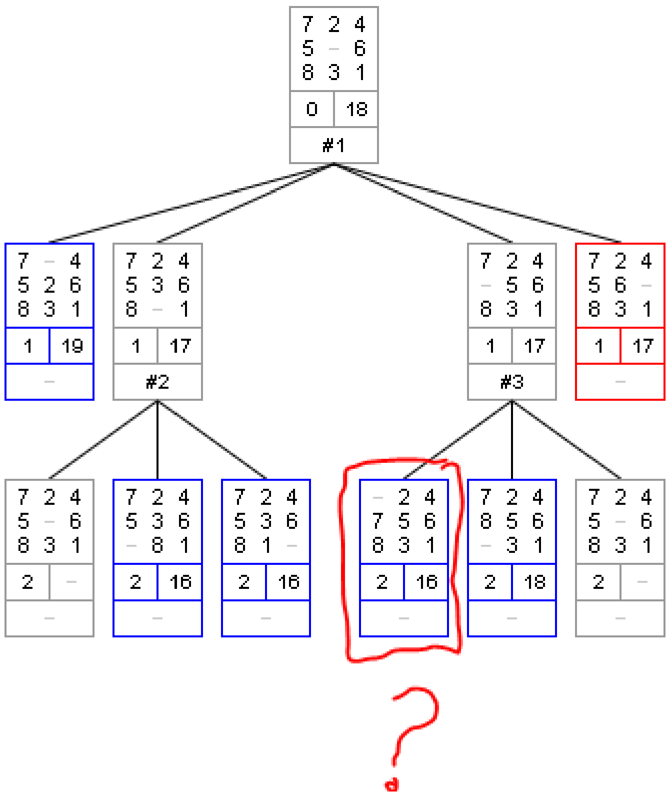
Support

Now we have 4 boards with f()=18,



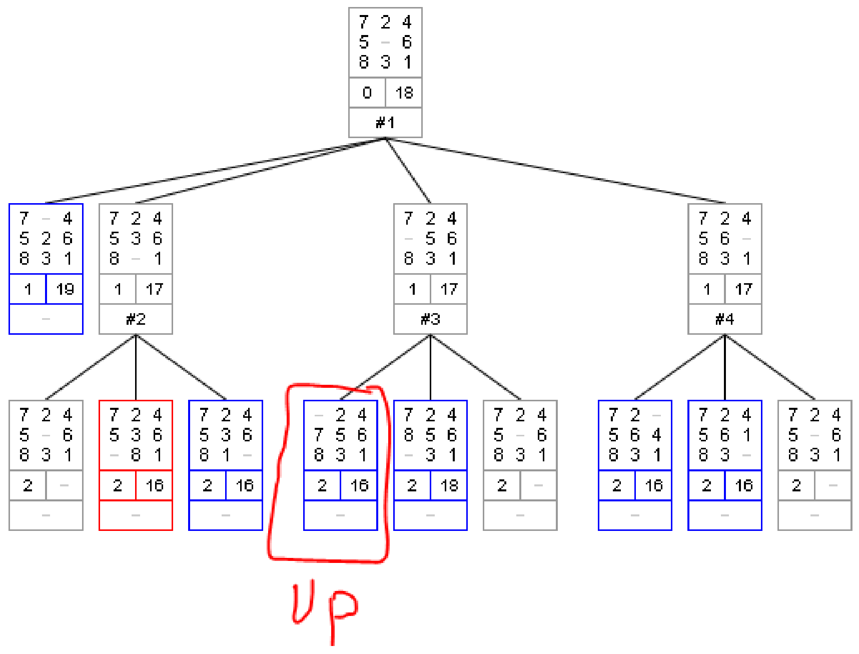
two of them are in deep 1 and the moves are LEFT and RIGHT, and two in deep 2 and the moves are LEFT and RIGHT too, so far so good, because both moves are LEFT, we pick the small depth.

The next board it will be



Now we have 5 boards with f()=18, one of them is deep 1 and RIGHT, and the other ones are in deep 2, one is UP (marked with ?) one is LEFT, and one is RIGHT, wich of these wins? We store these boards in order so in the frontier we have, first the LEFT, then RIGHT and finally UP. which is the next board? its the board with small depth or the one with UP action?

