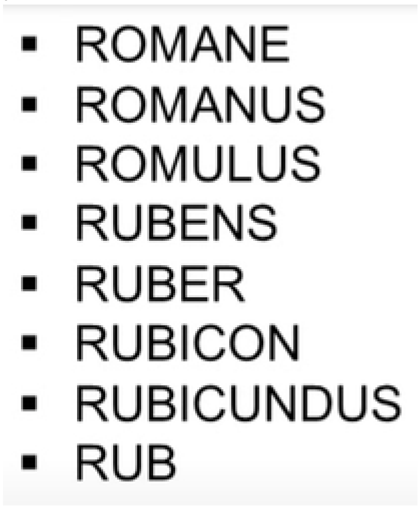
RLP, MPT

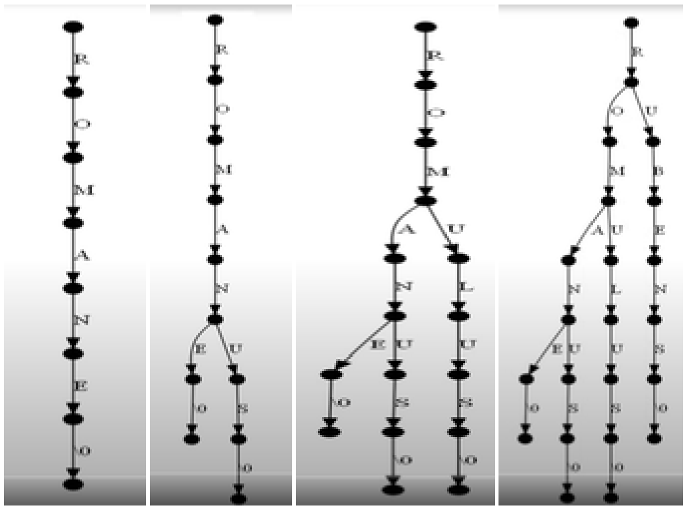
- RLP(Recursive Length Prefix) encoding
  - Data Types
    - String(bytes)
    - int(big-endian, no leading zeros)
  - Input
    - Encoding function takes an item, where
      - A string(byte array) is an item
      - A list of items is an item
  - Encoding

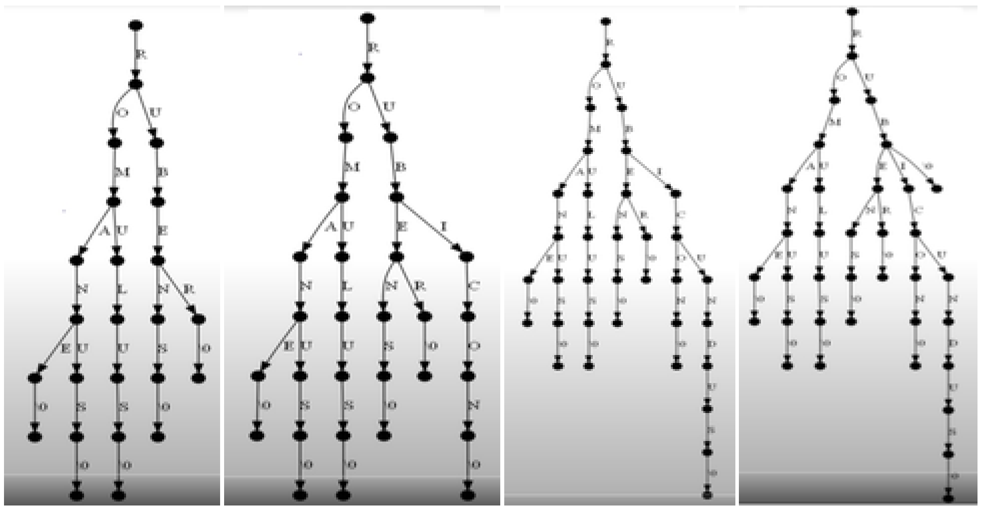
code range	code encoding	details	examples	example
0x00 ~ 0x7F	Single byte value	See ASCII table. http://www.asciitable.com/	a A	0x61 0x41
0x80 ~ 0xB7	Multiple byte value(0 ~ 55)	(First byte - 0x80) = length of following string	dog marshall (52 length)bos coin is the korea number one coin. go and buy it	0x83:64: 0x88:6D: 0xB4:62:
0xB8 ~ 0xBF	Multiple byte value(56 ~ 2**64-1)	(First byte - 0xB7) = length of bytes for encoding length of following string. Could be 1 ~ 8 2nd ~ 9th bytes = length of following string (encoded as int)	(57 length)bos coin is the korea number one coin. go and buy it dude "a" * 255 "a" * 256	0xB8:39: 75:64:65 0xB8:FF: 0xB9:01:
0xC0 ~ 0xF7	A list for a payload(0 ~ 55)	(First byte - 0xC0) = length of following payload	["d", "o", "g"] [ ] ["cat", "dog"] ["a"] * 55	0xC3:64: 0xC0 0xC2:83: 0xF7:61:
0xF8 ~ 0xFF	A list for a payload(56 ~ 2**64-1)	(First byte - 0xF7) = length of bytes for encoding length of following payload. Could be 1 ~ 8 2nd ~ 9th bytes = length of following payload (encoded as int)	["a"] * 56 ["a"] * 255 ["a"] * 256	0xF8:38: 0xF8:FF: 0xF9:01:
COMPLEX EXAMPLE: ["cat",["puppy","cow"],"horse",[[]],"pig",[""],"sheep"] → 0xC7:83:63:61:74:C2:85:70:75:70:70:79:83:63:6F:77:85:68:6F:72:73:65:C1:C0:83:70:69:67:C1				

```
• def rlp_encode(inp):
