IA Solutions to sheet 1

Exercise 1

It is possible. Let's represent wolves as w, goats as g, the boat as b and the river as $|$, so we can take the starting state as $wwwgggb|$ (with everything on the left side of the river). One route to the solution which never has more wolves than goats on the same side of the river is

$$\begin{aligned} wwwgggb| &\to wggg|wwb \to wwgggb|w \to ggg|wwwb \\ &\to gggwb|ww \to gw|ggwwb \to ggwwb|wg \to ww|wgggb \\ &\to wwwb|ggg \to w|wwgggb \to wwb|wggg \to |wwwgggb \end{aligned}$$

This introduces the idea of a *state space* in which we search for a solution. For this specific example we would most likely be interested in the *path* to the solution as well. Sensible search methods here involve keeping track of what states you have already visited in order to avoid pointless loops: i.e. it is a good idea to draw a graph of what states are accessible from what other states. Note that searching a graph (keeping track of visited nodes) requires more memory space than some of the bounds given in Exercise 3 below: to speak generally, there is often a tradeoff between time and space.

Exercise 2

In the pseudo-code as it is given, it is not specified in which order we choose the nodes from the frontier. If we treat the frontier as a *stack*, or a *last in first out* structure, the pseudo-code performs depth-first search. If we treat the frontier as a *queue*, a *first in first out* structure, the pseudo-code performs breadth-first search.

Exercise 3

For this question, we call a search method *complete* if it always finds some solution (if a solution exists). A search method is *optimal* if it only finds solutions of *minimal depth* (this is the same as finding a minimal cost path when the cost function is constant). Note this question concerns searching using a *tree*.

	BFS	DFS	Depth-limited	Iterative-deepening	Bidirectional
complete	✔	✔*	✘	✔	✔
time complexity	$\mathcal{O}(b^{d+1})^\dagger$	$\mathcal{O}(b^m)$	$\mathcal{O}(b^l)$	$\mathcal{O}(b^d)$	$\mathcal{O}(b^{\frac{d}{2}})$
memory complexity	$\mathcal{O}(b^{d+1})^\dagger$	$\mathcal{O}(bm)$	$\mathcal{O}(bl)$	$\mathcal{O}(bd)$	$\mathcal{O}(b^{\frac{d}{2}})$
optimal	✔	✘	✘	✔	✔

∗ If the maximal depth is finite.
† If $d < m$ (otherwise b^m).

Here is a bit more explanation of why BFS has time complexity $\mathcal{O}(b^{d+1})$, whereas Iterative-deepening only has time complexity $\mathcal{O}(b^d)$. We assume that our algorithm only checks whether a node is a solution state directly before it is expanded. When we expand all the nodes at depth k, we might have to add b^{k+1} nodes to the frontier, so we might perform at least b^{k+1} "operations". In particular, for BFS it is possible that $b^{d+1} - b$ nodes at depth $d + 1$ are added to the frontier before we check the solution. In total, we might have to have "seen" $1 + b + b^2 + \cdots + b^d + (b^{d+1} - b)$ nodes, thus we get at least $\mathcal{O}(b^{d+1})$ (a bit more reasoning shows that this bound is tight).

For Iterative-deepening, we perform successive Depth-limited searches, each time increasing l by one. For the time complexity, we simply add up the time taken by each Depth-limited search with $l = 1, 2, \ldots, d$. We argue that this sum is dominated by the time taken by the Depth-limited search with $l = d$.

In a Depth-limited search up to l, we effectively assume that the nodes at level l have *no* children. When we expand a node on level l there are no children to add to the frontier, so (roughly speaking) we perform only b^l operations for all the nodes at level l (we still perform at least b^{k+1} operations for all the nodes at a level $k < l$). This means that we get a time complexity of $\mathcal{O}(b^l)$ for Depth-limited search. Altogether, the time complexity of Iterative-deeping is therefore the same as that of Depth-limited with $l = d$, thus we get $\mathcal{O}(b^d)$ for Iterative-deepening.