

Go deh!

Mainly Tech projects on Python and Electronic Design Automation.

Thursday, April 02, 2009

(Ab)use of Python's in-built data structures

I like to use Python's in-built data structures quite a lot, and tend to force myself to ask whether I should create my own classes, which allows you to use meaningful names for fields and add comments/docstrings to the datastructure but usually at the cost of adding more lines of text.

After writing about Huffman codes and posting solutions to http://www.rosettacode.org/w/index.php?title=Huffman_codes&oldid=27804#Python Rosetta code and <http://paddy3118.blogspot.com/2009/03/huffman-encoding-in-python.html> my blog:

```

29 color="#804040">def color="#008080">codecreate(symbol2weights, tutor= False):
color="#804040"> 30 ''' color="#ff00ff"> Huffman encode the given dict mapping symbols to weights '''
color="#804040"> 31 global decode
color="#804040"> 32
color="#804040"> 33 heap = [ [float(wt), [[sym, []]], repr(sym)] color="#804040">for sym, wt color="#804040">in symbol
color="#804040"> 34 heapify(heap)
color="#804040"> 35 if tutor: color="#804040">print " color="#ff00ff">ENCODING:", sorted(symbol2weights.iteritems())
color="#804040"> 36 while len(heap) >1:
color="#804040"> 37     lo = heappop(heap)
color="#804040"> 38     hi = heappop(heap)
color="#804040"> 39     color="#804040">if tutor: color="#804040">print " color="#ff00ff"> COMBINING:", lo, ' color='
color="#804040"> 40     color="#804040">for i color="#804040">in lo[1]: i[1].insert(0, ' color="#ff00ff">0')
color="#804040"> 41     color="#804040">for i color="#804040">in hi[1]: i[1].insert(0, ' color="#ff00ff">1')
color="#804040"> 42     lohi = [ lo[0] + hi[0] ] + [lo[1] + hi[1]]
color="#804040"> 43     lohi.append(' color="#ff00ff">(s if nextbit() else %s)' % (hi[2], lo[2]))
color="#804040"> 44     color="#804040">if tutor: color="#804040">print " color="#ff00ff"> PRODUCING:", lohi, ' color
color="#804040"> 45     heappush(heap, lohi)
color="#804040"> 46 wt, codes, decoder = heappop(heap)
color="#804040"> 47 decode = eval(' color="#ff00ff">lambda : ' + decoder, globals())
color="#804040"> 48 decode.__doc__ = decoder
color="#804040"> 49 for i color="#804040">in codes: i[1] = ''.join(i[1])
color="#804040"> 50 #for i in codes: i[::] = i[:2]
color="#804040"> 51 return sorted(codes, key= color="#804040">lambda x: (len(x[-1]), x))
color="#804040"> 52

```

Someone posted a [href="http://www.rosettacode.org/w/index.php?title=Huffman_codes&oldid=27804#Java"](http://www.rosettacode.org/w/index.php?title=Huffman_codes&oldid=27804#Java) Java solution, that was over three times as long. Instead of just counting and comparing line counts, I took a closer look to see what the extra code was for.

```

import java.util.*;

color="#2e8b57">abstract color="#2e8b57">class HuffmanTree color="#2e8b57">implements Comparable<HuffmanTree> {
color="#2e8b57">public color="#2e8b57">int frequency; color="#0000ff">// the frequency of this tree
color="#2e8b57">public HuffmanTree( color="#2e8b57">int freq) { frequency = freq; }

color="#0000ff">// compares on the frequency
color="#2e8b57">public color="#2e8b57">int compareTo(HuffmanTree tree) {
color="#804040">return frequency - tree.frequency;
}
}

color="#2e8b57">class HuffmanLeaf color="#2e8b57">extends HuffmanTree {
color="#2e8b57">public color="#2e8b57">char value; color="#0000ff">// the character this leaf represents

color="#2e8b57">public HuffmanLeaf( color="#2e8b57">int freq, color="#2e8b57">char val) {
color="#2e8b57">super(freq);
value = val;
}
}

color="#2e8b57">class HuffmanNode color="#2e8b57">extends HuffmanTree {
color="#2e8b57">public HuffmanTree left, right; color="#0000ff">// subtrees

color="#2e8b57">public HuffmanNode(HuffmanTree l, HuffmanTree r) {

