Introduction

This is a continuation of homework 4a. The same background readings and dynamic program are applicable on this part of the assignment. While you have an additional 2 and a half weeks to complete this part of the assignment, be aware that this part will require more thought and coding than the Homework 4a. In particular, the derivation semirings will require non-trivial modifications to the semiring definitions. Therefore, we recommend that you

Get started now.

4 Derivation Semirings [40 points]

For each semiring in this section, you will implement the semiring in Python (starting with `starter.py` and modifying `semiZero`, `semiOne`, `semiPlus()`, `semiTimes()`, `A()`, and `R()`) and answer the following questions (10 points per semiring):

- [6 points] Submit your working code (required to get points for the following).
- [2 points] Describe your values of $\bigcirc$ and $1$ and how you compute $\otimes$ and $\oplus$.
- [2 points] Answer the space/time question described for each semiring below.

Your implementation of each semiring should be named `semiringN.py` where N is the number of the derivation semiring. (e.g., `semiring4.py`). In this section, you should keep in mind that the dynamic program stores the intermediate results of $\oplus$ in chart “cells”, one per LHS nonterminal per span $(i,j)$.

Like Section 3, you will not need to use a custom agenda comparator in this section. However, since the result of your semiring operators may not be limited to real numbers, it is important for you to keep in mind how our default agenda comparator works: it first checks to see if the value is numeric. If so, it simply uses it. Otherwise, it checks if it is iterable (tuples and lists fall into this category). If so, it uses the first numeric value in that iterable. Finally, if no numeric value can be found, it just uses the constant 1. **Hint:** It is therefore wise to keep any numeric value in the first element of your Python tuple, if the semiring requires a value.
Important clarifications (read after the main assignment text):

- Please do not add extra arguments to functions defined in starter.py. Since we will be running your code with an unmodified parser.py, your function definition needs to match what the parser expects or Python will raise an exception.

- Some questions below ask you about the space complexity (space questions). Please, ignore items stored in the agenda, compute only chart-related memory usage.

- Note that you may modify A, the word axiom just as you do R, the rule axiom. You may not need to redefine A for every semiring, so it is commented out in starter.py.

- Derivation Forest (semiring 4): For this problem, you are tracking all possible parses of the input sentence and keeping a set of all rules used in any parse. Valid parses cover the entire sentence span with a [S] non-terminal. You do not need to output constituent start and end positions, so if you encounter rule "[A] -> a" for two different indices, you only need to list the rule once. See the example output in 4.4. Another way to visualize this is that you are outputting a subset of the grammar where (1) any valid parse of the input can be built up using only rules from this subset and (2) every rule is part of at least one valid parse.

- Direct Viterbi semiring (semiring 5) To meet the requirements of the problem, you need to add another line to print the derivation. You may add exactly one additional pair of parentheses to join the left and right sides of your goal derivation during printing. This is the extent of post-processing that is required and allowed. The note "Each derivation should be formed entirely in the semiring" means that you may only modify the semiring definitions explicitly listed in the homework: semiZero, semiOne, semiPlus(), semiTimes() and R. All computation and formatting must take place within the semiring. You may not modify any other code.

- Some semirings require you to iterate over rules or derivations to print them. Namely, you are supposed to modify the script starter.py so that it produces output in the correct format. For semiring 4, you may format rules during printing. For semiring 6, you may add parentheses as in semiring 5, but you may NOT truncate the derivation list at print-time. Tracking the k-best derivations is part of the semiring. For semiring 7, you define a new backtrace() function that you call during printing. As stated in the directions, you may not perform string concatenation or formatting in your semiring. These are the only operations that are allowed outside of semiring definitions.

- The examples given in the problem descriptions are intended to demonstrate the output format only. They are not necessarily correct answers.

