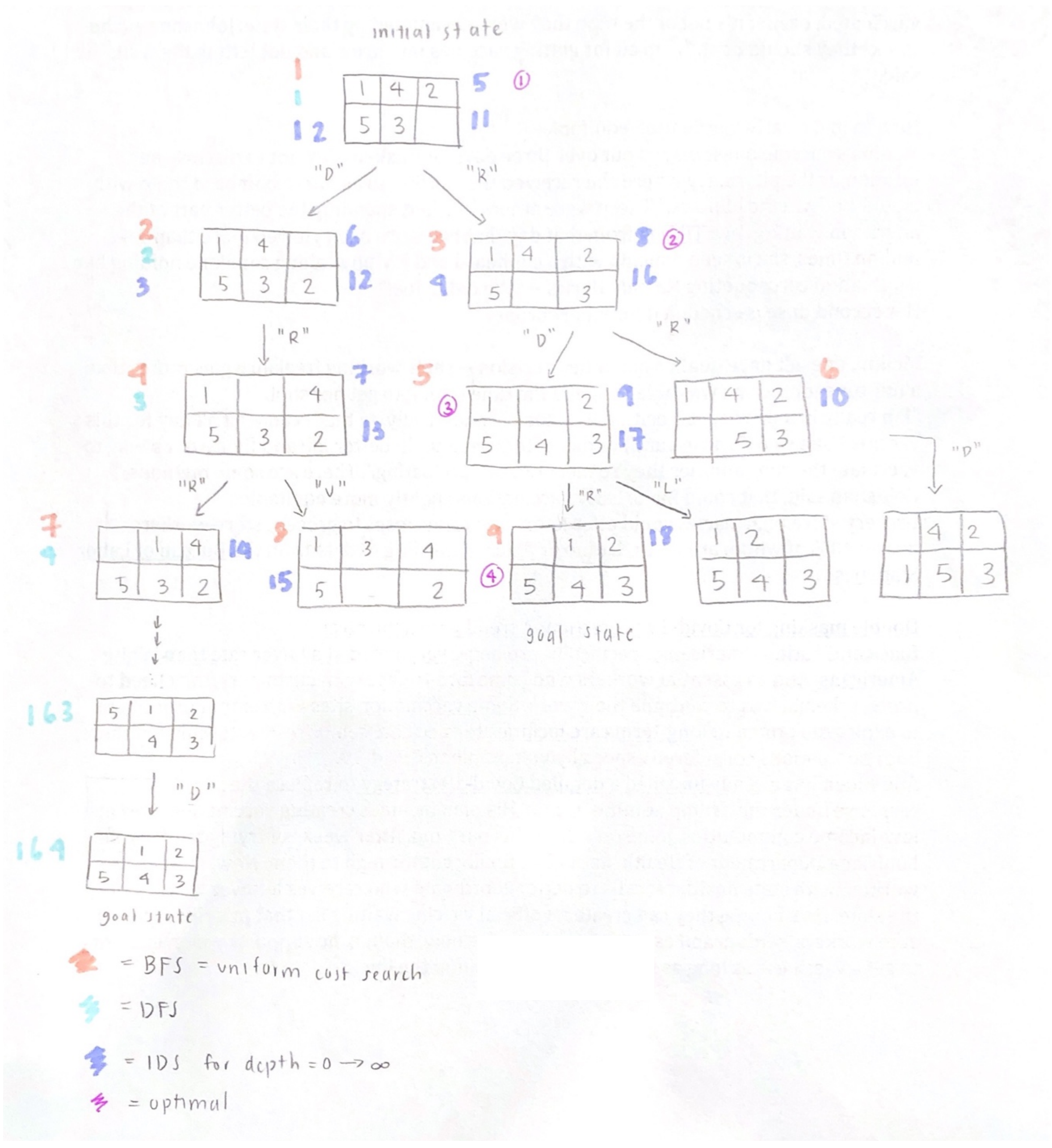
COMP424 Assignment 1

Anne-Sophie Fratzscher
ID: 260705446

1. Six-Puzzle

a. Below are the solution paths found by breadth-first search (BFS), uniform cost search, depth first search (DFS), and iterative deepening search (IDS). Additionally, the optimal solution is also shown. Paths can be found by going to the next number i.e. 1->2->3… until the goal state is reached, with 1 representing the initial state. Note for uniform cost search that visited children at a specific depth were added based on when they were visited, NOT which tile was moved when transitioning from parent to child. Therefore, uniform cost search has the same solution path as BFS. The code can be found in Q1.py



b. The Manhattan distance for a given tile is the number of moves necessary for that tile to reach its location in the goal state. Therefore, the total Manhattan distance is the sum of Manhattan distances for each tile to its goal position.

