Week 6

Week 6

SUBFORUMS

All

Assignment: Advanced Shortest Paths

Advanced Shortest Paths project, problem "Compute Distance Faster using Coordinates"

Michael Levin Instructor Week 6 · 4 years ago

Please use this thread to discuss Problem 2, "Compute Distance Faster using Coordinates", of Advanced Shortest Paths Programming Assignment (make sure to review forum rules before posting).

0 Upvotes Reply Follow this discussion

Earliest Top Most Recent

RA **Rajmohan Asokan** · 3 years ago

My solution is failing in test case #6. I'm trying to submit the bidirectional version of the A* algorithm. I have stress tested the solution and it's not revealing the issue. I have read through previous posts and I'm lost here. I assume test case #6 is just another large test case. Any hint would be appreciated.

0 Upvotes Hide 7 Replies

RA **Rajmohan Asokan** · 3 years ago

Actually, my solution failed in a large test from the stress testing.

0 Upvotes

Pradyumn Agrawal · 3 years ago

Did you managed to pass test case #6? I am trying too. Unable to debug my program.

1 Upvote

Pradyumn Agrawal · 3 years ago

I got this testcase from the forum. I think this failing the implementation. Can you tell me what is ouput of your program?

1 Upvote

RA **Rajmohan Asokan** · 3 years ago

No, I still haven't passed this test case. And, I checked the test case that you have provided. The query part looks like this,

1

1 56854

466 106

261228 259707 389

259707 261228 389

1

1 56854

6 106

261228 259707 389

259707 261228 389

1

1 56854

07 389

259707 261228 389

1

1 56854

4

1228 389

1

1 56854

4

Is this correct? Would you give me some help on understanding this? Anyway I looked at this after getting a weird result of 6 when I ran my solution on this.

0 Upvotes

Pradyumn Agrawal · 3 years ago

@Rajmohan I am not able to get your problem?

0 Upvotes

RA **Rajmohan Asokan** · 3 years ago · Edited

@Pradyumn, I couldn't make sense of this query section in the test case that you provided. So the input should be q (the no. of queries) and the next q lines will have the start nodes and the target nodes for which we have to compute the shortest distance. But the query section is messed up. Or am I missing something? Btw, I switched to unidirectional A* and was able to pass test case #6 and now I'm stuck in test case #8 (time limit exceeded).

0 Upvotes

Pradyumn Agrawal · 3 years ago

Query section is at very last with only one query. Go through the test case thoroughly.

0 Upvotes

Reply

Reply

Anthony Khong · 4 years ago

Michael, is there anything special about test case 9? Any chance I could get some tips on how to get my test to fail? I keep getting the wrong answer after zero seconds:

Failed case #9/9: Wrong answer

(Time used: 0.00/50.00, memory used: 17657856/2147483648.)

1 Upvote Hide 1 Reply

I **Michael Levin** Instructor · 4 years ago

Please try again, your solution should actually pass, looks like we had an unexpected technical error.

1 Upvote

Reply

Reply

Nataliia Khodiakova · 4 years ago

What is the bottleneck of the algorithms? I implemented carefully both unidirectional and bidirectional algorithms in Java, but both are getting time limit on test case #8.

Failed case #8/9: time limit exceeded (Time used: 200.32/100.00, memory used: 439898112/2147483648.)

Unidirectional algorithm is as follows:

1. Calculate potentials for all nodes - potential is euclidean distance from the start node to current node
2. Visit start node and add it to queue
3. While queue is not empty or the top node is not target node, take the top node u, go through its not visited neighbors v and compare dist[u] + cost[u][v] + potential[u] - potential[v] to dist[v].
4. If it improves the result, add v to the queue if it's not already there.
5. After the loop is finished, return dist[target] + potential[start] - potential[target].

Bidirectional algorithm is as described in lectures, potential function is (distance(start, node) + distance(node, target))/2. The sign of potential function depends on the side, of course.

In an endless loop I process the queues of both sides. I take a top node from a queue, try to relax it. Then if it was visited on the opposite side, loop breaks and I find the shortest distance.

I believe, the algorithms compute the distance correct, because they got as far as test #8. But still there is some bug which causes time limit on big data sets.

I tried to avoid Java-specific issues. java.util.Scanner is pretty slow, so I replaced it with FastScanner implementation that uses StringTokenizer and BufferedReader, so this shouldn't be an issue.

Do you have any ideas where the problem might be?

Many thanks in advance.

2 Upvotes Hide 8 Replies

Anthony Khong · 4 years ago

Hey, I'm on the same boat. Did you finally manage to solve it?

0 Upvotes

Nataliia Khodiakova · 4 years ago

Hi, nope, still struggling with it

0 Upvotes

Anthony Khong · 4 years ago

FWIW, I managed to clear test case 8 using C++. The mistake I made was not returning vertices and edges by reference, and that made it slow. But now I'm getting wrong answer on test case 9 :(

0 Upvotes

Nataliia Khodiakova · 4 years ago

Thanks for the hint, I changed Java classes to primitives and also got to Wrong Answer on test case 9 :)

Any ideas what it might be?

0 Upvotes

I

Ivan · 4 years ago

Looks like invalid input.

0 Upvotes

Michael Levin Instructor · 4 years ago

If you had a problem with test 9, please submit again, your solution should actually pass, looks like we had an unexpected technical problem.

0 Upvotes

Nataliia Khodiakova · 4 years ago

Thank you, Michael, it passed indeed :)

0 Upvotes

Dhruv Khandelwal · 3 years ago

Hey could you tell me a little bit more how you got the unidirectional astar to work for case #8(timeout). I have implemented the same unidirectional algorithm as you have mentioned above.

0 Upvotes

Reply

Reply

C

Clare · 4 years ago · Edited

This a very frustrating problem. I thought I'd finally cracked this but my bidirectional solution fails on test number 9 with a wrong answer error.

