## Indexed-Based Lists — Array

O(n)
O(N)

| | |
|---|---|
| size() | O(1) |
| get(i) | O(1) |
| set(i,e) | O(1) |
| add(i,e) | O(n) |
| remove(i) | O(n) |

## Positional lists

- 只能查 1st 和 last element
- 每个 node 知道它前一个和后一个是谁
- 通常是 Doubly linked list

## Stack

LIFO

- Implement by using
  - ① array: 有一个 pointer 指向最后 element
- Push and Pop take O(1)
  - ② singly linked list: push/pop happens at head, all take O(1)

# Queue FIFO

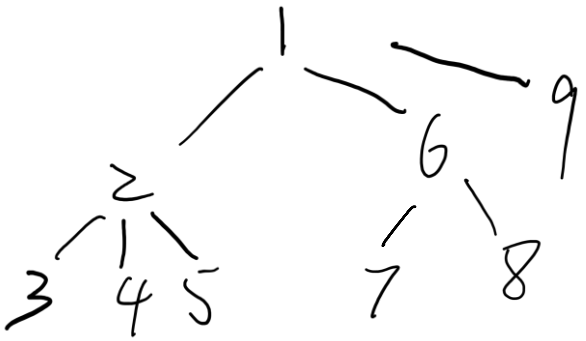
- Implement by
  1. Array: 有2个 var 分别记录最先和最后
  2. Singly linked list
     - 有一个 pointer 指向 singly linked list 的末尾
     - Enque 加在末尾 O(1)
     - Deque 加在 head O(1)

## General def of Tree

External node: leaf node

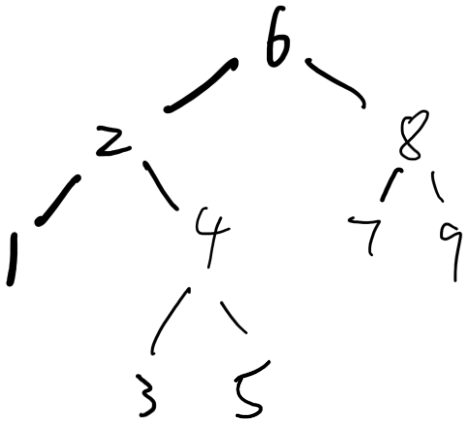
- Depth: number of ancestors not include itself. Root has 0 depth
- Height: maximum depth (数线  )

## Pre-order 中左右



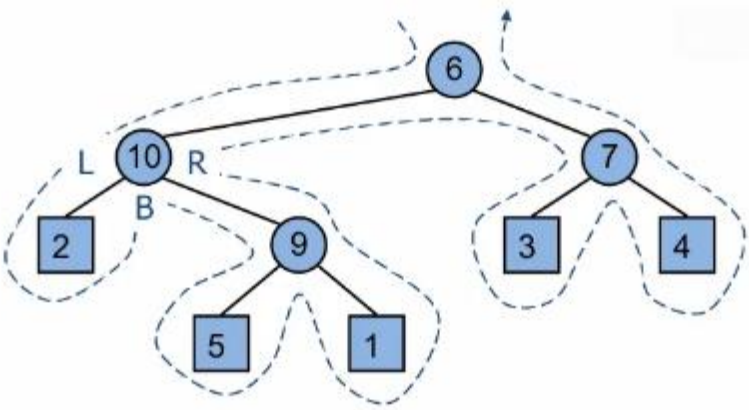
**Post-order** 左右中


**Inorder** 左中右 only works for binary Tree




**Euler Tour Traversal** only work for binary Tree

- leaf node 连续访问3次
- 其它 node 间接访问3次




6,10,2,2,2,10,9,5,5,5,9,1,1,1,9,10,6,7,3,3,3,7,4,4,4,7,6

1st 1 2 3 1 2 3 2nd 3rd look order

6,10,2,2,2,10,9,5,5,5,9,1,1,1,9,10,6,7,3,3,3,7,4,4,4,7,6

Preorder (first visit): 6, 10, 2, 9, 5, 1, 7, 3, 4

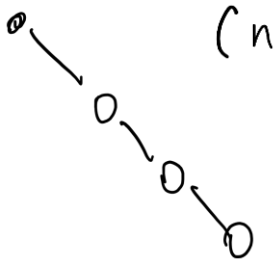
Inorder (second visit): 2, 10, 5, 9, 1, 6, 3, 7, 4

