Name: CHAN King Yeung
SID: 1155119394
CSCI3230 Assignment 2

---

a)

i. Genetic algorithm is an example of stochastic search algorithm. It evaluates an optimal point for an objective function. It initialises a set of individuals randomly and generates next generation repeatedly. For every generation, the population will experience selection, crossover and mutation (each of which consists of probabilities) to search through the whole state space. It stops at some reasonable generation or the fitness function (function to evaluate the objective function) no longer change substantially.

ii. $g(n)$ is a function which records the total cost spent from the first node to n node.
$h(n)$ is a heuristic value function which estimates the (minimal) cost from n node to the goal node.

iii. Both depth-first search and depth-limited search favour the Minimax algorithm.

iv. Breadth-first search (BFS)

With b limited nodes in the search tree, if there is a solution in it, BFS eventually finds its depth. Thus, BFS is complete. In the view of worse case, BFS needs to go through all the depth for searching a solution. If there are d depths in the search tree, the time complexity will be $1 + b + b^2 + \cdots + b(b^d - 1)$, that is $O(b^{d+1})$. Moreover, BSF memories every nodes while searching the solution. The space complexity in the worse case is also $O(b^{d+1})$. In general, BFS is not optimal since BFS does not consider the cost when replacing the fringe. However, if all the step cost is the same, BFS becomes uniform-cost search which guarantees optimality.

Depth-first search (DFS)

Similar to BFS, if the depth is finite, DFS eventually reaches to a solution. Yet, if there are cycles in the depth space, the depth space becomes infinite. DFS may get stuck in the looping the cycles and DFS no longer complete. Given a search tree has m maximum depths, DFS goes through all the depth to finds a solution in the worse case, and thus, the time complexity is $O(b^m)$. DFS searches a branch of subtree at a point in time. The space forms a linear space and the space complexity is $O(bm)$. DFS ignores the step cost to search for a solution in the leftmost subtree. Hence, DFS is not optimal.

v. Suppose $h(\cdot)$ is consistent, $h(n_g) = 0$ where n_g is a goal node and $h^*(\cdot)$ is the actual cost. We have

$h(n_g - 1) \le c(n_g - 1, n_g) + h(n_g)$ where $c(a, b)$ is the cost from a node to b node
$h(n_g - 1) \le c(n_g - 1, n_g) + h^*(n_g)$ given we know $h(n_g) = h^*(n_g)$ at the goal node
$h(n_g - 1) \le h^*(n_g - 1)$ given we know $c(n_g - 1, n_g) + h^*(n_g) = h^*(n_g - 1)$

If the fact $h(n_i) \le h^*(n_i)$ holds for any arbitrary node $n_i \coloneqq n_g - i$, consistency implies admissibility by definition. Thus, we have

$h(n_i - 1) \le c(n_i - 1, n_i) + h(n_i)$
$h(n_i - 1) \le h^*(n_i - 1)$ from the proof above
$h(n_i - 1) \le c(n_i - 1, n_i) + h^*(n_i)$
$h(n_i - 1) \le h^*(n_i - 1)$

Thus, consistency implies admissibility.

On the other hand, admissibility does not imply consistency. Consider a single way out path which has g nodes, that is $\langle n_1 \quad n_2 \quad \cdots \quad n_g \rangle$ where n_1 is the first node and n_g is the goal node. Suppose $c(a, b) = 1$ for every step and $h^*(n_1) = g$, we know $h(\cdot)$ is admissible if we let $h(n_1) = g - 1$ and $h(n_i) = 1 \; \forall \; i \ne 1 \cap g$. In the situation, $h(n_i) \le h^*(n_i) \; \forall \; i$, hence, $h(\cdot)$ is admissible without doubt. However, we know that

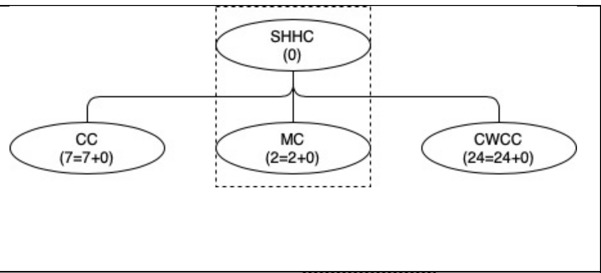
$h(n_1) > c(n_1, n_2) + h(n_2)$
$g > 1 + 1$
$g > 2$

Thus, consistency implies admissibility but not the other way around.