When calculating the Manhattan distance, one is assuming that the cost of each move is equal to 1. Additionally, the Manhattan distance assumes NO other pieces are in the way. Therefore, the Manhattan distance would always be less than the cost of a 6-puzzle (or 8-puzzle, as stated in class) with cost of 1 for each move, so Manhattan distance is an admissible heuristic for A* search. For the new variant in this question, it was added that the cost of a transition is equal to the number of the moved piece. This cost would be even larger than the cost for a traditional 6-puzzle with cost of 1 for each transition, since the tile numbers range from 1 to 6 (and we do not have negative tile numbers). Therefore, Manhattan distance is also an admissible heuristic for A* search in this new variant.

c. Let h_{new} be the Manhattan distance but with cost of each move equalling the number of the tile being moved. For example, for the '4' tile in our given example, the heuristic for '4' would be 4 * 2 = 8 (as opposed to 1 * 2 for the normal Manhattan distance). This heuristic would dominate h_2 because it multiplies h_2 by the number of the tile, and the tiles have a range of 1 to 6, so $h_{new}(n) \geq h_2(n)$ for all n. Therefore, h_{new} would dominate h_2.

Additionally, we need to check that h_{new} is an admissible heuristic. Similarly to how Manhattan distance is an admissible heuristic for 6-puzzle with cost of transition = 1, h_{new} is admissible because it assumes that no pieces are in the way (so the number of moves will always be less than or equal), so, since the transition cost is equal in the heuristic and the problem, the Manhattan distance with cost equal to the tile # will always be less or equal to the cost for the A* search variant.

2.

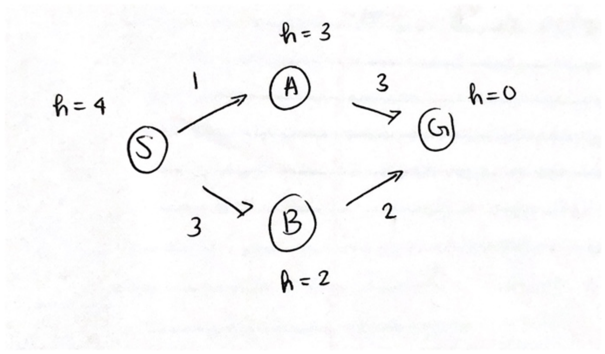
a. Let the state space be a path, meaning each parent has exactly one child. Let this tree have n nodes. Depth-first search will visit each node once, so it will have O(n) time. Iterative deepening search, on the other hand, will first visit the root, then the root + its child, and so on. This would lead to visiting $\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n = \frac{1}{2}n(n+1) = O(n^2)$. Therefore, in the case where the state space is a path, iterative deepening search will perform worse than depth-first search.[1] ■

b. TRUE. Breadth-first search (BFS) is the special case of uniform-cost search where all actions have the same cost (also called unit cost). We will prove this by showing that the unit cost version of uniform-cost search is equivalent to BFS. Let us assume the cost of a transition = i. We start at the start state. When we pop the start state, we add the children (C_1, C_2, … C_n) of the start state to the priority queue. We would not prioritize any of them since they all have the same cost = i. Since C_1, C_2, … C_n all have the same cost, we would then add the children of C_1 to the queue (call them $C_{1,1}$, $C_{1,2}$, … $C_{1,m}$). However, since $C_{1,1}$, $C_{1,2}$, … $C_{1,m}$ have cost = i+i (vs cost = i for C_2, … C_n), we would first visit the children of C_2, then C_3, … then C_n, before visiting the children of $C_{1,1}$. This is the same as running BFS on the graph. ■

