MENU

FIND MEMBERS BY USERNAME OR SKILL

survival07

EDIT

DASHBOARD

MY PROFILE

PAYMENTS

SETTINGS

LOG OUT

COMPETE

DESIGN CHALLENGES

DEVELOPMENT CHALLENGES

DATA SCIENCE CHALLENGES

COMPETITIVE PROGRAMMING

LEARN

GET STARTED

DESIGN

DEVELOPMENT

DATA SCIENCE

COMPETITIVE PROGRAMMING

COMMUNITY

OVERVIEW

TCO

PROGRAMS

FORUMS

STATISTICS

Support

Forums

Solutions and reviews | Feedback: (+4/-0) | [+] [-] | Reply 1 edit | Fri, Mar 12, 2021 at 12:43 AM BDT

ccplus
45 posts

Hi! Post your solutions and thoughts on the marathon problem.

As for me, I liked this problem a lot. It's supercomplex and interesting. Sad I was only able to start basically in the last 2 days of the contest. At first I just submitted a very naive random walk and later added some optimisations on optimal amount of moves. After that I figured out a heuristic of looping back to previous node in case that node still has above current average treasures and optimised some parameters around that for a pretty massive boost to 85 provisional points. Optimising parameters was quick, as this was just a random walk, so it ran very very fast, I could run over 50k+ test cases in a couple of tens of seconds, collect statistic and retry with different parameters.

I expected that this naive approach would not score very well on cases with low Max Treasure number as for those it seemed possible to scan surroundings better, but I expected that it would score okay on cases with high Max Treasure as I figured you can't do much better there, as those didn't seem too good for scanning one can generally score pretty high points there anyway. I tested which Max Treasure contributed how many provisional points and was shocked to find that basically the reverse was true at least in terms of provisional testbench. I was getting 48.7 on Max Treasure <= 5 cases and for Max Treasure > 5 cases I got mere 36.1. To this time I'm not sure how this was possible, which is one of the reasons I'm so interested in how you solved this problem.

I got to this solution in a couple of hours and decided to attempt a complete rewrite, now deducing nodes, rerunning node deduction strategies over the whole previous experience on every move. I was hoping to do this up to some extent and then figure out some heuristics for navigating the space based on having some knowledge about it. Adapted to the quality of recognition and exploratory needs of the deduction algo whatever they would be. So I coded for like 2 days and only in the last few hours of the contest I got to some more or less decent node recognition algo. If I had a day or two more I'd tune it and integrate it with some movement heuristic, but as of the last 30 minutes of the contest my new solution was performing worse than the tuned heuristical random walk on just about every case, so I ended up having to just revert to using that one. Scores 90 points on provisional.

I was really surprised that such a simple random walk with minimal heuristic scored so well on this problem and I belive this problem got me a bit closer to the level at which I'll be able to attempt really complex solutions. I'll also probably try to attempt complex solutions in a more gradual fashion. When I started writing node recognition I didn't appreciate how complex it is going to be. I overestimated the amount of signal I could get about where a point is. In the end I wasn't able to fully program even half the recognition features I initially expected to on time, because it turned out that core features had lots of unexpected depth. For example, if you hit a node at some point and go back via -1, it might be that the node you hit on which you have precious little info was some node you previously visited and when you later move to it and it has a smaller amount of treasures, the only way you'll be able to figure out that it's maybe the same node is if you'll account for possible intermediary one-off nodes like this on which you definitely have too little info to deduce which node it is, unless, maybe, you have some extremely strong info like 11+ paths from the node..

So to sum it up, liked this marathon a lot, learned quite a few, pretty frustrated I wasn't able to launch my rocket into space and land it back safely, but lessons were still learned.

Re: Solutions and reviews (response to post by ccplus) | Feedback: (+4/-0) | [+] [-] | Reply 2 edits | Fri, Mar 12, 2021 at 4:52 AM BDT

Daiver19
95 posts

Initially I wanted to write a 'formal' solution and compute the distribution of possible graphs, but that proved to be infeasible due to sheer amount of possible states. Maybe I could make it work for smaller cases, but forgot to do that during the contest. So in the end my solution is a bunch of messy heuristics.