Suppose we expand the red one because we prioritize the depth, we obtain



Now again we have 5 boards with the same depth and the same heuristic. again wich order we must follow? we must open the red one, or we must open the board marked as UP (following the UDLR)
The readings do not explain in depth which is the correct way. The solutions are very different if we choose different paths.
In the section "IV. What Your Program Outputs" of the project page said "...All the other variables, however, will have **one and only one correct answer** for each algorithm and initial board specified. ... " except for IDA*
So, what is the correct way to implement A* algorithm? Thanks in advance.

Good question.

You say: "We store these boards in order so in the frontier we have, first the LEFT, then RIGHT and finally UP." Well, not really. The boards are stored in a heap, so there is no assurance of any particular order. It depends upon the heap implementation, which may vary between different library classes, different versions or revisions of Python, or individual implementations if one chooses not to use a library class.

In your last iteration, my implementation chooses the node you have marked as 'UP', but only because the tie is broken by the board's first tile, which is zero!

posted about a month ago by **AWCook**

Support

So, you have nodes in your priority queue:

(1) The main metric (i.e. the "key") to determine the priority is the **cost**.

(2) But suppose two nodes have the same cost. Then you defer to which one is of higher priority in the "**UDLR**" ordering.

(3) Now suppose you ask, what if two nodes have the same cost, and also the same direction (e.g. they are both "Up"). Then we suggest deferring to whichever one came into the queue first (but we would have liked you to think about it; how it happens, if it happens, and whether or not it makes a difference; if so, when, etc).

Does that make sense? By "cost" we are referring to f(n) = g(n) + h(n). You do not need to consider g(n) and h(n) separately.

posted about a month ago by **dkc2122** (Staff)

@dkc2122 First of all, thanks for all the help.
I am having difficulty with the order of 2nd and 3rd keys. Shouldn't we favor the 3rd key since it is likely to promote shorter path? Also, wouldn't the use of 3rd key be analogous to using g(n) as the key? What I mean is does it make sense to do tie-breaking based on UDLR order when two nodes in question are at different depth?
Edit: After further deliberation, I would go as far as to say that tie-breaking based on UDLR makes little sense unless the competing nodes belong to the same parent.
Please, answer because the whole project is on hold until this is cleared up. :\

posted about a month ago by **JoonhoPark**

Ah, I think I understand what you are asking. So, the main point first: the **essence** of A* Search is that the **cost function**, being f(n), is used in a **priority queue**. That is it. Beyond that, the characteristics of the cost function are all that matters for A* to produce the optimal solution (e.g. It must be admissible for a tree; more generally it must be consistent, etc. but that is not what we're focusing on at the moment).

In particular, for A* to be optimal, there is no general requirement to go beyond point (1) above. This is analogous to BFS, where, **per layer**, it generally really does not matter which what order you expand the nodes in. However, *for grading purposes* in this project we ask that you always expand in UDLR-ordering; this is an arbitrary choice made with the intention of producing one and only one correct output. Similarly for A*.

If you swap (2) and (3), you still have an A* algorithm, and it will find the optimal solution. Remember, (1) affects optimality. Everything else that you choose to use as secondary priorities may only affect [if they do] efficiency (i.e. the number of nodes expanded in the process). However, we simply had to decide on a specific way to do it, so we require (2) to be the next priority.

Does that make sense?

P.S. As an aside, suppose you find two "Up" nodes, n1 and n2, with the same f(n) on your priority queue. In the general case there is really nothing much we can say about how the f(n) values are broken down into g(n) and h(n). The node that came in earlier, n1, may actually have a higher g(n) than the one that came in later. The reason is that this depends on the heuristic function as well. The parent of n1 may have had a high g(n), but a very low h(n), so we visited it first, and as one of its children n1 got placed into the frontier, etc.

posted about a month ago by dkc2122 (Staff)

@dkc2122 Thanks for the clarification. I will use (cost, direction, ID) as the keys for my heap. I had implemented one which uses (cost, parent, direction, ID) and somehow correctly performs resolution of parent comparison but will comply to the now explicit "specification" of the problem. :) It is always good to work out these things to the dots before embarking on a journey. One related question, I am ignoring updates to the cost when the costs are equal. Does that make sense to you as well?

posted about a month ago by JoonhoPark

Do you mean when you re-encounter a node (n2) that already has a copy (n1) in the frontier, and this time round the cost of n2 is still the same as n1? In that case, you are correct; we don't need to do anything. You can go ahead and "update" it, but that wouldn't change anything anyway :)

posted about a month ago by dkc2122 (Staff)

