Data Structure

資料結構

Author:第9組
林楷傑　郭佳翰　宮能翔　陳文鑫　黃鵬宇　康譯文　吳欣誠

什麼是資料結構

- 電腦組織、儲存資料的方式
- 對於不同用途選擇不同的資料結構（演算法、複雜度）
- 資料結構是一種工具
- 排序、分類、紀錄、分析

Abstract

- Computing Complexity
- Array
- Dynamic Array
- Linked List
- Stack
- Queue
- Tree

Computing Complexity

計算複雜度

- 如何表示電腦的計算速度？哪一種演算法比較好？
- 衡量程式的效率（時間、空間）
- 做N次動作 花了N時間
- Ex: N^3 + 2*N^2 + N => O(N^3)
- O(n), O(n^2) ,O(log(N))表示計算量 常數用O(1)表示

```
n = 10000
for(int i=0;i<n; i++)
    for(int j=0;j<n;j++)
        cout<<"讓我看看"<<endl;
```

Ex:如果輸出“讓我看看”的時間是k秒 那我就需要O(n^2)的時間 k*n^2秒

Array

陣列

- 一次儲存多個變數
- 優點：儲存快、拿取也快都是O(1)
- 缺點：大小被限制、刪除速度慢、插入速度慢 O(n)

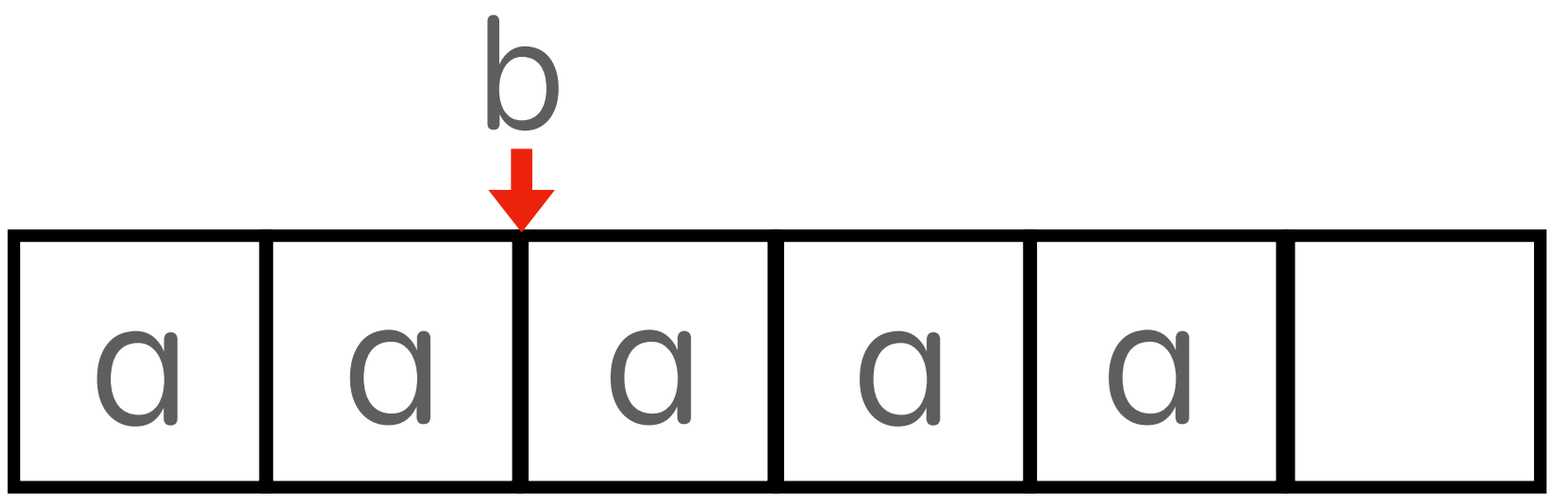
```
1  int a[5] = {0,1,2,3,4};
2  a[0] = 99;
3  for(int i=0;i < 5;i++){
4      cout<<a[i]<<" ";
5  }
6  #output: 99 1 2 3 4
```

宣告陣列

Question

問題大了

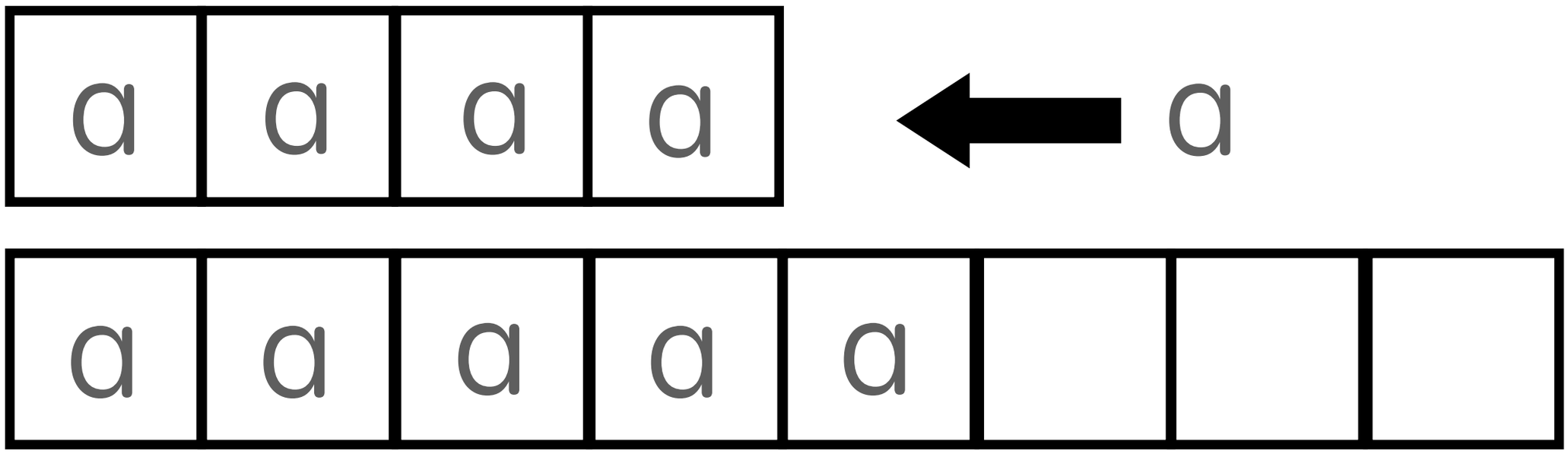
- 如何解決陣列大小被限制的問題？
- 如何解決插入速度很慢的問題？



Dynamic Array

動態陣列

- 可伸縮陣列
- 記憶體空間以2的冪次增加（不是無限增加）
- 優點：繼承陣列拿取快、存取快的優點，解決了大小限制問題
- 缺點：插入問題還是沒有解決



擴增兩倍！

Dynamic Array

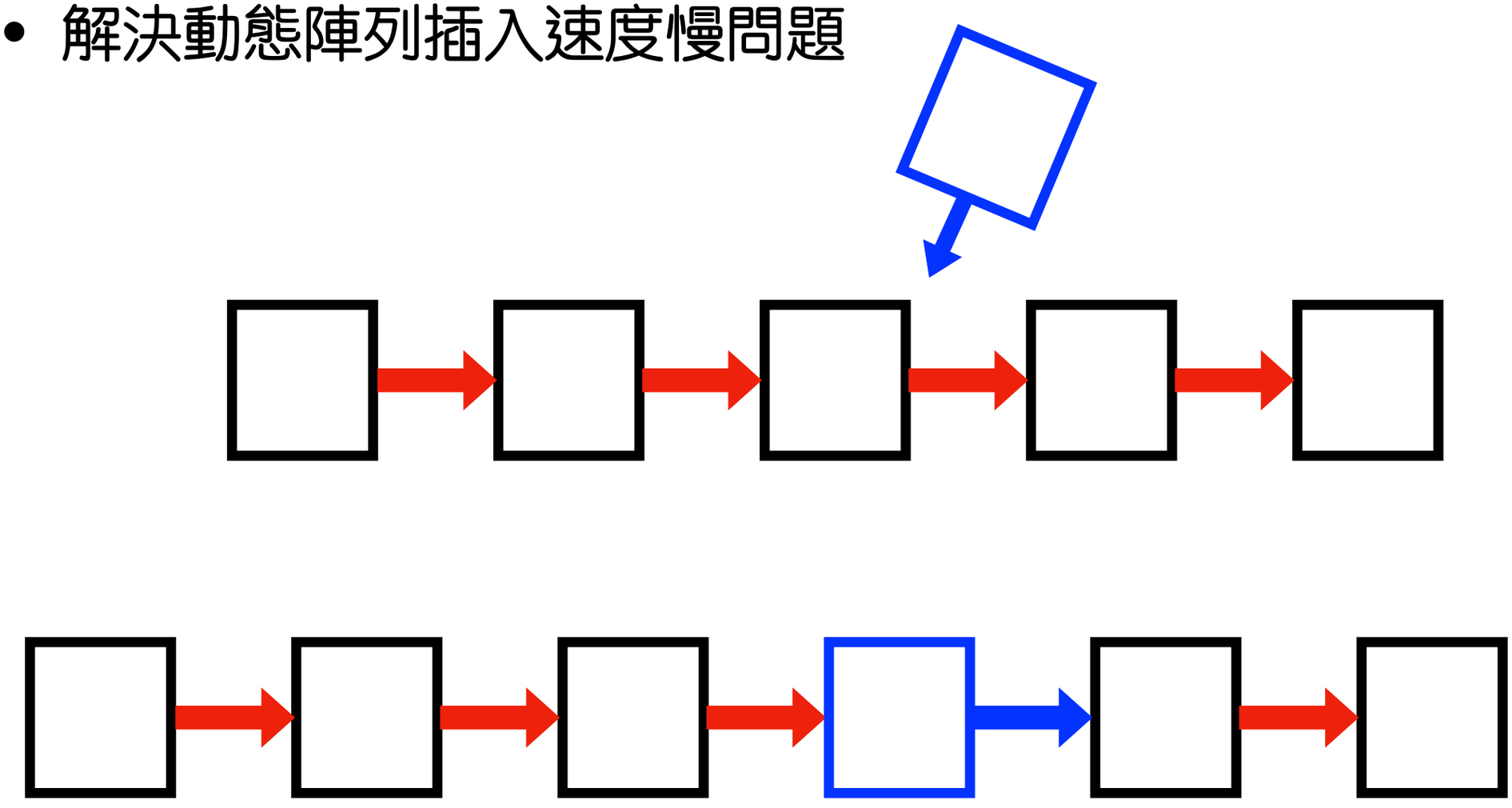
動態陣列

- member function
    - size()
    - capacity()
    - resize()
    - push()
    - pop()
    - del()
    - erase()
    - find()

Linked List

鏈結串列

- 以一個節點(Node)為單位，每個節點指向下一個節點
- 解決動態陣列插入速度慢問題



Linked List

Advantage and Disadvantage

- 插入、刪除 O(1)
- 但是取值、尋找 O(n) =>失去陣列結構的優點

```
1  struct node
2  {
3      int data;
4      node<int> *next;
5  };
6
7  list()
8  {
9      head = new node<T>;
10     tail = new node<T>;
11     tail->next = NULL;
12 }
```

沒有一個資料結構是完美的

視不同的需求和算法就有不同的變化

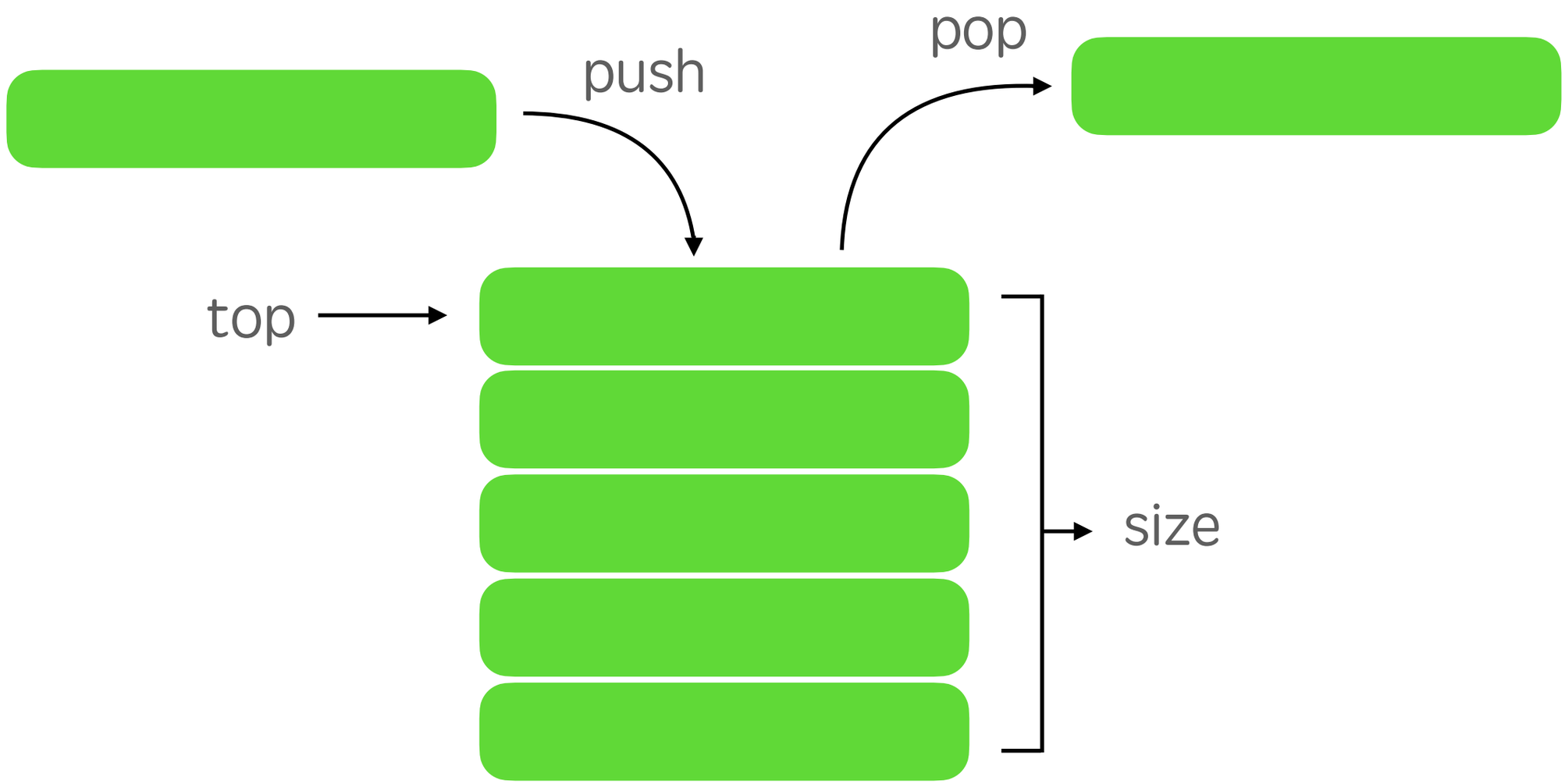
linked list

鏈結串列

- member function
  - begin()
  - end()
  - size()
  - empty()
  - push_front()
  - push_back()
  - pop_front()
  - pop_end()
  - insert()
  - del()
  - get()