0 Upvotes Reply

Yuri Bochkarev · 4 years ago

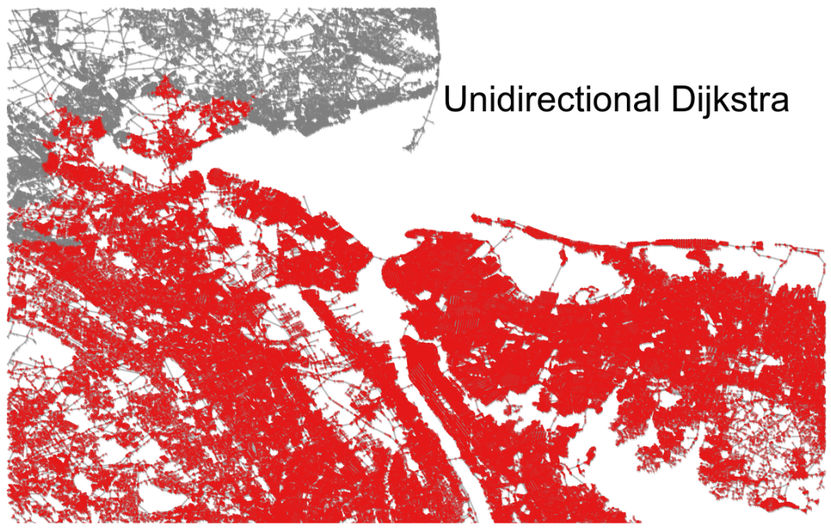
TL;DR: Why are vertices visited many times on graphs from 9th DIMACS Implementation Challenge? Is it because of errors edge weights of the graphs (DIMACS page warns about that) or because my algorithm is flawed?

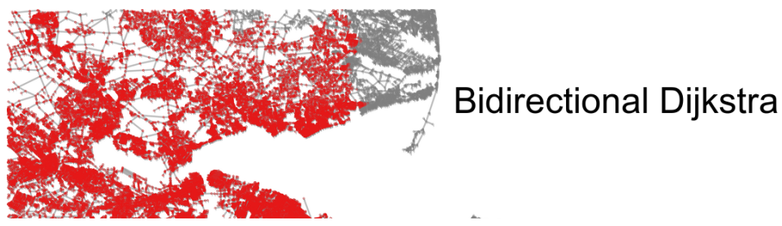
First of all, I want to once again thank our instructors and all those who participated in these discussions. These hints and explanations were very useful to me. Thank you.

In a couple of past week I've managed to advance more than I did in the previous 4 months. I've finally implemented Landmarks heuristics and I'd like to show visuals of my results.

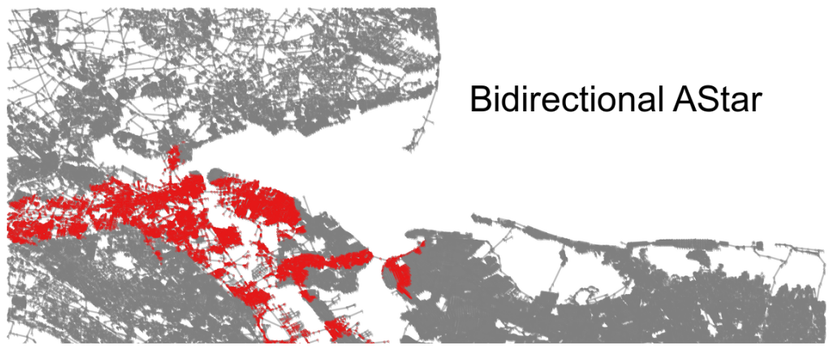
I took the smallest graph of New York City (264,346 nodes, 733,846 arcs) from http://www.dis.uniroma1.it/challenge9/download.shtml. There are two variants of edge weights: travel time and distance time. I used travel times version because I had had problems with distance times version.

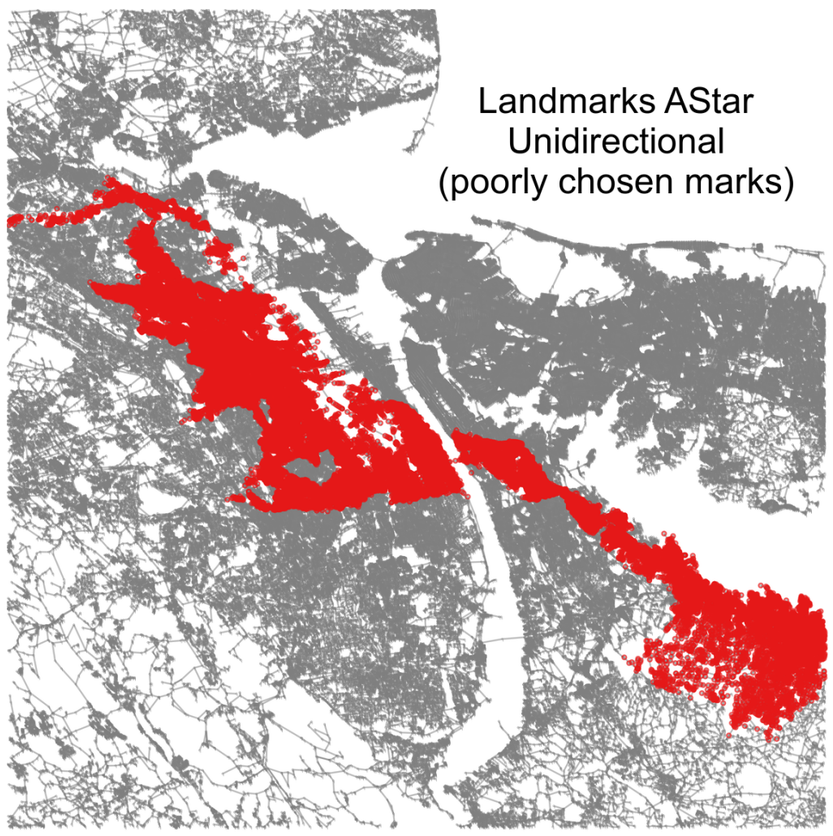
Pictures below were created by querying distance from 1 to 56854 (1-based). If anyone wants to look at the source graph, here is the file: https://www.dropbox.com/s/wfnoi9k7kj5sal6/usa-road.in.xz?dl=0. I've added labels, so hopefully they are self-explanatory.