Yes, but wouldn't it change the depth at which the board appears? I am leaning towards the new one being at deeper depth if we made update with the newer node but either way, thanks for clearing that up. Afterthought though... I guess if it has the same cost and board configuration, it would have to have happened at the same depth as well. @.@ I am beginning to think if I am overthinking this...

posted about a month ago by JoonhoPark

Suppose you encounter a node once, and calculate f(n1) = g(n1) + h(n1). Further suppose you encounter that node again, and calculate f(n2) = g(n2) + h(n2). The situation now is that we suppose f(n1) = f(n2). Now, since they represent the exact same board state, **by definition** the Manhattan distance function must yield h(n1) = h(n2). But then it must be the case that g(n1) = g(n2). So in the end it really does not matter. You are not overthinking this---this is all great to think out carefully!

posted about a month ago by dkc2122 (Staff)

Do you mean cost in the first argument of the key (cost, direction, ID)? If yes, I think it depends on the direction of n2 and n1. If n2's direction precedes n1's, it has to be updated, right? If the cost means the entire key, then the cost (key) will not be equal. No update is required as n1's ID is always smaller than that of n2's. However, if update is done, it will affects the nodes expansion order in case there are some other nodes with the same cost and direction and are inserted into the priority queue earlier than n2 and latter than n1.

posted about a month ago by fitcat

@fitcat That is a valid point. I just ran A* algorithm on 29 move board for each case. First of all, what @fitcat describes does happen quite a bit and definitely produces different answer.

- Run without the replacement of node based on UDLR priority:

  %run driver.py ast 8,5,2,7,1,4,6,0,3

  path_to_goal: *['Up', 'Up', 'Left', 'Down', 'Down', 'Right', 'Right', 'Up', 'Left', 'Left', 'Down', 'Right', 'Up', 'Up', 'Left', 'Down', 'Down', 'Right', 'Up', 'Up', 'Right', 'Down', 'Down', 'Left', 'Up', 'Right', 'Up', 'Left', 'Left'*] cost_of_path: 29 **nodes_expanded: 17399 fringe_size: 7730 max_fringe_size: 7733** search_depth: 29 max_search_depth: 29 running_time: 1.95200014114 max_ram_usage: 0.0
- Run with the replacement:

  funny thing just happend. path_to_goal: [*'Up', 'Left', 'Down', 'Right', 'Right', 'Up', 'Left', 'Left', 'Down', 'Right', 'Up', 'Up', 'Left', 'Down', 'Down', 'Right', 'Right', 'Up', 'Left', 'Up', 'Right', 'Down', 'Left', 'Down', 'Right', 'Up', 'Up', 'Left', 'Left'*] cost_of_path: 29 **nodes_expanded: 17453 fringe_size: 7711 max_fringe_size: 7717** search_depth: 29 max_search_depth: 29 running_time: 3.50699996948 max_ram_usage: 0.0

So what do we do now???? @dkc2122

posted about a month ago by JoonhoPark

Ah, I see what you are worried about now. No, the test cases we have selected will **not** produce multiple outputs depending on something as subtle as this. (If we ever do, in any project assignment, you can be sure submissions will be marked correct for any unspecified parameters like this one). The important point here; the solution is still optimal (29 steps). Don't worry too much more about it.

posted about a month ago by dkc2122 (Staff)

@dkc2122, Ok they both produced different 29 step solutions along with different stats. Having said that let me be crystal clear here since I am close to shutting the door on this. :), ignore tie-breaking based on UDLR order(key[1]) when scores(score here means cost/key[0] of (cost, direction, ID)) are equal. Only update/decreaseKey when new one has lower cost(key[0]). Is this correct?

posted about a month ago by JoonhoPark

Yes, that would be the "canonical" way to do it. The reason being, the use of a tuple (which includes things **other than** the path cost) as the priority key is entirely an artifact of a specific implementation. In this case, it is an artifact of your method of achieving UDLR ordering. However, if you already did it the other way, ceteris paribus it is no less "correct" of an A* algorithm!

posted about a month ago by dkc2122 (Staff)

@dkc2122, I am not sure if I understand you. Perhaps my use of heapq is different from your implementation. Perhaps, using Queue.PriorityQueue? Anyways, only update/decreaseKey when f(n) = g(n) + h(n) is lower for the new node, regardless of other nuances of implementation details? Is this correct?

posted about a month ago by JoonhoPark

Correct.

posted about a month ago by dkc2122 (Staff)

@dkc2122, Much appreciated! Now I can finally move on. :)

posted about a month ago by JoonhoPark

For me it seems that it happens a LOT (always) with huge lists of nodes with the same score. But if there is a sraight way to solution, the cost will remain equal, though each step will decrease h(n). It's too bad not to sort by depth after sorting by cost !

posted about a month ago by ELamy

@dkc21211: This test case is not subtle. It is valid and necessary in my opinion. The spec in "2. Orders of Visits" clearly states what the order of visits must be UDLR when there is a choice. Moreover, as the spec in "5. Grading and Stress Tests" said:

> We will be using a wide variety of inputs to stress-test your algorithms to check for correctness of implementation. So, we recommend that you test your own code extensively.