Stack

堆疊



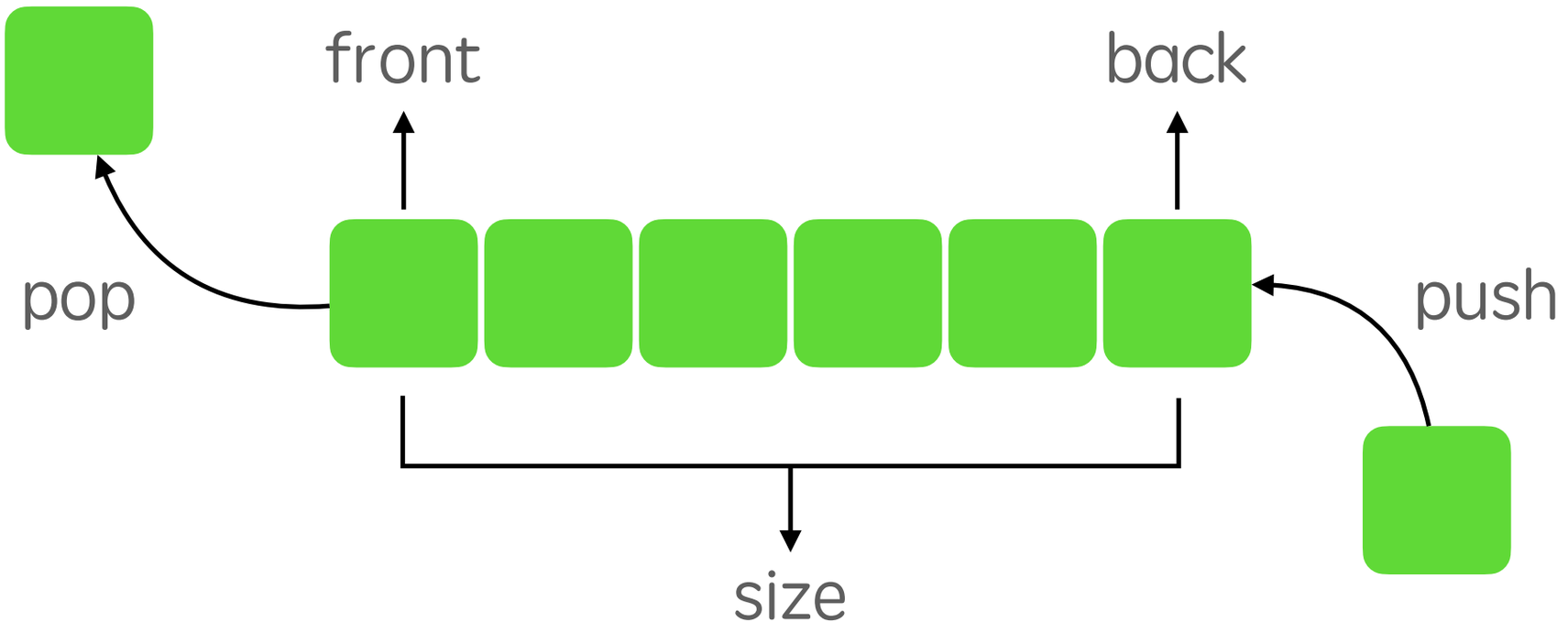
Stack

堆疊

- 後進先出
- 用 linked_list 實作
- member function
  - size()
  - empty()
  - push()
  - pop()
  - top()

Queue

佇列



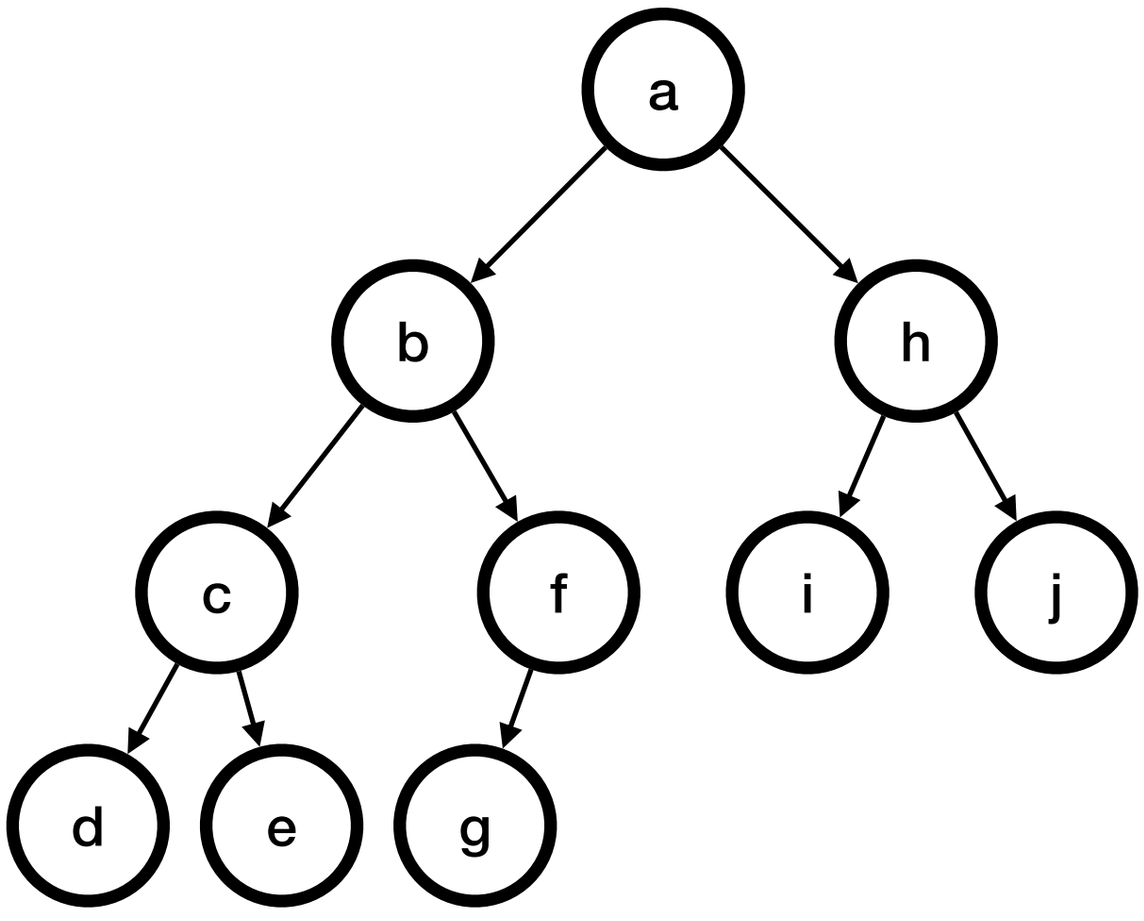
Queue

佇列

- 先進先出
- 用 linked_list 實作
- member function
  - size()
  - push()
  - pop()
  - empty()
  - front()
  - back()

🌳Tree

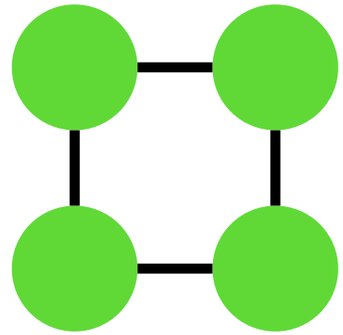
樹



🌳Tree

樹

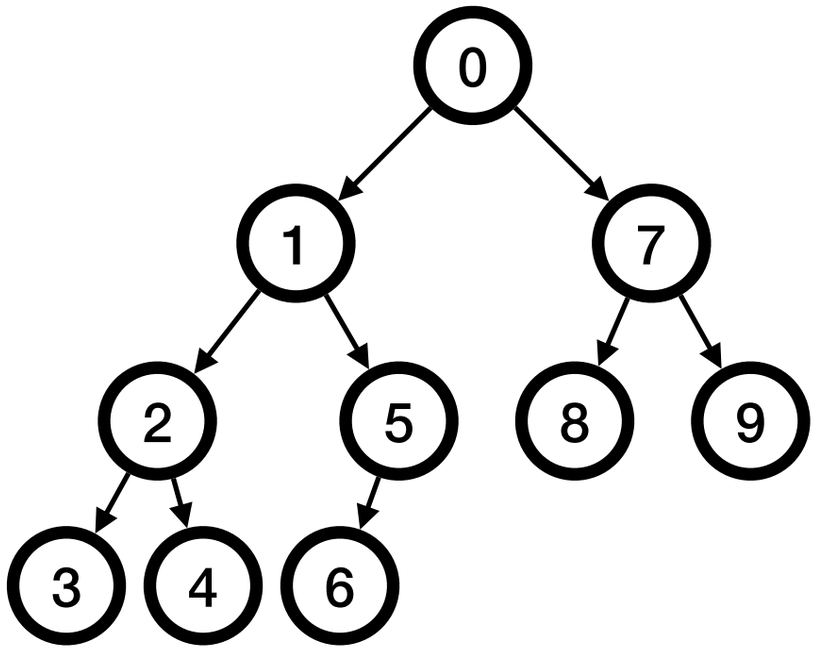
- 只有一個root，且不存在cycle
  - 對於所有的node有存在一條路到root
  - 每個node只有一個parent
- 檔案儲存方式是一種樹狀結構
- 族譜是一種樹狀結構
- 儲存方式
  - node
  - vector
  - matrix



🌳Tree

儲存_matrix

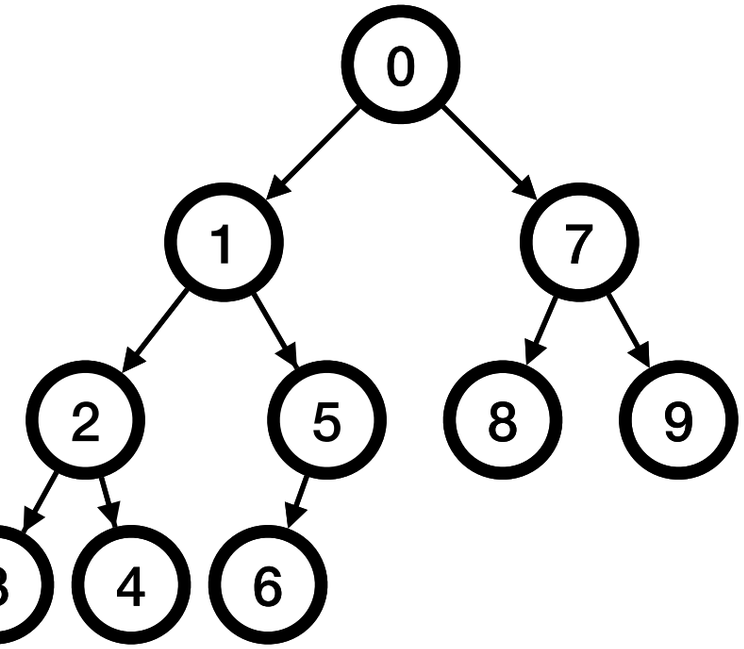
	0	1	2	3	4	5	6	7	8	9
0	F	T	F	F	F	F	F	T	F	F
1	F	F	T	F	F	T	F	F	F	F
2	F	F	F	T	T	F	F	F	F	F
3	F	F	F	F	F	F	F	F	F	F
4	F	F	F	F	F	F	F	F	F	F
5	F	F	F	F	F	F	T	F	F	F
6	F	F	F	F	F	F	F	F	F	F
7	F	F	F	F	F	F	F	F	T	T
8	F	F	F	F	F	F	F	F	F	F
9	F	F	F	F	F	F	F	F	F	F



