Here are optimal sequences of actions found for:

Problem 1

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

Problem 2

```
Load(C2, P2, JFK)
Load(C1, P1, SFO)
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)
```

Problem 3

```
Load(C2, P2, JFK)
Load(C1, P1, SFO)
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)
```

In the table below we compare the results of 3 uninformed planning algorithms (breadth-first, depth-first and uniform-cost searches) and 2 informed algorithms (A* ignore_preconditions and A* level sum) for the 3 problems.

PROBLEM 1					
	Breadth-first search	**Depth-first search**	**Uniform-cost search**	**A* ignore preconditions**	**A* level sum**
Expansions	43	12	55	41	11
Goal Tests	56	13	57	43	13
New Nodes	180	48	224	170	50
Plan Length	6	12	6	6	6
Time to run (s)	0.025500824	0.006925231	0.030145794	0.032513013	0.549799993

PROBLEM 2					
	Breadth-first search	Depth-first search	Uniform-cost search	A* ignore preconditions	A* level sum
Expansions	3343	1669	4852	1450	86
Goal Tests	4609	1670	4854	1452	88
New Nodes	30509	14863	44030	13303	841
Plan Length	9	1444	9	9	9
Time to run (s)	12.19837186	12.59970731	10.94149777	3.672941785	50.21670222

PROBLEM 3					
	Breadth-first search	Depth-first search	Uniform-cost search	A* ignore preconditions	A* level sum
Expansions	14663	592	18235	5040	318
Goal Tests	18098	593	18237	5042	320
New Nodes	129631	4927	159716	44944	2934
Plan Length	12	571	12	12	12
Time to run (s)	88.12566323	2.582764652	45.75844706	14.71490132	250.6627293

Optimality

Results

As we see, depth-first search is the only algorithm not to find an optimal plan. All the other plans find optimal plans of 6, 9 and 12 steps for Problems 1, 2 and 3 respectively.

Explanation

Breadth-first search explores the shallowest nodes first. This will result in finding an optimal plan, as long as all actions have the same cost, like here (AIMA book, 3d edition, 3.4.1).

We see that depth-first search is non-optimal and always returns plans that are much longer than for the other algorithms. This is because it explores the deepest node first, and thus might explore entire subtrees of the graph before finding a goal node.

Since all actions have the same cost, uniform-cost search behaves similarly to breadth-first search here and returns an optimal plan too.

A* algorithms minimize both the cost of the action to reach a particular node in the graph, as well as the cost to get from this node to the goal as determined by a specific heuristic function. In our problem, the cost of each action is equal to 1, and the heuristic chosen are either *ignore preconditions* or *level sum.*

A* *ignore preconditions* is optimal because the heuristic function is admissible, i.e., it does not overestimate. This is because ignoring preconditions makes this a relaxed version of the problem where any action can be accomplished no matter what its preconditions are (AIMA book, 3d edition, 10.2.3).

A* with *level sum* assumes that each subgoal is independent. Based on that, it uses the graph plan to estimate the number of levels needed to reach each subgoal and uses the sum of those numbers as a heuristic function. This is admissible when the subgoals are independent (AIMA book, 3d edition, 10.3.1), and we see here that this algorithm returns an optimal plan for each problem.

<u>**Number of expanded nodes**</u>

<u>Results</u>

We observe that breadth-first search and uniform-cost search expand a comparable number of nodes. Depth-first search expands significantly less nodes. Both A* algorithms expand significantly less nodes than breadth-first search and uniform-cost search, and A* with *level-sum* expands the least number of nodes for Problems 2 and 3.

<u>Explanation</u>

Breadth-first search and uniform-cost search expands comparable numbers of nodes since the cost of all actions is equal.

The difference of expanded nodes between depth-first search and breadth-first/uniform-cost searches comes from the position of the goal node in the graph. If the goal node is deep, then depth-first search will be much more efficient since it has a higher chance to reach it first compared to breadth-first search which has to expand all nodes from previous depths before reaching the goal. Alternatively, if the goal node is in early depths, depth-first search will probably have to expand much more nodes than bread-first search, since it might have to expand entire subtrees of the graph. It would thus seem here that the goal nodes are located deep in the graph.

The fact that A* algorithms expand significantly less nodes than breadth-first search and uniform-cost search is explained by the fact that with A*, the search is informed, i.e., the use of heuristic functions facilitates the search. In addition, *level-sum* expands significantly less nodes, as the cost that it returns is larger than the cost returned by *ignore preconditions* and is thus closer to the real cost, which makes it a better heuristic (Udacity lesson 11.39).

<u>**Time elapsed**</u>

<u>Results</u>

For this particular problem, depth-first search finds a solution much faster than all other algorithms. A* *ignore preconditions* comes second, followed by uniform-cost search, breadth-first search and finally A* *level-sum*.

<u>Explanation</u>

One factor that affects running time is the number of nodes that are being explored before finding a goal node. This explains why depth-first search and A* *ignore preconditions* are faster overall.

However, implementation of the algorithms can also make a difference, for example when comparing breadth-first and uniform-cost searches. Here, it seems that the implementation of uniform-cost search is more efficient since the two algorithms should in principle take more or less the same time to run according to the number of nodes that they explore.

Finally, A* *level-sum* takes significantly longer than all other algorithms. This may be explained by the fact that it needs to build the entire graph plan in order to be able to return the value of the heuristic function.