Unidirectional Dijkstra

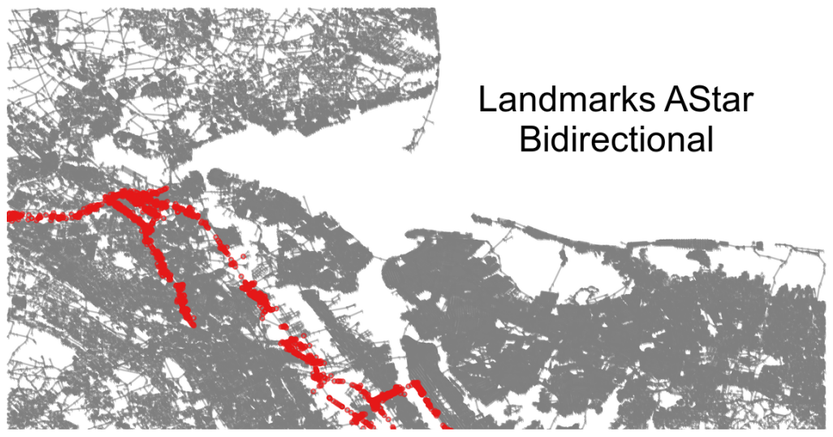

Bidirectional Dijkstra


Unidirectional AStar


Bidirectional AStar


Landmarks AStar
Unidirectional
(poorly chosen marks)


Landmarks AStar
Unidirectional



Now let's compare running time of these algorithms:

```
          DijkOne  AStarOne    ALTOne    DijkBi   AStarBi     ALTBi
<time>   3.29817   5.90689   0.59001   2.17007   1.20313   0.38343
<time>   3.56023   6.02202   0.02089   0.18924   0.04630   0.02870
<time>   3.48593   5.96014   0.02705   0.01701   0.01017   0.03909
<time>   3.69461   6.21951   0.00608   0.02138   0.00817   0.01095
<time>   3.56918   6.08143   0.01821   0.05103   0.02893   0.06390
<time>   3.60630   5.98832   0.04233   0.06175   0.03218   0.09057
<time>   3.48395   6.13712   0.01294   0.00907   0.00756   0.02220
<time>   3.61572   6.03906   0.03160   0.01882   0.01514   0.03970
<time>   3.60616   6.12585   0.02355   0.01299   0.01137   0.01662
<time>   3.54513   6.02290   0.02299   0.01184   0.01186   0.01414
```

I did't send Landmarks implementation to grader, but from my local stress testing I see that sometimes it misbehaves:

```
              BFS   DijkOne  AStarOne    ALTOne    DijkBi   AStarBi
               ALTBi    s -> t
           1328085   1328085   1328085   1328085   1328085   1328085
           1328085   1 56854
MISMATCH    145694    145694    145694    148376    145694    145694
 145694 3214 5138
            143270    143270    143270    143270    143270    143270
 143270 2201 3913
MISMATCH    128130    128130    128130    128472    128130    128130
 128472 3961 1786
MISMATCH    175503    175503    175503    178026    175503    175503
 175503 4902 2655
            301740    301740    301740    301740    301740    301740
 301740   34 3453
             80593     80593     80593     80593     80593     80593
   80593    1 3453
            210895    210895    210895    210895    210895    210895
 210895  100 3453
            165748    165748    165748    165748    165748    165748
 165748  200 3453
            154082    154082    154082    154082    154082    154082
 154082  300 3453
            235219    235219    235219    235219    235219    235219
 235219  400 3453
            152765    152765    152765    152765    152765    152765
 152765  500 3453
             73350     73350     73350     73350     73350     73350
  73350  600 3453
            115121    115121    115121    115121    115121    115121
 115121 1000 3453
            138221    138221    138221    138221    138221    138221
 138221 3000 3453
MISMATCH    145694    145694    145694    148376    145694    145694
 145694 3214 5138
            143270    143270    143270    143270    143270    143270
 143270 2201 3913
MISMATCH    128130    128130    128130    128472    128130    128130
 128472 3961 1786
MISMATCH    175503    175503    175503    178026    175503    175503
 175503 4902 2655
```

And finally (thank you for reading to this point!), the question: why are vertices visited multiple times by all algorithms? I want mention that I've already passed the grader tests by sending Bidirectional Dijkstra and Bidirectional AStar, so I thought the implementation is correct. What I see is this (as a reminder, my graph has 264,346 nodes, 733,846 arcs):

```
 120197 usa-road.abi.visited.txt
 274520 usa-road.aone.visited.txt
 493044 usa-road.bi.visited.txt
  15440 usa-road.labi.visited.txt
 114435 usa-road.laone.badmarks.visited.txt
  43828 usa-road.laone.visited.txt
 779220 usa-road.one.visited.txt
```

First column is the number of vertices scanned in any direction while doing query (1, 56854). "bi" (Bidirectional Dijkstra) and "one" (Onedirectional Dijkstra) have outstandingly large number of visited vertices. Now let's see what the top vertices that are visited often:

```
usa-road.abi.visited.txt
     12 134677
     10 1691
     10 134648

usa-road.aone.visited.txt
     22 134677
     18 194677
     17 140960

usa-road.bi.visited.txt
     12 162618
     11 194677
     10 61230

usa-road.ladi.visited.txt
      9 190803
      9 134677
      8 2559

usa-road.laone.visited.txt
     10 190409
     10 134972
      9 187960

usa-road.one.visited.txt
     12 162618
     11 194677
     11 141685
```

First column is the number of visits, second — vertex ID. In my pseudocode for onedirectional Dijkstra and AStar I have this:

```
def query(...):
    while not queue[0].empty():
        _, u = queue[0].get()
        self.visit(queue, u)

    if self.dist[0][target] != self.inf:
        return self.backtrack(source, target)
```

Notice that "if" is not in the while loop. If I move it there (simply indent it), the number of visited vertices reduces dramatically, but I occasionally get wrong (suboptimal) answers because I leave the loop too early.

2 Upvotes Hide 3 Replies

Michael Levin Instructor · 4 years ago

"Notice that "if" is not in the while loop. If I move it there (simply indent it), the number of visited vertices reduces dramatically, but I occasionally get wrong (suboptimal) answers because I leave the loop too early."