🌳Tree

儲存_vector

data	child	
0	1	7
1	2	5
2	3	4
3		
4		
5	6	
6		
7	8	9
8		
9		



🌳Tree

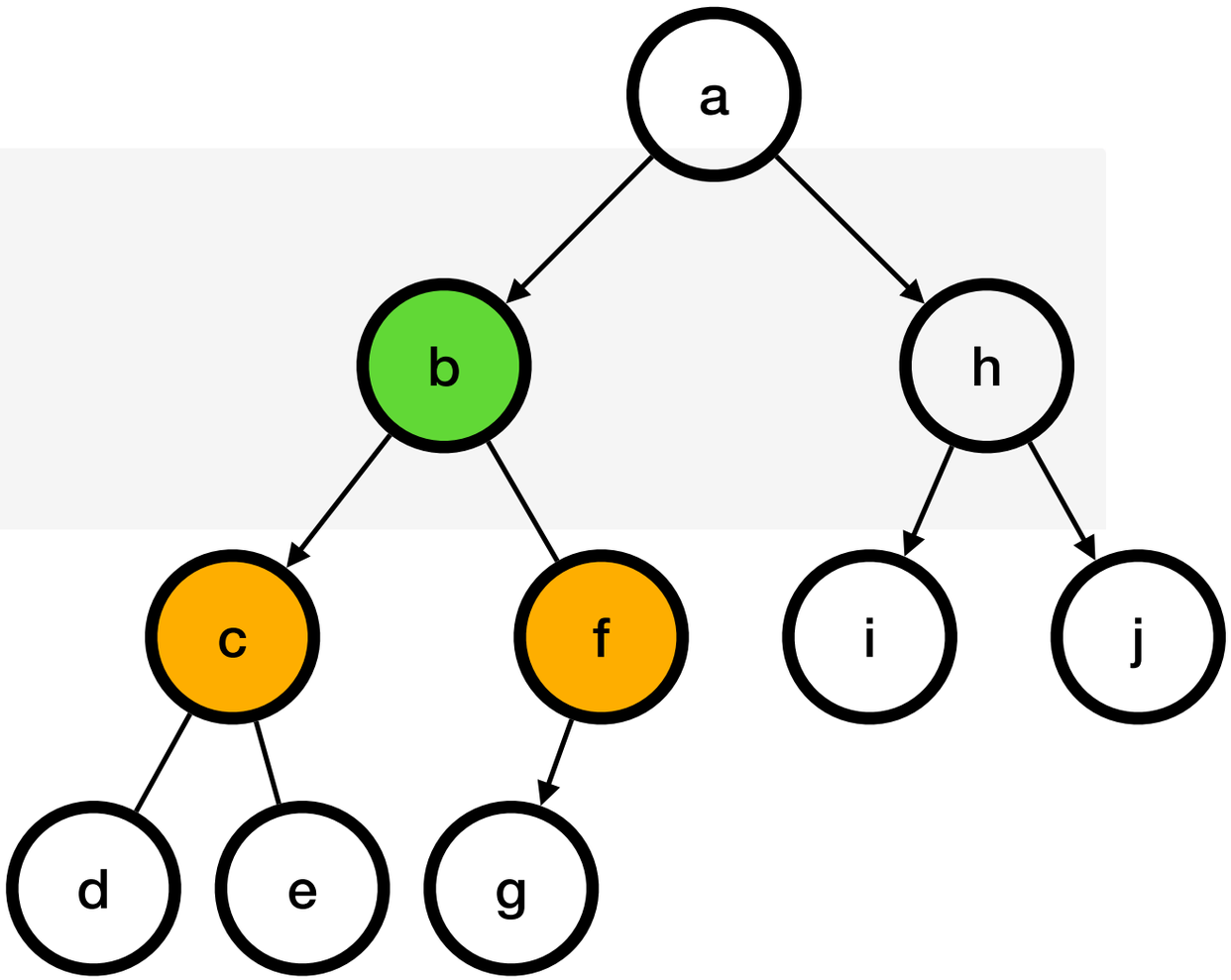
儲存_node

```
class node
{
    int *child_one;
    int *child_two;
    int data;
};
```

🌳Tree

儲存_node

```
class node
{
    int *child_one;
    int *child_two;
    int data;
};
```



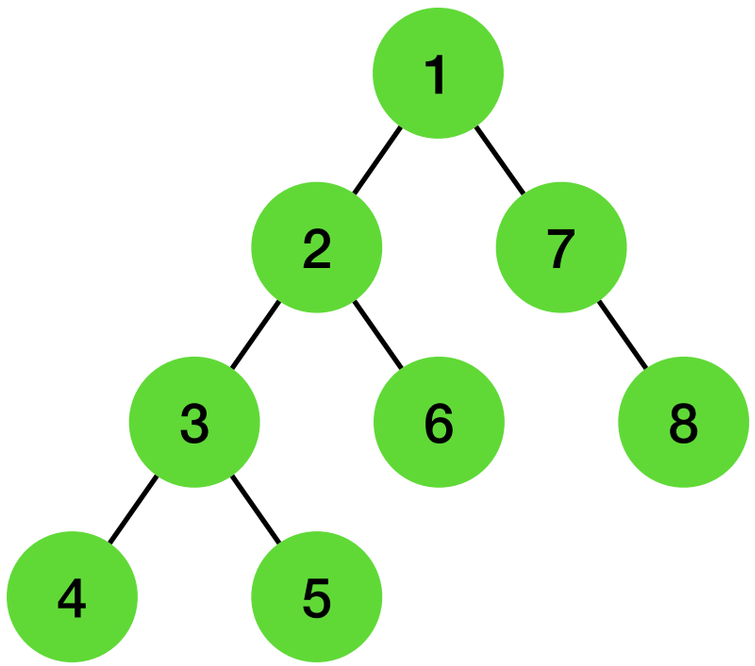
🌳Tree

遍歷

- depth-first-search (connect with stack)
  - pre-order-traversal
  - in-order-traversal
  - post-order-traversal
- breadth-first-search (connect with queue)
  - level-order-traversal