We are asked to ready for such test case and may be others that are tricky/subtle. So, we are doing tests extensively. Now, you tell us that there are no such test cases. Did we waste our time?

posted about a month ago by fitcat

@Elamy, I am not sure if I understand your last statement. Mind expounding on it a bit? :)
@fitcat, I share your sentiment. :'(

posted about a month ago by JoonhoPark

I'm not sure what you mean by this test case is not subtle. We refer above to the subtlety of the *implementation detail*, not the test case itself. (It's unclear what "subtlety" could refer to with respect to test cases).

Again, to be clear, A* works off of a priority queue. The priority queue is keyed by cost. The only **definitive** requirement of A*, therefore, is the cost being the priority key. (The UDLR ordering is a requirement only for this assignment.) As such, the "canonical" way to do decreaseKey(), if you have to do it, is with respect to the cost being the priority key. Now, the use of a tuple as priority key instead, is an **artifact** of a specific choice of implementation to handle UDLR ordering. It certainly works, but it seems that it has muddled the definition of what the "key" in decreaseKey() should be.

To repeat, the key here is the cost. If you choose to implement decreaseKey() by regarding the key as some tuple as well (hence updating a frontier node even when the cost is the same), then you may find that you have different (but still optimal) results for certain boards. All we're saying is, that is **completely fine**; you will not be docked points for that. We have decided that it is a subtle implementation detail that has little in the way of understanding what A* itself does, and will accommodate for variations to this effect.

Does that make sense?

posted about a month ago by **dkc2122** (Staff)

@dkc2122: I am afraid it does not. In the "Notes of Correctness" under Part IV of the spec:

> All the other variables, however, will have **one and only one** correct answer for each algorithm and initial board specified.

This thread is talking about what should we do when same f(n) is found. This is not a subtle implementation issue. The spec explicitly and clearly stated the requirement. We have to stick to it rigidly; otherwise, the implementation is considered as incorrect. Now, are you trying to say the spec is misleading or incorrect?

posted about a month ago by **fitcat**

[Staff] @dkc2122,

could you please clarify as True or False:

"All the other variables, however, will have one and only one correct answer for each algorithm and initial board specified."

posted about a month ago by **JohnA88**

Support

Ok, Thanks to all for the answers and specially to @dkc2122 for the patience.

About @dkc2122 POST

> BUT WE WOULD HAVE LIKED YOU TO THINK ABOUT IT; HOW IT HAPPENS, IF IT HAPPENS, AND WHETHER OR NOT IT MAKES A DIFFERENCE; IF SO, WHEN, ETC

I had really thought of it, English is not my main languaje, and perhaps I could not explain myself in the best way, but precisely because I had been thinking on that, I asked the question. The difference in the implementation was not going to vary the path_to_goal but it varied a lot in the statistics. This, I think, means that the algorithm was more or less efficient depending on the priority list implemented. But the project specifications specified that there should be only 1 correct answer. Because of that i asked for clarifications.

Here is my current result with a priority (cost function f(), UDLR, incremental UniqId)

```
path_to_goal: ['Left', 'Up', 'Right', 'Down', 'Down', 'Left', 'Up', 'Right',
cost_of_path: 26
nodes_expanded: 3264
fringe_size: 1753
max_fringe_size: 1754
search_depth: 26
max_search_depth: 26
running_time: 0.5699129104614258
max_ram_usage: 0
```

Obviously the fringe_size number is without duplicate boards. Like the max_fringe_size. I hope it will be ok, now i will dedicate some time to IDA*

posted about a month ago by **NataliaRequejo**

@JohnA88 @fitcat

(1) We **specified** the following input boards:

python driver.py ast 3,1,2,0,4,5,6,7,8

python driver.py ast 1,2,5,3,4,0,6,7,8

Do you get different results by keying your priority queue on f(n), versus some tuple you came up with? If you interpreted "specified" as "*anything you choose to come up with*", that does not make sense, since you can come up an *unsolvable* initial board and then what would a "correct" answer be?