```

```

        color="#2e8b57">super(l.frequency + r.frequency);
        left = l;
        right = r;
    }
}

color="#2e8b57">public color="#2e8b57">class HuffmanCode {
    color="#0000ff">// input is an array of frequencies, indexed by character code
    color="#2e8b57">public color="#2e8b57">static HuffmanTree buildTree( color="#2e8b57">int[] charFreqs) {
        PriorityQueue<HuffmanTree> trees = color="#804040">new PriorityQueue<HuffmanTree>();
        color="#0000ff">// initially, we have a forest of leaves
        color="#0000ff">// one for each non-empty character
        color="#804040">for ( color="#2e8b57">int i = color="#ff00ff">0; i < charFreqs.length; i++)
            color="#804040">if (charFreqs[i] > color="#ff00ff">0)
                trees.offer( color="#804040">new HuffmanLeaf(charFreqs[i], ( color="#2e8b57">char)i));

        color="#804040">assert trees.size() > color="#ff00ff">0;
        color="#0000ff">// loop until there is only one tree left
        color="#804040">while (trees.size() > color="#ff00ff">1) {
            color="#0000ff">// two trees with least frequency
            HuffmanTree a = trees.poll();
            HuffmanTree b = trees.poll();

            color="#0000ff">// put into new node and re-insert into queue
            trees.offer( color="#804040">new HuffmanNode(a, b));
        }
        color="#804040">return trees.poll();
    }
}

color="#2e8b57">public color="#2e8b57">static color="#2e8b57">void printCodes(HuffmanTree tree, Stack<Character> prefix) {
    color="#804040">assert tree != color="#ff00ff">null;
    color="#804040">if (tree color="#804040">instanceof HuffmanLeaf) {
        HuffmanLeaf leaf = (HuffmanLeaf)tree;

        color="#0000ff">// print out character and frequency
        System.out.print(leaf.value + color="#ff00ff">"\t color="#ff00ff">" + leaf.frequency + color="#ff00ff">"\t colc

        color="#0000ff">// print out code for this leaf, which is just the prefix
        color="#804040">for ( color="#2e8b57">char bit : prefix)
            System.out.print(bit);
        System.out.println();
    }
    color="#804040">else color="#804040">if (tree color="#804040">instanceof HuffmanNode) {
        HuffmanNode node = (HuffmanNode)tree;

        color="#0000ff">// traverse left
        prefix.push( color="#ff00ff">'0');
        printCodes(node.left, prefix);
        prefix.pop();

        color="#0000ff">// traverse right
        prefix.push( color="#ff00ff">'1');
        printCodes(node.right, prefix);
        prefix.pop();
    }
}

color="#2e8b57">public color="#2e8b57">static color="#2e8b57">void main(String[] args) {
    String test = color="#ff00ff">"this is an example for huffman encoding";

    color="#0000ff">// we will assume that all our characters will have
    color="#0000ff">// code less than 256, for simplicity
    color="#2e8b57">int[] charFreqs = color="#804040">new color="#2e8b57">int[ color="#ff00ff">256];
    color="#0000ff">// read each character and record the frequencies
    color="#804040">for ( color="#2e8b57">char c : test.toCharArray())
        charFreqs[c]++;

    color="#0000ff">// build tree
    HuffmanTree tree = buildTree(charFreqs);

    color="#0000ff">// print out results
    System.out.println( color="#ff00ff">"SYMBOL\t color="#ff00ff">WEIGHT\t color="#ff00ff">HUFFMAN CODE");
    printCodes(tree, color="#804040">new Stack<Character>());
}
}

```

I

noticed that the Java was tidy, and that they had defined their own classes for a HuffmanLeaf structure used when constructing a HuffmanTree.

