Thanh Cong Nguyen

Question 1:

Algorithm	Time Complexcity	Space Complexcity	Complete	Optimal
BFS	$O(b^d)$	$O(b^d)$	Yes	Yes
UCS	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	$O(b^{1+\lfloor C^*/\epsilon \rfloor})$	Yes	Yes
DFS	$O(b^m)$	$O(bm)$	No	No
DLS	$O(b^l)$	$O(bl)$	No	No
IDS	$O(b^d)$	$O(bd)$	Yes	Yes
A*	$O(b^d)$	$O(b^d)$	Yes	Yes

* Note: DFS is complete in finite space

IDS: time complexity is $O(b^d)$ when there is a solution, or $O(b^m)$ when there is none

IDS: space complexity is $O(bd)$ when there is a solution, or $O(bm)$ on finite space with no solution

Question 2:

Part 3)

- Breadth first search: Total states generated: 33
- Depth first search: Total states generated: 18

Part 4)

- Breadth first search: Total states generated: 33
- Depth first search: Total states generated: 18
- Depth limited search: Total states generated: 18 (limit = 10)

Part 5)

- Breadth first search: Total states generated: 87
- Depth first search: Total states generated: 48
- Depth limited search: Total states generated: 48 (limit = 10)

Part 6)
Breadth first search:
Total states generated: 10
Total states generated: 11
Total states generated: 59
=> Total: 10 + 11 + 59 = 80

Depth first search:
Total states generated: 12
Total states generated: 22
Total states generated: 20
=> Total: 12 + 22 + 20 = 54

Depth limited search: (limit = 10)
Total states generated: 12
Total states generated: 22
Total states generated: 20
=> Total: 12 + 22 + 20 = 54

- When a problem is subdivided into smaller components, the number of states generated in Breadth-First Search decreases from 87 to 80, while in Depth-First Search, it increases from 48 to 54. Similarly, in Depth-Limited Search, the number of states generated also increases from 48 to 54.

Question 3:
USC algorithm: Total states generated: 62
A* algorithm: Total states generated: 62

Question 5:

The engineering advances that led to Deep Blue's success were:

- Specialized hardware: There were 480 of single-chip chess, and each one could process 2 to 2.5 million chess positions per second. This chip was in a system composed of a 30-node IBM RS/6000 SP computer, with 16 chess chips working in parallel per SP processor.
- Hand-crafted heuristics and grandmaster database: Deep Blue used a carefully curated opening book of about 4,000 positions, created by grandmasters. Additionally, the extended book, based on a 700,000-game database of Grandmaster games, applied bonuses or penalties to moves based on their frequency, success rate, and strength.
- Alpha-beta pruning and search extensions: The alpha-beta pruning algorithm used by Deep Blue helped optimize its search by cutting off unnecessary branches of the game tree, while search extensions allowed the system to focus more deeply on critical positions.
- Complex evaluation function: comprehensive system that assigns values to approximately 8,000 distinct chess patterns, incorporating both static and dynamic feature values to assess board positions.
- Parallelism and optimization: Deep Blue employed a parallel architecture, consisting of 30 processors working in parallel to search the game tree. This allowed the system to explore multiple branches simultaneously, significantly speeding up the search process. Each processor was responsible for evaluating a portion of the game tree, ensuring that the system could search a vast number of positions more efficiently.

Transferability to other problems: Parallelism and optimization, Search extensions and alpha-beta pruning.
Chess-specific: Specialized hardware, Hand-crafted heuristics and grandmaster database, Complex evaluation function.

b) The reasons that AlphaZero is able to defeated StockFish even when it is given only 1/100 of the computing time:

- Stockfish uses alpha-beta search, evaluating 70 million positions per second, but its efficiency relies on the order of moves, handled by iterative deepening. In contrast, AlphaZero, with Monte-Carlo tree search (MCTS), evaluates just 80,000 positions per second but focuses more selectively on the most promising variations using a deep neural network. This neural network helps compute move probabilities and expected outcomes, guiding AlphaZero's search more efficiently than Stockfish. While Stockfish evaluates a much higher number of nodes, AlphaZero compensates by evaluating fewer but more relevant nodes, resulting in better decision-making. Instead of a minimax evaluation like Stockfish, AlphaZero averages over positions in a subtree, using non-linear function

approximation to evaluate positions, which is more powerful than the linear method used by typical chess programs. While this may cause approximation errors, MCTS averages them out, unlike alpha-beta search, which can propagate these errors to the root of the subtree. This allows AlphaZero to make better decisions with fewer evaluations and less computing power.

=> Therefore, the combination of MCTS and deep neural networks allows AlphaZero to explore fewer nodes while still making stronger strategic decisions, enabling it to outperform Stockfish even with only 1/100th of the computing time.