Please excuse the formatting. This PDF document was printed from the following README.md file available within my Github repo:

https://github.com/tommytracey/udacity/blob/master/ai-nano/projects/3-planning/results/README.md

Part 3: Written Analysis

- *Provide an optimal plan for Problems 1, 2, and 3.*
- *Compare and contrast non-heuristic search result metrics (optimality, time elapsed, number of node expansions) for Problems 1,2, and 3. Include breadth-first, depth-first, and at least one other uninformed non-heuristic search in your comparison; Your third choice of non-heuristic search may be skipped for Problem 3 if it takes longer than 10 minutes to run, but a note in this case should be included.*
- *Compare and contrast heuristic search result metrics using A* with the "ignore preconditions" and "level-sum" heuristics for Problems 1, 2, and 3.*
- *What was the best heuristic used in these problems? Was it better than non-heuristic search planning methods for all problems? Why or why not?*
- *Provide tables or other visual aids as needed for clarity in your discussion.*

Problem 1

Below are the initial goal and state for Problem 1. This problem is relatively simple as it only involves 2 cargos, 2 airplanes, and 2 airports (JFK, SFO).

```
Init(At(C1, SFO) ∧ At(C2, JFK)
        ∧ At(P1, SFO) ∧ At(P2, JFK)
        ∧ Cargo(C1) ∧ Cargo(C2)
        ∧ Plane(P1) ∧ Plane(P2)
```

```
        ∧ Airport(JFK) ∧ Airport(SFO))
Goal(At(C1, JFK) ∧ At(C2, SFO))
```

Here are the results from all the searches that I performed, including both uninformed and heuristic searches.

	Search Method	Optimal	Plan Length	Expansions	Goal Tests	New Nodes	Time (sec)
Problem 1	1. breadth_first_search	Yes	6	43	56	180	0.041
	2. breadth_first_tree_search	Yes	6	1458	1459	5960	1.220
	3. depth_first_graph_search	No	12	12	13	48	0.010
	4. depth_limited_search	No	50	101	271	414	0.112
	5. uniform_cost_search	Yes	6	55	57	224	0.043
	6. recursive_best_first_search h_1	Yes	6	4229	4230	17029	3.603
	7. greedy_best_first_graph_search h_1	**Yes**	**6**	**7**	**9**	**28**	**0.009**
	8. astar_search h_1	Yes	6	55	57	224	0.049
	9. astar_search h_ignore_preconditions	Yes	6	41	43	170	0.036
	10. astar_search h_pg_levelsum	Yes	6	11	13	50	1.138
	averages		11	601	621	2433	0.626

Here is the optimal plan (6 steps total), and presumably the two sets of actions could be done in parallel.

```
Load(C1, P1, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)
```

```
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
```

Given the results above, the `greedy_best_first_graph_search h_1` method is the best choice. While there are several search methods which produce an optimal plan, the `greedy_best_first` does it more efficiently than the others. As highlighted during the lectures, The Greedy Best First approach works by expanding nodes closest to the goal first and therefore minimizes the number of fluents it needs to process.

The results also indicate that for simple problems like this (i.e. ones with a small search space), non-heuristic methods like `breadth_first_search` and `uniform_cost_search` are viable options. They provide reasonable efficiency and yield optimal paths without the added complexity of a heuristic.

Among the heuristic methods, it's worth noting that all of them produced optimal paths, with `A*_ignore_preconditions` being the most efficient. But, given the small search space, it's hard to tell which of the heuristic search methods will perform best as the Air Cargo plans increase in complexity.

Problem 2

Below are the initial goal and state for Problem 2. This problem is slightly more complex as it now involves 3 cargos, 3 airplanes, and 3 airports (ATL, JFK, SFO).

```
Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(C3, ATL)
        ∧ At(P1, SFO) ∧ At(P2, JFK) ∧ At(P3, ATL)
        ∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3)
        ∧ Plane(P1) ∧ Plane(P2) ∧ Plane(P3)
        ∧ Airport(JFK) ∧ Airport(SFO) ∧ Airport(ATL))
Goal(At(C1, JFK) ∧ At(C2, SFO) ∧ At(C3, SFO))
```

Here are the results from all the searches that I performed, including both uninformed and heuristic searches.