Well, this part is clear. If you stop your algorithm as soon as your target vertex gets some finite distance estimate, it will often be wrong. The moment when the vertex gets a correct distance estimate which is equal to the actual distance from source is when you *.get()* it from the queue, not the first time when you update the distance estimate. You should stop the one-directional algorithm when the vertex that you pop from the queue is the target vertex (however, the stopping criterion for the bidirectional Dijkstra is different and is given in the lecture, and it follows that other bidirectional algorithms should also stop earlier). The distance estimates for the vertices are and should be updated multiple times in any of the above algorithms. However, each vertex should be pushed into the queue at most once, and consequentially, it should be popped from the queue at most once, and this should be ensured by checking whether you've already pushed a vertex into the queue before pushing it in. If you make these changes, it will ensure that the vertex is not processed multiple times (but it still will get multiple updates of its distance estimate $dist[v]$, and this is totally OK).

0 Upvotes

Yuri Bochkarev · 4 years ago

Wow, somehow I missed this. Apparently, I was computing distances from source to all, which is often described in the articles on the Internet.

I think I was mislead by some StackOverflow answer (probably this http://stackoverflow.com/a/31123108/258421). At first I didn't find any use in self.visited array in the python starter kit, friend_suggestion.py module. Because Python's queue does not support decrease_key, I thought having some duplicates in the queue is ok.

Now I've modified one-directional Dijkstra after your suggestion. Just to be sure we're on the same page, pseudo code is as follows now:

```
def query(s, t):  ...  dist[s] = 0  queue.put((0, s))
visited[s] = True  while queue is not empty:    u =
queue.extract_min()    if u == t:      return backtrack(...)
visit(u)  return -1def visit(u):  for v in neighbors of u:
alt = new estimate distance to v    if alt < dist[v]:
dist[v] = alt      ...      if not visited[v]:
queue.put((alt, v))       visited[v] = True
```

A better name for `visited` could be `enqueue` but I left it as is. I believe this is what you were talking about: I only push a vertex to the queue once. Unfortunately, using my utility for stress testing, I manage to find such queries, where returned result is suboptimal (the table looks better here http://sprunge.us/ccdX):

```
→ time pypy3 stress.py --random-queries 1000 --skip-embedded
--mismatch-only < test_astar/usa-road.10k-req1.t.in
  master                        DijkOne        AStarOne
 DijkBi         AStarBi         RefDijk          s -> tExecuting
1000 random random queries (seed 883965046)       MISMATCH
    210798          209046          209046          209046
    209046        4986 443        MISMATCH          376447
   376447          376447          376447          380542
5785 5586        MISMATCH          121952          120779
  120779          120779          120779        2241 1614
MISMATCH          219947          219947          219947
  219947          225358        3490 5957       MISMATCH
  62469           62247           62247           62247
  62247        2088 3158        MISMATCH          179917
179372          179372          179372          179372
4623 3866        MISMATCH          98068           98068
   98068           98068          102163        5765 6136
MISMATCH          110215          109670          109670
  109670          109670        2946 4083       MISMATCH
 209931          205978          205978          205978
 205978         2261 211        MISMATCH          245468
244471          244471          244471          244471
5825 3668        MISMATCH          99069           97727
   97727           97727           97727        3772 3271
MISMATCH          221306          214627          214627
  214627          214627        4082 591
```

Of course there is a chance that I made a mistake in three other algorithms. But there is also RefDijk — so called "reference" Dijkstra implementation by someone from the Internet (which of course is not guaranteed to be correct). As you can see from the table, there are cases where RefRijk is the out-lier, but there are also cases when all but one-directional Dijkstra agree on the result. Why it could be that?

I submitted this solution to the grader and it passed first 4 tests (and failed with a timeout on 5th). I had a feeling that the first 4 tests include tricky cases and edge corners and if you pass them, the algorithm is considered to be correct.

"it should be popped from the queue at most once, and this should be ensured by checking whether you've already pushed a vertex into the queue before pushing it in" — is this also valid for the other algorithms such as AStar and bidirectional Dijkstra?

0 Upvotes

Michael Levin Instructor · 4 years ago

" I had a feeling that the first 4 tests include tricky cases and edge corners and if you pass them, the algorithm is considered to be correct." - no, you should not assume this.

Some "reference" implementation from the Internet may well be wrong. If you submitted it and it got "Wrong Answer", then it is definitely incorrect.

For bidirectional algorithms, the property that each node is pushed into the queue at most once is still valid, but with a remark that we are talking only about one direction, forward or backward. In total, for bidirectional algorithms, they could push the same vertex twice - once in one direction and once in another. And yes, this is true for AStar as well, as you can consider AStar just the same Dijkstra with some potentials function on edges, and you can first apply the potential function to all the edge lengths, and then just run the regular Dijkstra (and then remember to subtract back the potential from the final answer).

0 Upvotes

Reply

Reply

Yuri Bochkarev · 4 years ago

When I was struggling with Bidirectional AStar, I looked for other explanations of the algorithm and I found these slides:

http://www.cs.princeton.edu/courses/archive/spr06/cos423/Handouts/EPP%20shortest%20path%20algorithms.pdf

In you look at slides 10 and 16, you will see that in contrast with the lectures they state slightly different stopping condition. They introduce mu and they suggest stopping when top_f + top_r >= mu. In lectures we stop when a vertex is visited on both sides (page 9?). Michael, could you please explain this difference? Thank you.

0 Upvotes Hide 5 Replies

Michael Levin Instructor · 4 years ago · Edited

The stopping criterion in the slides is very similar in action. Why does it work? The shortest path goes from s, passes through some nodes which are already processed in the forward search, then potentially goes into the area which is not processed neither by forward nor by backward searches, then goes into the area processed by the backward search and from there to t. If the shortest path goes through a node which is actually not processed yet neither by forward nor by backward search, then its length is at least $top_f + top_r \geq \mu$, so it cannot be shorter than the paths already considered. So, any improvement can only be through a path which goes from s, passes through some nodes processed by the forward search, and either one of these nodes is already processed by the backward search (then it is the "meeting point" v from the stopping criterion in my lecture), or there is a direct edge in this shortest path from the last node u on the shortest path from s to t which is processed by the forward search to some node w which is processed by the backward search, and in this case this path is already considered in the value of μ, as the distance estimates of both u in the forward search and w in the backward search are equal to the actual distances $d(s,u)$ and $d(w,t)$ correspondingly, and the distance estimate of u in the backward search is at most $d(w,t) + length(u,w)$, and the distance estimate of

w in the forward search is at most $d(s,u) + length(u,w)$. So, basically the criterion from the slides stops approximately at the same moment as the criterion from my lecture.

