**1)**

**Sudoku:** Fully, Single, deterministic, sequential, static, discrete

**Checkers with a Clock:** Fully, Multi, Deterministic, Sequential, Semi, Discrete

**Black Jack:** Partially, Multi, Stochastic, Sequential, Static, Discrete.

**Risk:** Fully, Multi, Stochastic, Sequential, Static, Discrete

**Walking:** Partially, Multi, Stochastic, Sequential, Dynamic, Continuous

**Flying a plane (solo, no other planes in the air):** Partially, Single, Stochastic, Sequential, Dynamic, Continuous

**Generating Top Sales Count for Movies**: Fully, Single, Deterministic, Episodic, Semi, Continuous

**Search Engine:** Partially, Single, Stochastic, Episodic, Dynamic, Continuous

**Navigating through a Cave:** Partially, Single, Stochastic, Sequential, Dynamic, Continuous

Partially, Multi, Stochastic, Sequential, Dynamic, Discrete

**RTS AIs (like Age of Empires):** Partially (can’t see opponent, hopefully), Multiple people playing, Stochastic, Sequential, Dynamic, Discrete (all players have a discrete amount of units, in discrete places).

**2)** Let’s do a proof by contradiction: Let’s assume Breadth-First Search is not optimal when the edgecosts are a nondecreasing function of depth. This means that for some path that BFS found, there was some better path that BFS didn’t choose. By better, I mean in terms of total edge cost for the path. Let’s assume that both Optimal and BFS paths were similar to some point *k* (this can be assumed, since it can be 0, right at the start). This means that at *k*, BFS expanded on a node *i*, while the optimal path expanded chose some other unexplored node, *i\_optimal.* In order for *i\_optimal* to be better, it has to be a lower cost. In order for it to be a lower cost, it has to be on a lower depth than *i*, since edge cost is a nondecreasing function of depth. BFS would have already expanded upon this node *i\_optimal* because it checks out all lower depths before going to a deeper depth (inherent property of a BFS). This means that *i\_optimal* was already expanded, which contradicts our previous statement that it wasn’t expanded.

Q.E.D, BFS is optimal when the edge cost is a nondecreasing function of depth.

**REPORT!**

**8 - Puzzle**

6) Generally speaking, the branching factor for 8-puzzle is on average, 3. At most, there is 4 directions that the blank space can move to: left, right, up and down. This only really happens when the blank state is in the middle. Otherwise, it is 3 or 2, but generally 3. There is only 9! permutations of tile sequences. Without any heuristic to guide the problem, uninformed searches are going to have huge problems, since it is 4^(depth). So by about 10 moves, you are sitting at a fringe of one million nodes expanded, but barely seeing anything with 10 moves.

7)

**Depth-first-search:** Doesn’t finish due to the depth it goes to, so it is neither complete nor optimal. Without a heuristic, it can get stuck in an infinite loop really easily, which is apparently what has happened here. Though, it only generated roughly 100 nodes, which is nearly the same as a guided heuristic, which is a plus, compared with Breadth first search which is in the tens of thousands.

**Breadth-first-search:** Due to the uniformity of the edgecosts, breadth first is both complete and optimal. However, a problem with Breadth first search, is that it generates way too many nodes with such a small problem. In table 4, you can see it generates 33,000 nodes. Doing something like 15-puzzle would obliterate any machine it is on.

**Uniform-cost-search**: Uniform cost is great in 8-puzzle because edge costs are all 1, just like with breadth first search. It will produce an optimal solution and is complete.

**Iterative-deepening-search**: In this case, iterative deepening is a mixed bag. It generates so many nodes, but not all at the same time. It seems to be the best uninformed search for 8-puzzle. While breadth first search has a huge fringe it needs to keep in memory, iterative deepening only had a max fringe of 20. It did a lot of redundant work though, as can be seen by the nodes generated (18,000). It did complete the problem, unlike depth-first search and it was an optimal solution.

**Best-first-search**: Best first search doesn’t really work for 8-puzzle. It fails every time because at first, all the h-values could be the same. As an example, moving two tiles might make them both still out of place, so it has the same h-value, which creates a possible infinite loop. In fact, best first search fails in every heuristic, so it is not a great informed search for 8-puzzle.

**A\* search**: A star is great for 8-puzzle. With a well thought out heuristic like manhatten distance, you get less nodes generated. Of course, the worst the heuristic, the slightly worst it gets. It is no breadth-first search in terms of amount of nodes on the fringe, but with a larger puzzle, and/or worst heuristic, it could get big--fast.