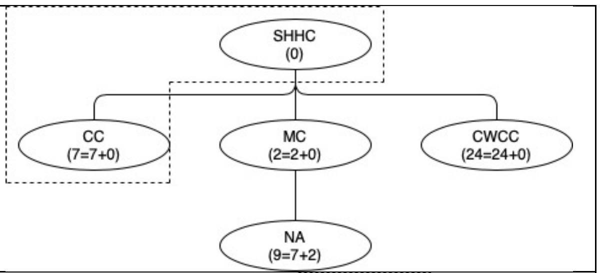
b)

i.

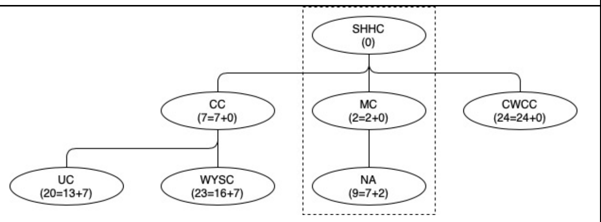
1.



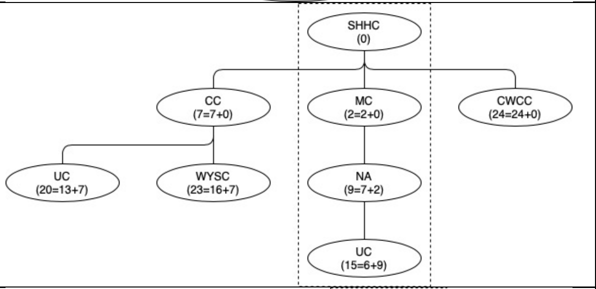
2.



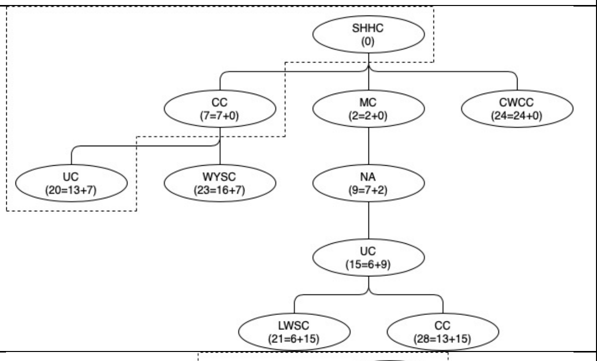
3.



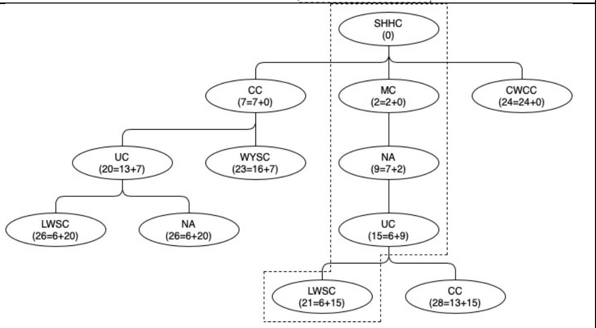
4.



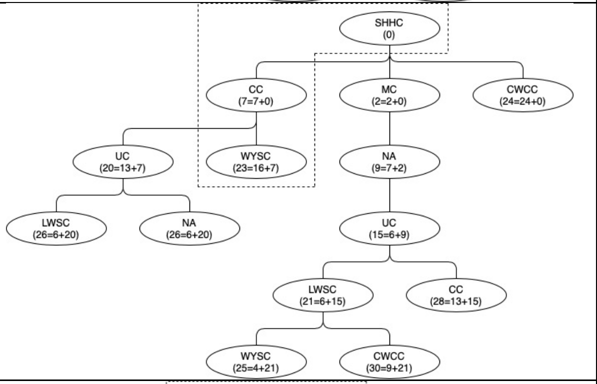
5.



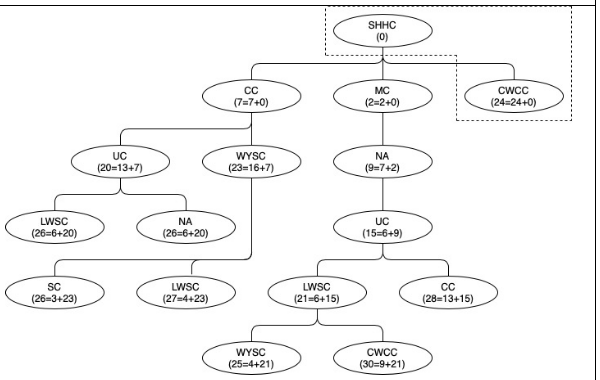
6.



