

usepackagefontspecDejaVu Sans Mono

utt

August 31, 2024

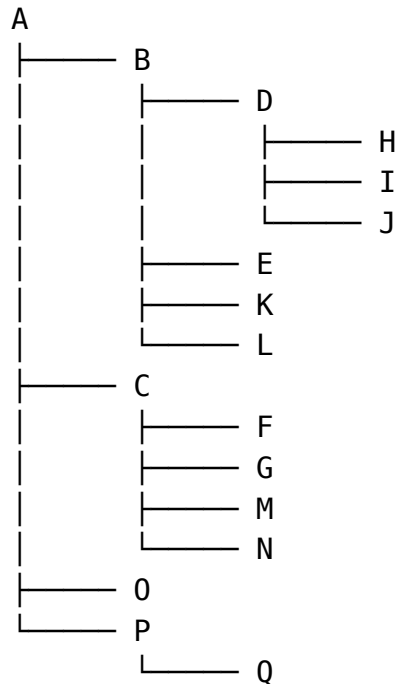
1 Universal Tree Traversal

We defined a grammar for a so-called universal tree traversal (UTT) grammar in `README.md`. This grammar is a relatively simple and contrained language that can be used to describe most tree traversals. In this notebook, we show how to use a simple parser to parse a UTT grammar and execute the traversal.

```
[1]: from treeprog.utt_eval import UttEval
import AlgoTree

def pp_results(res):
    for k, ns in res.items():
        print(f"{k}: {[n.name for n in ns]}")

tree = AlgoTree.FlatForest(
    {
        "A": { "data": "Data for A" },
        "B": { "data": "Data for B", "parent": "A" },
        "C": { "data": "Data for C", "parent": "A" },
        "D": { "data": "Data for D", "parent": "B" },
        "E": { "data": "Data for E", "parent": "B" },
        "F": { "data": "Data for F", "parent": "C" },
        "G": { "data": "Data for G", "parent": "C" },
        "H": { "data": "Data for H", "parent": "D" },
        "I": { "data": "Data for I", "parent": "D" },
        "J": { "data": "Data for J", "parent": "D" },
        "K": { "data": "Data for K", "parent": "B" },
        "L": { "data": "Data for L", "parent": "B" },
        "M": { "data": "Data for M", "parent": "C" },
        "N": { "data": "Data for N", "parent": "C" },
        "O": { "data": "Data for O", "parent": "A" },
        "P": { "data": "Data for P", "parent": "A" },
        "Q": { "data": "Data for Q", "parent": "P" }
    }
)
print(AlgoTree.pretty_tree(tree))
```



```
[2]: pre_order = [
      {"visit": "true", "result-name": "pre-order"},
      {"follow": "down", "select-order": "shuffle"},
    ]
    tree_eval = UttEval(False)
    pp_results(tree_eval(tree.root, pre_order))
```

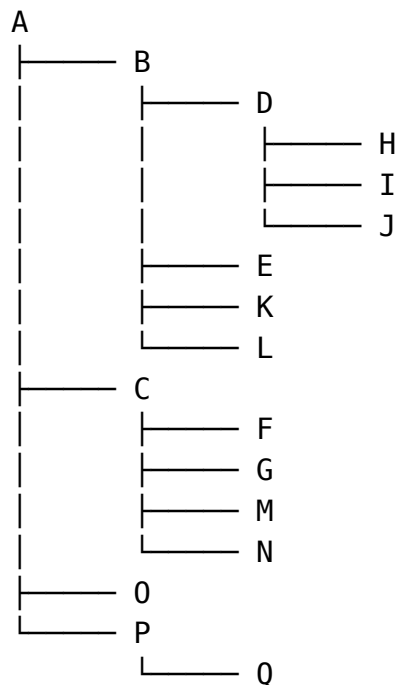
```
pre-order: ['A', 'B', 'D', 'H', 'J', 'I', 'E', 'L', 'K', 'O', 'C', 'M', 'N',
            'G', 'F', 'P', 'Q']
```

```
[3]: print(AlgoTree.pretty_tree(tree))
    post_order = [
        {"follow": "down"},
        {"visit": "true", "result-name": "post-order"}
    ]

    single_path = [
        {"visit": "true", "result-name": "a-random-root-to-leaf"},
        {"follow": "down",
         "select": {"name": "sample", "kwargs": {"n": 1}}}
    ]

    pp_results(tree_eval(tree.root, post_order))
    pp_results(tree_eval(tree.root, single_path))
```

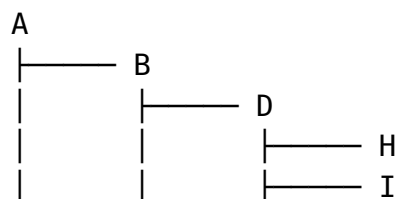
```
pp_results(tree_eval(tree.root, single_path))
pp_results(tree_eval(tree.root, single_path))
```