(2) From the **announcement** in this thread:

As a general issue, however, if there are choices you have to make in your programming projects that *not specified* in the instructions (check carefully!), but you sincerely believe that they will make a material difference in your output, you can bet that our test cases will happily accommodate correct answers :)

posted about a month ago by **dkc2122** (Staff)

Support

@dkc2122... Umm,,, I thought we agreed on three keys (f(n), UDLR, ID, item) or at least the fact that **when there is a tie on f(n) we do tie-breaking based on UDLR and then unique incremental ID**, *I remind you that this decision isn't just for update/decreaseKey operation, it is at the heart of heap operation itself.* @.@; What exactly are you saying in response (1)? **Just key it on f(n)** and let the chips fall where they may??? "If you interpreted 'specified' as anything you choose to come up with,' that doesn't make sense...," I thought we spent hours yesterday so that we have a clear specification... T.T; Or **do you mean the test cases**? If so, you mean to tell us to ignore the bit about fringe/stress testing our implementation? Lastly, I would like a confirmation on having to **maintain the heap in-variance**, however the implementation may be. It is a costly process O(n) for me anyways but is necessary according to documentation on python heapq implementation if I were to update/decreaseKey on my fringe instead of following their implementation suggestions completely due to concerns regarding plagiarism...

posted about a month ago by JoonhoPark

@dkc2122 :

> (1) We specified the following input boards: python driver.py ast 3,1,2,0,4,5,6,7,8 python driver.py ast 1,2,5,3,4,0,6,7,8 Do you get different results by keying your priority queue on f(n), versus some tuple you came up with?

Hard to really get much difference using the same basic algorithm when one of the input boards is solvable by exactly one move and the other one, which is solved by 3 moves, will have a clear difference in manhattan distance without any ties, so the other ways of comparing, including UDLR will never come into play. Your question doesn't make much sense because we could pick as second priority RLDU, for example, and still would get the same result.

The specs clearly recommend that we thoroughly test our code with different inputs but i'm not sure how we are supposed to test without any reference to correctness. Also, other than the two extremely trivial cases posted as examples, most of the A* tests i've seen here, in the discussion forums, produce wildly different results, even with the same optimal path, as far as fringe size and expansion goes. The differences are due to ways in which other things may or may not have been implemented, like following your recommendation for breaking cost function + UDLR ties, which will produce major differences in any board that may have any ties in Manhattan distance.

> As a general issue, however, if there are choices you have to make in your programming projects that not specified in the instructions (check carefully!), but you sincerely believe that they will make a material difference in your output, you can bet that our test cases will happily accommodate correct answers

This is NOT what the specs say, which is that there's **one and only one** correct solution. Please, if you insist that there's more than one possible correct solution, either change the course specifications or, please start a pinned thread in which that is clearly stated for everyone to see, instead of being buried here among a ton of replies.

Thank you

posted about a month ago by IreneNaya

@dkc2122, thanks for your patience and clarification, appreciate if you could confirm my understanding:

---

a. for the given specified input boards (3,1,2,0,4,5,6,7,8) and (1,2,5,3,4,0,6,7,8), there is one and only one correct solution; no matter the choice of 3rd and 4th keys in the priority queue. (1st key is specified as f(n); 2nd key is specified as UDLR)

---

b. for grading, the specified input boards (five of) may have one or more correct solutions; no matter the choice of 3rd and 4th keys in the priority queue. and these grading "test cases will happily accommodate correct answers".

---

c.for other boards (that we will be using during our extensive testing), there may be more than one correct solution, eg varying fringe_nodes, depending on the choice of 3rd and 4th keys in the priority queue. Or even no solution.

posted about a month ago by **JohnA88**

Support

@IreneNaya: You almost have said what I am trying to say. I believe there are nothing wrong in the spec. I just want to here someone from staff said "Thanks, we will strengthen the test cases." That's it!

posted about a month ago by **fitcat**

@joonhoPark, @NataliaRequejo

My heapq is sorting based on **lt** w/ first pass f=g+n and UDLR and I have vastly different results than you did. I am trying to understand, what is ID? I.D. as in iterative depth? or I.D. as in unique identification, aka adding another variable to my node instances as I create them for comparison purposes? How about use g as the 3rd pass in sorting heapq?