      if isinstance(inp, str):
          if len(inp) == 1 and ord(inp) < 0x80:
              return inp
          else:
              return encode_length(len(inp), 0x80) + inp
      elif isinstance(inp, list):
          output = encode_length(len(inp), 0xc0)
          for item in inp:
              output += rlp_encode(item)
          return output


  def encode_length(L, offset):
      if L < 56:
          return chr(L + offset)
      elif L < 256**8:
          BL = to_binary(L)
          return chr(len(BL) + offset + 55) + BL
      else:
          raise Exception("input too long")


  def to_binary(x):
      return '' if x == 0 else to_binary(int(x / 256)) + chr(x % 256)
```

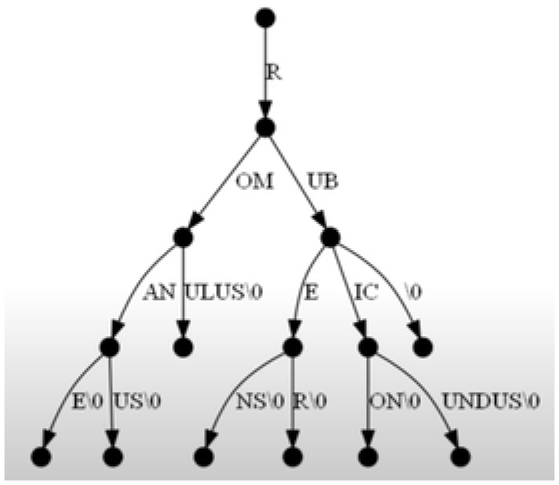
- Trie(Radix Tree)
    - A tree
        - A Node is a information node or a branch node.
        - A branch node can has 1 ~ M children, where M denotes the number of possible code
        - A information node can has only one child (null node)
        - Maximum depth of radix tree is M + 2(root, null)
        - Time complexity: O(logN) for inserts, lookups and deletes
    - Example)
        - 

          ▪ ROMANE
          ▪ ROMANUS
          ▪ ROMULUS
          ▪ RUBENS
          ▪ RUBER
          ▪ RUBICON
          ▪ RUBICUNDUS
          ▪ RUB



- Explain
    - If a code is alphabet(capital only) character then possible code range is 1 to 26 (a to z)
    - A branch node can has 2 to 26 children. Except ₩0(null) node.
    - A leaf node(or a node which has a ₩0 child node) shows a data by path from the root node to the node

- Patricia Trie
    - A tree
        - Same as Trie data structure with the exception of minor modification.