- For section 5, you are asked to transform output parse trees into trees that are consistent with the original grammar. To do this, you are writing your own script from scratch. You may use anything in the Python standard library. The resulting trees should be properly formed with respect to terminals, non-terminals, and parentheses. Here the example output is a correct example. Given the same input, your output should be character-by-character identical. To visualize what’s going on, you can draw out the parse trees to visualize the transformations. You can also draw out pairs of parse trees in the original grammar and CNF grammar that have chains of X non-terminals to think about how this is represented in LISP format.
Semiring 4: Derivation Forest (all rules that can participate in a complete parse)
This result of a parse under this semiring is the set of all unique rules that can participate in a full
derivation of the sentence. Your final output should contain lines of the following format for each
sentence:

SENT 0 AGENDA ADDS: 35
SENT 0 RULE: [X37] -> [VP] [.]
SENT 0 RULE: [NNP] -> Ms.
SENT 0 RULE: [.] -> .

You will receive points for “working code” based on grepping your solution’s output for “AGENDA
ADDS:”, “RULE:” and comparing against the correct answers. **Hint:** You may find the Python
set() data structure useful for maintaining unique collections. Also, the pipe operator | is useful
for producing the union of sets.

- **Space Question:** How many rule instances might be stored in memory for each sentence?
  Express your answer in terms of the input sentence length $L$, the number of nonterminals in
  the grammar $X$ and the number of rules in the grammar $R$.

Semiring 5: “Direct” Viterbi Derivation
You may break any ties by taking the first derivation you encounter. You may not use backpointers.
Each derivation should be formed entirely in the semiring. Remember that operator $\odot$ is not
commutative (see above). **Hint:** You can make good use of Python tuples here. Also, notice that
Python’s max() function works on tuples as long as the first item in the tuple is a float. Finally, if
you wish to add some additional type information to your tuples, don’t be afraid to embed strings
such as “RULE” or “AXIOM” inside your tuples.

Your final output should contain lines of the following format for each sentence:

SENT 0 AGENDA ADDS: 35
SENT 0 GOAL SCORE: 2.76059169625e-08
([NP] ([NNP] Elianti)))) ([.] .)))

You will receive points for “working code” based on grepping your solution’s output for “AGENDA
ADDS:”, “GOAL SCORE:”, and “GOAL DERIVATION:” and comparing against the correct an-
swers.

- **Space Question:** How many characters might be stored in memory for each sentence?
  Express your answer in terms of the input sentence length $L$, the number of nonterminals in
  the grammar $X$, the maximum number of characters in any word $W$, the maximum number
  of characters in any nonterminal $N$. You may ignore spaces, parens, and square brackets.
- **Time Question:** In terms of the length of the sentence, how many string concatenation
  operations might happen for each sentence in the worst case? You can use big O notation.
**Semiring 6:** Viterbi K-Best Derivations

A parse under this semiring will return the $k$ highest-scoring derivations under the grammar. Note that there may be less than $k$ derivations, which is fine. Use $k = 10$. By definition, a k-best list never has same derivation with the same score. Your final output should contain lines of the following format for each sentence:

```
SENT 0 AGENDA ADDS: 35
SENT 0 GOAL SCORE K=0: 2.76059169625e-08
```

You will receive points for “working code” based on us grepping your solution’s output for “AGENDA ADDS:”, “GOAL SCORE”, and “GOAL DERIVATION” and comparing against the correct answers. Obviously, you should have multiple “GOAL SCORE” and “GOAL DERIVATION” lines for sentences that have more than one possible derivation with lines numbered K=0 through K=9.

**Hint:** You may find the syntax `list(set(myList))` helpful for making the elements of a list unique. Also, the `+` operator may be used for concatenating lists.

- **Space Question:** How many (partial) derivations might be in memory for each sentence? Express your answer in terms of the input sentence length $L$, the number of nonterminals in the grammar $X$, and $K$.

**Semiring 7:** Viterbi Derivation with Backpointers

You may break any ties by taking the first derivation you encounter. You must use backpointers of the form (nonterm, i, j). You may not perform any string concatenation nor formatting inside your semiring – you should instead write a separate `backtrace()` algorithm that produces the derivation in the format described above. You need to figure out, which arguments this function should accept. **Hint:** The `parse()` function returns the chart to you, which can be indexed as `chart[lhs][([startPos, endPos])]`. Don’t be afraid to return more complex data structures using your semiring. You can add anything as long as you do not perform string concatenation or formatting inside the semiring.

