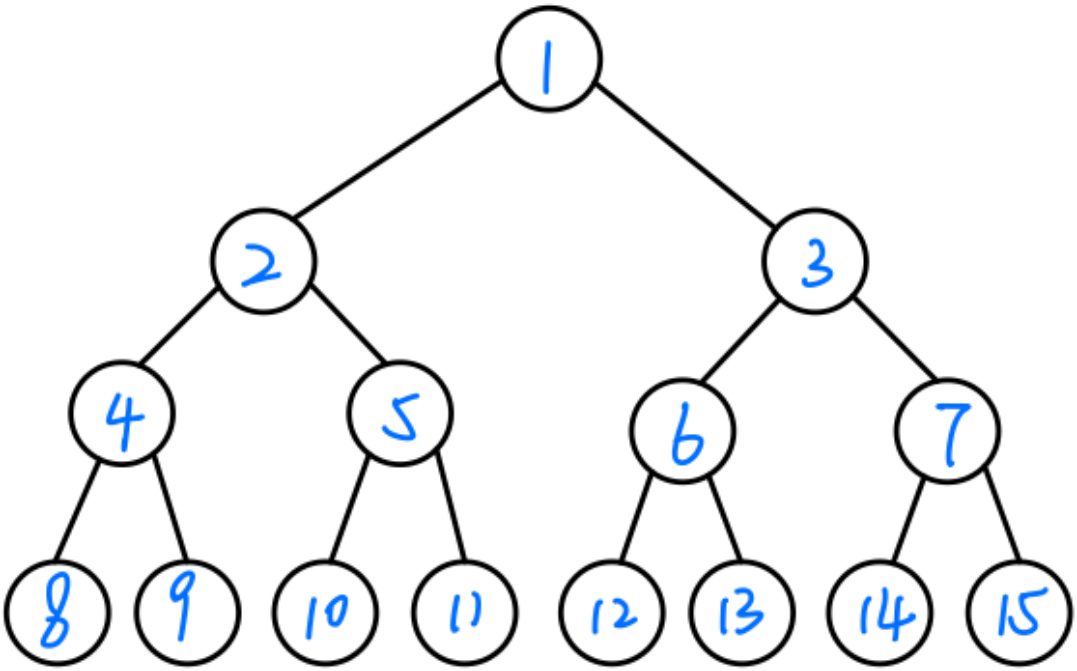
AI HW1

Name: 杨睦圳 **SID**：11813010

Q1

a.



b.

BFS: 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 11

DFS with depth-limited: 1 -> 2 -> 4 -> 8 -> 9 -> 5 -> 10 -> 11

IDS: 1; 1 -> 2 -> 3; 1 -> 2 -> 4 -> 5 -> 3 -> 6 -> 7; 1 -> 2 -> 4 -> 8 -> 9 -> 5 -> 10 -> 11

c.

Suppose we use the BFS in the bidirectional search, the step will be:

Direction	Visited Code	
Forward	1	Forward= {2, 3}
Backward	11	Backward = {5}
Forward	2	Forward= {3, 4, 5}
Backward	5	Backward = {2}

We can see that the bidirectional search only need 4 step to solve the problem, which is very efficient.

The branching factor for forward direction is 2 and for backward direction is 1.

d.

Yes, we can start from the goal state and search backward. Since the branching factor for backward direction is 1, the next state for i will be $\lfloor \frac{i}{2} \rfloor$. Therefore, it will need no search to go to the state 1 from the goal state.

e.

Yes, consider the process that convert a decimal number to the binary number, we can get the algorithm as below (the $goal\_state$ is a number represents the goal state and the $result$ is the order of the move):

```
find_way(goal_state):
    i<- goal_state
    Let reverse_order be a Stack, result be a List
    WHILE i > 1:
        IF i % 2 == 0:
            reverse_order.pull("Left")
        ELSE
            reverse_order.pull("Right")
        END IF
        i = i / 2
    END WHILE
    While reverse_order is not empty:
        result.append(reverse_order.pop())
    return result
```

Q2.

a.

TSP problem is to find a closed loop that contains all cities with the minimal length. The MST is a relaxed version of that problem since it is to find minimum costs of a link containing all cities without loop. Besindes, as a heuristic, MST is admmissible because the cost of it is always <= the solution for TSP.

b.