	Search Method	Optimal	Plan Length	Expansions	Goal Tests	New Nodes	Time (sec)
Problem 2	1. breadth_first_search	Yes	9	3343	4609	30509	16.799
	2. breadth_first_tree_search		*timeout*				
	3. depth_first_graph_search	No	575	582	583	5211	3.728
	4. depth_limited_search		*timeout*				
	5. uniform_cost_search	Yes	9	4761	4763	43206	15.701
	6. recursive_best_first_search h_1		*timeout*				
	7. greedy_best_first_graph_search h_1	**Yes**	**9**	**550**	**552**	**4950**	**1.733**
	8. astar_search h_1	Yes	9	4761	4763	43206	15.490
	9. astar_search h_ignore_preconditions	Yes	9	1450	1452	13303	4.681
	10. astar_search h_pg_levelsum		*timeout*				
	averages		103	2575	2787	23398	9.689

Here is the optimal plan (9 steps total), and presumably the three sets of actions could be done in parallel.

```
Load(C3, P3, ATL)
Fly(P3, ATL, SFO)
Unload(C3, P3, SFO)
```

```
Load(C2, P2, JFK)
Fly(P2, JFK, SFO)
Unload(C2, P2, SFO)
```

```
Load(C1, P1, SFO)
Fly(P1, SFO, JFK)
Unload(C1, P1, JFK)
```

Once again, we see that `greedy_best_first` is the recommeded algorithm since it produces an optimal path with the greatest efficiency. What's more, the gain in efficiency is now much more noticable compared to `breadth_first_search` and `uniform_cost_search`. The added complexity of Problem 2 (adding 1 cargo, 1 plane, and 1 airport) makes these non-heuristic methods less viable as evidenced by longer execution time and higher number of nodes. In fact, the added complexity of this problem caused four of the other search algorithms to timeout, including both heuristic and non-heuristic methods. This demonstrates how important the size of the search space is when choosing and configuring a planning algorithm. Specifically, at this stage it's now clear that the `A*_levelsum` algorithm is too inefficent to be useful moving forward.

Problem 3

Below are the initial goal and state for Problem 3. This problem is the most complex since it now involves 4 cargos and 4 airports (ATL, JFK, ORD, SFO), but only 2 airplanes are available to haul everything.

```
Init(At(C1, SFO) ∧ At(C2, JFK) ∧ At(C3, ATL) ∧ At(C4, ORD)
        ∧ At(P1, SFO) ∧ At(P2, JFK)
        ∧ Cargo(C1) ∧ Cargo(C2) ∧ Cargo(C3) ∧ Cargo(C4)
        ∧ Plane(P1) ∧ Plane(P2)
        ∧ Airport(JFK) ∧ Airport(SFO) ∧ Airport(ATL) ∧ Airport(ORD))
Goal(At(C1, JFK) ∧ At(C3, JFK) ∧ At(C2, SFO) ∧ At(C4, SFO))
```

Here are the results from all the searches that I performed, including both uninformed and heuristic searches. Although, note that some of the searches did not finish in the allotted 10-minute timeframe.

	Search Method	Optimal	Plan Length	Expansions	Goal Tests	New Nodes	Time (sec)
Problem 3	1. breadth_first_search	Yes	12	14491	17947	128184	131.447
	2. breadth_first_tree_search		*timeout*				
	3. depth_first_graph_search	No	1878	1948	1949	16253	26.679
	4. depth_limited_search		*timeout*				
	5. uniform_cost_search	Yes	12	17783	17785	155920	64.016
	6. recursive_best_first_search h_1		*timeout*				
	7. greedy_best_first_graph_search h_1	No	22	4031	4033	35794	15.866
	8. astar_search h_1	Yes	12	17783	17785	155920	62.893
	9. astar_search h_ignore_preconditions	**Yes**	**12**	**5003**	**5005**	**44586**	**19.119**
	10. astar_search h_pg_levelsum		*timeout*				
	averages		325	10173	10751	89443	53.337

Here is the optimal plan (12 steps total), and presumably the two sets of actions could be done in parallel.

```
Load(C1, P1, SFO)
Fly(P1, SFO, ATL)
Load(C3, P1, ATL)
Fly(P1, ATL, JFK)
Unload(C3, P1, JFK)
Unload(C1, P1, JFK)
```

```
Load(C2, P2, JFK)
Fly(P2, JFK, ORD)
Load(C4, P2, ORD)
Fly(P2, ORD, SFO)
Unload(C4, P2, SFO)
Unload(C2, P2, SFO)
```

Once again, we see that the larger search space causes the same four algorithms to timeout: `breadth_first_tree_search`, `depth_limited_search`, `recursive_best_first_search_h_1`, and `A*_h_pg_levelsum`. And, it appears that the added complexity now prevents the `greedy_best_first` method from producing an optimal path.

By this point we see an interesting pattern emerge: the `uniform_cost_search` and `A*_search_h_1` algorithms perform almost exactly the same across all three problems. This makes sense given the lecture discussion on how both of these methods are based on cost. Uniform Cost Search focuses on optimal path cost and will always find an optimal solution unless it gets stuck in an infinite loop. A* heuristic search expands the node of the lowest f-cost, adding nodes concentrically from the start node. However, as the search space expands, neither of these algorithms is very efficient.

With all that said, `A*_ignore_preconditions` is now the recommended algorithm since it does yield an optimal path with the greatest efficiency. And, overall, we can see that using informed search algorithms with well-designed heuristics is the best strategy.