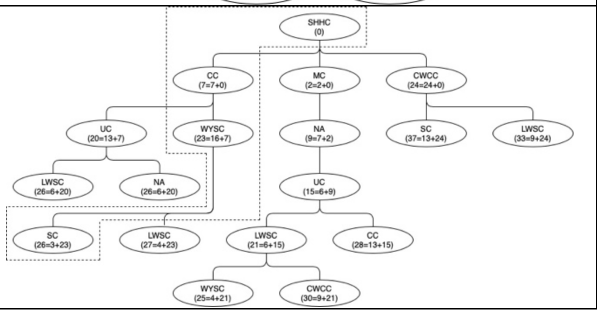
7.



8.

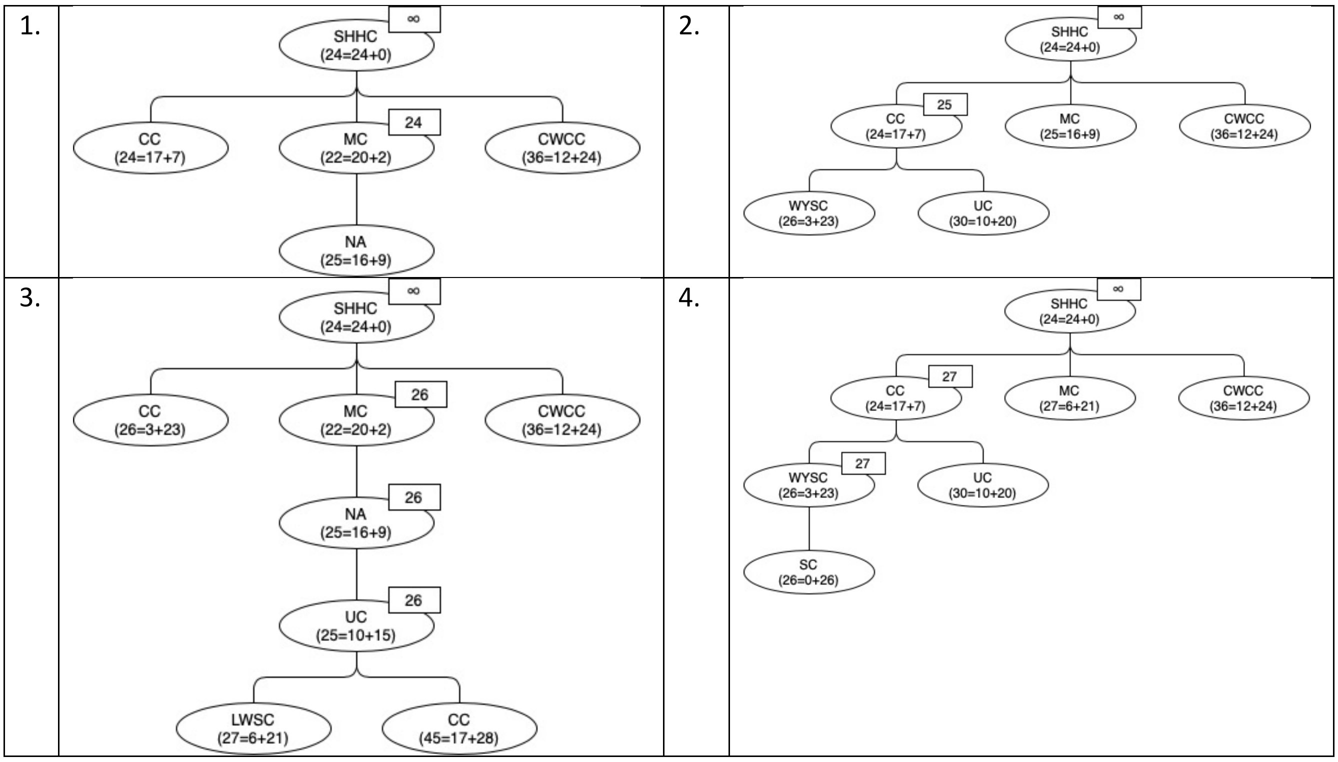


9.