To show that the MST heuristic dominates straight-line distance from the current city C back to the start city S, we need to show that MST never gives a smaller value. Add the direct edge E from C to S into the MST then we can get a loop containing C and S. If there exists an edge between C and S in the origin MST, then the value given by MST will at least equal to the value given by the straight-line distance. Else, the loop should be a polygon, which means that the new edge is a edge of the polygon so the value of it, i.e. the straight-line distance, will be smaller than the sum of all the value of other edges of the polygon, i.e. the value from MST. Therefore, MST heuristic dominates straight-line distance because it never gives a smaller value than the straight-line distance.

c.

The length of the cities sequence means the total length of the path given by the sequence.

```
Index the cities from 1 to n
Let f(s) be the total length of the loop sequence s
Let S(0) be the city sequence {1, 2, 3, ..., n, 1} and i = 0

REPEAT:
    Randomly exchange two nodes in S(i) for multiple time and store each new
sequence in a set called neighbour
    Get sequence with minimum length in neighbour, call it T
    IF the total length of T > the total length of S(i) THEN
        BREAK
    ELSE
        S(i + 1) <- T
        i <- i + 1

S(i) is the result given by this algorithm
```

d.

Gene: The cities sequence of the loop.

Fitness: Since we want the length of the loop to be small but the fitness to be large, we can let $fitness = \frac{1}{length\_of\_loop+1}$

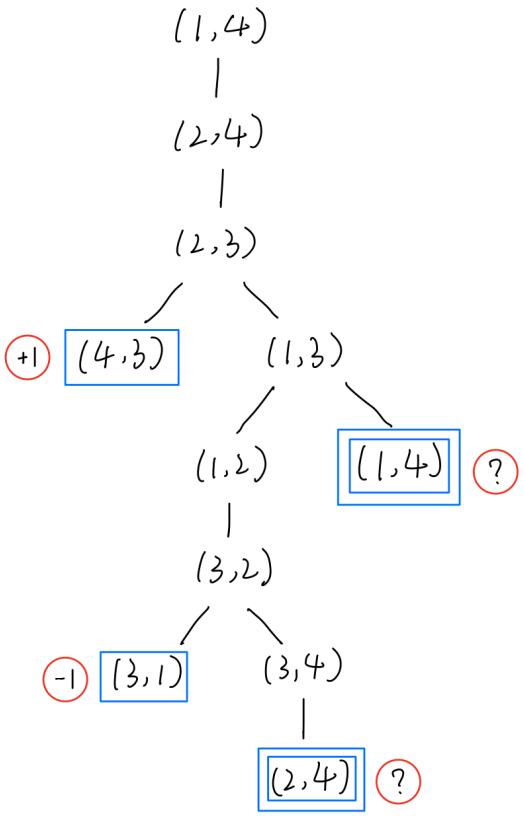
Crossover: The gene of the offsprings is a combination of the parents' gene, with the relative order of the gene from any parent doesn't change.

Mutation: For a small probability, randomly exchange two nodes of the sequence.

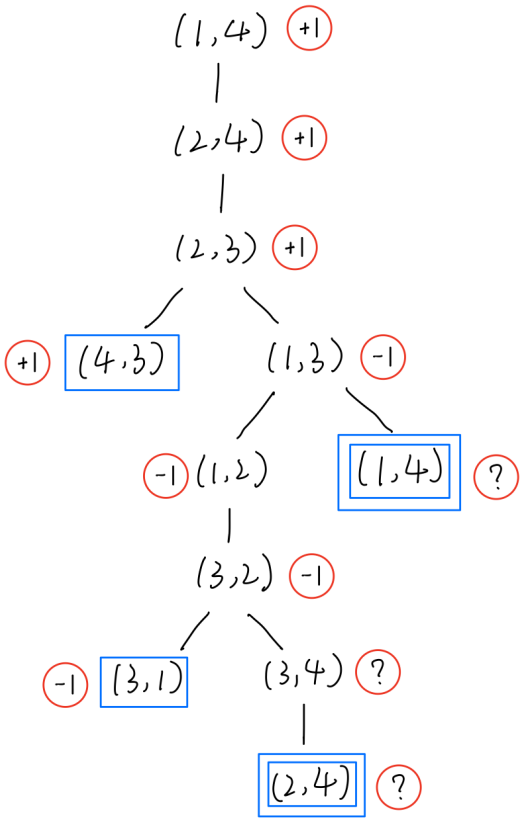
```
Index the cities from 1 to n
Randomly generate some loop sequences to be the initial population P(0)
Let i = 0

REPEAT:
    evaluate the fitness of each individual in P(i)
    select the parents from P(i) based on the fitness
    generate the offspring from the parents using crossover and mutation to
generate P(i+1)
    i <- i+1
    IF the best fitness keep stable for a number of iteration, THEN
        BREAK

The individual with the highest fitness in P(i) is the result given by this
algorithm
```

Q3.

a.



