Regular Expression Class based on OpenFST

The goal of this exercise is to write a small regular expression class that internally uses OpenFST to perform the matching.

In [ ]:
```
class OpenRE:
    def __init__(self,regex=None,cost=0.0):
        if regex is not None:
            self.add(regex,cost)
            self.compile()
        # IMPLEMENT ME
    def add(self,regex,cost=0.0):
        """Add a regular expression to the overall
        regular expression using a disjunction."""
        # IMPLEMENT ME
    def compile(self):
        """After adding component regular expressions,
        compile the internal fst."""
        # IMPLEMENT ME
        self.fst = something
    def cost(self,s):
        """Match the given string against the compiled
        regular expression and return the cost. Returns
        `inf` if there is no match."""
        # IMPLEMENT ME
        return cost
```

Your package should understand the following expressions:

- "ABC" - simple strings
- "AB|CD" - alternation
- "AB*C" - regex star (zero or more repeats)
- "AB+C" - regex plus (one or more repeats)
- "A(B|C)*D" - parentheses and optional operators

Assume that expressions are implicitly anchored at the beginning and end (no partial matches).

It's OK if you limit yourself to ASCII strings. Use `ord` to encode characters to integers. Do not worry about escape characters or wildcards.

Unit Tests

Write a set of unit tests demonstrating that your code works.

In [ ]:
```
assert OpenRE("abc").cost("abc") == 0
assert OpenRE("abC").cost("abc") == inf
assert OpenRE("ab").cost("abc") == inf # no anchoring
assert OpenRE("(a|b)").cost("a") == 0
assert OpenRE("(a|b)").cost("b") == 0
assert OpenRE("a|b").cost("a") == 0
# etc.
```

Parsing

For parsing the regular expression itself, you may want to use the `pyparsing` module.

Here is a simple example of how you might go about this. Note that this is *not* a correct regular expression parser yet and that you may want to generate a different kind of structure.

Read the documentation to figure out how to deal with whitespace and more characters.

In [ ]:
```
from pyparsing import *
postfix = Literal('+') | Literal('*')
alt = Literal( '|' )
lpar  = Literal( '(' ).suppress()
rpar  = Literal( ')' ).suppress()
lit = Regex('[^()|+*]+')
expr = Forward()
term = lit | alt + expr | Group( lpar + expr + rpar + Optional(postfix) )
expr << ZeroOrMore( term  )
expr.parseString("hello, (world|there)+|(a(b)c)")
```

In [ ]:
```

```