In my original Python, I used a nested list as the equivalent to the HuffmanLeaf of the Java example. It worked, but I had to introduce 'magic constants' to access the fields of the Python leaf, which is also un-named as a structure in the program (line 33 of

the Python code creates a list of Leaf structures):

```
33 heap = [ (float(wt), [sym, []]) for sym, wt in symbol2weights.iteritems() ]
```

Now I didn't want to go to the trouble of creating my own

classes, but that's where the new

<http://docs.python.org/dev/py3k/library/collections.html#collections.namedtuple> namedtuple class factory of Python 2.6 came to the rescue.

namedtuple

namedtuple

will generate a subtype of the tuple class that allows the fields of the tuple to be named and accessed via subscription, [], or as if the fields are instance variable names.

I modified my Python program to use two named tuples:

1. For the Leaf structure as a whole in line 3.
2. For a component of the Leaf structure that holds the symbol and the Huffman code (accumulated so far), in line 4.

```
1 color="#a020f0">from collections import namedtuple
color="#804040"> 2
color="#804040"> 3 Leaf = namedtuple(' Leaf', 'weight, symbols')
color="#804040"> 4 SH = namedtuple(' SH', 'sym, huff')
color="#804040"> 5
color="#804040"> 6 def color="#008080">codecreate2(symbol2weights, tutor= False):
color="#804040"> 7     ''' color="#ff00ff"> Huffman codecreate2 the given dict mapping symbols to weights '''
color="#804040"> 8     heap = [ Leaf(weight=float(wt), symbols=[ SH(sym, []) ])
color="#804040"> 9         for sym, wt in symbol2weights.iteritems() ]
color="#804040">10     heapify(heap)
color="#804040">11     if tutor: color="#804040">print " color="#ff00ff">ENCODING:", sorted(symbol2weights.iteritems())
color="#804040">12     while len(heap) >1:
color="#804040">13         lo = heappop(heap)
color="#804040">14         hi = heappop(heap)
color="#804040">15         color="#804040">if tutor: color="#804040">print " color="#ff00ff"> COMBINING:", lo, ' color="#
color="#804040">16         color="#804040">for sh color="#804040">in lo.symbols: sh.huff.insert(0, ' color="#ff00ff">0')
color="#804040">17         color="#804040">for sh color="#804040">in hi.symbols: sh.huff.insert(0, ' color="#ff00ff">1')
color="#804040">18         lohi = Leaf(weight = lo.weight + hi.weight,
color="#804040">19             symbols = lo.symbols + hi.symbols)
color="#804040">20         color="#804040">if tutor: color="#804040">print " color="#ff00ff"> PRODUCING:", lohi, ' color=
color="#804040">21         heappush(heap, lohi)
color="#804040">22         symbols = heappop(heap).symbols
color="#804040">23         symbols = [SH(sym, ''.join(huff)) for sym, huff in symbols]
color="#804040">24     return sorted(symbols, key= color="#804040">lambda sh: (len(sh.huff), sh))
color="#804040">25
```

The

class generation is succinct in lines 3 and 4, and I use the field names for clarity for example when creating the original list of Leaves that will form the heap in line 8

Printing namedtuples

The print statements stay the same, but before you got output like this:

```
style="font-family: monospace;"> style="font-family: monospace;">...
```

```
COMBINING: [2.5, ['C', []]] style="font-family: monospace;">
```

```
AND: [5.0, ['A', []]]
```

```
PRODUCING: [7.5, ['C', ['0']], ['A', ['1']]] style="font-family: monospace;">
...
```