With the criterion from the slides, your algorithm could stop slightly earlier, without having to find the "meeting point" v. For example, when the graph is just two s-t paths. One is with edges $(s,u,2), (u,w,3), (w,t,2)$ (where the third number in the tuple is the length of the edge) and the other one is $(s,v,4), (v,t,4)$. Then the shortest path is $s-u-w-t$ of length 7. The "meeting point" would be v, since the forward search would process in the order s,u,v, and the backward search would process in the order t,w,v. My version would stop after both of them find v. The criterion from the slides would stop after forward search found u and then backward search found w, and the path $s-u-w-t$ would be already considered when the backward search processed w, as the distance estimate of w would be already 5 in the forward and 2 in the backward search, $\mu = 5+2 = 7$, and $top_f = 4, top_r = 4$.

0 Upvotes

Yuri Bochkarev · 4 years ago

Thank you for the clarification! From the practical point of view, what exit condition would you recommend using in practice?

I see at as follows. Exit condition from the lecture is easier to implement and it's cheaper computationally. Even though condition with mu stops earlier and saves us time by avoiding certain vertices, it requires booking of mu on the every node visit, which sounds like more CPU work to me. Without any actual measuring and taking all above into account, I'd pick exit condition from lectures.

Please correct me on this.

0 Upvotes

Michael Levin Instructor · 4 years ago

When you know the difference is anyway small, without measuring that it's important you should just implement the easier version.

0 Upvotes

Tom Kenny · 3 years ago

I realize that I am replying to year-old posts here, but thanks for posting these discussions. I was timing out on test 8 using A*, so I adapted my Bi-Directional Dijkstra from the earlier problem to a Bi-Directional A*. I had numerous problems where I would then fail earlier tests. But I did a close reading of the slides from Princeton posted above and (finally) passed. Thanks to you all.

Good job! (Max time used: 64.19/100.00, max memory used: 444076032/2147483648.)

0 Upvotes

vlad · 2 years ago

I've **Python 3** and my results are the following:

- **A*** >> **6.66/50.00**
- **bidirectional A*** >> **time used: 4.64/50.00**

I'm amazed how this algorithms work and I still think there's lots to learn, including different optimizations and "flavors" of A* algorithm. But so far my takeaways are these:

1. Make sure you review slides very carefully and implement potential functions correctly for regular and reversed Dijkstra. You can also use **math.hypot** to speed up the calculation a little bit.
2. Use **heapq module**.
3. **Reverse graph** only **once** before processing the queries.
4. Use **set and dict** for storing processed vertices and distances instead of arrays.
5. Calculate potential "lazily", only when you need it. Don't store precomputed results. And you're gonna need it when you push data to the heap.

Good luck!

0 Upvotes

Reply

Reply

Arseny Antonov · 4 years ago

Is this problem supposed to be sovled with bidirectional astar algorithm, or its enough to implement unidirectional astar? i'm asking it because i got timeout on 6 test for python and i want to understand is it something wrong with my code or i just selected wrong algo?

```
1  Failed case #6/6: time limit exceeded (Time used: 95.93/50.00, memory
   used: 13275136/2147483648.)ə
```

0 Upvotes Hide 3 Replies

Arseny Antonov · 4 years ago

Just ignore this message. The problem was in my code:

```
Good job! (Max time used: 19.22/50.00, max memory used:
13127680/2147483648.)
```

Done with unidirectional AStar

0 Upvotes

Pradyumn Agrawal · 3 years ago · Edited

Can you tell me how did you resolved this issue. I am too facing the same problem. Implemented Unidirectional A^* algorithm. I have tried with big testcases and the test cases that were provided by Instructor. They are all working fine with my algorithm.

1 Upvote

RA **Rajmohan Asokan** · 3 years ago

Pradyumn - in this case, the reason for failure is exceeding the time limit. In my case, test case #6 failure says wrong answer. Is yours a wrong answer or time limit issue?

0 Upvotes

Reply

Reply

KP **Konstantin Pachuev** · 4 years ago

Hello!

My approach is to test the parts then combine them.

1. Test bidirectional algorithm without potential functions (from first assignment)
2. Test unidirectional algorithm with potential functions.

So

1. I run bidirectional Dijkstra from the first assignment and received Test #4: wrong answer. (I didn't expected that).
2. When I run unidirectional A* I've reached Failed case #8/8: time limit exceeded (Time used: 99.94/50.00, memory used: 76361728/2147483648.) That's what I expected in the first item also.

Can I rely that Test #4 in this assignment use same data as previous assignment ?

0 Upvotes Hide 3 Replies

I **Michael Levin** Instructor · 4 years ago

No, of course you cannot rely on such things. There were no such restrictions. Tests can be the same or different for different problems. For this problem, even the input format is different from the previous one, so there is no reason data should be the same.

If you got wrong answer, this only means that you've implemented something incorrectly in the first place, however, the tests in the first assignment may be too simple to catch your mistake.

0 Upvotes

KP **Konstantin Pachuev** · 4 years ago

Thank you, one more thing. Is it correct approach if I'll try fix my bidirectional Dijkstra until I get time limit error, just to be sure that bidirectional algorithm works well, then add potential function to it?

0 Upvotes

I **Michael Levin** Instructor · 4 years ago

Better approach would be to use stress testing to make sure that your your Bidirectional Dijkstra works. Relying just on the tests in the system is wrong. You should learn to test yourself. Also, most of the tests in the system are real-world graphs, so it's cool, but from the other hand, the test set is not that rigorous.

0 Upvotes

Reply

Reply

SB **Sergey K Bessmertnyy** · 4 years ago · Edited

I couldn't figure out Bidirectional A*. In one direction it works like a charm, first single-directional version I uploaded passed all tests with max time 23s (Python3). On test#3 it explored 10463 vertexes total for all 1000 queries.

The same test #3 for Bidirectional-A* version was giving me 75 errors out of 1000 queries (where result wasn't optimal solution) with the same potential function as for one directional search. I played with different potential functions and could eliminate errors, however even with the same approach as 1-directional A* it didn't give me any performance benefits, it relaxed about 6500 vertexes in each direction before meeting in the middle, so total number of vertexes explored was actually about 25-30% higher than with one-directional A*. I also checked one-directional backward search, it relaxed 1000 vertexes more than forward search, still not nearly as bad as bidirectional version. Honestly I didn't expect bidirectional A* to be twice faster, because A* search already goes to the right direction, but I didn't expect it to be significantly slower. So now I'm wondering is it actually slower on this dataset or I was doing it wrong way.