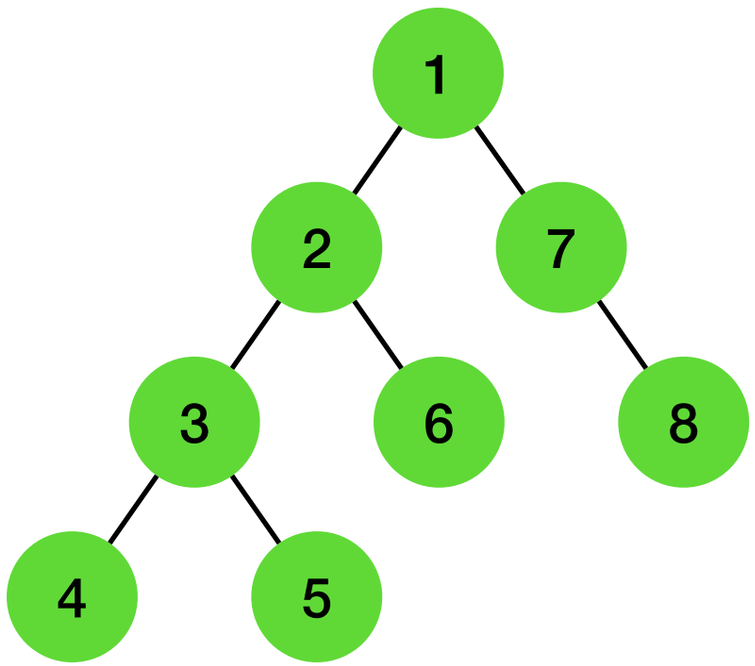
🌳Tree

遍歷__DFS



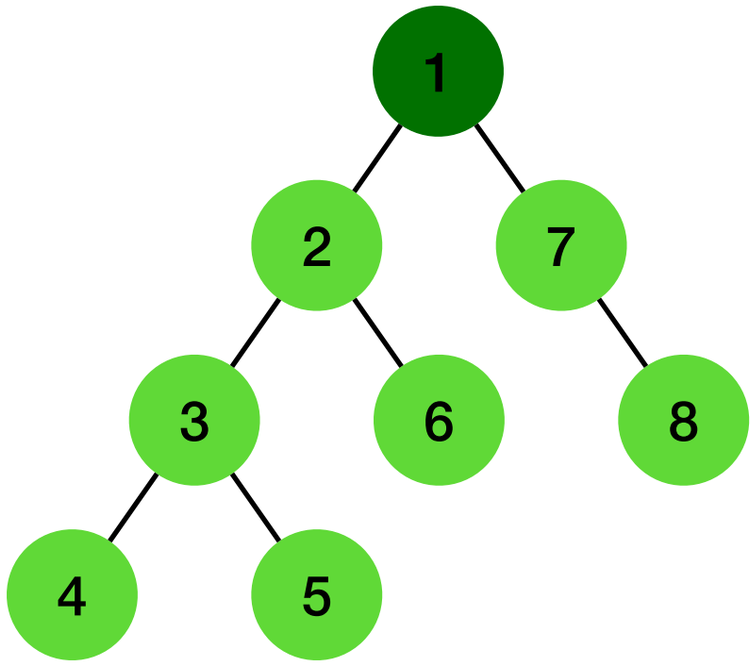
🌳Tree

遍歷__DFS



🌳Tree

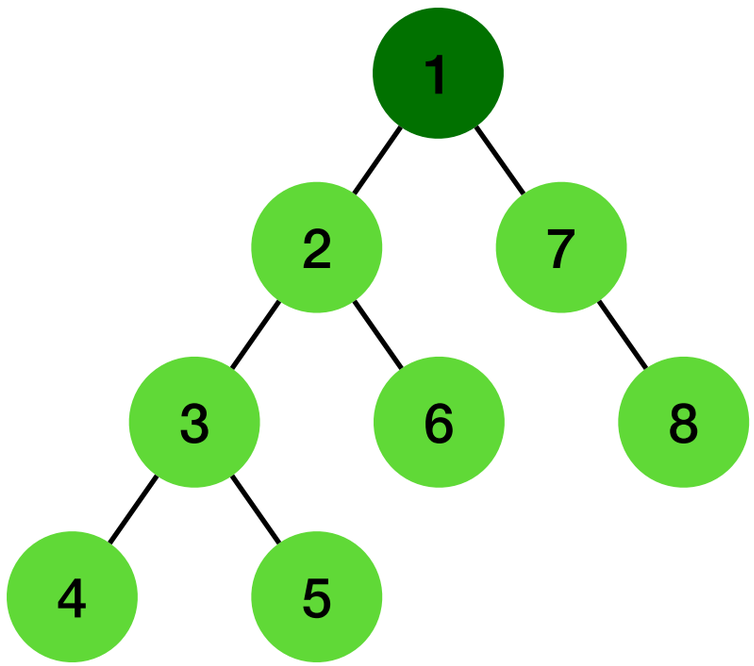
遍歷__DFS

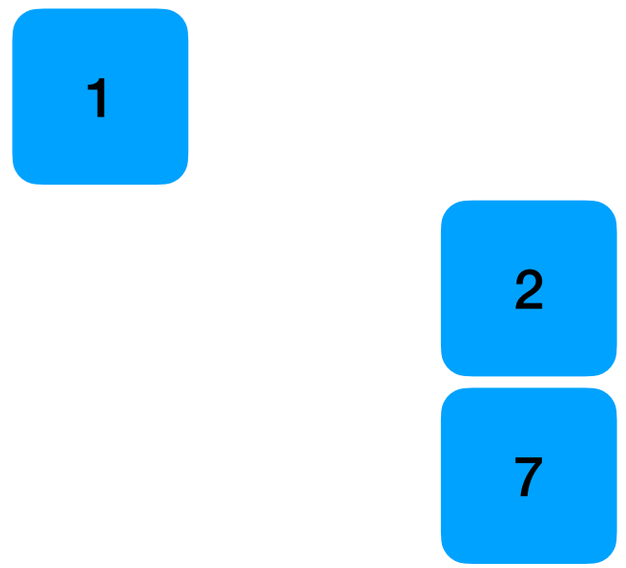


1

🌳Tree

遍歷__DFS

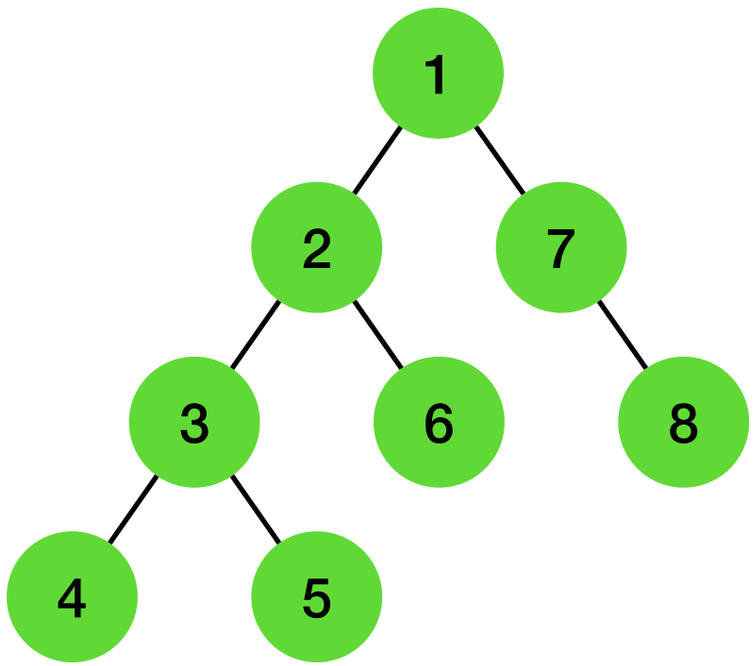


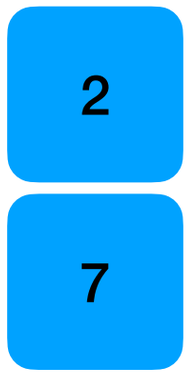


1

🌳Tree

遍歷__DFS

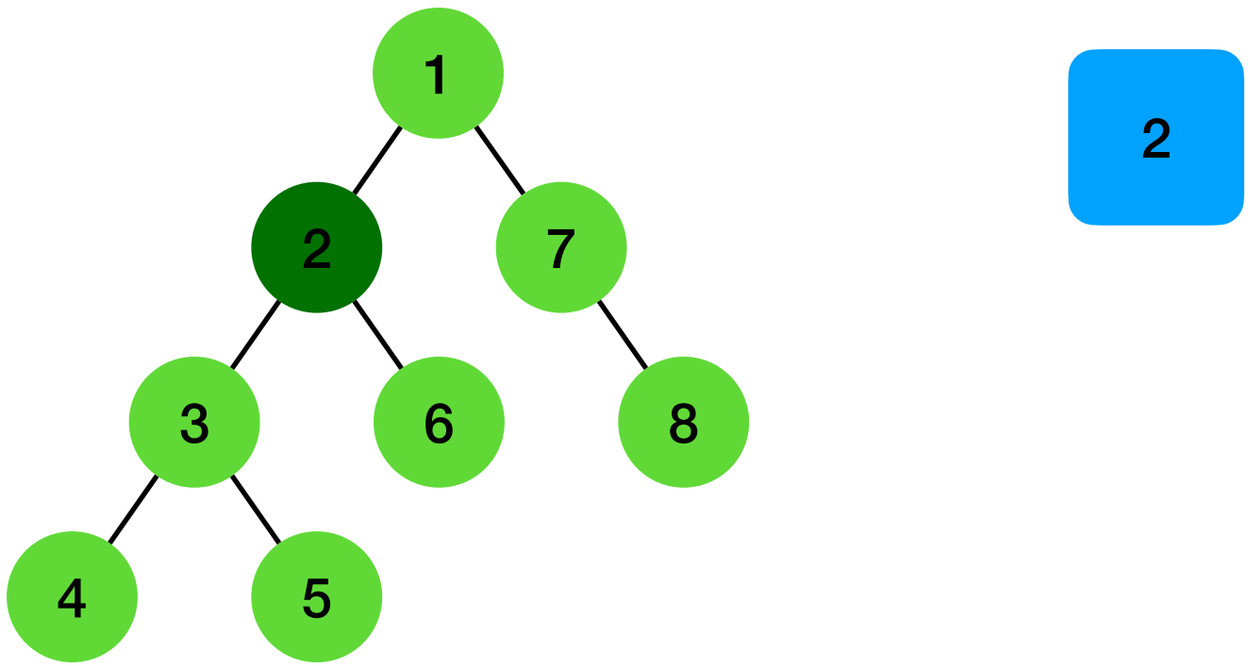




1

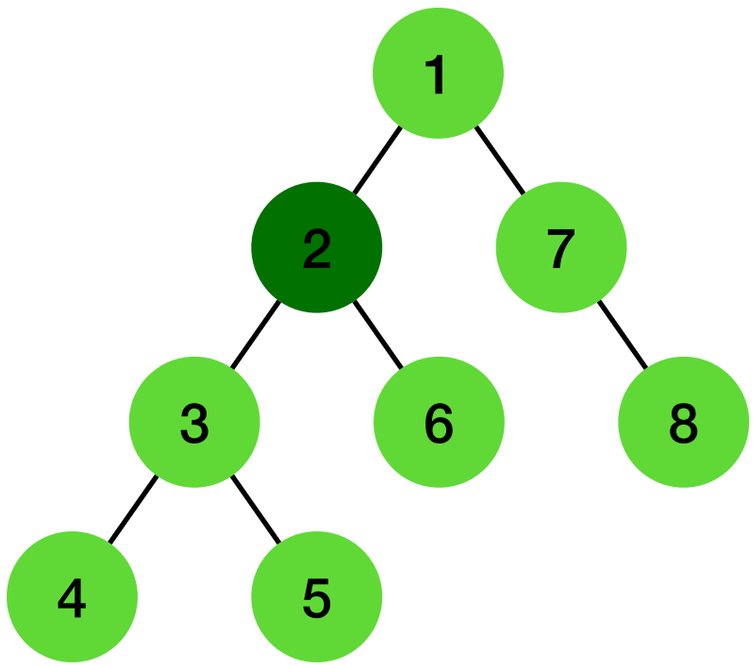
🌳Tree

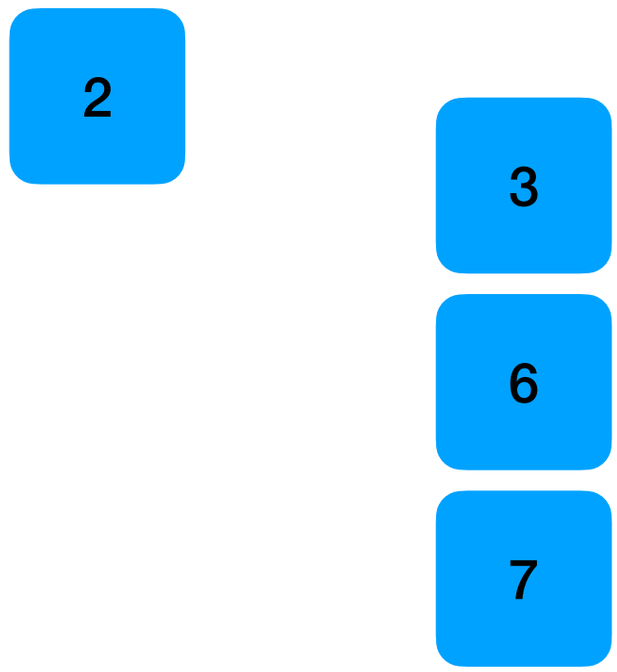
遍歷__DFS



🌳Tree

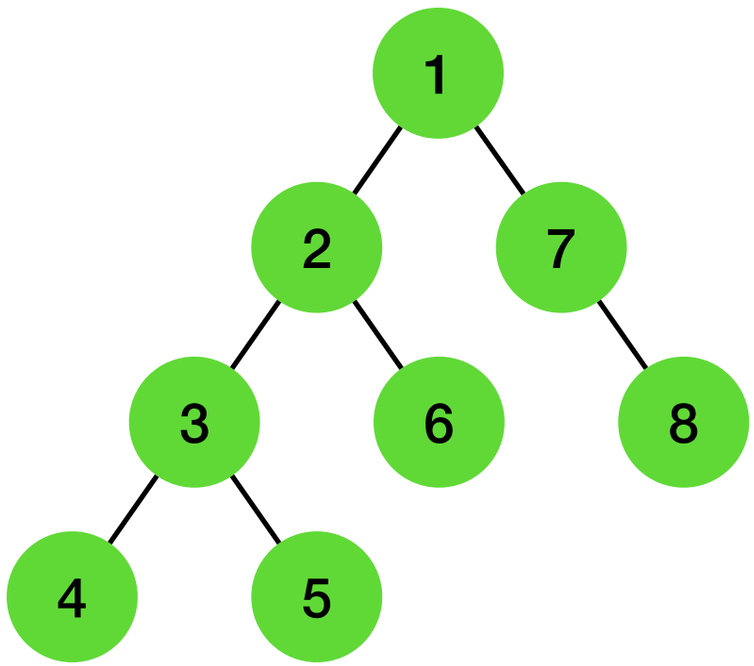
遍歷__DFS