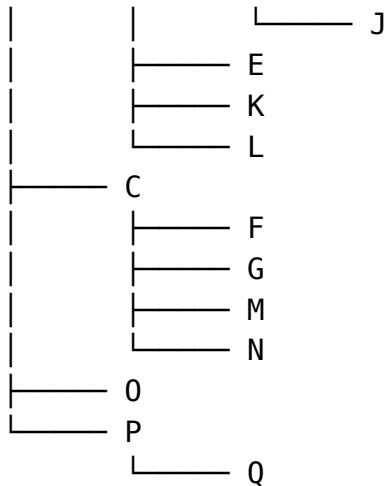


```
post-order: ['H', 'I', 'J', 'D', 'E', 'K', 'L', 'B', 'F', 'G', 'M', 'N', 'C', 'O', 'Q', 'P', 'A']
a-random-root-to-leaf: ['A', 'C', 'M']
a-random-root-to-leaf: ['A', 'P', 'Q']
a-random-root-to-leaf: ['A', 'O']
```

The next example is a little unusual, as we have two follow actions: **up** and **down**. Normally, you only see one, but we are mixing them together to show how the parser can handle it.

```
[4]: B = tree.node("B")
custom_order = [
    {"follow": "up"},
    {"visit": "true", "result-name": "custom-order"},
    {"follow": "down"}
]
print(AlgoTree.pretty_tree(tree))
pp_results(tree_eval(B, custom_order))
```





```
custom-order: ['A', 'B', 'D', 'H', 'I', 'J', 'E', 'K', 'L', 'C', 'F', 'G', 'M', 'N', 'O', 'P', 'Q']
```

This may not have the expected behavior for more complex **down** actions. Let's only sample a single node for the down action to see a simple example.

```
[5]: custom_order = [
    {"follow": "up"},
    {"visit": "true", "result-name": "custom-order"},
    {"follow": "down", "select": {"name": "sample", "kwargs": {"n": 1}}}
]

pp_results(tree_eval(B, custom_order))
```

```
custom-order: ['A', 'B', 'K', 'E']
```

```
[6]: custom_order_2 = [
    {"visit": "true", "result-name": "custom-order-2"},
    {"follow": "down", "select": {"name": "sample", "kwargs": {"n": 2}}, "select-order": "shuffle"}
]

pp_results(tree_eval(tree.root, custom_order_2))
```

```
custom-order-2: ['A', 'O', 'C', 'F', 'G']
```

```
[7]: cond_order = [
    {
        "cond": [
            {
                "pred": "less?",
```

```

        "args": ["$depth", 2],
        "order": [
            { "visit": "true", "result-name": "shallow-nodes"},
            { "follow": "down"}
        ]
    },
    {
        "pred": "true",
        "order": [
            { "visit": "true", "result-name": "deep-nodes"},
            { "follow": "down"}
        ]
    }
]

tree_eval_debug = UttEval(False)
pp_results(tree_eval_debug(tree.root, cond_order))

```

```

shallow-nodes: ['A', 'B', 'C', 'O', 'P']
deep-nodes: ['D', 'H', 'I', 'J', 'E', 'K', 'L', 'F', 'G', 'M', 'N', 'Q']

```

```

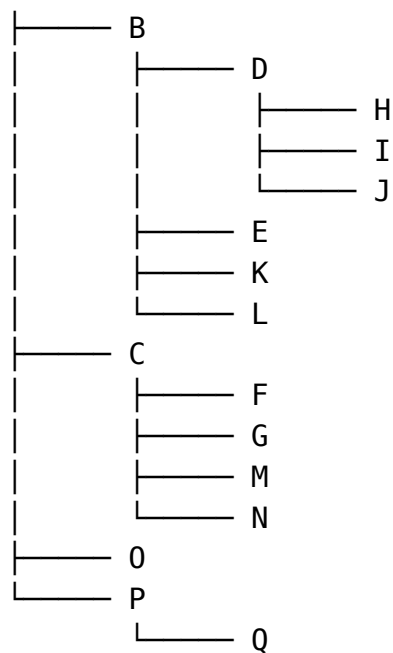
[8]: cond_pre_order = [
    {
        "cond": [
            {
                "pred": "true",
                "order": [
                    { "visit": "true", "result-name": "preorder"},
                    { "follow": "down"}
                ]
            },
        ],
    }
]

# this is the same as `pre_order`, but when we only have a single_
# automatic condition
# that is always true, we can just use the order directly

tree_eval_debug = UttEval(False)
print(AlgoTree.pretty_tree(tree))
pp_results(tree_eval_debug(tree.root, cond_order))

```

A



shallow-nodes: ['A', 'B', 'C', 'O', 'P']

deep-nodes: ['D', 'H', 'I', 'J', 'E', 'K', 'L', 'F', 'G', 'M', 'N',
 ↳ 'Q']