        - A edge can denotes more than one code
        - Space complexity of Patricia trie is better than trie
    - 

- Merkle-Patricia-Tree(MPT)
    - A tree
        - A node is one of bellows
            - null - represented as the empty string
            - branch - a 17-item node, where first 16 elements correspond to the 16 possible hex characters in a key, and final element hold a value
            - extension - a 2-item node, encodedPath and value(key), where value is a hash of some other node
            - leaf - a 2-item node, encodedPath and value
    - nibble
        - 4bit → 2**4 = 16 possible code
    - A branch node
        - has a list of 17 items.
    - A extension node/leaf node
        - has a list of 2 items
        - the first nibble of a first item has meanings
            - 0010 → leaf node
            - 0001 → extension node. nibbles[1:]. When a key of the node length is odd. remove first 1 nibble
            - 0000 → extension node. nibbles[2:]. When a key of the node length is even. remove first 2 nibbles

- Explain
    - Every item is treated as (key, value), where key is hash key and value is RLP encoded string
    - Put an item to MPT
        - First Put (0x010102)
            - key is divided into nibles. nibble is a 4bit data.
            - Append the nodetype nibble into front of the key. In this case, the nibble is 0010(leaf)
            - encode the node by RLP. And put (key, value) to the kv-store, where key is the hash of the value, and value is RLP encoded node
        - Replace (0x010102)
            - Do the same process above
        - Put node share common prefix (0x010103)
            - 0x010102 and 0x010103 share first five nibbles
            - Make a parent node of both nodes.
                - Key(hex prefix is 0x01010) is 0x101010. Because hp length is odd. add 0x1 into the front
                - Value is a hash for the branch node
            - Branch node has a 0x01010 HP implicitly
                - children nodes are placed in 2 and 3 position.
                - This should be a hash key for the node. But a node less than 32 bytes is placed at that position without hashing
        - ...
            - See bellow code and its result
            - https://github.com/kfangw/trie_example

```
from src import trie, rlp

(
    NODE_TYPE_BLANK,
    NODE_TYPE_LEAF,
    NODE_TYPE_EXTENSION,
    NODE_TYPE_BRANCH
) = tuple(range(4))


def traverse(hash, tab=0):
    if not hash:
        return
    if type(hash) is list:
        return
    node = state._decode_to_node(hash)
    node_type = state._get_node_type(node)
    node_type_str = ''
    if node_type == NODE_TYPE_BLANK:
        node_type_str = 'BLNK'
    elif node_type == NODE_TYPE_BRANCH:
        node_type_str = 'BRCH'
    elif node_type == NODE_TYPE_EXTENSION:
        node_type_str = 'EXTN'
    elif node_type == NODE_TYPE_LEAF:
        node_type_str = 'LEAF'
    print_node(node, hash.encode("hex"),
node_type_str, tab)

    if node_type == NODE_TYPE_EXTENSION:
        traverse(node[1], tab=tab+1)
    elif node_type == NODE_TYPE_LEAF:
        pass
    elif node_type == NODE_TYPE_BRANCH:
        for n in node[:15]:
            traverse(n, tab=tab+1)


def print_node(node, hash, node_type_str,
tab=0):
    print "{0}[{1}][{2}]{3}".format(" "*tab,
node_type_str, hash, [n.encode('hex') if type(n)
is not list and len(n) == 32 else n for n in
node]).upper()


print 'x01x01x02', ['VALUE_010102']
state = trie.Trie('triedb', trie.BLANK_ROOT)
state.update('\x01\x01\x02',
rlp.encode(['VALUE_010102']))
traverse(state.root_hash)
print '\n'

print 'x01x01x02', ['VALUE_010102']
state.update('\x01\x01\x02',
rlp.encode(['VALUE_010102_REPLACE']))
traverse(state.root_hash)
print '\n'

print 'x01x01x03', ['VALUE_010103']
state.update('\x01\x01\x03',
rlp.encode(['VALUE_010103']))
traverse(state.root_hash)
print '\n'

print 'x01x01', ['VALUE_0101']
state.update('\x01\x01',
rlp.encode(['VALUE_0101']))
traverse(state.root_hash)
print '\n'

print 'x01x01x02x55', ['VALUE_01010255']
state.update('\x01\x01\x02\x55',
rlp.encode(['VALUE_01010255']))
traverse(state.root_hash)
print '\n'

print 'x01x01x02x57', ['VALUE_01010257']
state.update('\x01\x01\x02\x57',
rlp.encode(['VALUE_01010257']))
traverse(state.root_hash)
print '\n'
print 'x01x01x03x57', ['VALUE_01010357']
state.update('\x01\x01\x03\x57',
rlp.encode(['VALUE_01010357']))
traverse(state.root_hash)
print '\n'

print 'x22x02x03x57', ['VALUE_22020357']
state.update('\x22\x02\x03\x57',
rlp.encode(['VALUE_22020357']))
traverse(state.root_hash)
print '\n'


print rlp.decode(state.get('\x01\x01'))
print rlp.decode(state.get('\x01\x01\x02'))
print rlp.decode(state.get('\x01\x01\x03'))
print rlp.decode(state.get('\x01\x01\x02\x55'))
print rlp.decode(state.get('\x01\x01\x02\x57'))
print rlp.decode(state.get('\x01\x01\x03\x57'))
print rlp.decode(state.get('\x01\x01\x03\x58'))
print rlp.decode(state.get('\x22\x02\x03\x57'))
```

-
  ```
  /Users/kfangw/.virtualenvs/trie/bin/python
  /Users/kfangw/workspace/trie_example/ex.py
  x01x01x02 ['VALUE_010102']
  [LEAF][687A900FD04F0161D8B94D143A6DB6F1D404D357A
  5AEE9AB3009DF7B06432354][' \X01\X01\X02',
  '\XCD\X8CVALUE_010102']


  x01x01x02 ['VALUE_010102']
  [LEAF][0CE6613DFFFDEFDB975B85B0FA85153031D1DE91A
  DBA954C196291831B24DCF7][' \X01\X01\X02',
  '\XD5\X94VALUE_010102_REPLACE']


  x01x01x03 ['VALUE_010103']
  [EXTN][60C58EABBA7BF7ABD37390224ED2E54F1E197F59A
  828BB046F6EC7F090B377C1]['\X10\X10\X10',
  '294A7E0E2EE3176F4B31C1431A6F270CEC68041756F077A
  D9C2D316A6E47931A']
   [BRCH][294A7E0E2EE3176F4B31C1431A6F270CEC680417
  56F077AD9C2D316A6E47931A]['', '', [' ',
  '\XD5\X94VALUE_010102_REPLACE'], [' ',
  '\XCD\X8CVALUE_010103'], '', '', '', '', '', '',
