Austin Ray HW3

1.|a.| $n = 1 \Rightarrow$ 1-grams

u = cost function

ex: $u(\text{"the"}) = 0.005$ $u(\text{"(ortaitous"}) = 0.9$

- more frequently found in english
- less freq. found in english

⇓ lower score ⇑ higher score

$u(\text{"me"}) = 0.008$ $u(\text{"s"}) = 1000$

$u(\text{"mes"}) = 1000$ ↳ b/c not english word

$u(\text{"themes"}) = 0.8$

u

$\text{greedy}(\text{"themes"}) = u(\text{"the"}) + u(\text{"me"}) + u(\text{"s"})$

$= 1000.013$

$>> u(\text{"themes"}) = 0.8$

∴ greedy gives sub-optimal answer by optimizing for current word only, which makes other words very sub-optimal

HW3

1. b.

goal state = all letters used

state = (cur word start ind, cur word end ind)

actions = {end word, add letter}

HW3

2. [a.] "my fst dg"

1.) previous word is "my"
partial
current word is "fst"
↳ infer "fist" for "my fist"

2.) previous word is "fist"
current partial word is "dg"
↳ infer "dug" for "fist dug"

"my fist dug" is the result, with "my fist" and "fist dug" as the bigrams.

However, "my fast dog" would have made more sense, since "my fast" is close to "my fist" and "fast dog" is way better than "fist dug"

HW3

3. [a.]

actions = { cut current word and fill in its vowels (≥0 actions), add a letter to the current word }

(actually ≥20 actions, where each is the word w/ its vowels filled in some way)

State = { previous word, current word, start char. index, end char. index }

initial state = { "-BEGIN-", 0, 0 }

final state = { anything..., strlen(input), strlen(input) }

costs = { 0 for adding letters to current word
otherwise (bigramCost(prevWord, action)

{ action is some way to Vowel-fy current word

HW3

3.1 C.1 (python syntax used below)

Simpler Search (P) Problem

- w = current word
- w' = previous word
- u(w) = min([b(w', w) for w' in word_corpus])
- $\vec{U}$ = [u(w) for w in word_corpus]
- $\vec{U}$ can be pre-processed as long as {all possible fills} ⊆ {word corpus}
- u(w) ≤ b(w', w) for all w' in word_corpus because b/c u(w) was defined to be equal to the minimum b(w', w) value over all w' in word_corpus
- state = {start char ind of current word, end " " " "}
  = {s_ind, e_ind}
- actions(state=[s_ind, e_ind])
  → = combined array of:
  - possibleFills(queryString[s_ind : e_ind])
  - if(e_ind != len(queryString)):
    - 1 # ← 1 as action means we're adding a char to the current word, not letting it alone

3C. (cont'd)

Simpler search problem P' (cont'd)

- goal state = [s_ind, e_ind]
  where e_ind == len(queryString)
- succ(s=[s_ind, e_ind], type(a): 1 = delAction : String)
  → = [e_ind, e_ind]
  — a == 1 here
- succ(s=[s_ind, e_ind], type(a) = addAction : Integer)
  → = [s_ind, e_ind + 1]
- startState = [0,0] OR whatever start state is given to P' by P
- Basically, states, goal states, and successor states are same for simpler P' 3(c). as for 1(a-b).

Costs: cost'(adding a letter) = 0 = cost(adding a letter)

cost'(deleting word + filling as word w) = u(w)

↑ cost' used b/c this is a relaxation (P') of the original problem P

u(w) ≤ b(w', w) for all w'

∴ cost'(s, a) ≤ cost(s, a) for all possible s + a

3c. (cont'd)

The heuristic $h_u(s_i)$ is just to solve the relaxed version of the problem in its current state (s_i). That is, feed the current state, s_i, into the relaxed problem P' as its starting state, then solve the relaxed problem as defined on the previous pages and return the total cost of solving the relaxed problem as $h_u(s_i)$.

Thus, if the query string for the original problem P is "thridg" and the current state is ["the", 2, 4] then h_u(["the", 2, 4]) is the total cost of solving P' with start state [2, 4]

As stated, P is the problem as defined in 3a. and P' is the relaxed version of that problem, which has a condensed state space and uses $u(s,a)$ instead of $b(s,a)$