Julia Moska 256519

1.Desription of the problem and A* algorithm

A* is an searching algorithm that can be distinguished with it's 'prediction' element. The idea is, that each step has to be reasonable and promising for the future. That way, the search is narrowed to steps that seems to be optimal . The cost of the next step is represented as a *f(n)* function, calculated as *f(n)=g(n)+h(n)* where *g(n)* is a cost to reach to the node, and *h(n)* is a heuristics function. Heuristics function is supposed to estimate the future costs of the road we are about to choose. If we set h as a constant equals to zero, than we would get a Dijkstra algorithm. Usually, a* is used to minimize the costs of reaching the lst node. However, in our case we need to obtain the time to finish all the work, which means finding the highest cost path.

2.Desription of chosen Heuristics

Heuristic Depth First Search

Following: *http://aranne5.bgu.ac.il/others/CohenYossi19812.pdf* description- Simply, h represents all reachable nodes, so the number of nodes that are possible to visit till finding the ending point. DFS algorithm is used to obtain these values.

Heuristic Children

Heuristics that sums weights of all children nodes with parent node weight. I thought of it as a measurement of which part of the graph is in general 'heavier', which direction it leans towards to. The bigger is such a heuristics, the more nodes we have as children, which would mean more possible nodes to be compared on the next step, more possible choices, which might lead us to a better next step. Such a heuristics would also allow situation such that singular child node has enourmously huge weight, and should be chosen in the next step, while if we were f.e just calculating number of children as h value, then that option would be rejected.

Heuristic Children&Parents

H value represented as a sum of node's children and parents. As we've discussed on laboratory, more parents mean a longer processed time before coming to that node, while amount of children create more possible choices in the future as in *children heuristics.*

Heuristic Depth Search First +

Basic DSF heuristic modified by adding element's weight to it. I've decided to check it out, because I didn't think that in graphs,especially the huge ones, where some nodes h value would be in hundreds, adding the node weight would change the balance. Apparently, that was the worst performance heuristics

3.Obtained result

For small graphs the time of procedures was so small, that even with 30 digits precision, I couldn't get any another value apart of row of 0's. Therefore I've decided to not include that nformation for small graphs. Similarly for some bigger graph, some values were like that to, so there I've used 0.0 as a value

Green values are the correct results

test_small.dag

	result
DFS	1.6176527184423595
DFS+	0.9980982099350494
Children	1.6176527184423595
childrenParent	0.9980982099350494

test_small_sparse.dag

	result
DFS	1.5383073446920403
DFS+	1.5383073446920403
Children	1.5383073446920403
childrenParent	1.5383073446920403

test_medium.dag

	result	Heuristics calculation time<nanoSec>	Graph searching time<nanoSec>
DFS	18.2182805254555	93728100.0	15621100.0
DFS+	9.894496589203042		0.0
Children	18.2182805254555	0.0	0.0
childrenParent	18.2182805254555	0.0	62486300.0

test_medium_sparse.dag

	result	Heuristics calculation time<nanoSec>	Graph searching time<nanoSec>
DFS	15.680812285842553	78553500.0	0.0
DFS+	4.992894160724692		0.0
Children	9.71900853777253	0.0	0.0
childrenParent	2.4119022960517364	0.0	0.0

test_large.dag

	result	Heuristics calculation time<nanoSec>	Graph searching time<nanoSec>
DFS	29.52727333677888	1000176000.0	31240500.0
DFS+	12.641456438605989		15621300.0
Children	29.52727333677888	0.0	62483700.0
childrenParent	29.52727333677888	0.0	765745600.0

test_large_sparse.dag

	result	Heuristics calculation time<nanoSec>	Graph searching time<nanoSec>
DFS	21.25355490309097	156168000.0	0.0
DFS+	9.497644157439776		0.0
Children	12.650100695314027	0.0	0.0
childrenParent	6.930028096566502	0.0	0.0

test_xlarge.dag

	result	Heuristics calculation time<nanoSec>	Graph searching time<nanoSec>
DFS	39.97942956819456	9379491000.0	187453600.0
DFS+	22.01866007760561		46863600.0
Children	39.97942956819456	31241800.0	281186100.0
childrenParent	39.97942956819456	0.0	6137387500.0

test_xlarge_sparse.dag

	result	Heuristics calculation time<nanoSec>	Graph searching time<nanoSec>
DFS	29.277590025902363	562369200.0	15621200.0
DFS+	9.893776325223591		0.0
Children	11.00889437273781	0.0	0.0
childrenParent	10.996793692985479	0.0	15670000.0

4.Obtained results analysys

Apparently, the two of the heuristics I've came up myself with works perfectly fine for the the dense graphs, but none of them worked for medium, large and lxlarge sparse graphs. I think there is no point in comparing performance time in small and medium graphs, because the bigger the graph is, the harder the problem becomes and more mistakes might be done, where small and medium graphs results in realy inconsistent data. Children heuristics and children&parents heuristics have definately shortest time of calculating it, because it affects just a current node and it's surroundings, so there is no much complexity in calculations. DFS search is called recursively, so the bigger is the graph, the longer it takes to get through it. If we compare total required time to find a path (heuristics calculation+searching time), then it's the smallest for two last heuristics. To summ up, DFS heuristics is a safe option when it comes to accuracy of the results, but we need to be aware that it's really time taking. If we would know on the beggining that we deal with a dense graph, than we could use children or children&parents heuristics because of the better performance time. However, with no information about the graph's structure, it could be a really bad choice resulting in huge miscalculations. Variation of DFS with node's weight is a total disaster in each case.

5.Sources

Links that I've used to understand the algorithm behaviour and heuristics choice procedure. I've mostly inspired myself by pseudocode and written description of the next steps of the algorithm.

https://www.geeksforgeeks.org/search-algorithms-in-ai/

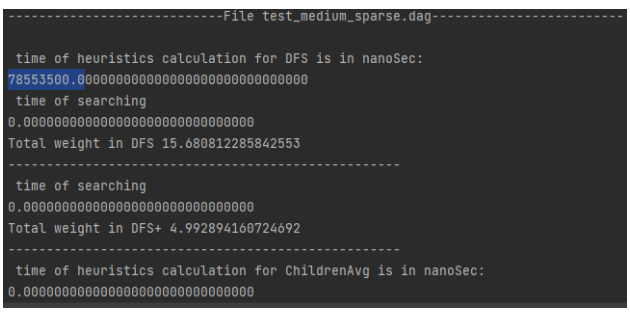
https://www.pythonpool.com/a-star-algorithm-python/

https://stackabuse.com/basic-ai-concepts-a-search-algorithm/?fbclid=IwAR1wiP82so2Y7eepQR8bdXfcMcstHXUJpQkFAV4UFj2DJUKipO-GN8fDb9E

https://cs.stackexchange.com/questions/28336/longest-path-a-admissible-heuristics-and-optimalness

http://aranne5.bgu.ac.il/others/CohenYossi19812.pdf

*https://en.wikipedia.org/wiki/A*_search_algorithm*



Small proof that I've really gotten rows of 0's as result