  '', '', '', '', '', '', '']


  x01x01 ['VALUE_0101']
  [EXTN][447B22BE286A680F4B589F05BB8CC47ED04D38FF1
  9414D2D46675A9F02771E30]['\X00\X01\X01',
  '47A9B3634EC6409AEC81AF2A50003A0EAB8FDAD1E8B58B2
  8390E9DCF7E3AF139']
   [BRCH][47A9B3634EC6409AEC81AF2A50003A0EAB8FDAD1
  E8B58B28390E9DCF7E3AF139]['294A7E0E2EE3176F4B31C
  1431A6F270CEC68041756F077AD9C2D316A6E47931A',
  '', '', '', '', '', '', '', '', '', '', '', '',
  '', '', '', '\XCB\X8AVALUE_0101']

  [BRCH][294A7E0E2EE3176F4B31C1431A6F270CEC6804175
  6F077AD9C2D316A6E47931A]['', '', [' ',
  '\XD5\X94VALUE_010102_REPLACE'], [' ',
  '\XCD\X8CVALUE_010103'], '', '', '', '', '', '',
  '', '', '', '', '', '', '']
  ```

```
x01x01x02x55 ['VALUE_01010255']
[EXTN][161ECBE0B5F7DFA6E5071BF7465225D2EEBCE14BF
FABAC72E3C75682918405A7]['\X00\X01\X01',
'7A9920557AFAAE6C0B32556581AEC1272268F9A204E4568
68BB502BA28523CB7']
 [BRCH][7A9920557AFAAE6C0B32556581AEC1272268F9A2
04E456868BB502BA28523CB7]['94A5ACB78A076F15FBABB
A7555FBA6AF35545A3808A055F39D3D5019A7BF5AA6',
'', '', '', '', '', '', '', '', '', '', '', '',
'', '', '', '\XCB\X8AVALUE_0101']

[BRCH][94A5ACB78A076F15FBABBA7555FBA6AF35545A380
8A055F39D3D5019A7BF5AA6]['', '',
'B906A1999DF68A93564DF79F66BD8BF131A9878C4F69444
0E2D466F57B61F897', [' ',
'\XCD\X8CVALUE_010103'], '', '', '', '', '', '',
'', '', '', '', '', '', '']

[BRCH][B906A1999DF68A93564DF79F66BD8BF131A9878C4
F694440E2D466F57B61F897]['', '', '', '', '',
['5', '\XCF\X8EVALUE_01010255'], '', '', '', '',
'', '', '', '', '', '',
'\XD5\X94VALUE_010102_REPLACE']


x01x01x02x57 ['VALUE_01010257']
[EXTN][95EEAD42EC2299166DBAA6B7829CECEE07E2456D1
DDDC0CEA374E6DD616EE66D]['\X00\X01\X01',
'7CDB5A8960C1BA18E8B46E5D1078C0EFD95FD3D9F459D62
85D82AEF02C5B43F3']
 [BRCH][7CDB5A8960C1BA18E8B46E5D1078C0EFD95FD3D9
F459D6285D82AEF02C5B43F3]['DAF4B952B63167235533A
430D3D9E72EA2F5EFD185CCE4C5BC9498AD3AEF859B',
'', '', '', '', '', '', '', '', '', '', '', '',
'', '', '', '\XCB\X8AVALUE_0101']

[BRCH][DAF4B952B63167235533A430D3D9E72EA2F5EFD18
5CCE4C5BC9498AD3AEF859B]['', '',
'6E3A703201033B60A06829C48CBCBEE065333C3B691C28E
A289C22377A27689E', [' ',
'\XCD\X8CVALUE_010103'], '', '', '', '', '', '',
'', '', '', '', '', '', '']

[BRCH][6E3A703201033B60A06829C48CBCBEE065333C3B6
91C28EA289C22377A27689E]['', '', '', '', '',
'79DAEE00D6CBD9B6F8DD1E37EC406A2B8506DE9AC16A400
96658C2A4F7BBF163', '', '', '', '', '', '', '',
'', '', '', '\XD5\X94VALUE_010102_REPLACE']
```

```
[BRCH][79DAEE00D6CBD9B6F8DD1E37EC406A2B8506DE9AC
16A40096658C2A4F7BBF163]['', '', '', '', '', ['
', '\XCF\X8EVALUE_01010255'], '', [' ',
'\XCF\X8EVALUE_01010257'], '', '', '', '', '',
'', '', '', '']


x01x01x03x57 ['VALUE_01010357']
[EXTN][C5E604E4805008F6AA00349CC59B682364BC52B49
484E43C04D895DC79D8474C]['\X00\X01\X01',
'EE4BF2735EF0F062D483750ED40CA2660584878B95D2DE7
37AE62EFB8A710047']