0 Upvotes Hide 3 Replies

Michael Levin Instructor · 4 years ago · Edited

There is no theoretical guarantee that bidirectional version will always work faster than single-directional. However, it seems that you've missed the point which was covered in the lectures that for bidirectional A* you should use a special potential function that is a combination of forward potential and backward potential. Try to fix that and see what's the new result on the same test. You shouldn't play with potential function, you should choose it so that there are theoretical guarantees that your algorithm will make no errors.

1 Upvote

SB **Sergey K Bessmertnyy** · 4 years ago

I haven't missed it, I was able to get correct results however with even slower performance. Just out of curiosity was trying to find faster version. Correct (optimal) result for all quries was obtained simply by (d(v,t) - d(v,s)) /2. Here is my results for test #3. It shows number of vertexes explored for all 1000 queries, for one-directional ones I provided two versions for forward and backward direction.

One-directional Djikstra: 49957 / 50997

Bi-directional djiksta: 9725+9311 = 19036

One-directional A* with euclidean distance: 10463/11065

Bidirectional A* [pf = (d(v,t) - d(v,s)) /2]: 7632+7116 =14748,

And yes, I also tried Bidirectional A* with d(v, t) for forward and d(v,s) for reversed one. Only because it gave best results for one-directional A*. I received 6160+5630=11790 (still not better) however with 75 errors. It proved to be wrong.

So Bidirectional A* here was just a little bit better than bi-directional Djikstra, and one-directional A* was still significantly faster (10463 vs 14748).

Later I'll compare on different, larger dataset.

0 Upvotes

CW **Connor Wong** · 3 years ago

@Michael Is there any guideline/experience on when to choose uni- over bi-directional A*star? For example, which types of network (social, transport, etc.) in which one would perform on average better than the other? Is there any good example where bi-directional approach would be slower because the two paths would hardly meet?

It also seems in assignment 2 (with coordinates) the bidirectional approach works extremely well (about 5s out of 50s for Python).

0 Upvotes

Reply

Reply

Alexander · 4 years ago · Edited

Hello!

Can someone help me out with this problem?

I don't understand why when I update the length with the following heuristic it doesn't work with bi dircetional A star.

```
1  for(int v : neighbors(u)){
2    d= dist[u] + cost(u,v);
3    if(d < dist[v]){
4      dist[v]=d;
5      queue.add(new Entry(d + distance(v, target),v);
6    }
7  }
```

It works with A star, but it doesnt work with bi directional A star.

Bi directional itself works fine without A star

Who can give me a counter example, when just adding to actual distance the distance what left to the target produce wrong answer in bi directional A star.

I understand that the length of the edges in both direction should be equal, I just cannot come up with an example, when it fails

0 Upvotes Hide 1 Reply

Michael Levin Instructor · 4 years ago

It's not clear what is your "dist" here. You should remember that in A* you should forget completely about the initial edge lengths until the very end. All your distances and all your costs should be the ones applied to edge lengths with potentials added. You do a bidirectional dijkstra on the graph with such changed edge lengths, you find the shortest path in this graph with a usual bidirectional dijkstra, and then you just correct the distance with the corresponding difference of potentials of the source and the target vertices, and the path itself stays the same.

Regarding the counterexample when using the changed edge lengths for costs and the initial edge lengths for the distances, or something like this,- you should come up with all such couterexamples yourself, by trying to prove that this should work, finding a flaw in the proof that you don't know how to fix, and then finding a "local" counterexample which makes the proof impossible because of the specific statement that is false, and then extending this counterexample to become "global" - proving the algorithm wrong.

1 Upvote

Reply

Reply

SF **Steven Fraser** · 4 years ago

What's a respectable time for this? I got

Good job! (Max time used: 35.02/50.00, max memory used: 76259328/2147483648.)

which seems slow!

1 Upvote Hide 2 Replies

CW **Connor Wong** · 3 years ago

This is what I got for Python 3:

Good job! (Max time used: 4.75/50.00, max memory used: 19185664/2147483648.)

I adopted basically everything from assignment 1 (bidirectional Dijkstra) , modifying only the edge (i.e. adding the potential). I tried saving also the distance calculation (square, squareroot, etc.) but it didn't shave off too much time.

I suggest anyone encountering difficulty in this assignment to look closely at the lecture slides, following it closely has helped me solve this problem.

1 Upvote

DJ **Du Fong Jie** · 4 months ago

You are right. Only need modify code of problem 1 that can easily fit to problem 2.

Good job! (Max time used: 10.61/50.00, max memory used: 17375232/2147483648.)

0 Upvotes

Reply

Reply

NB **Nadav Blum** · 4 years ago

Hi all,

In the java starter file, edge cost is held in this data structure:

ArrayList<Integer>[][] cost;

But when calculating potentials based on coordinates, the edge cost is not expected to be an Integer, is it ? Should I change this definition to use floats / doubles instead ? Should round edge cost ?

Thanks

0 Upvotes Hide 4 Replies

NB **Nadav Blum** · 4 years ago

Ok. I can avoid taking the square-root in distance calculation.

0 Upvotes

NB **Nadav Blum** · 4 years ago

Ok. Done. In Java

Good job! (Max time used: 72.31/100.00, max memory used: 463368192/2147483648.)

0 Upvotes

Anthony Khong · 4 years ago

Hey, how did you avoid taking the square root?

0 Upvotes

CW **Connor Wong** · 3 years ago

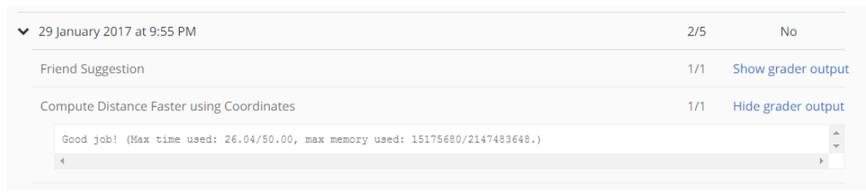
I was thinking about avoiding the square root as well, but it seems that I ran into integer overflow in Python 3 for some cases. Not too sure if that was the cause, but I passed the grader as soon as I used square root.