Basically, there are 3 questions we need to answer every turn: how many treasures to take, where to go next, when to stop.

1) How many treasures to take?
- I always take as much as possible. Tried to leave a small random amount for information gain, but that never worked.

2) Where to go next? Pick the first option which apply:
- If previous vertex has at least (maxTreasurePickup + 4) treasures just go back. Changing this threashold from maxTreasurePickup to maxTreasurePickup + N was probably the most efficient score gain per character - it gave more than 0.5 points on 1000 seeds locally and almost 1 point on the provisional tests. Interestingly, the flat + N has performed better than some formulas depending on maxTreasurePickup.
- Go to the path which we are sure we've never tried before
- Go back if there is more than 2 treasures in previous vertex
- If there is at least one edge for which we aren't sure if it has been visited before, pick one of such edges based on some weight function
- Lastly, just pick a random edge.

3) When to stop?
- Keep track of negative score so far (basically add the score of each step but reset the result to 0 when it's positive). The idea is to check if we can ever make it positive with the remaining steps.
- Estimate how many treasures are left and roughly estimate the average number of treasures per step. Roughly simulate the remaining steps using these values and some magic adjustments.
- Add the score estimation to negative score so far and stop when the result is negative.
- I've iterated on this quite a bit and think it's decent, but I suspect it's possible to do better.

Tracking - For 2) and 3) we need some tracking.
- For 3) I simply look at the vertices for which we can be sure they're unique (i.e. treasures+edge count doesn't match any other unique vertices) and number of vertices which definitely have 0 treasures (i.e. vertices where I took more that 0 treasures and which became 0 now).
- For 2) keep track of the graph in the following fashion: create a new vertex for each new edge and mark the vertices which have the same treasures+edge count at the moment (let's call them 'compatible'). Also update the compatibility list every step (e.g. if v1 edge 1 leads to vertex with degree 3, but for v2 edge 1 leads to vertex with degree 4, then they're definitely not compatible; if they lead to the same degree vertex, they're likely to be the same vertex).
When need to pick an unvisited edge from the current vertex, recursively traverse the graph of vertices 'compatible' with the current one. The edge which hasn't been visited from any compatible vertex is definitely unvisited. Otherwise use some magic formula to come up with probability of every edge being already visited.

Again, I'm pretty sure it's possible to do better here, but didn't have time for that.

Scoring - I normalized the scores locally by dividing them by the maximum possible score (as if you knew all the vertices an could teleport between them). My final solution gets about 0.917 points on average relative to the maximum possible score, which is quite interesting given that we have much less information.

Re: Solutions and reviews (response to post by Daiver19) | Feedback: (+4/-0) | [+] [-] | Reply

2 edits | Fri, Mar 12, 2021 at 6:02 AM BDT

AmAtUrECoDeR
135 posts

I also used lots of messy heuristics.
At each node, I always take as much treasure as possible at each node.
Then I do one of three things:
1. If we have not gone back to a previous node in the last step (i.e. if we have not chosen the path number -1 in the previous step), and the previous node has more than X treasures, where X is sort of an estimate of the average number of remaining treasures in the entire graph (more detail on that later), go back to it.
2. Otherwise, look at the explored neighbor of our current node with the highest known number of treasures (this can be known over time, more detail below). If this number is X or more, then go there.
3. Otherwise, go to a random neighbor.

For step 2, we can gain information over time about the number of treasures in all neighbors of a specific "center" room if we repeatedly go back to that room (which will only happen while that room has sufficiently many remaining treasures). This is because going down a fixed-numbered path from a fixed room always leads to the same next room (which can be seen by reading the tester source code). We store this information in an array. If at some point we don't go back to the "center" room, we clear our array entirely and designate a new room as the "center".