Postorder (third visit): 2, 5, 1, 9, 10, 3, 4, 7, 6

---

Binary Tree

- 0 or 2 children for proper binary Tree
- Worst case (not proper)



# BST — unbalanced

- Finding() O(h) could be o(n), since unbalance
  if reach an external node, means Tree does not have it
- Inserting() O(h) could be o(n)
  always insert to leaf node
- Deletion() O(h) could be O(n)
  use right tree 最小值 replace current value
  如果没有 right child, 就用当前 node 的 left child 作为新的

## Range Query in BST

- return all nodes less than $k_1$, large than $k_2$
  - a. If key(k) < $k_1$, Recursively search right subtree
  - b. If $k_1$ < key(k) < $k_2$, add current node to output, recursively call left and right sub tree respectively
  - c. If key(k) > $k_2$, recursively search left subtree

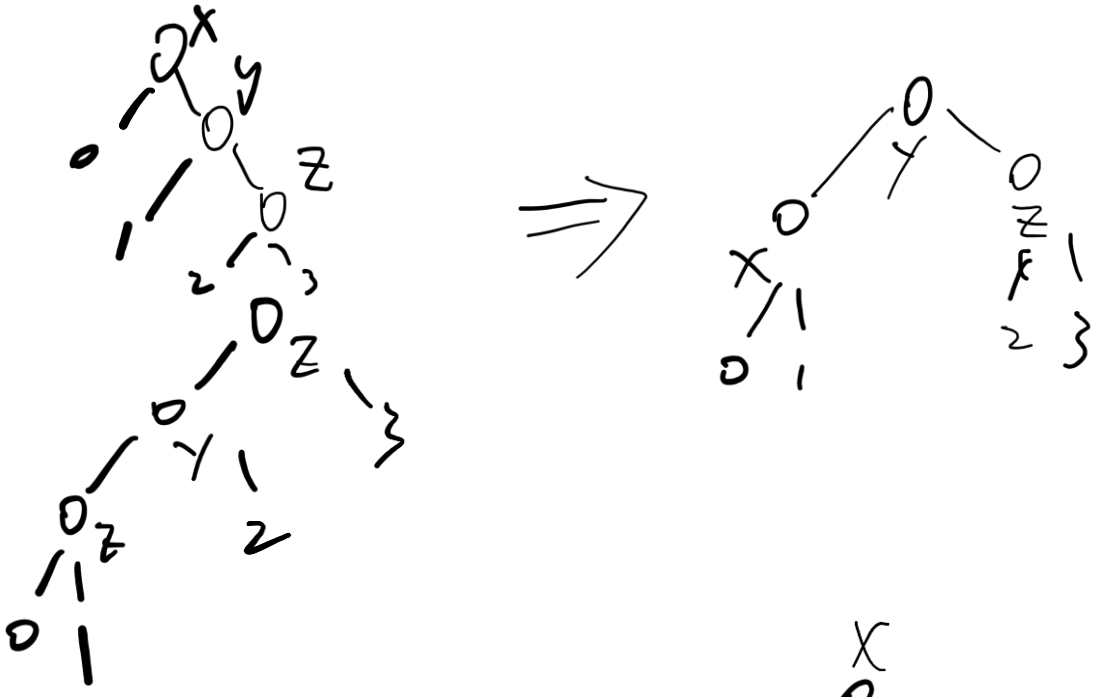
O(|size of output| + tree height)