0 Upvotes

Reply

Christopher Bruner · 4 years ago

Dunno if this counts for competing, but I finally cracked this problem :)



0 Upvotes Reply

Yuri Bochkarev · 4 years ago · Edited

After solving Bidirectional Dijkstra, I'm stuck with Bidirectional AStar.

Basically, the larger graph gets, the more the algorithm spends time oscillating between pairs of two vertices, e.g. here is a list of visited vertices:

```
1   65
2   43
3   65
4   43
5   65
6   43
```

I'm using simple version of priority queue where items can be added many times.

It's not entirely clear to me what our augmented edge weight in Bidirectional AStar is. Lecture notes mention l_pi_f(u, v) = l(u, v) - pi_f(u) + pi_f(v). Then p_f and p_r are introduced, and they rely on pi_f/pi_r. Does it mean that we switch to l_p_f which is now as follows: l_p_f(u, v) = l(u, v) - p_f(u) + p_f(v) ? Is that so?

I implemented p_f/p_r as noted in the lecture and checked the condition p_f(u) + p_r(u) == 0, it holds true for all u in my generated graph (see below).

As for the Relax function, it is as follows in pseudocode:

```
1   Relax(front, side, u, source, target):
2       for v in neighbors(u):
3           actual_dist = dist(u) + cost(u, v)
4           alt = dist(u) + cost(u, v) - p_f(u) + p_f(v)
5
6           if alt < dist(v):
7               dist(v) = actual_dist
8               parent(v) = u
9
10              front[side].add(alt, v)
11              workset.add(v)
12
13      visited[side][u] = true
14      workset.add(u)
```

As you can see, I'm computing alt using current distance + edge cost between u and v + potential functions (I switch between p_f/p_r depending on the side).

I'm not sure I got it right, though. I'm using potentials (alt) only to compare it to the actual distance dist(v), which never uses potentials, only actual edge weights. Is it correct?

For my tests I'm using a generated graph thats looks like a grid of size 10x10. Coordinates are numbers (1, 1), ..., (10, 10). Horizontal and vertial edges are of weight 1 (which is equal to their Euclidian distance), diagonal edges are 2 (which is larger than Euclidian distance (2 > 1.4), but this should still work, right?).

On this sample graph p_f is as follows:

```
1   p_f(u=0, source=0, target=99) = 6.36
2   p_f(u=1, source=0, target=99) = 5.52
3   ...
4   p_f(u=8, source=0, target=99) = 0.52
5   p_f(u=9, source=0, target=99) = 0
6   ...
7   p_f(u=90, source=0, target=99) = 0
8   ...
9   p_f(u=98, source=0, target=99) = -5.52
10  p_f(u=99, source=0, target=99) = -6.36
```

Does this look reasonable?

I'm clueless on where to go next because the only difference from Dijkstra is this potential function and how to use its results, so the mistake is somewhere out there. There is so little code to check, I still can't find the reason of incorrect bevahior.

Could you give me any hints on this one? Am I missing something?

0 Upvotes Hide 16 Replies

See earlier replies

Michael Levin Instructor · 4 years ago · Edited

Well, the main idea is that when you're implementing AStar, you should switch completely from the initial graph to the graph with new edge lengths. New edge length is old edge length plus difference of potentials on its ends. All new edge lengths must be non-negative (check that!) if your potentials are correct - because otherwise Dijkstra's algorithm cannot work, it's a prerequisite. All edge lengths must be the same regardless of computing them "forward" or "backward" (check that!) if you've used correct updated potentials for the forward pass and backward pass - this is the idea for p_f and p_r based on pi_f and pi_r. You should forget about initial edge lengths, initial distances whatsoever during AStar. You will get the correct distances in the end, after your AStar has finished doing Dijkstra with Potentials by adding the potential of one end and subtracting the potential of the other, according to the formulas from the lecture. Currently you're still computing some "actual" distances - don't do this. If you implement everything correctly, your algorithm won't visit any node twice in the same pass (forward or backward), because Dijkstra's algorithm doesn't do this, if all the edge lengths are non-negative.

1 Upvote

Yuri Bochkarev · 4 years ago

Thank you for the clarification. I still don't quite get how I can completely switch to new edge lengths. If I do that, my answer (length of the best path) will be affected, won't it? As far as I understand, the answer is still expected to be in terms of the original edge lengths given in the input graph. That's why I thought that I should not put estimates with potentials into dist array. Please correct me if I'm wrong.

> You will get the correct distances in the end, after your AStar has finished doing Dijkstra with Potentials by adding the potential of one end and subtracting the potential of the other

Do you mean that I need to do "best_path_len - p_f(source) + p_f(target)" one more time in the end before returning the answer?

0 Upvotes

Michael Levin Instructor · 4 years ago · Edited

Yes, of course it will be affected. But this is very easy to fix. You just need to correct with this difference of potentials once in the end, after AStar and its Dijkstras have finished. Whether you need to add p_f(target) - p_f(source) or p_f(source) - p_f(target) to the final answer depends on the formulas you use to change the edge lengths initially: in the end you need to add the reverse.

0 Upvotes

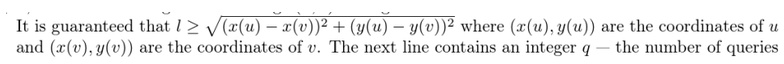
Yuri Bochkarev · 4 years ago

Still being stuck on dist_with_coordinates, I decided to check edge lengths. From the grader I have the output of case2 that is two months old (captured last year): http://sprunge.us/fAQF The output is rather small and does not look shortened, at least it does not contain "..." in the middle, thus I assume it's complete.

Now, the handout PDF in problem 2 states that:

l >= Eucledian dist(u, v)

where l = direct edge (u, v) length (from the input file).


It is guaranteed that $l \geq \sqrt{(x(u) - x(v))^2 + (y(u) - y(v))^2}$ where $(x(u), y(u))$ are the coordinates of u and $(x(v), y(v))$ are the coordinates of v. The next line contains an integer q — the number of queries

However, this is simply not true for case2.

-74258291 40695601

-74259991 40694901

1 2 1838

```
1  R> sqrt((-74258291 + 74259991) ^ 2 + (40695601 - 40694901) ^ 2)
2  [1] 1838.478
```

How's this possible? Is it wrong grader case that is already fixed or am I missing something?