[1] Definition of path for Q2a from https://math.stackexchange.com/questions/865381/how-to-call-a-tree-with-a-single-branch

c. TRUE. For A* search, the estimate of cost of current path, or f(n), equals the cost of the current path so far + the heuristic function (or the estimated cost to go from n to goal state), i.e. f(n) = g(n) + h(n). If we let h(n) = 0 for all n, then f(n) = g(n). This is still a case of A* search because h(n) would still be an admissible heuristic since 0 < shortest path from n to goal state for all n (assuming cannot have negative edges). Then, from the algorithm of A* search from class with h(n) = 0, we would take the node from the front of the queue (like in UCS), then add the successors n' of n with priorities f(n') = g(n') + h(n') = g(n') + 0 (the same as in UCS), and then terminate when the goal state is popped from the queue (same as in UCS). Therefore, you would get the same result running UCS and A* search with h(n) = 0 for all n. Therefore, UCS is a special case of A* search[2]. ■

d. FALSE. Below is a counterexample:



First note that the heuristic is perfect:
h*(S) = 1+3=4 = h(S)
(NOTE: h*(S) = 4 because h* = shortest path)
h*(A) = 3 = h(A)
h*(B) = 2 = h(B)

The optimal path would be S->A->G, with cost = 4

Best-first search will expand the most promising node (node with smallest heuristic), so best-first search would give the path S->B->G (because h(B) = 2 < h(A) = 3), which has a cost = 5. This is clearly NOT the optimal path.

Therefore, best-first search is NOT optimal in the case where we have a perfect heuristic[3]. ■

e. TRUE. In A* search, nodes are sorted by f(n'), where f(n') = g(n') + h(n') = cost from start to n' + estimated costed from n' to goal, and the node with the smallest f(n') is expanded. In the case of a perfect heuristic, h(n) = h*(n) = shortest path from n to goal state for all n. Therefore, f(n') = cost from start to n' + shortest path from n' to goal state. For this example, we have been told to assume a unique optimal solution (otherwise, we would need to account for tiebreaking). At the first iteration, A* search will pick the node where f(n') is minimal, meaning the cost of going to node n' + the shortest path from n' to goal will be minimal. The optimal path must contain the node n' with minimal f(n'). If we continue until the goal state, only nodes that are part of the optimal path will be expanded. As a result, A* search with a perfect heuristic will never expand nodes that are NOT in the path of the optimal solution. ■

[2] Referenced https://ai.stackexchange.com/questions/9182/how-do-i-show-that-uniform-cost-search-is-a-special-case-of-a for Q2c

[3] Referenced https://stackoverflow.com/questions/53311457/is-best-first-search-optimal-and-complete for Q2d

3. The code for question 3 can be found in Q3.py. The results are shown below:

For 7 cities, 100 instances:

	MIN	MAX	MEAN	STDEV	# OPTIMAL
Brute-force	1.4886	3.66881	2.51718	0.358308	
Random	2.00548	5.92662	3.76011	0.70977	0
Greedy	1.4886	3.66881	2.51901	0.360451	97

a. Brute-force search code can be found in def bruteForce(distances, numCities). The min distance was 1.4886, the max distance was 3.66881, the mean distance was 2.51718, and the standard deviation was 0.358308.

b. Random tour code can be found in def randomTour(distances, numCities). The min distance was 2.00548, the max distance was 5.92662, the mean distance was 3.76011, and the standard deviation was 0.70977. No random tours happened to be the optimal solution. The average distance from random tour was greater than the optimal mean distance found in brute-force serach, and there was more variation (larger standard deviation).

c. Greedy local search/ hill climbing using the 2-change neighbourhood function can be found in def greedy(distances, randcost, path, numCities). The min distance was 1.4886, the max distance was 3.66881, the mean distance was 2.51901, and the standard deviation was 0.360451. The number of instances where the algorithm found the optimal solution was 97 (or 97%). The min and max were the same as for the brute-force search, with the mean slightly higher. The standard deviation was also similar to that for the brute-force search.

d. It was not possible to compute the cost of the optimal solutions using brute-force search in reasonable time. However, below are the min/max/mean/standard deviation using the random tour approach and greedy local search approach for 100 instances involving 100 cities:

	MIN	MAX	MEAN	STDEV
Random	45.671	59.2345	52.3549	2.51471
Greedy	7.68322	8.93237	8.37079	0.30295