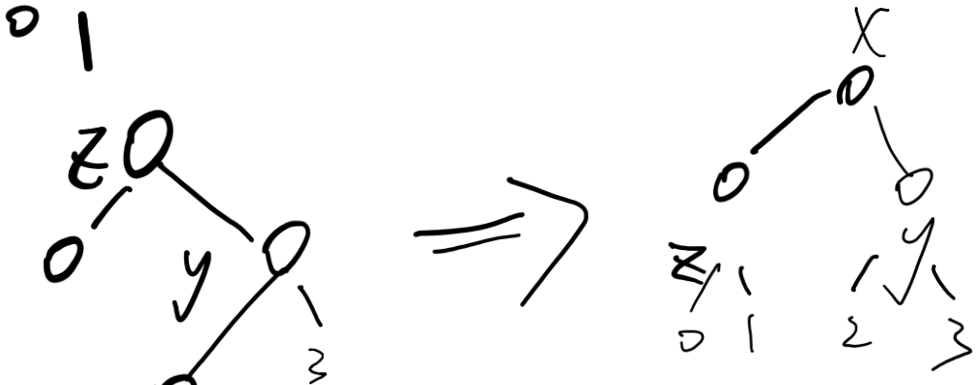
## AVL

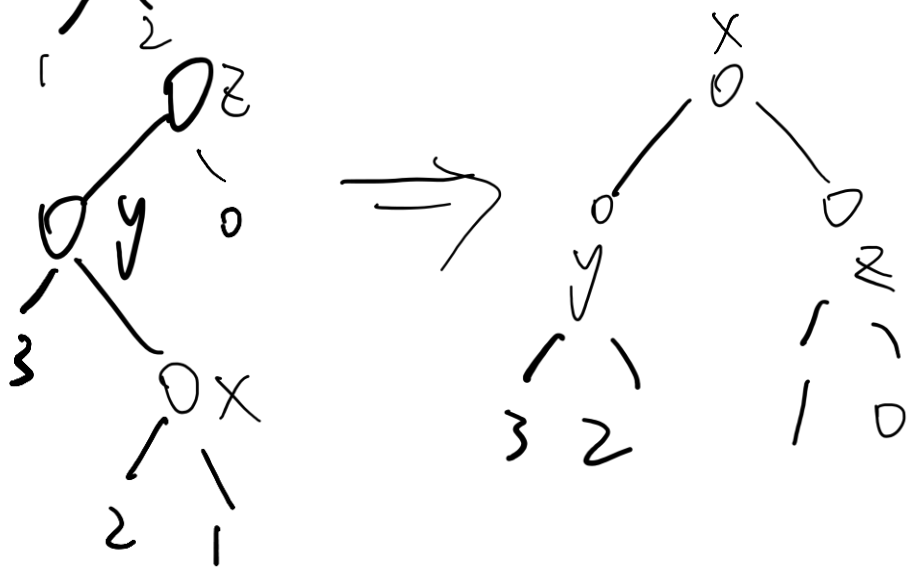
O(log n) for all operation

- Balanced BST . need have new variable height for each node

- The ranks of the two children of every internal node differ by at most 1
- 关键 step, rebalancing tree after every operation







- **Delete**: 1. let w be the parent of deleted node

$O(\log(n))$

2. let z be w's lowest ancestor
3. let y be the child of z with larger height
4. Let x be the child of y with large height
5. rotate

- **Insert**: $O(\log(n))$
  - let w be the newly inserted node
  - let z be the ancestor of w, whose subtree height differ by 2
  - let y be the child of z and ancestor of w
  - let x be the child of x and ancestor of w
  - rotate

**List-based Map**:

↳ Double LL

O(n) for all operation, not very good

# Tree-Based Map

- AVL
- O(logn) for all operation

## Heap

它是 priority queue 的模板

- Heap sort() O(nlogn) 因为有n个element need to 执行 insert + remove_min

- A complete binary tree，所以不支持 search

- Properties: Construct takes O(n)
  1. The last node is the rightmost node of maximum depth
  2. Root is always the smallest

3. Search any element (not min) takes O(n)

4. 最后一层前的所有 level 都是满的

5. Height is $\log_2 n$

6. have pointer indicate to last node

Insertion $O(\log n)$

step screenshot

Remove $O(\log n)$ :