Your final output should contain lines of the following format for each sentence:

```
SENT 0 AGENDA ADDS: 35
SENT 0 GOAL SCORE: 2.76059169625e-08
SENT 0 GOAL BACKPOINTERS: [('NP-SBJ', 0, 2), ('[X37]', 2, 5)]
```

Goal backpointers include only backpointers participating in the last derivation. Please, refer to the above example.

You will receive points for “working code” based on us grepping your solution’s output for “AGENDA ADDS:”, “GOAL SCORE:”, “GOAL BACKPOINTERS:” and “GOAL DERIVATION:” and comparing against the correct answers.

- **Space Question:** How many backpointers might be stored in memory for each sentence? Express your answer in terms of the input sentence length $L$ and the number of nonterminals in the grammar $X$. 

5 Reverse CNF Transformation [10 points]

Although our parser requires grammars to be in Chomsky normal form, we would like to produce output that is more consistent with linguistically motivated parse trees written out by human annotators. To accomplish this, you will write a Python script to reverse the CNF conversion on the trees output by the parser to make them consistent with the original grammar extracted from the Penn Treebank. You can identify which nonterminals came from the CNF transformation because they have the form Xn where n ≥ 1 (see cnf-convert.py for more details).

Your script will read parses, one per line, from standard in. Parses will be in the format specified for the Viterbi derivation semiring. Your script will apply the reverse transformation, then write parses in the same format to standard out. The parses you will be processing are provided in the file trees.out.

Example input line:

```plaintext
```

Example output line:

```plaintext
```

You should submit your solution as uncnf.py.

- [7 points] Submit your working code as uncnf.py. It will be run with the following command line:
  ```plaintext
  python uncnf.py < trees.out > trees.original
  ```

- [3 points] Is transforming the parse trees sufficient to reverse the CNF transformation or do you also need to recalculate the derivation scores? Why or why not?

6 Agenda Pruning [10 points]

In this section, you will explore a very basic method of performing approximate inference: pruning items from the agenda. The basic idea is that we can predict which items are not likely to be useful in producing a final solution and avoid processing them at all. Again, start with starter.py and modify the prune() method.

You will again use the left-to-right agenda ordering with increasing span sizes and will experiment with the provided Viterbi score semiring.

Answer the following questions:

1. [2 points] Using agenda pruning, are we guaranteed to produce a parse if one exists? Why or why not?

2. [2 points] Using agenda pruning, are we guaranteed to produce the correct score under this semiring? Why or why not?

3. [3 points] Implement a simple pruning function: prune any agenda item with value less than 1e-12. What is the number of agenda-add operations when you apply this pruning function?
Do you produce parses for all sentences? Are the final scores identical to the true scores (without pruning)? Why might pruning with a constant threshold be a bad idea? Submit your solution as `pruning1.py`.

4. [3 points] Now implement a slightly smarter pruning function that uses average rule weight: prune any agenda item with value less than $0.01725 \rho$ where $\rho$ is the number of rules applied in the derivation thus far. Recall that for a given span length, there is a closed form solution for the number of rules applied in a CFG in Chomsky normal form. What is the number of agenda-add operations when you apply this pruning function? Do you produce parses for all sentences? Are the final scores identical to the true scores (without pruning)? Submit your solution as `pruning2.py`.

**Submitting Your Code**

**Note:** The submission site for this homework will be online after Nov 9th.

In addition to submitting two hard copies of your assignment, all code for this assignment should be submitted at `http://demo.clab.cs.cmu.edu:8265/turnin/`. You should submit a single `.tar.gz` file containing exactly the following files:

- `semiring[4-7].py`
- `uncnf.py`
- `pruning*.py`

Your tar file should be called `submission.tar.gz` and should not contain any internal directory structure. You can create such a tarball using:

```
tar -cvzf submission.tar.gz semiring[4-7].py uncnf.py pruning*.py
```