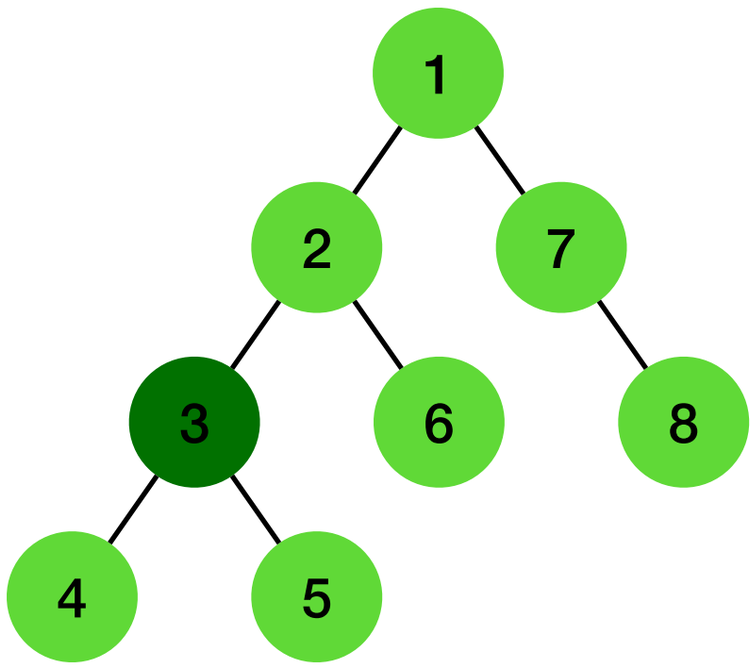
🌳Tree

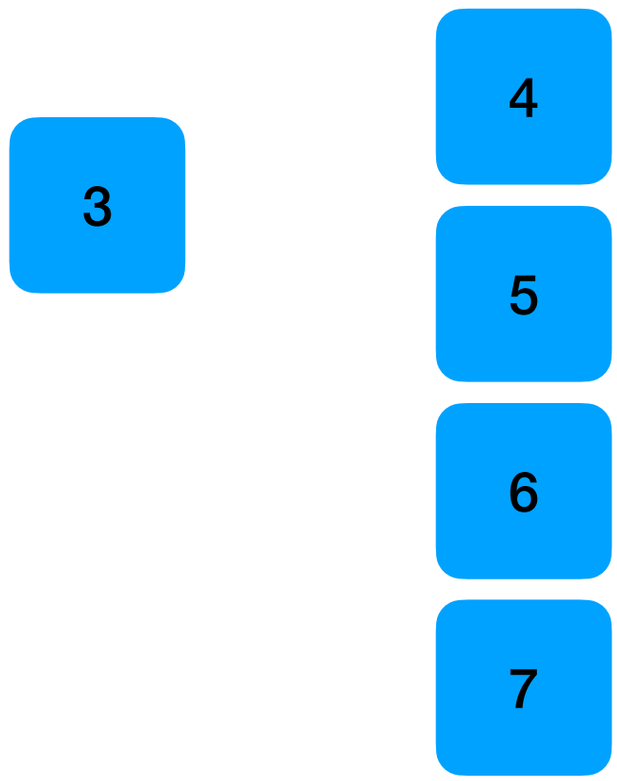
遍歷__DFS



🌳Tree

遍歷__DFS

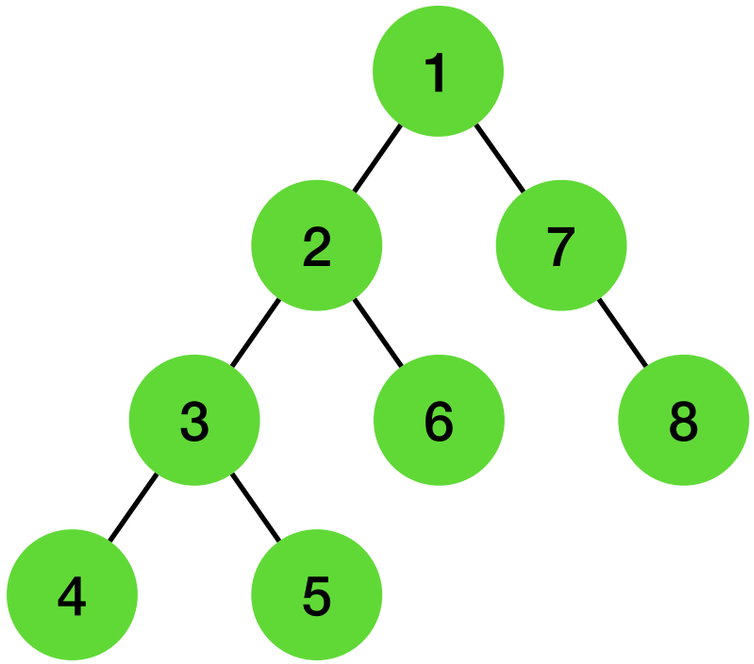


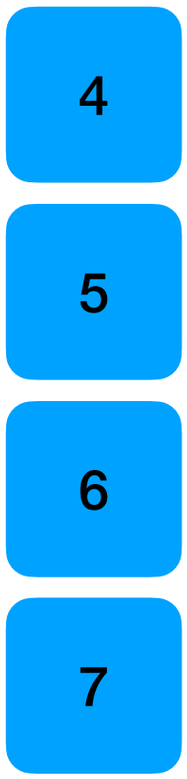


1 2 3

🌳Tree

遍歷__DFS

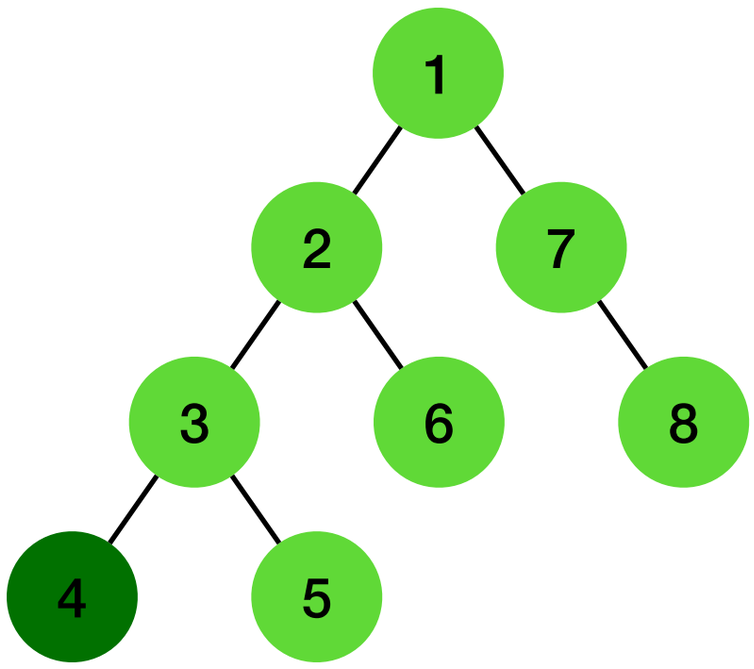


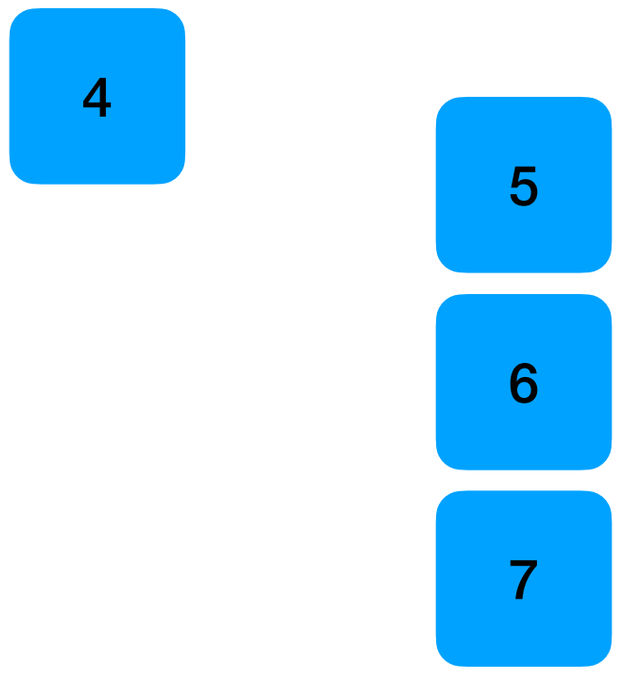


1 2 3

🌳Tree

遍歷__DFS

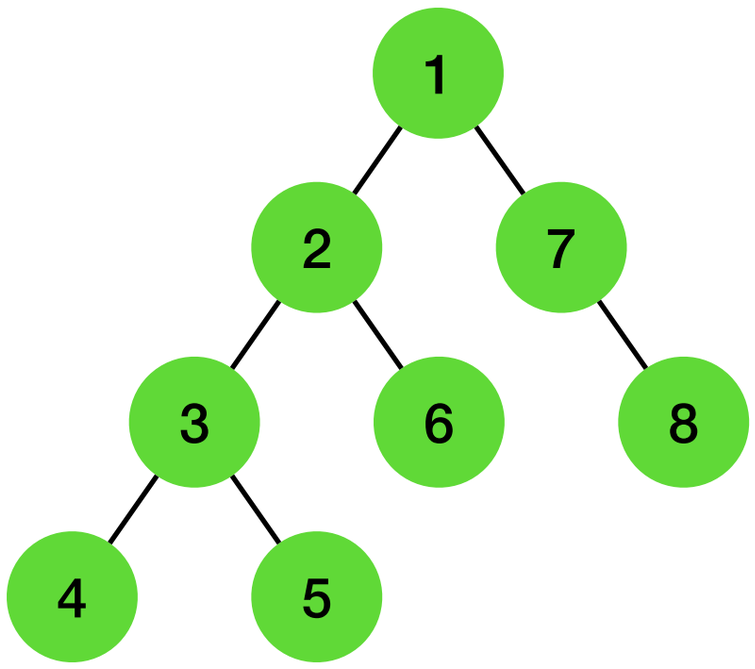




1 2 3 4

🌳Tree

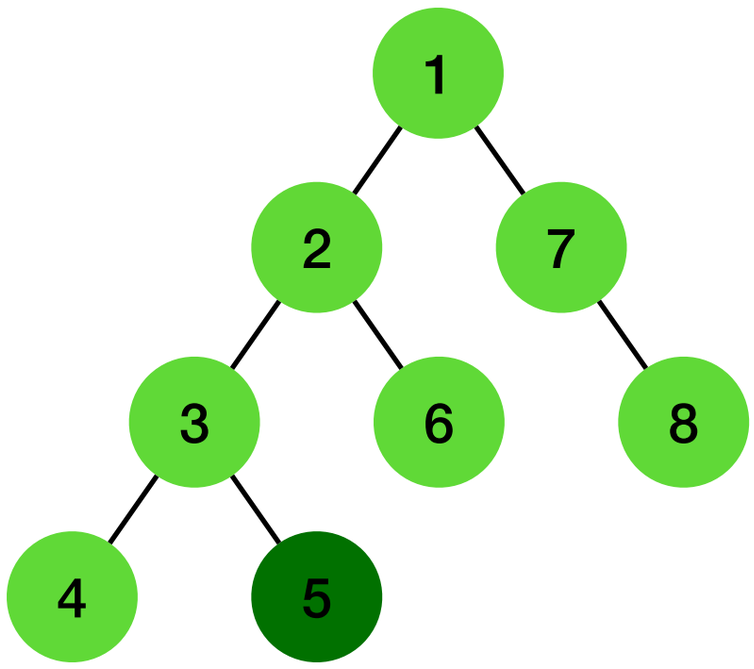
遍歷__DFS

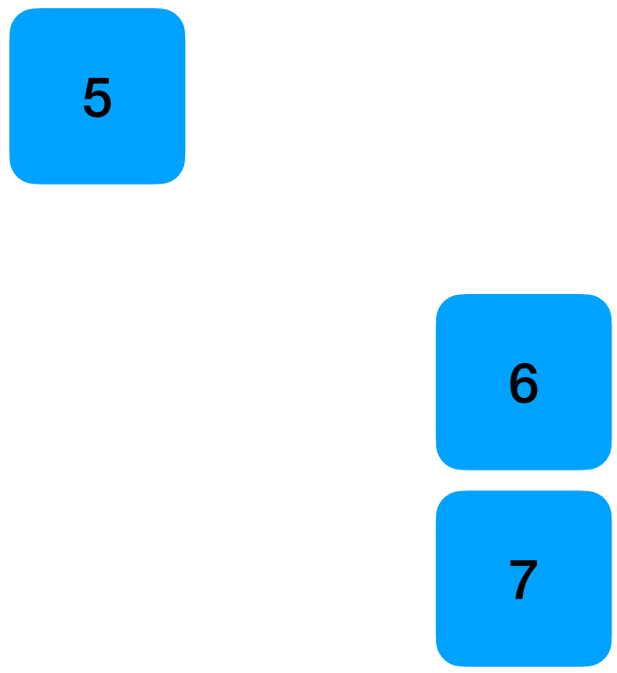


1 2 3 4

🌳Tree

遍歷__DFS

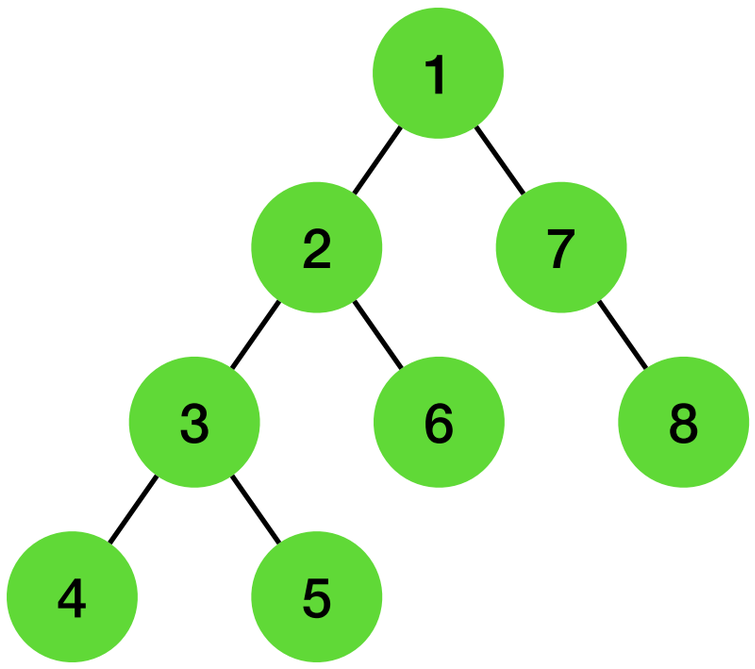