- Remove the root with the last node
- Remove last node
- Downheap + Search
  ↓ find new last node pointer

Selection Sort $O(n^2)$

bad than heap sort

merge sort

**Insertion Sort** $O(n^2)$

**Map** put(k,v) 会替换掉旧有 key

- 主要 time complexity 由 collision solving method 决定

**Separate chaining** $O(n)$

- array of singly indexed list

**Linear Probing** $O(n)$ 从 h(k) 开始往后找次对 value

- single array implement
- rule 抄印出来

**Cuckoo hashing** $O(1)$ if no evicting cycle

- Evicting cycle. 头尾和开头连上了

· 两个 Hash function: $H_1, H_2$

两个 Hash table $T_1, T_2$

Map 没有 iter 的功能，但我们可以同步 creat 一个 doubly linked list 来记录 insert order

**Graph**

m: # of edges

n: # of vertices

adjacent vertices are connected

· Simple graph: no parallel edge, no self-loops — can have cycles

**Connected graph** → only have one connected component

: You can reach any vertex from any other vertex without leaving graph

**Cut edge**: if removed, would increase the number of connected components in a graph

## Degree

deg(v)

undirected diagram:

# of edges of a vertex

directed

in-degree

# of the vertex →

out-degree

# of in vertex ←

- Tree is connected, no cycles

## property

① $\sum_{v \text{ in } V} deg(v) = 2m$ 每一条 edge 算了2次

② In a simple undirected graph $m \leq n(n-1)/2$

Implementation

. screenshot

Linked list
- Edge list: each vertex doesn't know the edge
- Adjacency list: each vertex knows the edge

space O(n+m)

Adjacency Matrix:

space $O(n^2)$

DFS $O(m+n)$ → 用 adjacency list

DFS edge: an edge used to discover a new vertex

back edge: not a DFS Edge

· 从图的某一个 vertex 开始尽可能 explore 直到 stuck，再回到上一个 node 探索 new branch

We use DFS to identify cut Edges: $O(n+m)$

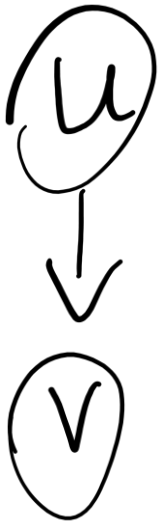
1. using DFS form a tree

only DFS-Tree edge can be cut edge

2. Compute height for each node in the tree (level)

3. For each node, 我们尽可能下降并进行一个 edge up 能达到的最高高度 (down_and_up)

4. compare down_and_up(v) & level(u)




edge(u,v) is not a cut edge

$\Updownarrow$

$down\_and\_up(v) \leq level(u)$

is a cut edge

$\Updownarrow$

$down\_and\_up(v) > level(u)$

#不能是≥ 因为会连到u本身

#两点uv不能disconnect graph

# BFS

$O(m+n)$ 没有考虑 weight of edge,

**BFS-Edge**:

if an edge is used to discover a **new vertex**

**Cross Edge**: connect two vertices, both have already been visited

$L_0$ the start point, only has one vertex

**Property**

For each $v$ in $L_i$, there is a path in $T_s$ from $s$ to $v$ with $i$ edges

- $T_s$: Spanning Tree
- $i$: 因为在 $L_i$

Use BFS to find shortest path from S to destination $O(n+m)$

## Bipartie Graph

a undirected graph can be divided into two sets A and B such that no 2 vertices within same set are adjacent

## Weighted Graph

MST 和 SPT 都是 Spanning Tree，而 Spanning Tree 一定包含所有 n 个顶点和 n-1 条边

- Each edge has an associated numeric value
- Using Dijkstra method

Goal: to find shortest path from root to every other vertex

The tree called SPT

- Time complexity is $O(n^2)$ for array
  $O(m \log n)$ for heap

$O(m + n\log n)$ for Fibonacci heap

- Not work for negative weight edge

## MST

minimum spanning Tree

- Goal: form a T whose sum of edge weight is minimum. (include every vertex)