Now you get the clearer:

```
style="font-family: monospace;"> ENCODING:
(['A', '5'), ('B', '25'), ('C', '2.5'), ('D', '12.5')]

COMBINING: Leaf(weight=2.5, symbols=[SH(sym='C', huff=[])])

AND: Leaf(weight=5.0, symbols=[SH(sym='A', huff=[])])

PRODUCING: Leaf(weight=7.5, symbols=[SH(sym='C',
huff=['0']), SH(sym='A', huff=['1'])])

COMBINING: Leaf(weight=7.5, symbols=[SH(sym='C',
huff=['0']), SH(sym='A', huff=['1'])])

AND: Leaf(weight=12.5, symbols=[SH(sym='D', huff=[])])

PRODUCING: Leaf(weight=20.0, symbols=[SH(sym='C', huff=['0', '0']),
SH(sym='A', huff=['0', '1']), SH(sym='D', huff=['1'])])

COMBINING: Leaf(weight=20.0, symbols=[SH(sym='C', huff=['0', '0']),
SH(sym='A', huff=['0', '1']), SH(sym='D', huff=['1'])])

AND: Leaf(weight=25.0, symbols=[SH(sym='B', huff=[])])

PRODUCING: Leaf(weight=45.0, symbols=[SH(sym='C', huff=['0', '0',
'o']), SH(sym='A', huff=['0', '0', '1']), SH(sym='D', huff=['0', '1']),
SH(sym='B', huff=['1'])])
```

Conclusion

Although you can do so much with Python lists, if each position in the list has a fixed meaning then you might be best to use tuples, and if you should be using tuples, then namedtuples can make your code more readable without bloating it with long class definitions.

- Paddy.

Posted by Paddy3118 at 10:07 pm 

Reactions: interesting (0) reactionary (0)
waste of space (0) disinterested (0)

4 comments:




racitup Sun Jul 26, 11:46:00 pm

Please can you fix your html?!

Reply

Unknown Mon Sep 30, 06:18:00 am


 **Hi**
Have you implemented the decoder as well?
[Reply](#)

 **NieDzejko** **Fri Dec 13, 09:31:00 pm**
Hi. Some HTML problems made your post completely unreadable.
[Reply](#)

[Replies](#)

 **Paddy3118** **Sat Dec 14, 01:02:00 pm**
Yep. That's an issue with blogger and I haven't got around to reposting it. It was fine when posted, and I am aware. Thanks for your concern.

[Reply](#)

 **Comment as:** Ashok B (Goog ▾) Sign out

Publish Preview ☐ Notify me

Links to this post

Create a Link

[Newer Post](#)

[Home](#)

[Older Post](#)

Subscribe to: [Post Comments \(Atom\)](#)

About Me

Paddy3118

[View my complete profile](#)

Followers

Followers (16)

Subscribe Now: google



Go deh too!

whos.amung.us

Blog Archive

- [2019 \(8\)](#)
- [2018 \(10\)](#)
- [2017 \(10\)](#)
- [2016 \(3\)](#)

- ▶ 2015 (8)
 - ▶ 2014 (18)
 - ▶ 2013 (14)
 - ▶ 2012 (12)
 - ▶ 2011 (7)
 - ▶ 2010 (12)
 - ▼ 2009 (42)
 - ▶ December (1)
 - ▶ November (4)
 - ▶ October (3)
 - ▶ September (3)
 - ▶ August (4)
 - ▶ July (1)
 - ▶ June (4)
 - ▶ May (6)
 - ▼ April (6)
 - Fight for the community you want
 - Monitoring a linux process as it reads files
 - Bimap. Bi-directional mapping/dictionary for Pytho...
 - Knapsack solution by OO Calc
 - Memoization and stack use.
 - (Ab)use of Pythons in-built data structures
 - ▶ March (3)
 - ▶ February (5)
 - ▶ January (2)
 - ▶ 2008 (28)
 - ▶ 2007 (23)
 - ▶ 2006 (7)
 - ▶ 2005 (1)
-