Ryan Worley

2a) The greedy input is suboptimal beause it will do what it thinks is best for the short term, and will not take long term into consideration.

For example:

-"Failure": Greedy alorithm will split @ fail, leaving -ure at the end, which is not a word. The optimal case wald be to return the word "failure" unsplit, which the greedy algorithm will not do.

b) Code.

2) The greedy algorithm is suboptimal for this search problem, due to the fact it cannot account for ~~seer~~ insertions beyond the current word.

For example:

"th rd rd" will return the same word for both entries, whatever word is lowest cost. Maybe the word "red" has the lowest cost associated with "rd." However, the sentance "the red red" doesn't make much sense. A better sentance "the red rod" could be found from an algorithm that looks into the future, and takes cost associated w/ two words next to each other

3a) States:

1. Letter index (current)
   - Current index in query
2. Letter index of end of previous word
   - End index of previous word
3. Previous word
   - Used to calculate bigrams cost

Use 3 states, not too many

- Use ①, ② to index consonents for possible fill
- Use ③ to calculate bigrams cost for prev word, current word

Actions:

- Create new word from possible fills
- Don't create new word, move to next index

Costs:

If new word created from possible Fills:

Cost = biGrams(prevWord, currentWord)

Initial:

CurrentIndex = 0

Prev Word Index = 0

PrevWord = -BEGIN-

Final Test:

Ci == len(query) ✓

Pre== len(query) ✓
word index

If both word indices reach end of string solution is found.

3c) Use A* search, use h(s) function for a relaxed function.
where $cost_{relaxed} \leq cost(s,a)$.

- Use u_b
- $h(s) \leq FutureCost(s)$

Solution: We define $u_b(w) = min_{w'} b(w', w)$ where w' is all possible new words via function possibleFills. Thus, this will return a lower predicted end cost than the actual.

$$u_b(w) = cost'(s;a) = min_{w'} b(w';w)$$

The state and actions are the same for this heuristic problem.

$$min\ b(w';w) \leq b(w';w)$$

$cost_{rel}(s,a) \leq cost(s,a)$, so relaxed problem is consistent.

(Not sure if right, took best guess)

3d) - UCS is just a basic case of A* search. If h(s) in A* search = 0, A* turns into the UCS method. h(s) functions narrows path of UCS search, but if 0, h(s) doesn't do any thing.

- BFS is a somewhat special case of uniform cost where all actions have same cost. Since all actions are same, frontier (first state in frontier) will be shallowest path in tree search. As soon as solution is found for both algorithms, we know this most cost effective because it is first on frontier.