 [BRCH][EE4BF2735EF0F062D483750ED40CA2660584878B
95D2DE737AE62EFB8A710047]['94CC8761229D492F600FC
7599FE422A412CC89ECA2C8B05ADA6C4B6DB5C22AC4',
'', '', '', '', '', '', '', '', '', '', '', '',
'', '', '', '\XCB\X8AVALUE_0101']

[BRCH][94CC8761229D492F600FC7599FE422A412CC89ECA
2C8B05ADA6C4B6DB5C22AC4]['', '',
'6E3A703201033B60A06829C48CBCBEE065333C3B691C28E
A289C22377A27689E',
'112CBB736E83DD954D2FD1B0079F56F5A5013F44A002D99
DC57F1D79741C28DB', '', '', '', '', '', '', '',
'', '', '', '', '', '']

[BRCH][6E3A703201033B60A06829C48CBCBEE065333C3B6
91C28EA289C22377A27689E]['', '', '', '', '',
'79DAEE00D6CBD9B6F8DD1E37EC406A2B8506DE9AC16A400
96658C2A4F7BBF163', '', '', '', '', '', '', '',
'', '', '', '\XD5\X94VALUE_010102_REPLACE']

[BRCH][79DAEE00D6CBD9B6F8DD1E37EC406A2B8506DE9AC
16A40096658C2A4F7BBF163]['', '', '', '', '', ['
', '\XCF\X8EVALUE_01010255'], '', [' ',
'\XCF\X8EVALUE_01010257'], '', '', '', '', '',
'', '', '', '']

[BRCH][112CBB736E83DD954D2FD1B0079F56F5A5013F44A
002D99DC57F1D79741C28DB]['', '', '', '', '',
['7', '\XCF\X8EVALUE_01010357'], '', '', '', '',
'', '', '', '', '', '', '\XCD\X8CVALUE_010103']


x22x02x03x57 ['VALUE_22020357']
[BRCH][AD8CE09423C45DEE29672BC7AC49162A41F0E6E65
806F784AA34CF2D84F2F0B5]['CD7853F730A49FCDB6B58E
C49505AB2FFDC234234A711B507B475725E85EB383', '',
['2\X02\X03W', '\XCF\X8EVALUE_22020357'], '',
'', '', '', '', '', '', '', '', '', '', '', '',
'']
 [EXTN][CD7853F730A49FCDB6B58EC49505AB2FFDC23423
4A711B507B475725E85EB383]['\X11\X01',
'EE4BF2735EF0F062D483750ED40CA2660584878B95D2DE7
37AE62EFB8A710047']

[BRCH][EE4BF2735EF0F062D483750ED40CA2660584878B9
5D2DE737AE62EFB8A710047]['94CC8761229D492F600FC7
599FE422A412CC89ECA2C8B05ADA6C4B6DB5C22AC4', '',
'', '', '', '', '', '', '', '', '', '', '', '',
'', '', '\XCB\X8AVALUE_0101']

[BRCH][94CC8761229D492F600FC7599FE422A412CC89ECA
2C8B05ADA6C4B6DB5C22AC4]['', '',
'6E3A703201033B60A06829C48CBCBEE065333C3B691C28E
A289C22377A27689E',
'112CBB736E83DD954D2FD1B0079F56F5A5013F44A002D99
DC57F1D79741C28DB', '', '', '', '', '', '', '',
'', '', '', '', '', '']

[BRCH][6E3A703201033B60A06829C48CBCBEE065333C3B6
91C28EA289C22377A27689E]['', '', '', '', '',
'79DAEE00D6CBD9B6F8DD1E37EC406A2B8506DE9AC16A400
96658C2A4F7BBF163', '', '', '', '', '', '', '',
'', '', '', '\XD5\X94VALUE_010102_REPLACE']

[BRCH][79DAEE00D6CBD9B6F8DD1E37EC406A2B8506DE9AC
16A40096658C2A4F7BBF163]['', '', '', '', '', ['
', '\XCF\X8EVALUE_01010255'], '', [' ',
'\XCF\X8EVALUE_01010257'], '', '', '', '', '',
'', '', '', '']

[BRCH][112CBB736E83DD954D2FD1B0079F56F5A5013F44A
002D99DC57F1D79741C28DB]['', '', '', '', '',
['7', '\XCF\X8EVALUE_01010357'], '', '', '', '',
'', '', '', '', '', '', '\XCD\X8CVALUE_010103']


['VALUE_0101']
['VALUE_010102_REPLACE']
['VALUE_010103']
['VALUE_01010255']
['VALUE_01010257']
['VALUE_01010357']
None
['VALUE_22020357']
```

```
Process finished with exit code 0
```