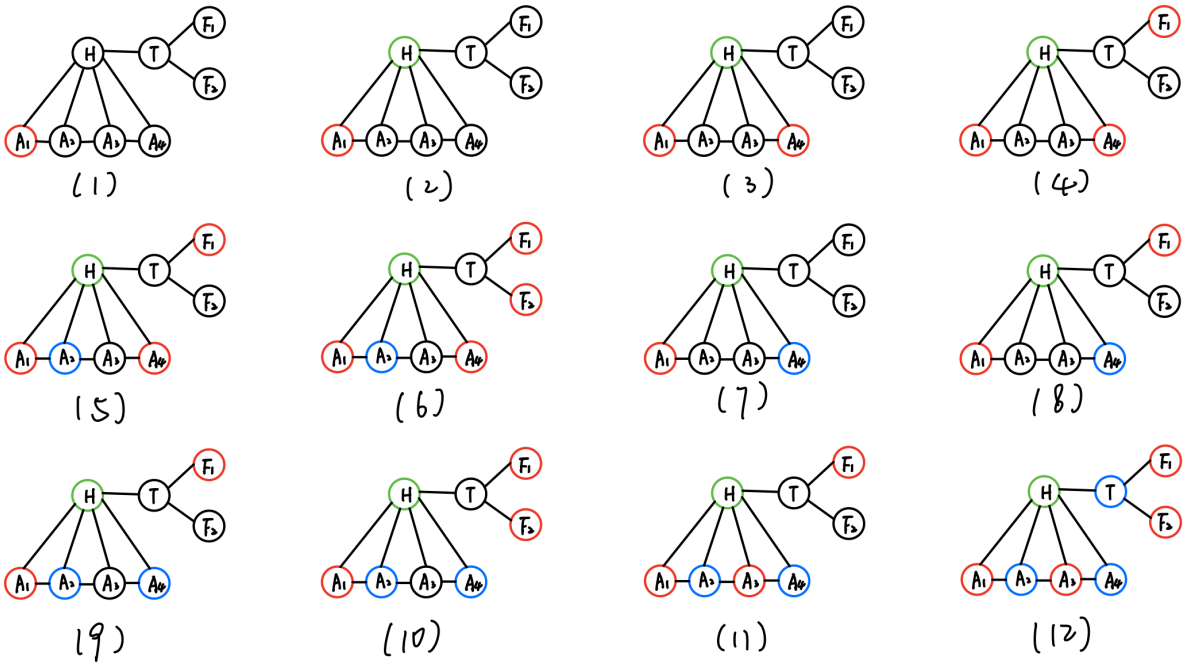
b.



Since "?" can be either "+1" or "-1", the strategy is give it the value between "+1" and "-1", i.e. we have

$$max(+1,?) = +1, min(?,-1) = -1, max(?,-1) =?, min(?,+1) =?$$

Q4.



(1) A_1 = R

(2) H = R conflicts with A_1 so H = G

(3) A_4 = R

(4) F_1 = R

(5) A_2 = R conflicts with A_1 and A_2 = G conflicts with H, A_2 = B

(6) F_2 = R

(7) A_3 = R conflicts with A_4, A_3 = G conflicts with H, A_3 = B conflicts with A_2. Go back to 5 and we can find that A_2 has no choice so we should then go back to (3). A_4 = G conflicts with H so A_4 = B.

(8) F_1 = R

(9) A_2 = R conflicts with A_1 and A_2 = G conflicts with H, A_2 = B

(10) F_2 = R

(11) A_3 = R

(12) T = R conflicts with F_1 and F_2, T = G conflicts with H, so T = B.

Then, we successfully color the node with three colors by the given order.