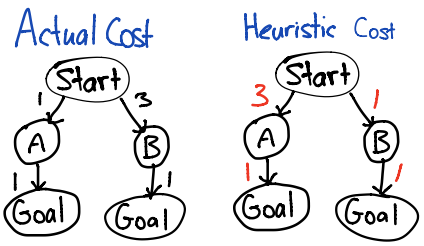
7.

Assume a heuristic H is consistent. Let H be at some state **s**, and denote the action of reaching the closest goal node, **g**, by **A**. The conditions for consistency guarantee that $H(s) \leq cost(A) + H(g)$, where cost(A) is the actual cost of action A. But $H(g) = 0$ so $H(s) \leq cost(A)$. Thus the heuristic will always be less than or equal to the actual cost of reaching the goal, and this is valid for any point s. Therefore, H is admissible.

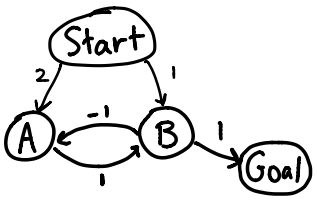
One example of a heuristic that is admissible but not consistent in the corner problem is a heuristic that computes the Manhattan distance to the closest corner that hasn't been reached yet. This is always less than or equal to the total cost, so the heuristic is admissible. However, it is not consistent when there are multiple unvisited corners. When you are a distance of 1 away from the closest corner, the heuristic evaluates to 1. Then when you move onto the corner, that corner is no longer unvisited so the heuristic evaluates to the distance to one of the remaining unvisited corners, which can be a distance *d* which is greater than 1 away. Thus, the heuristic jumped from 1 to d after a single movement with a cost of 1. Therefore, the heuristic is not admissible.

9



The actual optimal path would be going from the start through A to the goal with a total cost of 2. However, using A* with the inadmissible heuristic on the right would result in a path being found that goes through B to the goal with an actual cost of 4, which is not the optimal solution.

10



Assuming previously nodes can be visited, this is possible. From the start and using uniform cost search, the search will go to Node B. However, from B, it won't go to the goal. Instead, it will go to A since that reduces the overall cost. From A, it will go back to B. The search will continue oscillating between A and B infinitely, since the negative edge cost results in this path producing the lowest cost at all times.

11

Depth first search would be better since the alphabet contains thousands of symbols so each subsequent level of the search would contain thousands of branches. Breadth first search would expand all the nodes at a certain level at once, which would quickly exact memory at deeper levels. Depth first search only expands the nodes in its path and along the frontier of the path, which is far less than the number of nodes the BFS expands, so it works better when you have a large number of branches to ensure you have enough memory. The algorithm should be changed to suit the problem by modifying the maximum depth the DFS (Iterative deepening) can go to 5, since that is the size of the sequence we are looking for.

12

Instead of limiting the iteration by depth, you could limit the iteration by cost. To do this, you would use set a maximum cost that the DFS could expand to, and explore other branches once that cost is reached. If no solution is found, you would expand the maximum cost. There might be multiple solutions after an expansion, so to ensure it is the lowest cost, for each iteration, you would search the whole iteration and store all solutions if multiple are found in that iteration and return the lowest one out of the solutions found. This guarantees that the optimal solution will always be found.