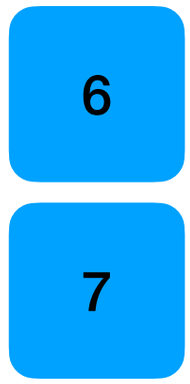


1 2 3 4 5

🌳Tree

遍歷__DFS

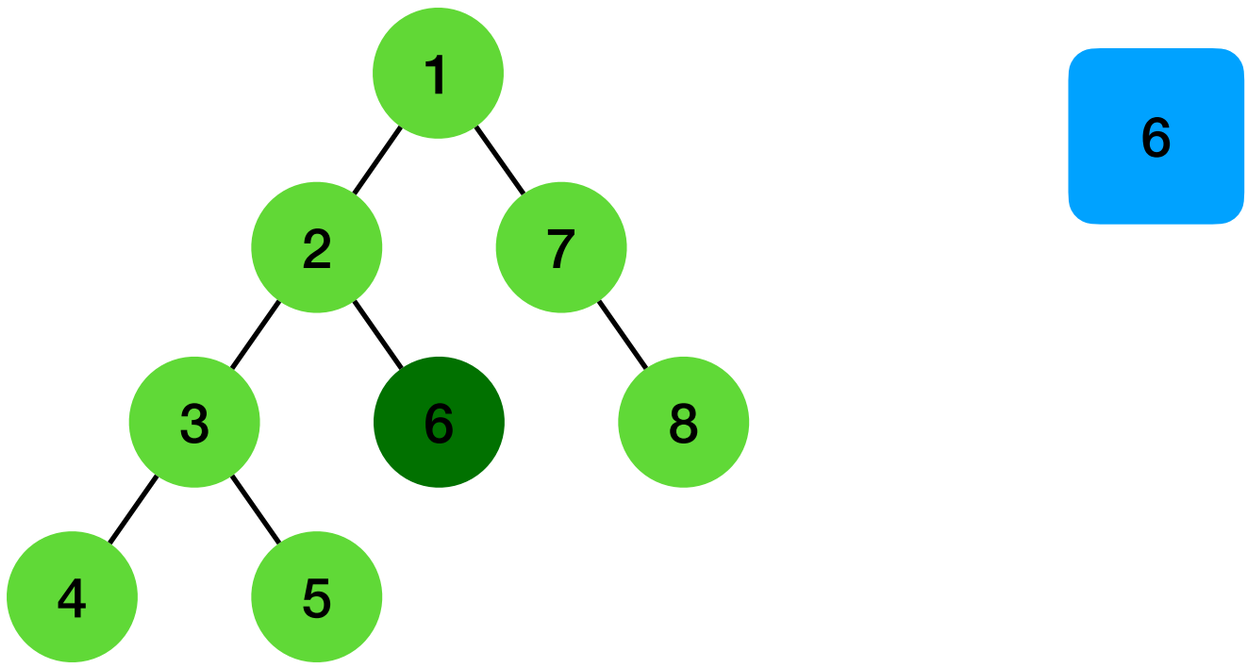




1 2 3 4 5

🌳Tree

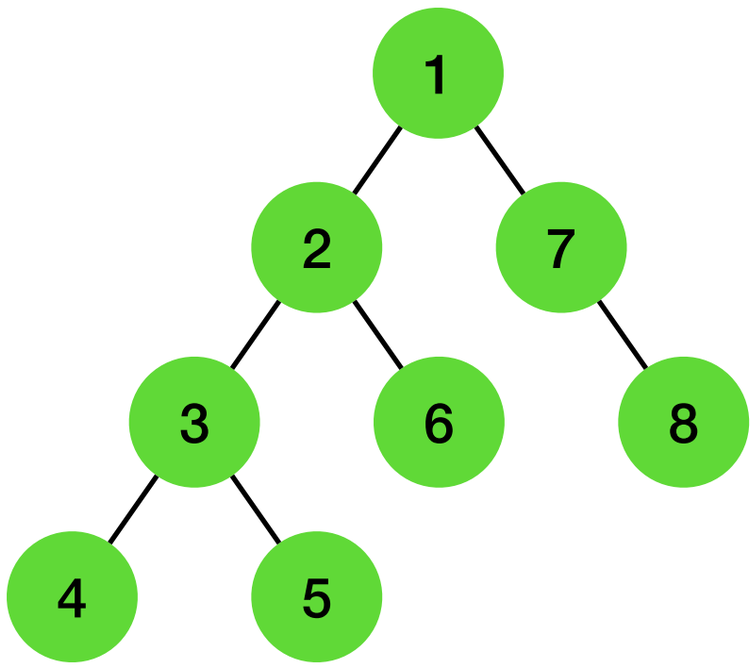
遍歷__DFS



1 2 3 4 5 6

🌳Tree

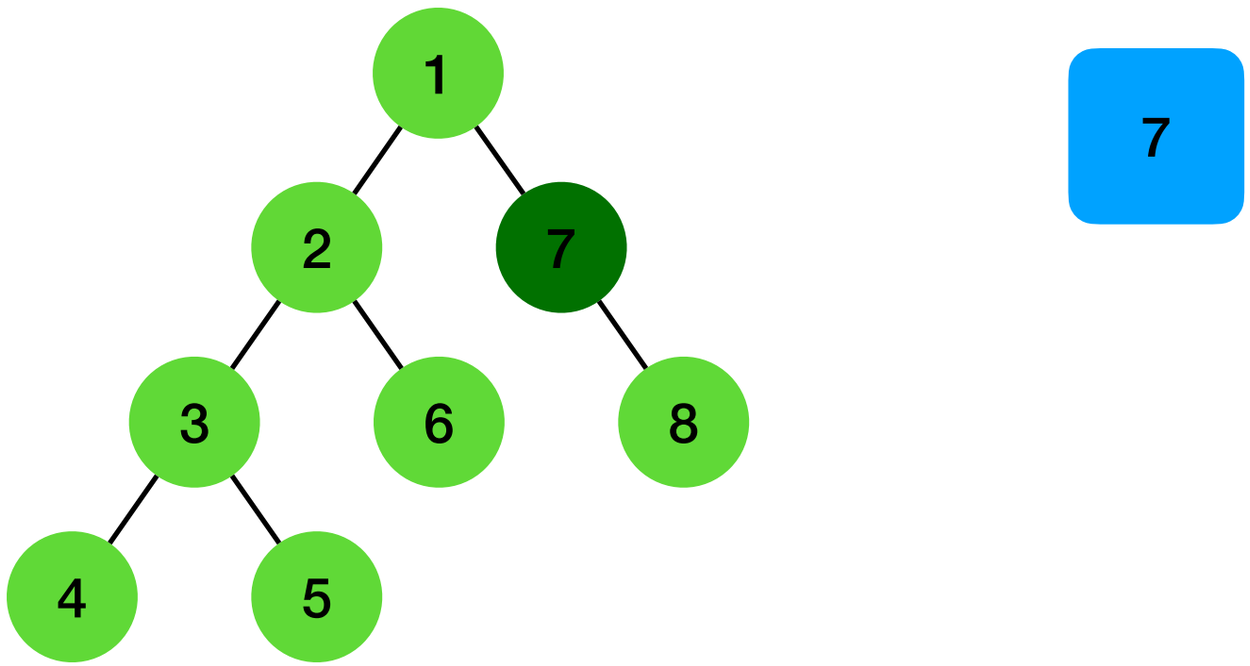
遍歷__DFS



1 2 3 4 5 6

🌳Tree

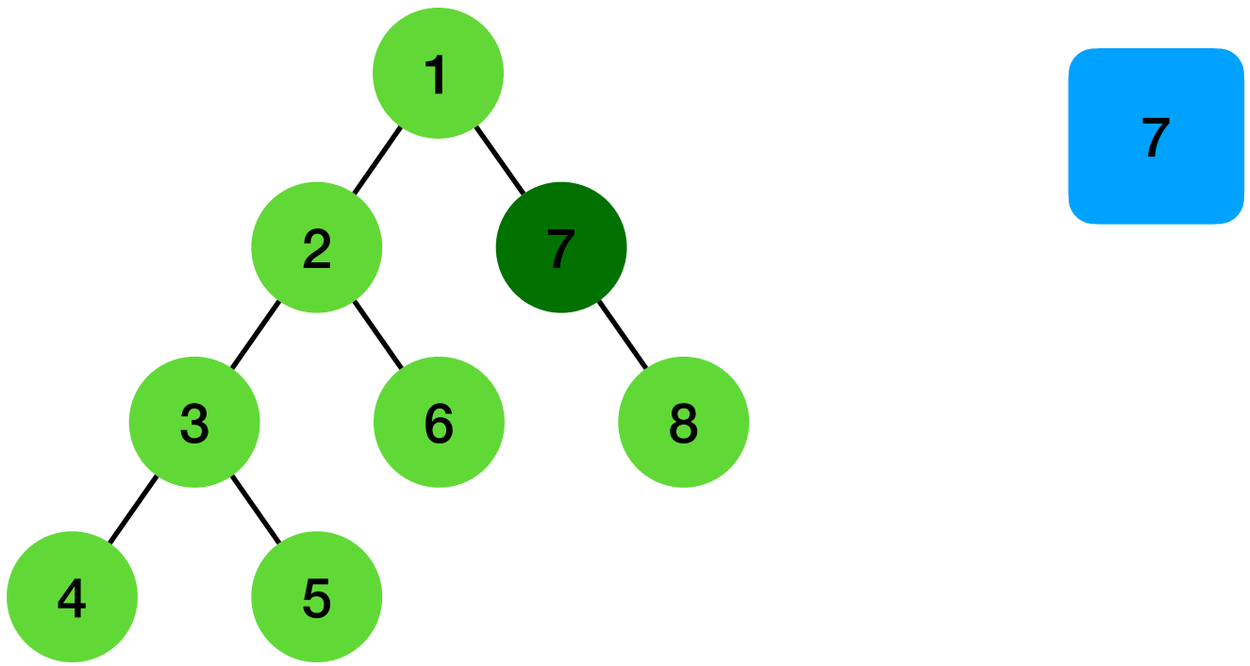
遍歷__DFS



1 2 3 4 5 6 7

🌳Tree

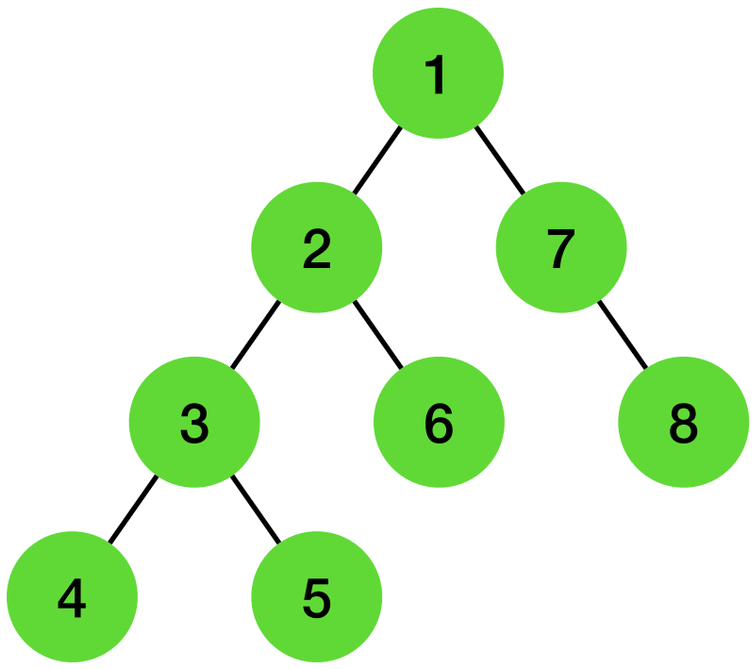
遍歷__DFS



1 2 3 4 5 6 7

🌳Tree

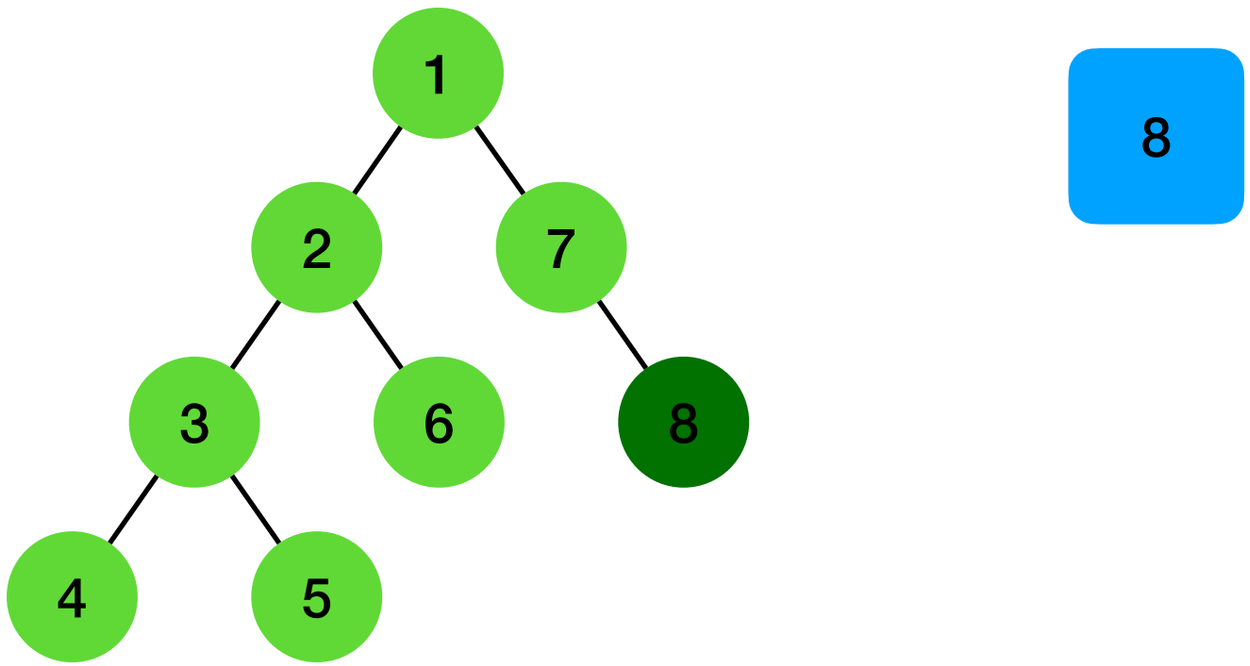
遍歷__DFS



1 2 3 4 5 6 7

🌳Tree

遍歷__DFS

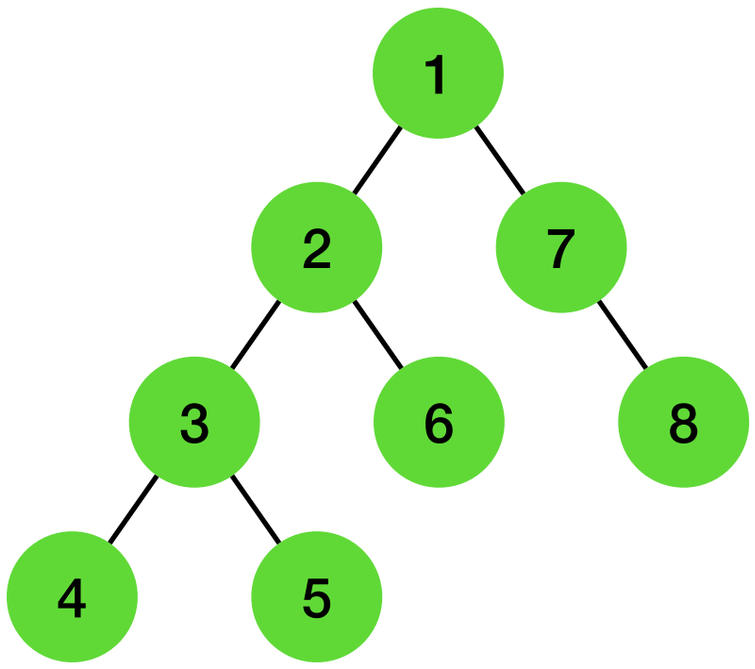


8