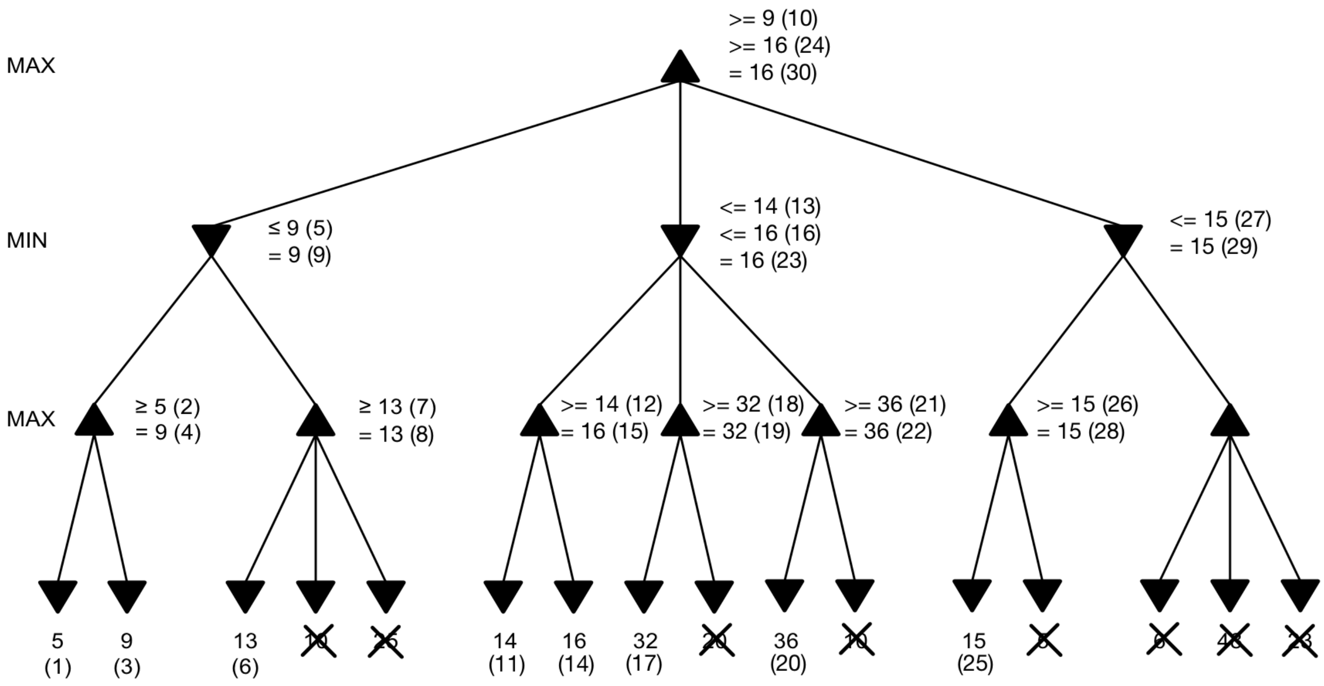
Therefore, the shortest path will be SHHC → CC → WSYC → SC.

ii.



Therefore, the shortest path will be SHHC → CC → WSYC → SC.

c) Pruning in minimax algorithm aims to eliminate a branch of a search tree if we have enough information about some nodes to draw the conclusion. Without checking all the nodes in the search tree, it reduces the exponent in time complexity by half. The beauty of pruning is that the final result remains the same even cutting branches. There are 2 parameters in α-β pruning, which are α and β respectively. α consists of the maximum value it has found and β records the minimum value it has found. When we found a value which is the best choice in the corresponding subtree, that is $\alpha \geq \beta$, we can ignore the remaining nodes in that subtree. Because we always select the best choice before pruning, ignoring nodes will not affect the final decision and makes the algorithm runs faster.


MAX
>= 9 (10)
>= 16 (24)
= 16 (30)
MIN
≤ 9 (5)
= 9 (9)
<= 14 (13)
<= 16 (16)
= 16 (23)
<= 15 (27)
= 15 (29)
MAX
≥ 5 (2)
= 9 (4)
≥ 13 (7)
= 13 (8)
>= 14 (12)
= 16 (15)
>= 32 (18)
= 32 (19)
>= 36 (21)
= 36 (22)
>= 15 (26)
= 15 (28)
5 (1)
9 (3)
13 (6)
14 (11)
16 (14)
32 (17)
36 (20)
15 (25)