To calculate X, I maintain two numbers, "E" and "avgroom", as well as a boolean "maybenew". While we travel in the graph, if we choose steps 1 or 2, we set maybenew to false for the next step, and set it to true otherwise.
Both E and avgroom are initialized to 25 at the beginning (as every room is expected to have 25 treasures without any knowledge).
In each room, let cnt be the current number of treasures in that room.
- If maybenew is set to true, then update avgroom to (1-0.175)*avgroom+0.175*cnt.
- Also, if maybenew is set to true and maybenewcnt<N, where maybenewcnt = # of times maybenew was set to true, then update E to E+(cnt-25)/N with a probability of (1-maybenewcnt/N); this is to try compensating for the fact that maybenew often gives false positives (i.e. it is often set to true when it shouldn't be).
- Then, we set X to (1-0.75)*E+0.75*avgroom.
- Finally, after we have decided our next step and updated maybenew accordingly, we take amt=min(cnt,maxTreasurePickup) treasures and update E to E-amt/N.

To decide when to stop, I store the score gains/losses of the previous 50 steps. At any point, if the weighted total 50*s_1+49*s_2+...+s_50 is less than 0, where s_1, s_2, ... are the score gains/losses 1 step ago, 2 steps ago, etc., than I stop.

The numbers 50, 0.175, and 0.75 were chosen by experimentation. My solution scores about 0.897 points on average, relative to the maximum possible score.>

Re: Solutions and reviews (response to post by ccplus) | Feedback: (+0/-0) | [+] [-] | Reply

Mon, Mar 15, 2021 at 2:29 PM BDT

Was it ever better to collect less than the maximum number of treasure? We were hoping that solution would use this approach to leave "bread

dimkadimon
4499 posts

crumbs" through the graph to help in deciding where you have already been.

Re: Solutions and reviews (response to post by dimkadimon) | Feedback: (+0/-0) | [+] [-] | Reply   1 edit | Thu, Mar 18, 2021 at 3:45 AM BDT

tuff
37 posts

My code, which assumed a randomized path list, was based on the observation that if you started at a random node and moved through the graph always using path 0 you would trace a path that eventually ended in a loop. If you removed all the treasure from your "current" node, you could use this as a mark that you had previously done a path-0 trace from this node. If a path-0 move lead to a zero treasure node, abandon this trace move randomly through a path that was not via path 0, hoping to find a node with treasure and start a path-0 trace from that node. Nodes that initially had no treasure slightly compromised this strategy.

When taking all the treasure from the "current" node, leave some treasure in each neighboring non-path-0 node so that it is not marked as being in a known path 0 trace. This is a partial answer to the above question, I did sometimes not take the maximum I could.

The intention was that this would have somewhat fewer wasted moves than a completely random walk as the next node to drain would be more likely to be non-empty (at least for the early moves).

I believe the effectiveness of this strategy was compromised with path 0 always leading to the smallest numbered node because it caused the traces to be much shorter, so the proportion of moves where you knew you were moving into a node likely to have treasure was smaller that it would have been if the paths were random. So you can see I was particularly disappointed to learn near the end of the contest that the paths were sorted and not random.

If the paths were completely random, I'm not sure this strategy would have actually been better than a random walk, but this idea was the only real insight I had about traversing a graph like this, so I used it.

Re: Solutions and reviews (response to post by tuff) | Feedback: (+1/-0) | [+] [-] | Reply   Thu, Mar 18, 2021 at 9:56 PM BDT

sullyper
279 posts

I think your strategy would not have been impacted too much if instead of taking always path 0 you took the middle path, or even better, for each number of paths, you pick a random number at the beginning of the program (or first time you see it) and always pick it:
2 paths (have to be room 0!) -> 1
3 paths -> 2
4 paths -> 0
5 paths -> 4
...

This will also eventually loop as it's deterministic, the length of the cycle should be much longer than picking always path 0 (which would only have 2-cycle at the end). I am not sure if the path would be comparable to a random graph or much shorter, maybe you can give it a quick attempt as it should not be much changes.