1 2 3 4 5 6 7 8

🌳Tree

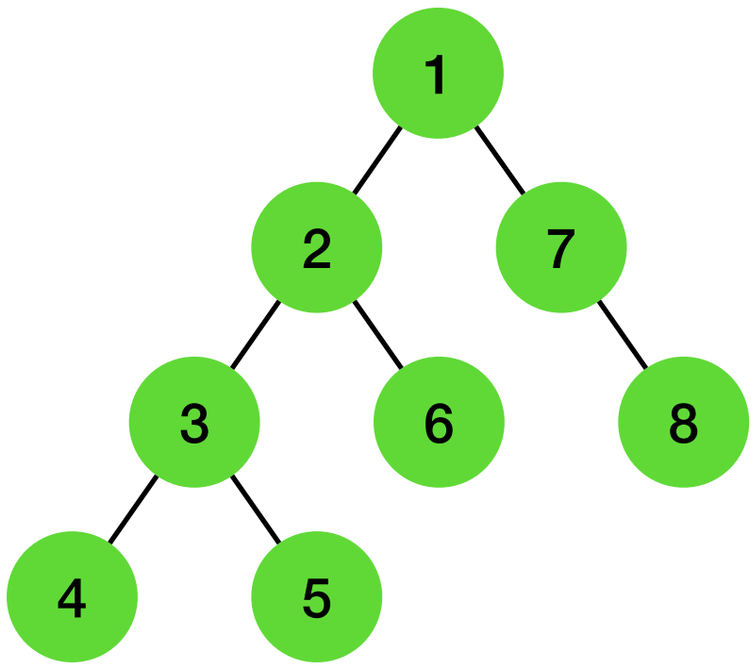
遍歷__DFS



1 2 3 4 5 6 7 8

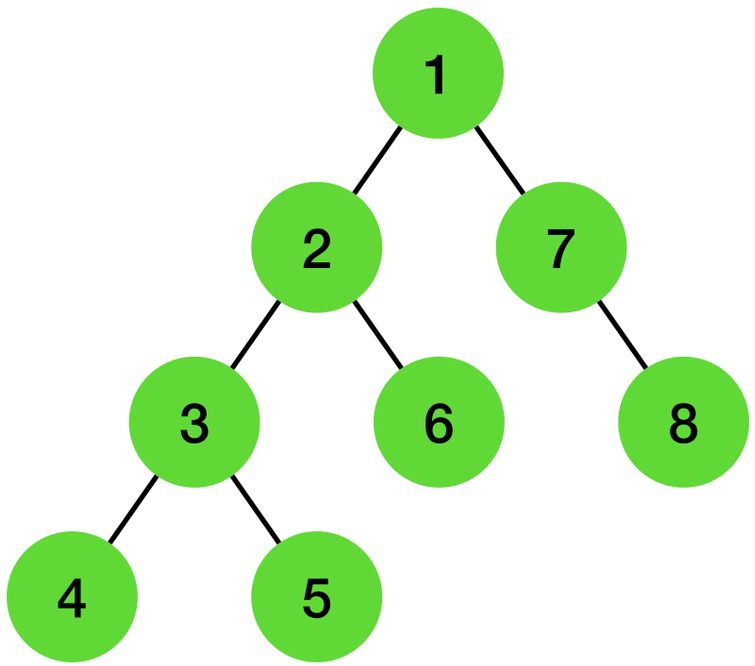
🌳Tree

遍歷__BFS



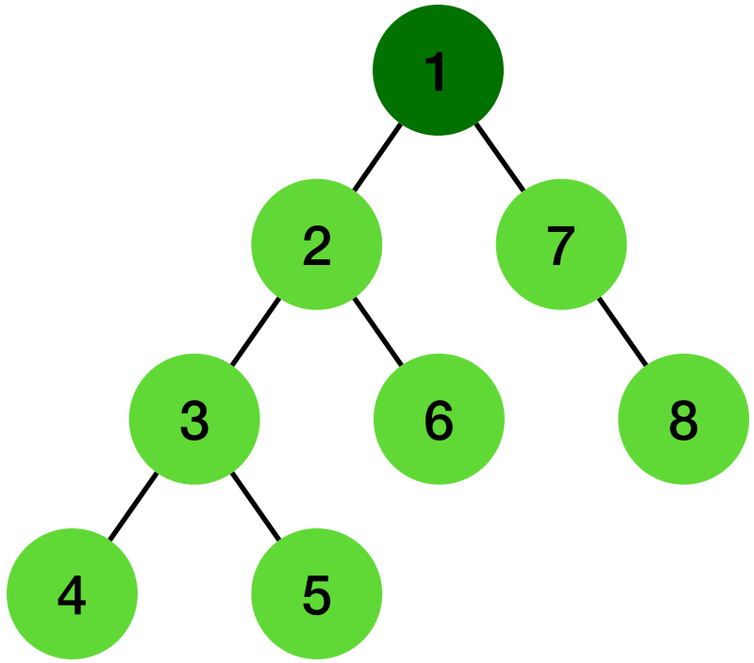
🌳Tree

遍歷__BFS



🌳Tree

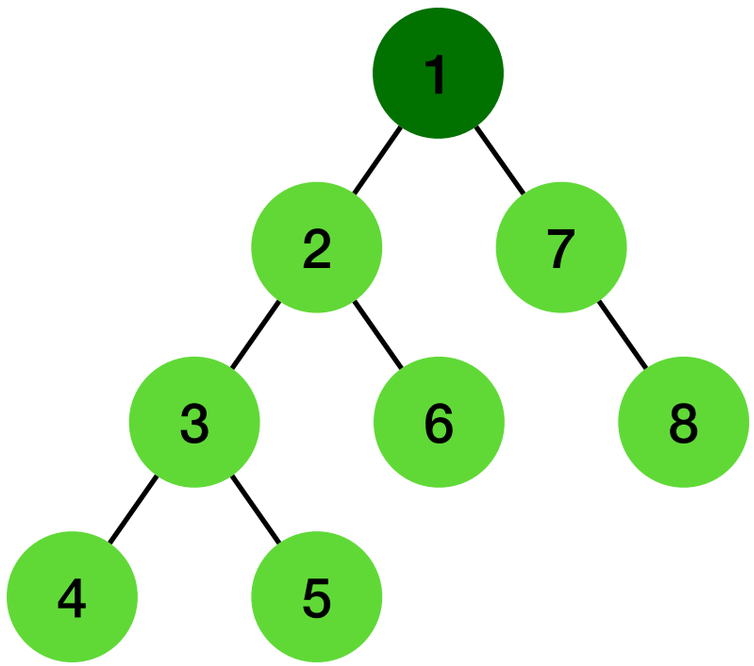
遍歷__BFS

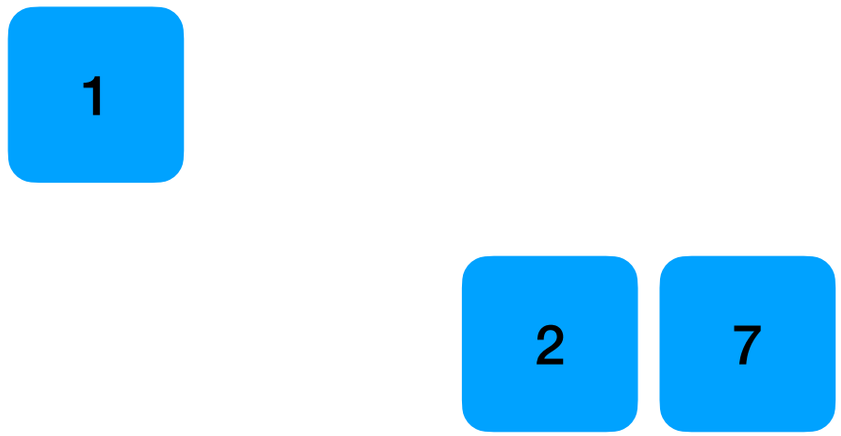


1

🌳Tree

遍歷__BFS

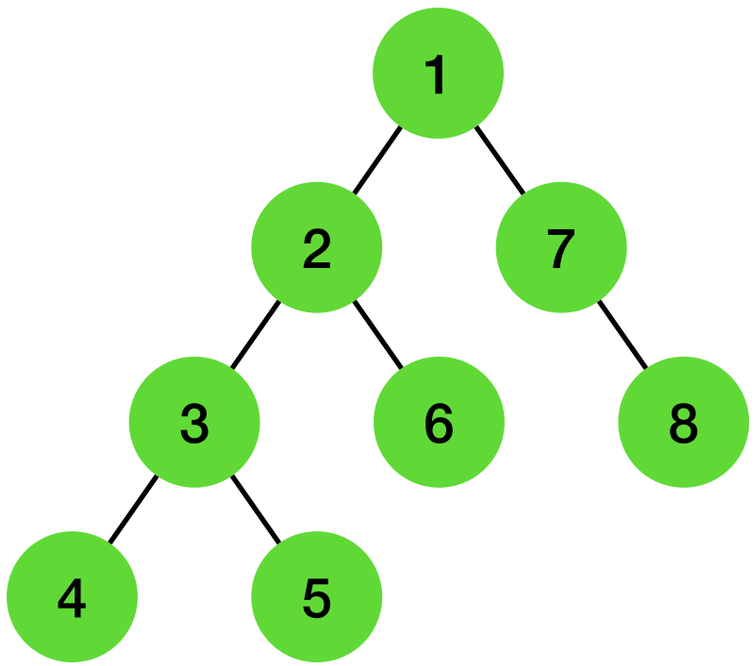


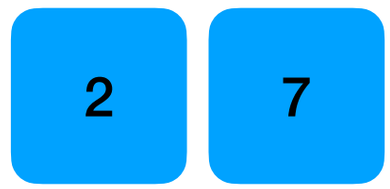


1

🌳Tree

遍歷__BFS

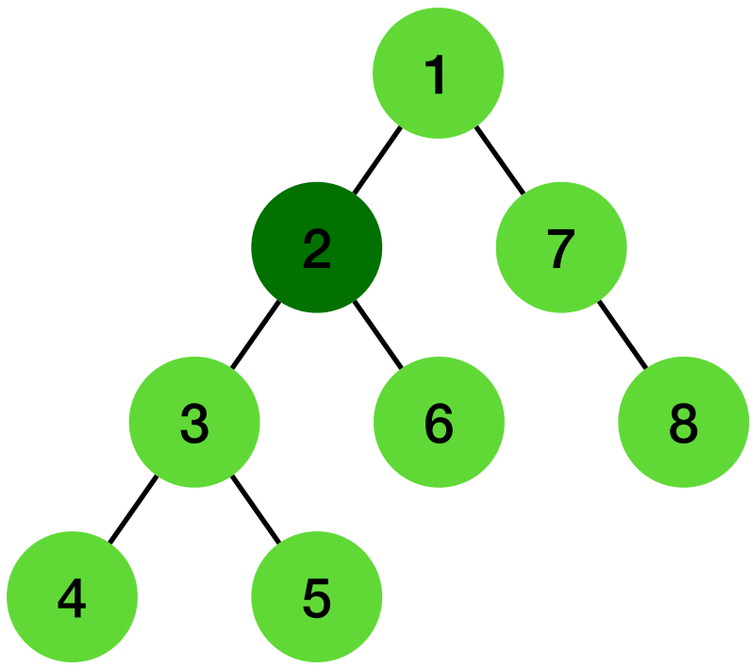


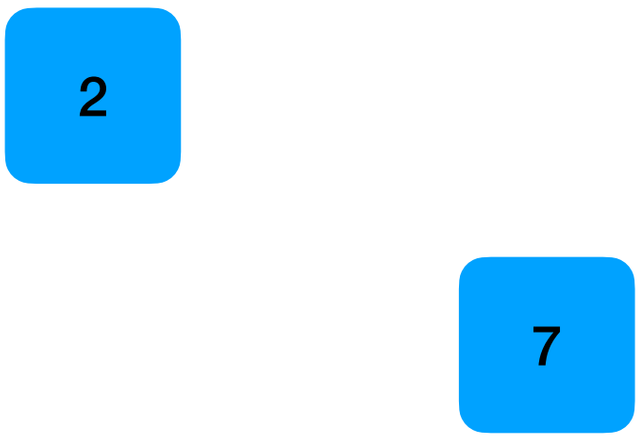


1

🌳Tree

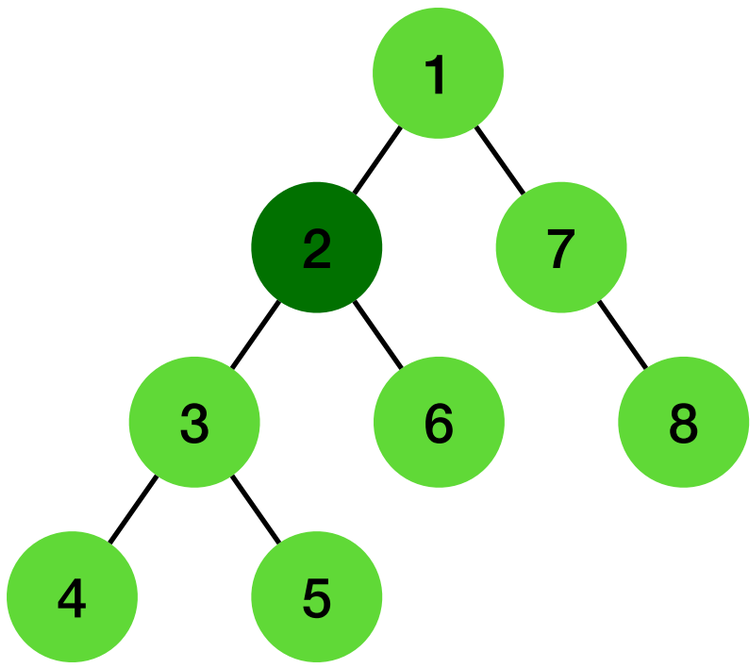
