Planning Problems

For this project three planning problems were given with the following action schema:

```
Action(Load(c, p, a),
       PRECOND: At(c, a) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
       EFFECT: ¬ At(c, a) ∧ In(c, p))
Action(Unload(c, p, a),
       PRECOND: In(c, p) ∧ At(p, a) ∧ Cargo(c) ∧ Plane(p) ∧ Airport(a)
       EFFECT: At(c, a) ∧ ¬ In(c, p))
Action(Fly(p, from, to),
       PRECOND: At(p, from) ∧ Plane(p) ∧ Airport(from) ∧ Airport(to)
       EFFECT: ¬ At(p, from) ∧ At(p, to))
```

Problem 1:

```
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))
```

Problem 2:

```
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))
```

Problem 3:

```
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))
```

Optimal Plan Lengths

Problem 1 can be solved in ***six (6)*** steps, as follows:

```
Load(C1, P1, SFO) // load up C1 into P1 at SFO
Load(C2, P2, JFK) // load up C2 into P2 at JFK
Fly(P1, SFO, JFK) // move things were they need to go
Fly(P2, JFK, SFO)
Unload(C1, P1, JFK) // unload C1 from P1 at JFK (subgoal 1)
Unload(C2, P2, SFO) // unload C2 from P2 at SFO (subgoal 2)
```

Problem 2 can be solved in ***nine (9)*** steps, as follows:

```
Load(C1, P1, SFO)
Load(C2, P2, JFK)
Load(C3, P3, ATL)
Fly(P1, SFO, JFK)
Fly(P2, JFK, SFO)
Fly(P3, ATL, SFO)
Unload(C1, P1, JFK)
Unload(C2, P2, SFO)
Unload(C3, P3, SFO)
```

Problem 3 can be solved in ***twelve (12)*** steps. This one is special in that both planes need to pick-up 2 cargos before flying to their final destinations as follows:

```
Load(C1, P1, SFO) // load up cargo where an airplane (P1) already is
Load(C2, P2, JFK) // load up cargo where an airplane already is
Fly(P1, SFO, ATL) // move P1 to ATL
Fly(P2, JFK, ORD) // move P2 to ORD
Load(C3, P1, ATL) // load up more cargo into P1
Load(C4, P2, ORD) // load up more cargo into P2
Fly(P1, ATL, JFK) // fly P1 to final destination (JFK)
Fly(P2, ORD, SFO) // fly P2 to final destination (SFO)
Unload(C1, P1, JFK) // unload C1 at JFK (subgoal 1)
Unload(C3, P1, JFK) // unload C3 at JFK (subgoal 2)
Unload(C2, P2, SFO) // unload C2 at SFO (subgoal 3)
Unload(C4, P2, SFO) // unload C4 at SFO (subgoal 4)
```

Non-heuristic Search Metrics

Below are tables tabulating the results of four non-heuristic search strategies for all three problems. Of the four search strategies chosen, only breadth_first_search and uniform_cost_search gave optimal plans across all three problems.
The fastest and most efficient memory wise is depth_first_graph_search. The downside of depth_first_graph_search is that it will typically give us the longest path out of the four.
Given only these metrics to use, breadth_first_search is the best strategy to use because it is complete, and uses less time and memory (node expansions) than uniform_cost_search. breadth_first_search may fail to be the best out of these given strategies when the branching factor of the problem is high, as its memory usage will also be high.

Problem 1

Search Strategy	Optimal	Path Length	Execution Time (s)	Node Expansions
breadth_first_search	Y	6	0.0312	43
depth_first_graph_search	N	20	0.0137	21
uniform_cost_search	Y	6	0.0718	55
greedy_best_first	Y	6	0.00509	7

Problem 2

Search Strategy	Optimal	Path Length	Execution Time (s)	Node Expansions
breadth_first_search	Y	9	14.142	3346
depth_first_graph_search	N	105	0.353	107
uniform_cost_search	Y	9	12.501	4853
greedy_best_first	N	21	2.484	998

Problem 3

Search Strategy	Optimal	Path Length	Execution Time (s)	Node Expansions
breadth_first_search	Y	12	111.977	14663
depth_first_graph_search	N	392	1.888	408
uniform_cost_search	Y	12	55.299	18235
greedy_best_first	N	22	16.919	5614

Heuristic Search Metrics

Below are tables tabulating the results of three variants of A* heuristic search strategies for all three problems.

Of these three variants, A* w/ h_ignore_preconditions is fastest, but A* w/ h_pg_levelsum uses the least memory.

Given this information, A* w/ h_ignore_precoditions is the best of these strategies to use when time is critical but memory is not, and A* w/ h_pg_levelsum is the best to use when memory is critical but time is not.

Problem 1

Search Strategy	Optimal	Path Length	Execution Time (s)	Node Expansions
A* w/ h_1	Y	6	0.039	55
A* w/ h_ignore_preconditions	Y	6	0.0506	41
A* w/ h_pg_levelsum	Y	6	0.976	11

Problem 2

Search Strategy	Optimal	Path Length	Execution Time (s)	Node Expansions
A* w/ h_1	Y	9	12.439	4853
A* w/ h_ignore_preconditions	Y	9	4.0587	1450
A* w/ h_pg_levelsum	Y	9	205.0559	86

Problem 3

Search Strategy	Optimal	Path Length	Execution Time (s)	Node Expansions
A* w/ h_1	Y	12	54.311	18235
A* w/ h_ignore_preconditions	Y	12	15.259	5040
A* w/ h_pg_levelsum	Y	12	1032.116	325

Non-Heuristic versus Heuristic Search Metrics

In comparing Non-Heuristic and Heuritstic Search Metircs I ran the best strategy from each (breadth_first_search and A* w/ h_ignore_preconditions) through the program one more time. The results of these runs are tabulated in the tables below in order to more readily demonstrate the differences.

From the results we can say that of the two, A* w/ h_ignore_preconditions is better both memory-wise and time-wise for all three problems, which demonstrates it to be the best strategy to use for these problems.

Problem 1

Search Strategy	Execution Time (s)	Node Expansions
breadth_first_search	0.0311	43
A* w/ h_ignore_preconditions	0.396	41

Problem 2

Search Strategy	Execution Time (s)	Node Expansions
breadth_first_search	13.913	3346
A* w/ h_ignore_preconditions	3.758	1450

Problem 3

Search Strategy	Execution Time (s)	Node Expansions
breadth_first_search	102.45	14663
A* w/ h_ignore_preconditions	15.347	5040

Conclusion

The results obtained demonstrate the benefits of heuristic search strategies. Heuristic search strategies perform better both in terms of time and memory usage, but the actual best strategy may depend on the problem and constraints (time or memory).