Homework 2

The assignment is to implement the chart parser described in Handout 4. Submit a single python file named `hw2.py`.

1. Your parser should behave as follows:

```
>>> from hw1 import Grammar
>>> from hw2 import *
>>> g = Grammar('g0')
>>> p = Parser(g)
>>> trees = p('I book a flight in May'.split())
>>> len(trees)
2
>>> for s in sorted(str(t) for t in trees):
...     print(s)
...
(S
  (NP I)
  (VP
    (V book)
    (NP
      (NP
        (Det a)
        (N flight))
      (PP
        (P in)
        (NP May)))))
(S
  (NP I)
  (VP
    (VP
      (V book)
      (NP
        (Det a)
        (N flight)))
    (PP
      (P in)
      (NP May))))
```

Some notes:

- The input to the parser is a list of strings (words), not a single string (sentence).
- To get the parser to support the function call syntax, define the special method `__call__()`.

- Be sure that the words, chart, and edge table get created fresh each time the parser is called. Do not let information from one sentence bleed over into the next.
- The output must be a list of `Tree` instances (Handout 2, HW 1).

2. The specification for `Node` is given in Handout 4 #12. For debugging convenience, define `__repr__()` so that a `Node` prints as `[`*i cat j*`]`. For example:

```
>>> d = Node('Det', 'the', 0, 1)
>>> n = Node('N', 'dog', 1, 2)
>>> np = Node('NP', [d, n], 0, 2)
>>> np
[0 NP 2]
>>> np.cat
'NP'
>>> np.i
0
>>> np.j
2
>>> d.expansions
['the']
>>> np.expansions
[[[0 Det 1], [1 N 2]]]
```

3. The chart should be stored in the parser's `chart` attribute. It should be a `dict` containing `Node` instances, indexed by (X, i, j), where X is the category, i is the start position, and j is the end position.

```
>>> p.chart['NP', 0, 1]
[0 NP 1]
>>> ('S', 2, 5) in p.chart
False
```

4. The specification for `Edge` is given in Handout 4 #13. For debugging convenience, define `__repr__()` so that an `Edge` prints out as shown in the following example:

```
>>> from hw1 import Rule
>>> r = Rule('NP', ['Det', 'N'])
>>> r
NP -> Det N
>>> e = Edge(r, [d])
>>> e
(NP -> [0 Det 1] * N)
>>> e.rule
NP -> Det N
```

```
>>> e.expansion
[[0 Det 1]]
```

5. The edge table should be in the `Parser` attribute `edges`. Edges are indexed by (k, Z), where the edge is looking for a node with category Z and start position k. Edges with the dot at the end are not stored in the table.

```
>>> p.edges[4, 'PP']
[(VP -> [1 VP 4] * PP), (NP -> [2 NP 4] * PP)]
>>> p.edges[4, 'X']
[]
```

6. The parser `__call__()` should also accept an optional argument `tracing`. If `True`, the parser should print out:

- `Add Node` *node expansion,* for each node that it adds to the chart,
- `Add Expansion` *node expansion,* for each expansion that it adds to an existing node, and
- `Add Edge` *edge,* for each edge that it adds to the edge table.

For example:

```
>>> trees = p('I book a flight'.split(), tracing=True)
Add Node [0 NP 1] I
Add Edge (S -> [0 NP 1] * VP)
Add Edge (NP -> [0 NP 1] * PP)
Add Node [1 N 2] book
Add Node [1 V 2] book
Add Edge (VP -> [1 V 2] * NP)
Add Node [2 Det 3] a
Add Edge (NP -> [2 Det 3] * N)
Add Node [3 N 4] flight
Add Edge (NP -> [2 Det 3] [3 N 4] *)
Add Node [2 NP 4] [[2 Det 3], [3 N 4]]
Add Edge (VP -> [1 V 2] [2 NP 4] *)
Add Node [1 VP 4] [[1 V 2], [2 NP 4]]
Add Edge (S -> [0 NP 1] [1 VP 4] *)
Add Node [0 S 4] [[0 NP 1], [1 VP 4]]
Add Edge (VP -> [1 VP 4] * PP)
Add Edge (S -> [2 NP 4] * VP)
Add Edge (NP -> [2 NP 4] * PP)
```

7. Test the parser on `g1`. Make sure it parses the sentences in `g1.sents`.