1) a). Proceed with proof by induction:

BC: $h(G) = 0 \leq h^*(G)$

IS: Suppose shortest path from A to G has i steps, then denote the successor of A to be A'.

Then, shortest path from A' to G has (i-1) steps.

According to the inductive hypothesis, $h(A') \leq h^*(A')$

∴ ~~Cons~~ h is consistent

~~$h(A) \leq c(A, A') + h^*(A')$~~

$c(A, A')$

∴ $h(A) \leq c(A, A') + h(A') \leq c(A, A') + h^*(A') = h^*(A)$ ∎

2. For a configuration, denote the second to last node as n'. Then, h is exactly the same with the given h* except that $h(n') = \frac{1}{2} h^*(n')$. (which means $h(n') < h^*(n')$ since $h^*(n') \neq 0$).
In this way, h is still admissable because $h(a) \leq h^*(a)$ for all node a. (Since $h(a) = h^*(a)$ for all node other than n'. and $h(n') < h^*(n')$.). It's also inconsistent because for all node a and its successors (other than n'). $h(a) - h(a') = c(a, a')$
but $h(n) - h(n') < h^*(n) - h^*(n') = c(n, n')$.
(n' is n's parent)

3. b). i). Start State: {a:a, b:b, c:c, ..., z:z}.
The root will have 26 children which are {a:a, ... z:z}, {a:b, b:b, ... z:z} ... {a:c, b:b, ..., z:z}, {a:z, ..., z:z}.
And the tree has 27 layers in total, For ith layer below the root, we flip all possible mapping of the ith letter.
DFS is gauranteed to run because this tree contains all possible states and DFS doesn't stop unless it tries all states or it reaches the goal state (given that we have the perfect goal-test).

ii). O(1).

iii). Yes, by using the given perfect goal-test, DFS only halts when it reaches the goal state or if it tries all possible states and cannot find a goal state.

c). cost function: we sum up the possibility of all adjacent letters in the parent state, call it S1, and do the same thing for its successor, call it S.
cost of changing from parent to children is S1 - S2.
heuristic: get the length of the word (i.e. "hi" has length of 2). then sum up the adjacent possibilities, call it S'. Then, for the current state n, h(n) = length - S'
It can guide A* because the heuristics give correct direction of whether the word is getting closer of farther from the target. (Since the h(n) gets larger as the word becomes less likely to be the target.)
This is not admissible because h(n) could be larger than h*(n),
And this is consistant because h(a) - h(b) is always equal to the cost according to the definition of cost and heuris

2

3. a).

i). Goal Test: check if every word in X is contained in W or differed by at most one letter with a word in W. If yes, then it's then goal. Otherwise, it's not.

~~This~~ This works because it tolerate typos (I define typo as misspelling at most one letter.) and it also return true if every words can be found in the dictionary.

ii). No. Some words only differ in one letter, and the typo tolerance property of the goal test function may ~~cause~~ return true even if the mapping is not correct.

For instance, if the original msg is: "It's big!"

The test will return true even if the ~~resp~~ state is: "It's bid!"

iii). Original Msg, "It's big!"

State: "It's bid!"

The test will return true because "bug" also exsist in the dictionary and it differs from "big" by only one letter.

4. a). State Representation:

A list of list, where each list inside ~~repre~~ contains the information of a discussion section.

ex: [[True, Jack, 8am slot], [False, Mary, 10am slot]----].

↗ Exam/Normal ↘ TA ↘ Timeslot.

Successor function:

Flip the entry of the lists inside to change the configuration of a specific discussion section.

Start state:

~~can be~~ each discussion section's configuration being randomly assigned. (Without having to worry about the constraints.).

Goal Test:

Iterate over~~#~~ the list, and check whether the arrangement of discussions satisfy the constraints listed in the problem. If so, the state being tested is the goal. Otherwise not.

b). $O(2^N \cdot M)$ → State Space Size.

Branching factor: $2 \cdot N$.

Algorithm. ~~def function: <check overlap>~~

~~# check if more~~

def function "check":

- - - -.

// check if the lists satisfy the constraints.

// if yes return 1

// if no return 0.

~~[illegible]~~