- Cut - S — a subset of vertices in G
- Cutset - D — a collection of edges with exactly one vertex in cut-S
- Cycle C — a collection of edges

## Properties

(1) Cut Property

Lets be any subset of node

② Cycle Property

③ A cut set interact

Prim's Algorithm 从一点开始，每次找 visited vertex 最近的 neighbor

heap $O(m \log m)$

$O(m + n \log m)$ Fibonacci heap

m 最大为 $n^2$

Kruskal's Algorithm 每次找最小的 edge

↳ $O(n^2)$ when use union find

Greedy Algo

usually $O(n \log n)$ because we need to sort first

① Task Scheduling

$O(n \log n)$

- sort according to task beginning time
- need to assign tasks that finish first in the sorted array

$O(n + d \log d)$

② Text compression - Huffman Encoding

· 0表示不同字符的放置 · 0左1右

· " " 也占一个字符

· frequency 越大的字符越靠近 root

**property**

· non-leaf node : sum of frequency of its sub tree

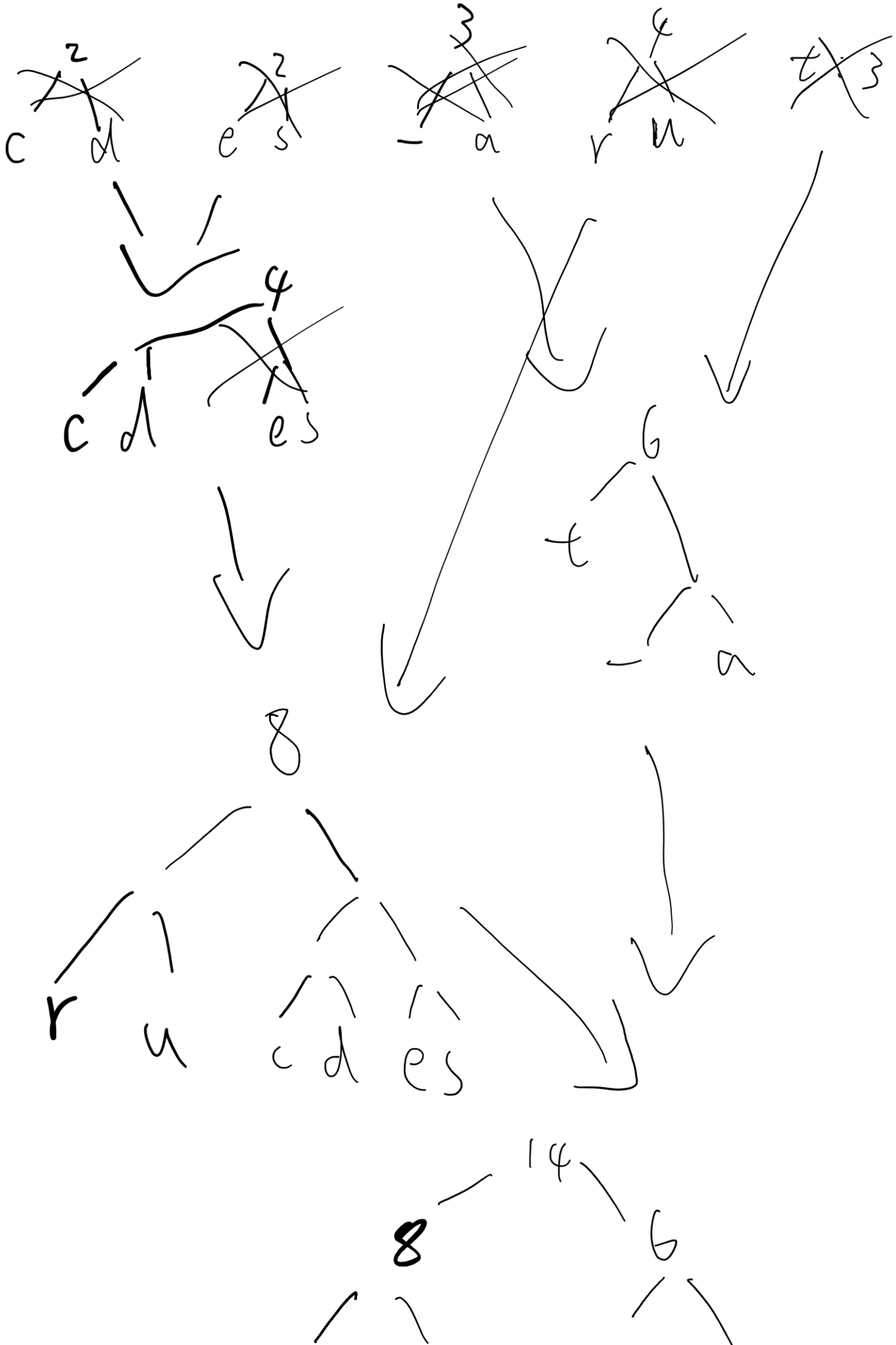
· No code word is a prefix of another code word