Much thanks! There's no 'official' test cases which makes verifying my ast implementation hard...

posted about a month ago by **HCL33**

ID is probably a sequence number to differentiate two boards. In effect a sequence number will prioritise the earlier board if all other elements are equal: cost, heuristicsanf move... or did I miss another heuristic variable :-)

posted about a month ago by **skofic**

@HCL33, unique ID so you can find it later without ambiguity during decreaseKey operation. You probably should use heapq internal sorting mechanics while maintaining heap invariants during decreaseKey operation. Use of 3 keys just means we wanted to be clear on how to break ties, we first look at f(n) then UDLR and then unique ID. There have been much talk about correct way of doing this but in the end those three things are what the staff is looking for. Read the documentation on heapq till the very end to understand correct implementation.

posted about a month ago by **JoonhoPark**

Inspecting my results using heapq as my priority queue represenation:

In case:

1. The cost is the same (say 8)
2. The movement is the same (say Up)

then you don't have to set a third priority constraint as the heap pops the oldest by default, which is fine (like if you held a creation timestamp and you wanted to pick the min timestamp (firstIn) first). Am I right @dkc2122?

posted about a month ago by **StathisPeioglou**

Add a comment

Support

battila_hu
about a month ago

Hey,
first I solved the A* with a dict, then as I read this comment I made a new one with priority Q.

```
Solution with dict:
656 function calls in 0.002 seconds

Solution with Priority Q:
771 function calls in 0.002 seconds
```

So I don't see the advantage of the pQ so far, it seems that use more resource. Maybe with longer routes, bigger datasets.

So, I checked the example above, and found a bug in my priority queue solution. I'm using the python built in pq, but that does not support item removing just the pop. So I had node in my Q with different costs. And that caused an issue. Now it is solved, and seems pretty fast:)

```
$ python driver.py astQ 7,2,4,5,0,6,8,3,1
path_to_goal: ['Left', 'Up', 'Right', 'Down', 'Right', 'Down', 'Left', 'Left'
cost_of_path: 26
nodes_expanded: 2215
fringe_size: 1220
max_fringe_size: 1221
search_depth: 26
max_search_depth: 29
running_time: 0.048
max_ram_usage: 8.6328125

$ python driver.py astDict 7,2,4,5,0,6,8,3,1
path_to_goal: ['Left', 'Up', 'Right', 'Down', 'Right', 'Down', 'Left', 'Left'
cost_of_path: 26
nodes_expanded: 3106
fringe_size: 1664
max_fringe_size: 1665
search_depth: 26
max_search_depth: 30
running_time: 0.244
max_ram_usage: 8.62109375
```

posted about a month ago by **battila_hu**

Support

@battila_hu, I don't know if you are following our discussion a bit above but my implementations, currently we are down to what to do when cost is equal but has different move which took the board to the contested state, definitely have different stats than yours. I would encourage participation in the said conversation. :)

```
%run driver.py ast 7,2,4,5,0,6,8,3,1
path_to_goal: ['Left', 'Up', 'Right', 'Down', 'Down', 'Left', 'Up', 'Right',
cost_of_path: 26
nodes_expanded: 3264
fringe_size: 1753
max_fringe_size: 1754
search_depth: 26
max_search_depth: 26
running_time: 0.200000047684
max_ram_usage: 0.0
```

posted about a month ago by JoonhoPark

Could it be possible to have max_search_depth over 26 in AST for your run ? Think twice at the order of expanding nodes. Here is the start of my frontier list when finding solution (not same as yours)

```
:
   ---> Current state : LURDDLURRULLDRRDLURULDDLUU
0 1 2
3 4 5
6 7 8
    ---> Neighbors :
   ------- Frontier -------
Cost  State             Path
26  7,4,0,3,2,1,6,8,5 LDRURDLLURDLURUR
26  2,5,4,6,3,0,7,1,8 LURDRDLLURRDLUR
26  7,2,5,3,1,0,6,4,8 LURDRULLDRDLURRDLUR
26  7,6,0,5,4,1,8,2,3 URDDLUUR
26  2,6,0,7,4,1,8,5,3 LURRDDLUUR
26  7,1,0,5,4,3,8,2,6 RDLUURDDLUUR
26  2,1,0,7,4,3,8,5,6 RDLULURRDDLUUR
26  7,6,0,3,4,1,5,2,8 DLURURDDLUUR
26  2,6,0,7,4,1,5,3,8 DLUURRDDLUUR
26  6,0,2,7,1,3,5,8,4 RULDDRULDLUUR
26  3,0,2,7,4,6,5,8,1 DRUULDRDLLUUR
26  7,1,0,6,4,3,5,2,8 RDLLURURDDLUUR
```

