Discussion 5: Huffman tree, Heap, BST

CS101 Fall 2024

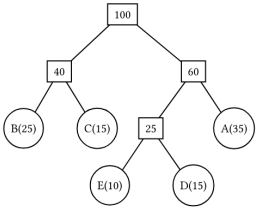
CS101 Course Team

Nov 2024

Huffman tree

motivation: minimize average code length after encoding. (Actually, Huffman coding is the optimal encoding method, which could be proved by greedy.)
Average length:

$$\overline{L} = \sum_{x} p(x)l(x)$$

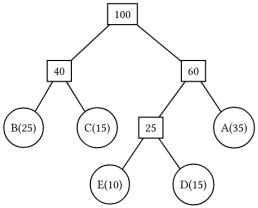


字符	频率	编码
A	35	11
B	25	00
C	15	01
D	15	101
E	10	100

Huffman tree

property:

- Huffman code is not unique(same probability, which side to be 0/1...)
- The variable with the longest length is not unique.



字符	频率	编码
A	35	11
B	25	00
C	15	01
D	15	101
E	10	100

Heap

- Min-heap: the value of every node in the subtree larger than the root.
- max-heap: the value of every node in the subtree smaller than the root.
- Operation: Top, Pop, Push.
- Pop: Remove objects. rotate every [small] key up.
- Push: add an object as a leaf, then rotate it up.
- **A naive binary heap generally may be incomplete, while a binary heap specifically refers to something that is complete**

Heap

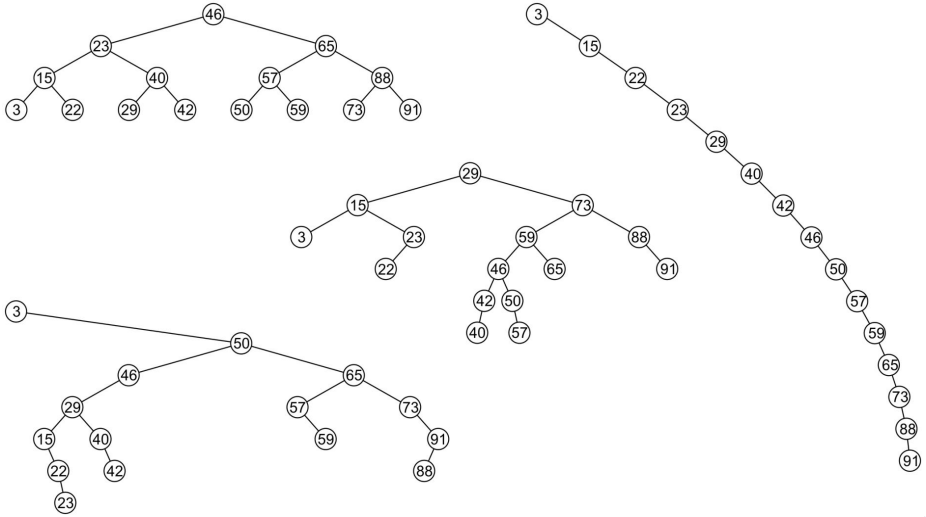
- Complete trees: the height is limited. so Time complexity will be $O(\log n)$
- Push operation needs to satisfy the rule of CT.
- Pop: copy the last entry in the heap to the root, then rotate it down.
- As it's a complete tree, we can store it using an array.

BST

- all objects in the left sub-tree to be less than the object stored in the root node
- all objects in the right sub-tree to be greater than the object in the root object
- the two sub-trees are themselves binary search trees

problem: The height of a BST is $O(n)$, to avoid this problem, we need to add more constrains to the structure, such as AVL-tree.

BST $O(n)$ height



BST Operations

- insert
  Search for the proper position to insert, and then add a new node.
- find
  Similarly with insert, without adding the new node.
- erase
  Search for the proper position, and use the precursor(biggest element among the nodes with smaller value) or successor(smallest element among the nodes with bigger value) of the deleted node to fill in. (In our lecture note, we use the successor for uniform)

BST Complexity

- Find: $O(h)$
- Insert: $O(h)$
- Erase: $O(h)$

If the tree is perfect, these complexities will be $O(\log n)$.
If the tree is closed to a linked list, these complexities will be $O(n)$.