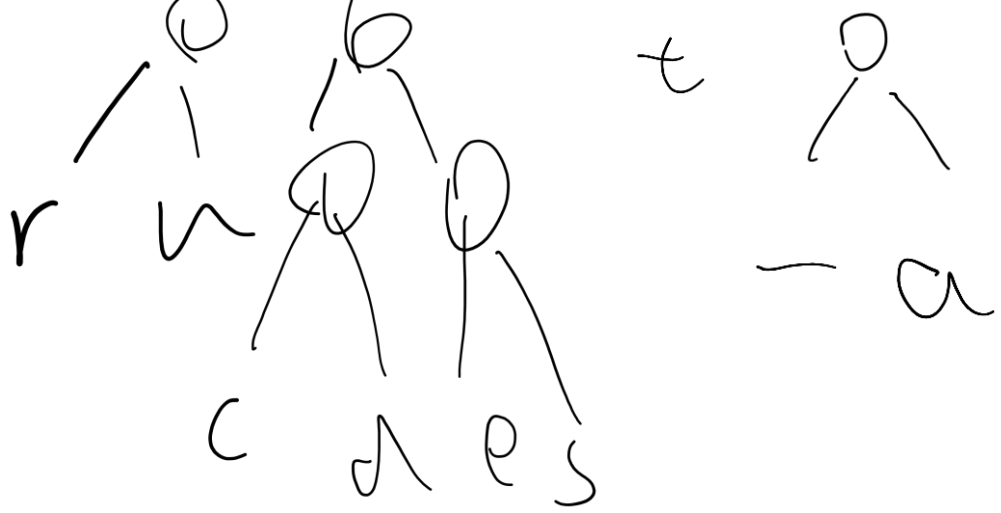
· 最深的 leaf node 的 frequency 最大

· $depth_i(a) < depth_i(b) \rightarrow f(a) \geq f(b)$

data " " s r u c e
r u
a t
t

每次取最小的两个 node :


c
d
e
s
—
a
r
u
t
3
2
4
6
8
14



# Divide and conquer

1. Table of $T(\ ) + O(\cdots)$
2. Unrolling ☆
3. master Theorem ☆
4. Binary search { linked list $O(n)$ / sorted list $O(\log n)$

⑤ Merge Sort $O(n \log n)$

⑥ Quick sort Worst $O(n^2)$

⑦ Majority in a Array

Can be done in $O(n)$

if we choose to discard half elements in each level

⑧ Maxima-Set (Pareto frontier) $O(n \log n)$

· Core: 找出 ls and rs 的 y-coord

⑨ $x \cdot y$ V 2.0 ☆ $O(n^{(\log_2 3)})$

Care about unrolling tree

(10) Find the $k^{th}$ smallest number in A $O(n)$

(11) Find the $k^{th}$ smallest number in

A and B $O(\log(m+n))$

↓ ↓

m n

Randomised Algorithm 不是重点...