Heuristic Analysis (Planning Search) :

Progression planning problems can be solved with graph searches such as breadth-first, depth-first, and A*, where the nodes of the graph are "states" and edges are "actions". A "state" is the logical conjunction of all boolean ground "fluents", or state variables, that are possible for the problem using Propositional Logic. For example, we might have a problem to plan the transport of one cargo, C1, on a single available plane, P1, from one airport to another, SFO to JFK

- Run uninformed planning searches for `air_cargo_p1`, `air_cargo_p2`, and `air_cargo_p3`; provide metrics on number of node expansions required, number of goal tests, time elapsed, and optimality of solution for each search algorithm. Include the result of at least three of these searches, including breadth-first and depth-first, in your write-up (`breadth_first_search` and `depth_first_graph_search`).
- If depth-first takes longer than 10 minutes for Problem 3 on your system, stop the search and provide this information in your report.
- Use the `run_search` script for your data collection: from the command line type `python run_search.py -h` to learn more

Problem	Algorithm	Expansions	Goal tests	New nodes	Time elapsed
P1	BFS	43	56	180	0.0946
P1	DFGS	12	13	48	0.01445
P1	UCS	55	57	224	0.0623
P2	BFS	3401	4672	31049	23.06
P2	DFGS	350	351	3145	2.27
P2	UCS	4761	4763	43206	17.684
P3	BFS	14491	17947	128184	154
P3	DFGS	1948	1949	16253	31.52
P3	UCS	17783	17785	155920	76.0043

Why a Planning Graph?

The planning graph is somewhat complex, but is useful in planning because it is a polynomial-size approximation of the exponential tree that represents all possible paths. The planning graph can be used to provide automated admissible heuristics for any domain. It can also be used as the first step in implementing GRAPHPLAN, a direct planning algorithm that you may wish to learn more about on your own (but we will not address it here).

- Run A* planning searches using the heuristics you have implemented on `air_cargo_p1`, `air_cargo_p2` and `air_cargo_p3`. Provide metrics on number of node expansions required, number of goal tests, time elapsed, and optimality of solution for each search algorithm and include the results in your report.

Problem	Algorithm	Expansions	Goal tests	New nodes	Time elapsed
P1	A_h1	55	57	224	0.0856
P1	A_h_ignore_pre	41	43	170	0.0482
P1	A_pg_level	11	13	50	1.115
P2	A_h1	4761	4763	43206	18.12
P2	A_h_ignore_pre	1450	1452	13303	5.79
P2	A_pg_level	86	88	841	76.09
P3	A_h1	17783	17785	155920	76.0043
P3	A_h_ignore_pre	5003	5005	44586	21.20
P3	A_pg_level	311	313	2863	364.19

Part 3: Written Analysis

TODO: Include the following in your 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.

For this analysis I used the 3 algorithms BFS,DFGS and UCS.

For the execution time and number of nodes expanded DFGS is the best choice.

For the optimality (plan length, execution time) both BFS and UCS are better in plan length than DFGSS But USC is better than BFS in time of execution

Problem	Algorithm	Plan length	Expansions	Goal tests	New nodes	Time elapsed
P1	BFS	6	43	56	180	0.0946

P1	DFGS	12	12	13	48	0.01445
P1	UCS	12	55	57	224	0.0623
P2	BFS	9	3401	4672	31049	23.06
P2	DFGS	346	350	351	3145	2.27
P2	UCS	9	4761	4763	43206	17.684
P3	BFS	12	14491	17947	128184	154
P3	DFGS	1878	1948	1949	16253	31.52
P3	UCS	12	17783	17785	155920	76.0043

- Compare and contrast heuristic search result metrics using A* with the "ignore preconditions" and "level-sum" heuristics for Problems 1, 2, and 3.

  For this analysis I used the 3 below heuristics for the astar search algorithm:

  - h_l : heuristic always returns l
  - h_ignore_preconditions: by ignoring the preconditions required for an action to be executed , estimate the minimum number of actions that must be carried in order to satisfy all goal conditions
  - h_pg_levelsum: use a planning graph representation of the problem state space to estimate the sum of all actions that must be carried out from the current state in order to satisfy each individual goal condition

in term of execution time a_h_ignore_precondition is the best one.

In term of nodes expansions, the heuristic h_pg_levelsum is the best.
In term of optimization (plan length and execution time), the 3 algorithms have similar plan length but a_h_ignore_pre come first in term of time execution

Problem	Algorithm	Plan Length	Expansions	Goal tests	New nodes	Time elapsed
P1	A_h_l	6	55	57	224	0.0856
P1	A_h_ignore_pre	6	41	43	170	0.0482
P1	A_pg_level	6	11	13	50	1.115
P2	A_h_l	9	4761	4763	43206	18.12

P2	A_h_ignore_pre	9	1450	1452	13303	5.79
P2	A_pg_level	9	86	88	841	76.09
P3	A_h_l	12	17783	17785	155920	76.0043
P3	A_h_ignore_pre	12	5003	5005	44586	21.20
P3	A_pg_level	12	311	313	2863	364.19

- 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?

Taking the execution time as a criteria , I noticed that for the heuristic planning methods the a_h_ignore_pre was the best one in time execution and for the non heuristics planning methods I noticed that DFGS was the best one in time execution between the others non heuristics algorithms used in this comparaison. By comparing the h_ignore_pre and DFGS (time execution and plan length) I noticed from the above tables that a_h_ignore_pre is better than DFGS in time execution (21.20 vs 31.52) and in plan length (max 12 vs max 1878)