posted about a month ago by ELamy

@Elamy, Ok after sticking to the "specification" ironed out with @dkc2122, I get the same sequence as you. This is the reason why I wanted to leave no stone un-turned when working out the spec. ;)

```
%run driver.py ast 7,2,4,5,0,6,8,3,1
    path_to_goal: ['Left', 'Up', 'Right', 'Down', 'Down', 'Left', 'Up', 'Righ
    cost_of_path: 26
    nodes_expanded: 3253
    fringe_size: 1755
    max_fringe_size: 1756
    search_depth: 26
    max_search_depth: 26
    running_time: 0.228999853134
    max_ram_usage: 0.0
```

As for max_search_depth I am not sure if that is possible.

Support

posted about a month ago by JoonhoPark

@JoonhoPark now you have 1755 and 1756 in fringe size and max fringe size ? What happened, I had like you initially.

posted about a month ago by ELamy

@ELamy, the latest result is based on specification worked out with @dkc2122. There are many subtleties we can consider, but the staff has laid out which ones we should consider and which ones we should ignore.

posted about a month ago by JoonhoPark

@JoonhoPark I have no problem with the instructions. It was clear: use priority queue. The UDLR does not matter, because after a while you are gonna have a big frontier, some nodes will have the same weight/cost, and there will be plenty of UP to get to those.

The only issue I had was the duplicated states in the q. But I solved it with a pretty simple method.

Ok, I had an another bug, but it was definitely my silly fault: I created a stupid false Manhattan scoring. But yesterday I fixed it. So now it is perfect.

posted about a month ago by battila_hu

I use heapq and prioritize with ((f(n), UDLR), time_added)

```
path_to_goal: ['Left', 'Up', 'Right', 'Down', 'Down', 'Left', 'Up', 'Right',
cost_of_path: 26
nodes_expanded: 162406
fringe_size: 9283
max_fringe_size: 21672
search_depth: 26
max_search_depth: 32
running_time: 6.54120779037
max_ram_usage: 179MB
```

I still get longer searches!

posted about a month ago by bicepjai

Are you maintaining heap invariance?

posted about a month ago by JoonhoPark

according to docs, heap invariant is maintained with heapq operations

posted about a month ago by bicepjai

How are you decreasing the key? Depending on how you do, it doesn't. For example, you can't just take an item out of the underlying list of heap and expect it to hold invariance without doing heapify on the whole list, which is expensive. Of course, The docs does give a workaround.

posted about a month ago by JoonhoPark

Support

The max search depth of yours is too high. The A* should always try the nearest/shortest route. If your algorithm went to 32 steps deep that means you missed the optimal solution which is just 26 steps from the start.

By the way the UDLR is really does not matter in this case. Because if you go UP the the cost will be +1, so the next pick should be DOWN, then LEFT, then RIGHT. IT is just the order. And on the other hand usually for example there are more then 4 possible choices (just for example) with 26 costs. So you have multiple up-s, down-s, left-s, right-s. And you pick a route by not checking what was the previous way to that child/state, maybe it was RIGHT but you should choose that one which has UP ;) I think it does not worth the effort to manage such a queue, it is just an illusion that you still pick the next child in the order you want, but you don't. You wanted to pick the kid whose parent transition was UP but the parents parent transition should be UP too, and ... And if there is no such, then you want to pick one which has a DOWN somewhere. Or Left, or Right. No way. Keep it simple!

posted about a month ago by **battila_hu**

@JoonhoPark, I have almost the same stats as you for 'python driver.py ast 7,2,4,5,0,6,8,3,1', but my fringe and max fringe are a bit bigger:

path_to_goal: ['Left', 'Up', 'Right', 'Down', 'Down', 'Left', 'Up', 'Right', 'Right', 'Up', 'Left', 'Left', 'Down', 'Right', 'Right', 'Down', 'Left', 'Up', 'Right', 'Up', 'Left', 'Down', 'Down', 'Left', 'Up', 'Up']
cost_of_path: 26
nodes_expanded: 3253
fringe_size: 1809 (yours was 1755)
max_fringe_size: 1810 (yours was 1756)
search_depth: 26
max_search_depth: 26
running_time: 0.559194
max_ram_usage: 13

Do you have any ideas why my fringe size is bigger? Perhaps it's due to keeping duplicate boards in the fringe when the same board with lower cost is encountered? That would mean in this test example, at the time of finding the solution, there are 54 boards in the fringe that are duplicates of lower cost boards, but have not been popped off and discarded yet.