**IDA\* search**: I would say, for 8-puzzle, this algorithm is the best. It generates about 5-10% more nodes than A\*, but the fringe is almost always 80% less than A\*. So for tiles misplaced, A\* came out with 147 and 214 for max fringe and nodes generated respectively. for IDA\*, it was 11 and 228. Easily worth the trade off.

8)

**Tile Displacement**: This heuristic is admissible. It is a relaxed heuristic that doesn’t take into account moving between borders, and acts as if you can just “jump” to another tile. As such, it will be less than Cost because of this.

**Manhattan Distance:** This heuristic is admissible. The reason is, at the most reasonable case, the manhatten distance is 1 if the tile is right next to the space. If there is a non-blank tile in that space, Manhatten distance doesn’t take into account the cost to get around it. This is the case with any tile out of place.

**X-Y finding**: This heuristic is admissible because it is an even more relaxed version of the Manhatten Distance problem. It doesn’t take into account the distance from a column, it only notes that it is not in the right column. Same for the rows.

9) In terms of informed-ness, The order is as follows: Manhatten Distance > X-Y finding > Tiles out of Place. It can be seen in every step of the goal states, seen in Table 3. A more terse reason is because each heuristic is a more relaxed form of the previous. Column-Row displacement doesn’t take into account the amount of tiles it has to move in order to get to the column / row that is out of place, which makes it less informed than Manhatten Distance. While the column-row displacement heuristic would have to make a possible two jumps (one for column, then another for row), Tiles out of Place heuristic is even more relaxed. It assumes any tile can jump to any location on the puzzle, making it the least informed of the group.

**Romanian Route Finding**

6) The possible branching factor for RRF is K, where K is the amount of cities there are. At least in Romania, the graph seems pretty sparse, so the branching factor will not be as large as 8-puzzle, which should keep state-space relatively small. Even if the branching factor was large, it would make it that much easier to get to the goal for algorithms that rely on breadth and not depth (BFS, iterative and IDA).

7)

**Depth-first-search**: Just like 8-puzzle, DFS gets into an infinite loop and doesn’t complete before reaching an “infinite” depth. So it is not right first this problem either. One of the things it does right is produce a small fringe, but so do others in this problem due to the sparseness of the graph.  
**Breadth-first-search**: This algorithm has a very low cost associated with it since the branching factor is so low to get to the goal state. The amount it produces is on par with some of the heuristically driven searches. The main problem is that it is not optimal. Since edge costs are not uniform, it will not find an optimal solution. It does this in the runs I performed, producing a GVal of 450 when the optimal is 418

**Uniform-cost-search**: Since uniform cost always chooses the least cost node to expand upon, it will find an optimal solution and is complete. The problem is that it generates the most nodes out of all the algorithms, due to the fact that it might expand greedily into an area that isn’t wise to choose in the end.

**Iterative-deepening-search:** Iterative deepening produces one of the lowest fringes, which is nice but is not optimal. It suffers the same fate as breadth-first search, by possibly not expanding upon a node that is optimal. It does this in the runs I performed, producing a GVal of 450 when the optimal is 418

**Best-first-search**: Best first search is very light in nodes generated to find the goal, since it relied only on straight line paths. The problem is that it was not optimal. It chose a GVal of 450 as well. The reason for this is because a straight line doesn’t necessarily let on to real flight paths.

**A\* search**: A\* search is optimal and complete. It barely has any fringe nor does it generate a lot of nodes (only 17 nodes generated). It does the least redundant work out of all the search algorithms, and would be recommended since the search space is so sparse for these types of problems.

**IDA\* search**: Also optimal, if it finds a solution. Luckily, IDA never reaches the upper inifinite bound of deepening in the problem, though I doubt it would. It does a lot of redundant work compared to A\*, which can be seen when the nodes generated are about 300% more (17 by A\* compared to 59 by IDA\*). It did keep the fringe at the lowest of any other search algorithm, which is nice for dense graphs.

8)

**Straight Line Distance**: Is admissible because the shortest distance is always a straight line. So either the cost will be the straight line itself, or more jagged lines that add to the cost.

**Jump Straight Line Distance**: Is admissible because it is an even more relaxed problem of the straight line, saying that if my neighbor has a straight line that is less than my straight line, then choose that.

9) Straight Line is always more informed than Jump Straight Line since it is the min of straight line or the neighbors of that current node’s straight line. So at all points it is either equal to straight line, or less, making it less informed.