**TERNARY SEARCH TREE**

Ternary search tree คือโครงสร้างต้นไม้เป็น trie (บางครั้งเรียก prefix tree) ที่เป็นโครงสร้างพิเศษที่ต่างจากโครงสร้างต้นไม้อื่นๆ จำนวน ซึ่งจัดอยู่ในลักษณะคล้ายกับ BST (binary search tree) แต่จะมี Child node มากกว่า BST 1 จำนวนTernary search tree สามารถใช้เป็นใครงสร้างแผนที่เชื่อมโยงกับความสามารถในการค้นหาสตริงที่เพิ่มขึ้น อย่างไรก็ตามการที่สามารถใช้เป็นโครงสร้างการค้นหามากขึ้นนั่นก็หมายถึงใช้พื้นที่เพิ่มมากขึ้นด้วย

**การดำเนินการของ TST**

* 1. ตัวชี้ซ้ายชี้ไปที่โหนดที่มีค่าต่ำกว่าค่าในโหนดปัจจุบัน **(LEFT NODE)**
* 2. ตัวชี้ที่เท่ากันชี้ไปที่โหนดที่มีค่าเท่ากับค่าในโหนดปัจจุบัน **(MID NODE or EQUAL NODE)**
* 3. ตัวชี้ด้านขวาชี้ไปที่โหนดที่มีค่ามากกว่าค่าในโหนดปัจจุบัน **(RIGHT NODE)**

นอกจากนี้จะต้องมี Node ที่เป็น **“Data”** และ Node ที่กำหนดจุดสิ้นสุด **“END”**ของข้อมูล ในกรณีที่ข้อมูลมีการซ้ำกัน

ตัวอย่างTST

c

/ | \

a u h

| | | \

t t e u

/ / | / |

s p e i s

จากกราฟนี้แสดงให้เห็นแต่ละโหนดที่ถูก insert หรือเพิ่มเข้ามาเป็นโครงสร้างของ TST โดยมีการเพิ่มข้อมูล "cute","cup","at","as","he","us" and "i" ตามลำดับ โดยข้อมูลที่ถูกเพิ่มเข้ามาก่อนนั้นจะเป็นกราฟ tree ที่อยู่ในตำแหล่งตรงกลาง และข้อมูลลำดับถัดไปหากลำดับตัวอักษรมีค่าน้อยกว่า parent node ก็จะเป็นกราฟ child node ที่อยู่ซ้ายมือ เช่นเดียวกันหากข้อมูลที่เพิ่มเข้ามามีค่าลำดับตัวอักษรมากกว่า parent node ก็จะอยู่ child node ขวามือ

CODE PYTHON คุณสมบัติของ NODE

class Node:  
 def \_\_init\_\_(self, data):  
 self.data = data  
 self.end = False  
 self.left = None  
 self.mid = None  
 self.right = None  
 def print\_(self):  
 return ''.join(['[', self.data,  
 ('' if not self.end else ' <end>'),  
 ('' if self.left is None else ' left: ' + self.left.print\_()),  
 ('' if self.mid is None else ' mid: ' + self.mid.print\_()),  
 ('' if self.right is None else ' right: ' + self.right.print\_()), ']'])

CODE PYTHON การเพิ่มข้อมูล

def add(self, node, string):  
 if len(string) == 0:  
 return node  
  
 head = string[0]  
 tail = string[1:]  
  
 if node is None:  
 node = Node(head)  
 if head < node.data:  
 node.left = self.add(node.left,string)  
 elif head > node.data:  
 node.right = self.add(node.right,string)  
 else:  
 if len(tail) == 0:  
 node.end = True  
 else:  
 node.mid = self.add(node.mid, tail)  
 return node

CODE PYTHON การค้นหาข้อมูล

def search(self, node, string):  
 if node is None or len(string) == 0:  
 return False  
  
 head = string[0]  
 tail = string[1:]  
 if head < node.data:  
 return self.search(node.left, string)  
 elif head > node.data:  
 return self.search(node.right, string)  
 else:  
 *# use 'and' for matches on complete words only,  
 # use 'or' for matches on string prefixes* if len(tail) == 0 and node.end:  
 return True  
 return self.search(node.mid, tail)