11-20 Lecture:

Lecture Notes: Key Topics and Concepts

---

1. Administrivia

- Introduction and brief logistics discussion.

---

2. QotD23: `revComp(S)` (Reverse Complement)

- **Problem Description**:
  - Input: A DNA string `S` (consisting of `A, T, C, G`).
  - Output: Reverse complement of the DNA string where:
    - `A → T`, `T → A`, `C → G`, `G → C`.
  - Reverse the order of the string in addition to complementing.
- **Biological Context**:
  - DNA strands bind to complementary strands forming a double helix.
  - Complementary strands can be used to replicate DNA.
- **Recursive Approach**:
  - **Base Case**: Empty string returns itself.
  - **Recursive Step**:
    - Consume the first character, compute its complement, and recursively process the tail.
    - Build the output by appending the complement in reverse order.
- **Optimization Notes**:
  - Constant strings are used for mapping complements instead of multiple conditionals.
  - Example of indexing:
    - Use a string like `"ACGT"` and an index calculation to find complements.
- **Runtime Consideration**:
  - Dictionary lookup for complements (constant time) is theoretically faster than string indexing (linear scan).
  - In practice, for small DNA strings, the indexing is simpler and effective.
- **His Solution**:

```
def revComp(S):
    if S == '':
        return ''
    return(revComp(S[1:]) + 'ACGT'['tgca'.index(S[0].lower())])
```

---

3. HW2: Overview

- **Primary Goals**:
  - Implement two methods for a `Wordle` class:
    1. `expandPattern(self, pattern, softCnt, match='')`
    2. `hint(self, S, target)`.
- **Key Points**:
  - Understanding the implementation of the provided `Wordle` template is critical.
  - Directly writing the methods without context leads to debugging challenges.

---

4. HW2: `expandPattern(self, pattern, softCnt, match='')`

- **Purpose**:
  - Converts a Wordle feedback pattern (with hard and soft constraints) into all possible "hard-only" patterns.
- **Parameters**:
  - `pattern`: String with uppercase letters (green), lowercase (yellow), and dots (gray).
  - `softCnt`: Dict counting occurrences of lowercase letters.
  - `match`: String used to construct the completed patterns.
- **Methodology**:
  - Recursive approach:
    - Process the pattern character by character.
    - Uppercase letters are fixed positions.
    - Dots can be replaced by any valid lowercase letter or a dot.
    - Lowercase letters are used to satisfy "soft" constraints.
  - Example:
    - Pattern: `S..t`
    - Soft constraints: `{'e': 1}`
    - Output: Set of valid hard patterns (e.g., `St.e`, `S..e`, etc.).
- **Output**:
  - Returns a set of hard constraint patterns.

---

5. HW2: `hint(self, S, target)`

- **Purpose**:
  - Computes and returns the word from `S` with the highest entropy.
- **Parameters**:
  - `S`: Set of possible target words based on constraints.
  - `target`: Actual target word (known internally).
- **Methodology**:
  - Shannon Entropy Calculation:
    - Each guess partitions `S` into subsets based on feedback patterns.
    - Calculate the probabilities of each subset based on word counts.
    - Compute Shannon entropy for each guess.
  - Select the word with the maximum entropy.
- **Key Example**:
  - `S = {27 words}`.
  - Feedback partitions `S` into subsets (e.g., `7 bins + 1 failure bin`).
  - Maximize information gain by choosing the word that results in balanced partitioning.

---

6. Algorithmic Considerations

- **Recursion and Efficiency**:
  - Recursive formulation emphasizes simplicity and modularity.
  - Tail recursion: Process head, recursively handle tail.
- **Runtime Trade-offs**:
  - Theoretical speed vs. practical implementation:
    - Small datasets → Favor simplicity.
    - Large datasets → Favor efficiency.
- **Entropy Optimization**:
  - Balancing subsets ensures faster narrowing of possible solutions.

---

7. Demonstrations

- **`expandPattern`**:
  - Tested with various patterns and soft constraints.
  - Outputs validated to match expected sets of hard constraints.
- **`hint`**:
  - Showcased effective narrowing down of possible solutions using entropy-based guesses.

---

8. Closing Remarks

- **Homework Advice**:
  - Start early; understand the provided codebase before implementing.
  - Focus on clarity in recursive logic for `expandPattern`.
  - Test incrementally to avoid confusion during debugging.
- **Support**:
  - Assistance available via Piazza or office hours (except during holiday breaks).

Questions/Unclear Points:

- How to efficiently integrate the entropy calculation into large datasets in `hint`.
- Handling edge cases where feedback patterns lead to ambiguous subsets.

HW2:
Youre only writing 2 methods but need to understand how the implementation works with rest of functions.
Copy patterns 9:00

```
#softCnt dictionary, everu lowercase letter in pattern. returns set, all paterns it matches
#soft lowercase, hard upperscase
#general idea is to descend the program recursively, exploring all feasible 'completions' of that pattern
#and using the third match parameter to accumulate the expanded pattern
#recursive formulation
```

#S set of words,computers entropy of word in S(set of remaining words from D consistent with target) against all of the words of S. Returns the word in S with the highest energy

#uses shannon entropy calculation to evaluare every remaining word in S with respect to it's ability to partition S: then choosef from the among tbe highest scoring words.
#We already have a pattern(the most recent feedback pattern) whihc tells us a lot about how the words in S can be partitiioned.
#Imagine |S| is 27, and the first word in S yields feedback 'st..Y' against the (hidden) target word

9:07 we count how many words in S end up in these seven bins(plus one more failure bin) and use those counts to compute the shannon entropy.

Look for solution xd

```
>>> w=Wordle()
5757 words read.
>>> S=set(w.D)
>>> w.expandPattern('S..te', {'t':1, 'e':1})
{'ST.E.', 'S.E.T', 'S..ET', 'SET..', 'S.TE.', 'STE..', 'SE..T'}
>>> w.expandPattern('CH..o', {'o': 1})
{'CHO..', 'CH.O.'}
```

Returns a set of expanded patterns.