————————Air Cargo Problem (ACP) Search Results————————

	Breadth first	Depth first	Greedy with 'h1'	A* with 'h1'	A* ignore precond	A* with level sum
ACP-1						
expansions	43	21	7	55	41	11
goal tests	56	22	9	57	43	13
new nodes	180	84	28	224	170	50
plan length	6	20	6	6	6	6
time elapsed	0.04	0.02	0.01	0.04	0.05	0.64
ACP-2						
expansions	3343	624	966	4849	1443	85
goal tests	4609	625	968	4851	1445	87
new nodes	30509	5602	8694	44001	13234	831
plan length	9	619	16	9	9	9
time elapsed	16.46	4.29	2.76	13.94	5.94	57.59
ACP-3						
expansions	14663	408	5462	18235	4945	290
goal tests	18098	409	5464	18237	4947	292
new nodes	129631	3364	48176	159716	43991	2670
plan length	12	392	21	12	12	12
time elapsed	122.42	2.12	18.49	63.03	21.37	262.05

Best result for each row is highlighted, as are some DFS results for ACP-3, which were close to best.

An optimal solution to ACP-1:
```
(6 actions)
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)
```

An optimal solution to ACP-2:
```
(9 actions)
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Load(C3, P3, ATL)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)
Fly(P3, ATL, SFO)
Unload(C3, P3, SFO)
```

An optimal solution to ACP-3:
```
(12 actions)
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Fly(P2, JFK, ORD)
Load(C4, P2, ORD)
Fly(P1, SFO, ATL)
Load(C3, P1, ATL)
Fly(P1, ATL, JFK)
Unload(C1, P1, JFK)
Unload(C3, P1, JFK)
Fly(P2, ORD, SFO)
Unload(C2, P2, SFO)
Unload(C4, P2, SFO)
```

Out of the non-heuristic searches, whose results are shown in the left 4 columns of the above table, only breadth-first and A* returned the solution with the shortest plan length for all 3 ACP. Greedy best-first and A* were both tested here with the 'h_1' "heuristic," which simply returns a constant 1 for estimated closeness to the goal, and therefore is not a proper heuristic. For that reason, Greedy and A* will be compared here to BFS and DFS. (A* with h_1 is the same as Uniform Cost Search, since adding a constant to all nodes' path costs does nothing to change their relative positions in the priority queue.) Greedy did in fact return the optimal solution of 6 actions for the smallest ACP, but failed to do so in the larger 2 problems, returning 16 instead of 9 for ACP-2 and 21 instead of 12 for ACP-3. This is because Greedy assigns the same priority to every node in the queue, so that its solution is in some ways random: It's a function of the possibly arbitrary order in which nodes are added to the frontier. Nevertheless, Greedy could be considered useful for estimating optimal solutions to harder problems, since it was about 6 times faster than BFS and within a factor of 2 in terms of solution optimality. Although Depth-First was even faster than Greedy on ACP-3 (2.12 secs vs. 18.49 secs), its solution of length 392 was so far from the optimal 12 that it really only serves as a check for whether any solution is possible at all. Which is actually very important, of course—For example, faced with a difficult problem, the first step might be to run DFS on it to check that it is even solvable. DFS had planes aimlessly flying back and forth across the country without loading or unloading any cargo in ACP-3, but at least it found in 2 seconds that there was a feasible solution, and it only expanded 408 nodes into 3364 new nodes total (vs. 5462 expanded to 48176 in Greedy; 14663 expanded to 129631 with BFS; and 18235 expanded to 159716 with A*). On a more complicated problem, it would be nice to know there was in fact a solution before dedicating the time and computational resources to finding the optimal one. Then the next step might be to run Greedy to see if the optimal solution was within the ballpark of acceptance. Maybe we're told that we need a solution no longer than 200 actions, and Greedy finds us one with 250, which convinces us to devote the resources to running BFS or A* for the optimal solution.

So if BFS and A* with h_1 both choose which node to expand next based on the same path costs to get to the node (since all actions have the same cost in the ACP problems, which makes a node's path cost equivalent to its depth), why does A* work faster and expand more nodes than BFS? It expands more nodes because it goal-tests nodes only when it pulls them off the frontier, whereas BFS goal-tests them before ever putting them on the frontier. Yet A* took 63 seconds to complete ACP-3 where BFS took 122 seconds. The reason is that A* uses a priority queue (heap) for its frontier where BFS uses a list. In this particular setting, where all nodes have the same priority key of 1, from the h_1 function constant, Python uses a tiebreaker key based on the order the nodes enter the heap, which results in a quick push onto the heap. But the timesaver versus BFS comes from searching each time to see if a potential new node's state is already on the frontier or not, which is done in logarithmic (of the number of nodes in the frontier) time for the heap but linear time for the list. So even though in ACP-2, for example, the maximum frontier size got up to about 1400 nodes with BFS, and about 1700 in A*, BFS took almost twice as long to finish (even after testing various other orderings of initial states for the problem, to make sure A* wasn't just getting luckier on the order). The disparity in search time grows with the size of the problem too, as can be seen by the advantage being only 16.46 to 13.94 secs in ACP-2 yet 122 to 63 in ACP-3. With a much larger problem, the difference would be much more noticeable, such that out of the non-heuristic searches discussed here, the relatively small extra space needed for the extra node expansions in A* is more than made up for by its increase in speed over BFS.

As for the two A* searches performed with real heuristics, shown in the last two columns of the table above, the "ignore_preconditions" heuristic was the timesaver of the two, while "level_sum" was the spacesaver. For ACP-3, "ignore" needed 43991 nodes, and "level" only 2670. Yet "level" took 262.05 seconds to find its solution, while "ignore" found its in just 21.37 secs. Level_sum turns out to be a more informed heuristic than ignore_preconditions, since it actually takes into account how many steps away from each goal a node is, where "ignore" simply tallies how many goals are unsatisfied by the node, regardless of how far away those goals might be. While "ignore" is rapidly expanding nodes with the little information it has, "Level" is methodically studying the details, building a new PlanningGraph object for every single new node, so that it can get better information on how far away that node actually is from the goals. So the choice between the two heuristics comes down to how much computer space you have for your search—As long as you have the room, "ignore_preconditions" will get you the result much faster, but otherwise you're going to need the extra time and "level_sum".

Is a heuristic even needed at all? Well, that depends on what you need from a solution. If it doesn't need to be optimal, then no, you don't need a heuristic—just use depth-first search for a quick and ugly solution that doesn't use much memory. If it doesn't need to be optimal, but it needs to give you a ballpark idea of what an optimal solution might look like, without taking up too much time and memory, then without heuristics you can still use something like Greedy. Beyond that, if you need optimality but don't have huge resources, you can solve a small problem without heuristics, but for larger ones you'll be better off using A* with "ignore_preconditions", for quicker solutions where you have sufficient memory, or A* with "level_sum", when all you have is time to spare.