0 Upvotes

SB **Sergey K Bessmertnyy** · 4 years ago

My opinion is since all distances are given as integers, you have to round euclidean distance as well. At least I did.

0 Upvotes

Yuri Bochkarev · 4 years ago

But then again there is division by 2 in p_f calculation. You don't use floating point numbers at all, do you?

0 Upvotes

Michael Levin Instructor · 4 years ago

The test case was wrong in this respect, it has been fixed.

0 Upvotes

Yuri Bochkarev · 4 years ago

Good news, I've finally passed:

Good job! (Max time used: 28.00/50.00, max memory used: 16883712/2147483648.)

There is one question though. My pseudocode looks as follows:

```
1  def p(side, u):
2    pi_f = eclidean(u, target)
3    pi_r = eclidean(source, u)
4    result = (pi_f - pi_r) / 2.
5    result *= (1 - side) + -1 * side
6    return result
7
8  def visit(side, u, ...):
9    ...
10   for v in neighbors(u):
11     potential = -p(side, u) + p(side, v)
12     alt = dist[side][u] + cost[side][u][v] + potential
13     if alt < ...
14   ...
15
16 def backtrack(side, source, target):
17   best_dist = <iterate workset to find best dist>
18   potential = -p(side, source) + p(side, target)
19   potential *= (1 - side) + -1 * side
20   return int(round(best_dist - potential))
```

This may be peculiarity of my implementation, but in the code above I don't understand why line 19 (flip sign of potential depending on the current side, in backtrack) is necessary. If I don't do that, the answer is wrong whenever the search ends on the reverse side of the graph. It seemed to me that doing reverse of what I do in the visit — subtracting potential from best_dist — should be enough, but it's not...

0 Upvotes

Michael Levin Instructor · 4 years ago

You're flipping the side, because in the forward direction your potential is $p(side, target) - p(side, source)$, but in the backward direction your source is $target$, and your target is $source$, so your potential is actually $p(side, source) - p(side, target)$. This is correct.

0 Upvotes

Zachary Luna · 3 years ago · Edited

Michael, wouldn't p(side,target)−p(side,source) just be the distance from s to t ? I don't see anywhere in the slides where it mentions adding p(side,target)−p(side,source) after the Dijkstra is complete.

0 Upvotes

Zachary Luna · 3 years ago

What I mean is, if side=0, then p(side,target)=(pi_f-pi_r)/2 where pi_f=distance from s-t and pi_r=distance from s to s which makes p(side,target)= dist from s-t/2

0 Upvotes

Michael Levin Instructor · 3 years ago · Edited

You should take a look at the slide with the Lemma, it is 6 (8 of 42) in the PDF I see. It explicitly shows the formula for converting between the path length in the initial graph and in the graph with edge lengths updated using potentials.

Potential function is not the distance, it is just some estimate of the distance. If we already knew the distances between nodes, we wouldn't need Dijkstra's algorithm to compute them in the first place.

0 Upvotes

Zachary Luna · 3 years ago

Hi Michael, Yes I already understand this point. What I don't understand is how can you calculate the potential p(side,target) when you are at node s? This is the meaning of the question below (from previous post). Would pi_r=0 here?

side=0, then p(side,target)=(pi_f-pi_r)/2 where pi_f=potential from s-t and pi_r=potential from s to s which makes p(side,target)= dist from s-t/2

0 Upvotes

Zachary Luna · 3 years ago

You see, I've been stuck for weeks on this and step. This step is: after you process a node in both the forward search and backward search and then stop the search to "find the shortest distance". The slides don't say clearly what to do in the step of "find the shortest distance" and this assignment is ruining my life. Can you please tell me what to do after you stop the dijkstra? When you say, "p_f(target) - p_f(source) or p_f(source) - p_f(target)" are you talking about the potential from the node processed in forward search and backward search to s and to t? or are you talking about the potential from s to t and from t to s?

0 Upvotes

Michael Levin Instructor · 3 years ago · Edited

You just need to implement the forward/backward potentials as functions and then forget about them forever during the execution of Bidirectional Dijkstra with Potentials. Now you are just doing the regular Bidirectional Dijkstra, but on a different graph - with edge lengths augmented using the corresponding potential functions. The functions are exactly as they are described by the formulas for π_f, π_r, p_f, p_r in the lectures.

When you've found the node v which is processed both in the forward and in the backward direction in the regular Bidirectional Dijkstra, what you do is you check the shortest path from s to t through v, and you compare it to all possible shortest paths $s - u - w - t$ where u is processed in the forward search, w is processed in the backward search, and there is a direct edge (u, w) between u and w. You choose the shortest of all those paths (including the path through v) and declare it the shortest path between s and t.

Now you need to remember that you've augmented the edge lengths, so your path lengths have changed. The only thing you need to do to correct for that is to add a corresponding difference of potentials to the resulting length. The path itself is still correct, it is the shortest one in the initial graph as well as in the graph with augmented edge lengths. Only the length of this path is different in these two graphs. Lengths of the paths between s and t in the initial graph and in the augmented graph satisfy equation $\ell_\pi(P) = \ell(P) - \pi(s) + \pi(t)$, as states the lemma in the slides. ℓ_π is the length in the augmented graph, while ℓ is the length in the initial graph. Use this lemma to come up with the formula to correct the final length to get the distance in the initial graph.

0 Upvotes

Reply

Reply

Ivan Lazarevic · 4 years ago

Can you please post test case #3 somewhere, because it doesn't fit completely in the grader output?

It says:

"Warning, long feedback: only the beginning and the end of the feedback message is shown, and the middle was replaced by " ... ". Failed case #3/6: (Wrong answer) Input: 100 1000 3097 9733 4950 275 ..."

What I mean is both input and output files (with correct answers).

Thanks.

Maybe on a cloud, or in the starter files zip archive...?

0 Upvotes Hide 3 Replies

Michael Levin Instructor · 4 years ago

I've added the first 3 tests for this problem to the starter files zip archive.

1 Upvote

Diego ARCIS · 4 years ago

My implementation fails here too. Any hint?

0 Upvotes

R

Robin · 3 years ago

I've also failed one of the output queries...any corner case that we should pay attention to?

0 Upvotes

Reply

Reply

Reply

Reply