EDIT: YEP! That was it. I now have the same stats as you for this case.

posted about a month ago by **ryanwc**

@JoonhoPark I am implementing the one mentioned in the Priority Queue Implementation Notes, still seems like my frontier PQ is wrong.

@battalia thats true, the depth is really high, trying to debug it

posted about a month ago by **bicepjai**

```
ast 7,2,4,5,0,6,8,3,1
cost_of_path: 26
nodes_expanded: 2675
fringe_size: 1542
max_fringe_size: 1543
search_depth: 26
max_search_depth: 26
running_time: 0.70999956
max_ram_usage: 16.15625000
```

posted about a month ago by **T-900**

could someone verify the following ?

> f(n) = manhattan_distance + search_depth

posted about a month ago by **bicepjai**

Add a comment

Support

MT3
about a month ago

[Staff] There's some discussion about whether or not the 'nodes_expanded' counter should be reset after each iteration within the IDA* algorithm. For example, see here. Could you please give the correct interpretation?

Responded here.

posted about a month ago by **dkc2122** (Staff)

thanks

posted about a month ago by **MT3**

Add a comment

skofic
about a month ago

In your post you say:

> Now you might ask, what if there are multiple "Up" choices with the exact same key? We don't want to give away too much, so we leave this case for you to think about. First off, does this happen with the 8-puzzle? Why or why not? Does this depend on your choice of heuristic?

When you say "Does this depend on your choice of heuristic?" do you mean that we are free to choose the best heuristic? In the instructions you say that the A* should use the Manhattan heuristic, but there is another heuristic that performs better. Are we free to use any heuristic we find best suited, or must we stick with Manhattan? Or should we use exclusively Manhattan for A* and our choice for IDA?

From everything I read and understood, I'd say that you have to stick to the Manhattan.
Those questions are more guidlines, they encourage you to think about the problem.

posted about a month ago by **PassiW**

This "does it depend on your choice of heuristic" means... If the heuristic were different, would the answers to the afore-mentioned questions be the same?

It does not suggest you're free to choose the heuristic! Manhattan priority calculation is the one to use.

posted about a month ago by **AHoebeke**

Add a comment

Support

RoderikV
about a month ago

given this exercise, the fringe size and max fringe size are completely arbitrary.
you are not testing if we understood the principle, but if we can replicate your implementation by trying to match your outcome. This then becomes an arbitrary test of how well you can code - to be specific: how well you can code in python on unix/linux.
In that sense, it would be more beneficial to say "yes you did get an answer" or "your answer was wrong as it did not lead to the expected outcome" and additionally add "but you did worse than us" or "great, you did better than our solution" additionally, you could post the solution you have implemented as study material to improve our coding skills.
May i suggest you should supply a starter code in driver.py giving you the goaltest, the State model and the solver with the 4 small changes needed to implement bfs, dfs, ucs and ids.

leaving off course the 4 small changes to be implemented

posted about a month ago by **RoderikV**

Tend to agree. Will be interesting to see the "perfect" solution???

posted 29 days ago by **geoffc**

Add a comment

misoa
17 days ago

Does anyone have a clue why I get a **Cost of 30** and not **26** for the "7,2,4,5,0,6,8,3,1" and 11 for 3,1,2,0,4,5,6,7,8 configuration. I did implement carefully the tuple(f(n), UDLR, increm_id) and the algorithm from the lectures with a set(). Any hint what I could check would be appreciated! :)

Support

Well ! Here what I think you are doing.

First: your difference between 30 and 26 lies in that you take into account the 0.

Now why do you want to obtain 26 ? In fact you count the manhattan distance in a 1D space: "7,2,4,5,0,6,8,3,1" -> "0,1,2,3,4,5,6,7,8", but you have to count in a 2D space:

```
7 2 4        0 1 2
5 0 6   ->   3 4 5
8 3 1        6 7 8
```

Then you get 18 !

Now I can't explain why you get 11 for "3,1,2,0,4,5,6,7,8" (the correct value is 1)

EDIT: Sorry I'm realizing that I certainly misunderstood the question (This project dates back some time now). Anyway try to get the correct value for the simplest configuration "3,1,2,0,4,5,6,7,8".

posted 15 days ago by **Berliere**

Add a comment

Showing all responses

Add a response:

Preview

Submit

Download on the App Store

ANDROID APP ON Google play

Support