Jason Park
ECE 473 Homework 6

1.

a. I chose to define my cost function as follows: a higher cost means a greater number of words can be formed from the word as a base / starting point. For example: he would have a very high cost because a lot of words can be formed with a starting point “he” (hell, hello, heap, hesitate, etc.). Conversely, a word like hello would have a lower cost because there are not many words, if any, that can be formed from hello as a starting point.
The greedy algorithm is not guaranteed to find the lowest-cost segmentation of the input string “helloworld”. Observe the following breakdowns:
Optimal algorithm finds -> hello world
Greedy algorithm (n=2) finds -> he ll ow or ld
Greedy algorithm (n=3) finds -> hel low orl d _ _
Greedy algorithm (n=5) finds -> hello world
This shows that although the greedy algorithm can find the optimal solution, it is <u>not guaranteed</u> that it finds it.

2.

a. The counter-example I have chosen is the Golden Rule, “Do unto others as you would have them do unto you.”
The input string would be “D nt thrs s y wld hv thm d nt y”. We can see right away that the greedy algorithm would reason that “Do” would be the most likely starting word. From there, the greedy algorithm would then reason that “Do not” would be the most likely next word if the first word was “Do”. It would not choose “Do unto”, as it is much more uncommon and would have a higher cost initially because of the rarity. It could also get stuck with “not” as the first word of the bigram, as there are not many words, if any, that can come after “not” and contain the letters [t, h, r, s]. The only words I can think of are theirs, threes, ethers, and others. None of these words would provide the optimal solution, and the greedy algorithm might even get stuck on the second half of the bigram “not ___”.

3.

a. As a search problem, I would define the states as possible steps in a solution to the problem. The start state would be a tuple with the first index and the “-BEGIN-” string. Each subsequent state would be the previous word and the index of the next letter. The end state would be whenever the state[0] value is equal to the length of the query. Costs would be determined with a bigram cost function with parameters (previous word, new proposed word).