遍歷__BFS





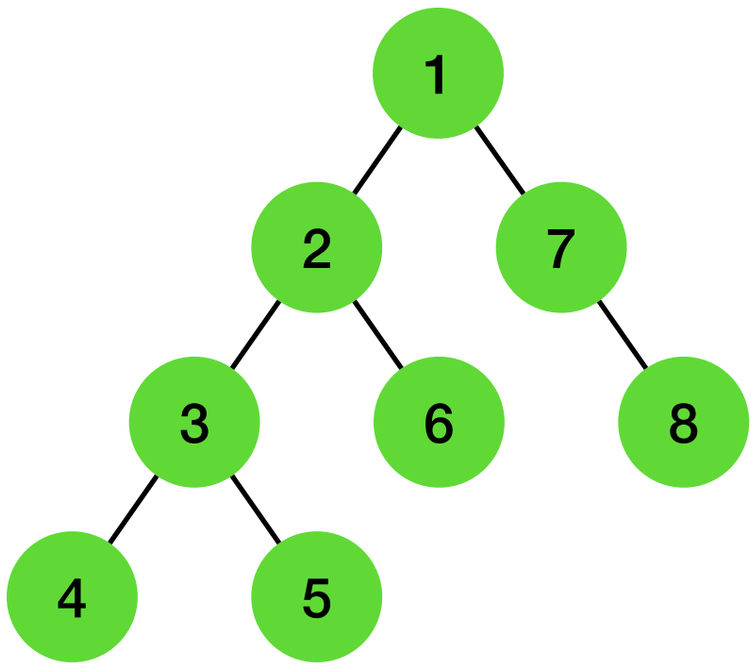
🌳Tree

遍歷__BFS



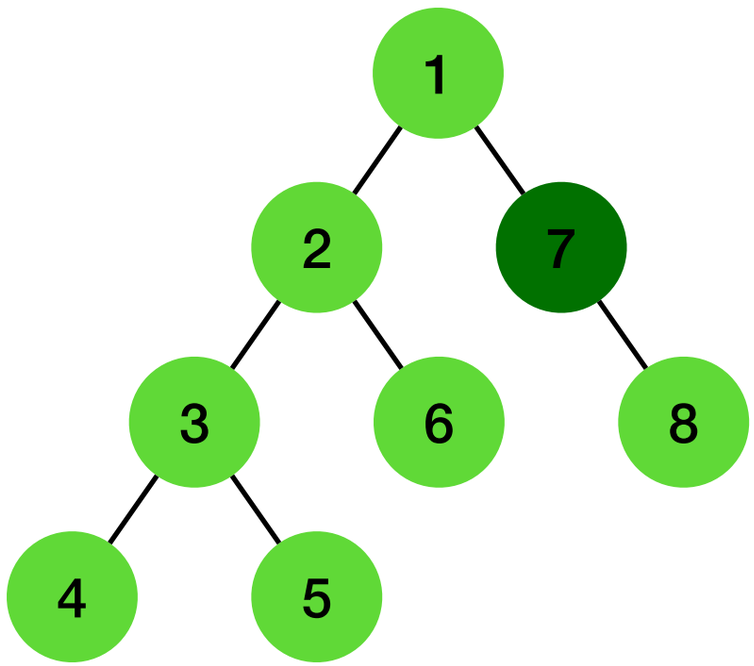
🌳Tree

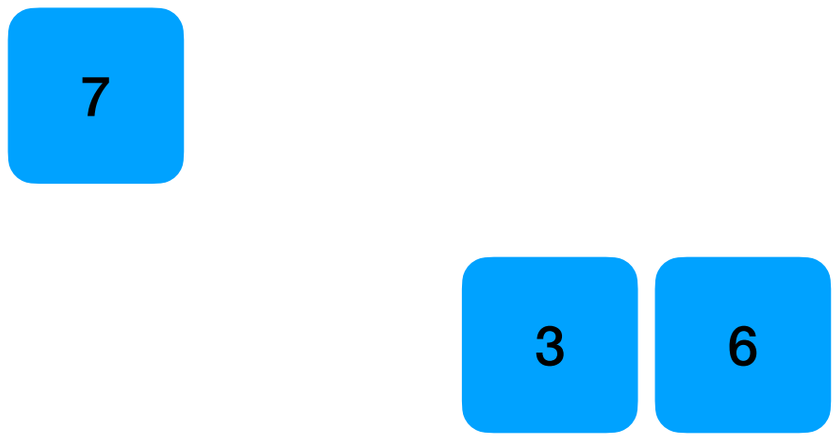
遍歷__BFS



🌳Tree

遍歷__BFS

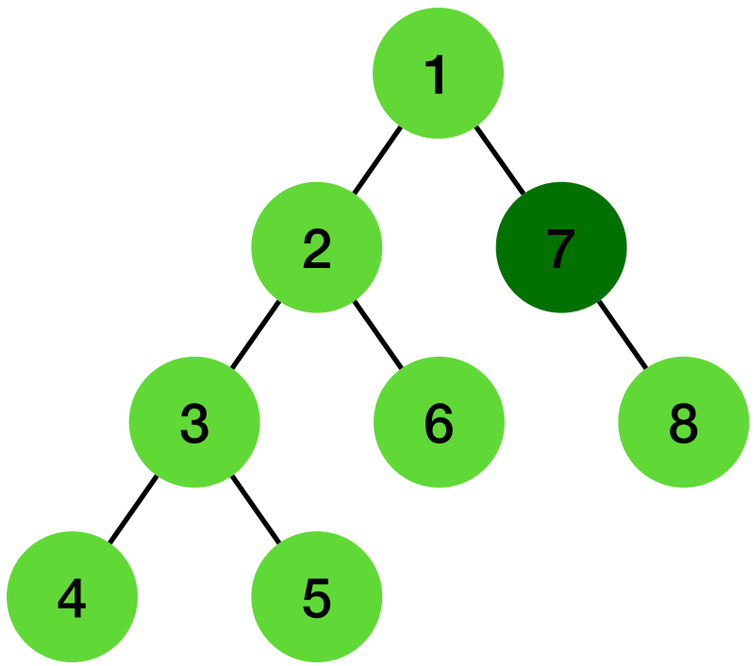


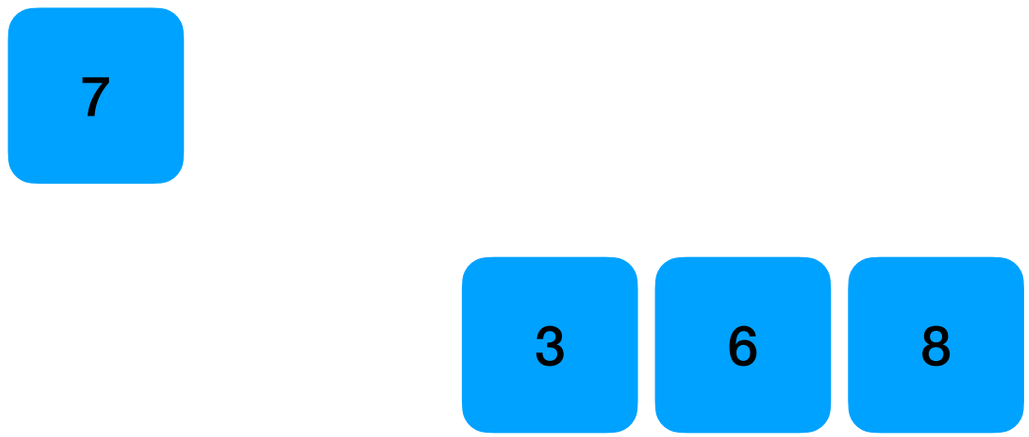


1 2 7

🌳Tree

遍歷__BFS

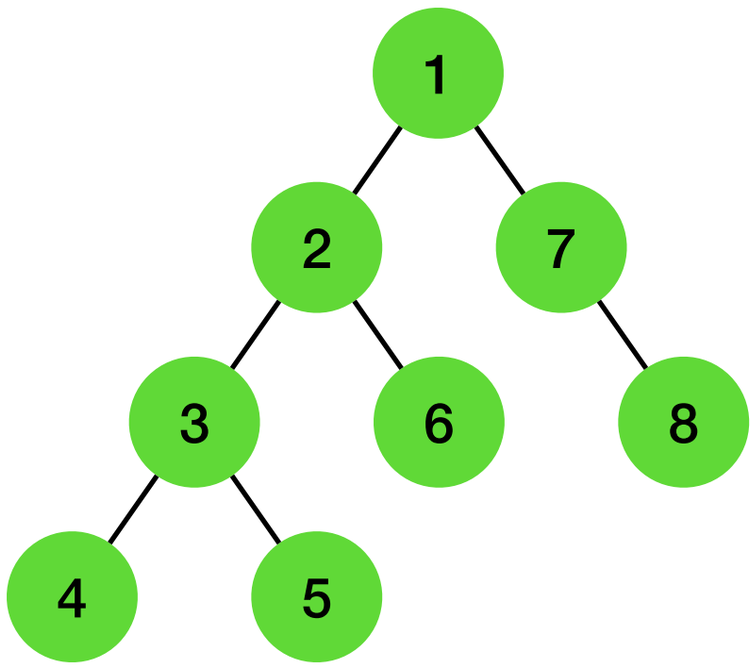




1 2 7

🌳Tree

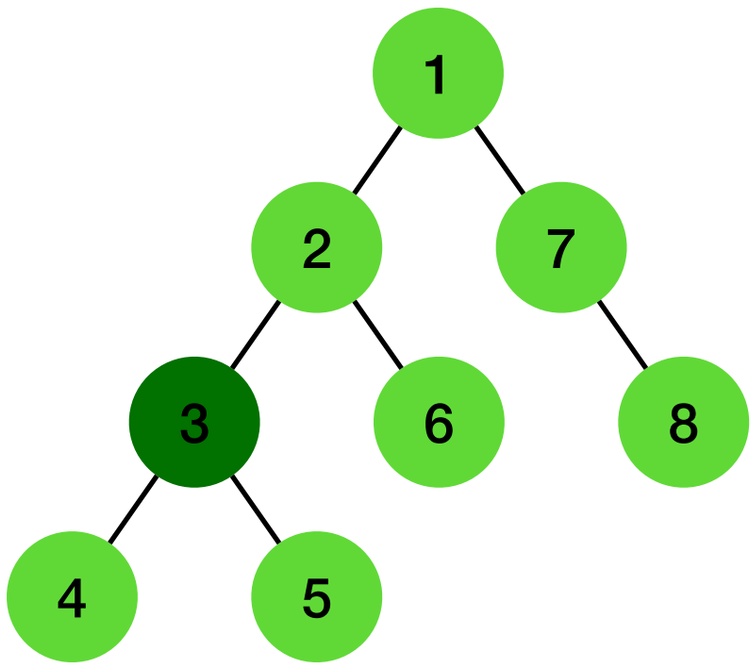
遍歷__BFS



1 2 7

🌳Tree

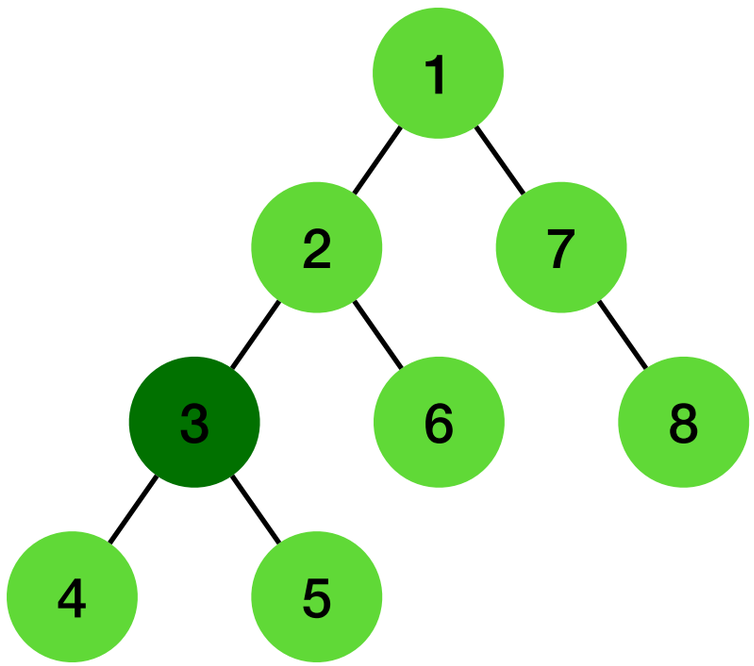
遍歷__BFS

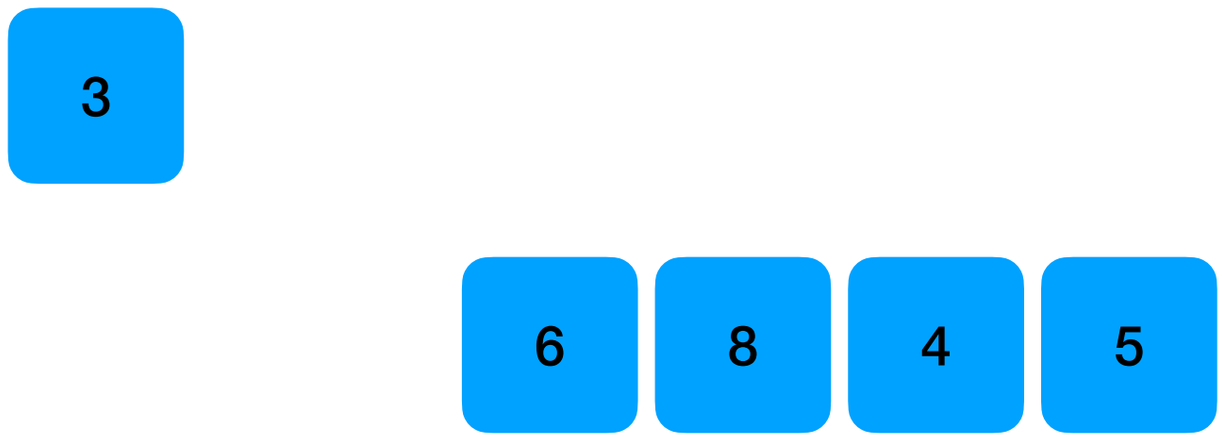


1 2 7 3

🌳Tree

遍歷__BFS

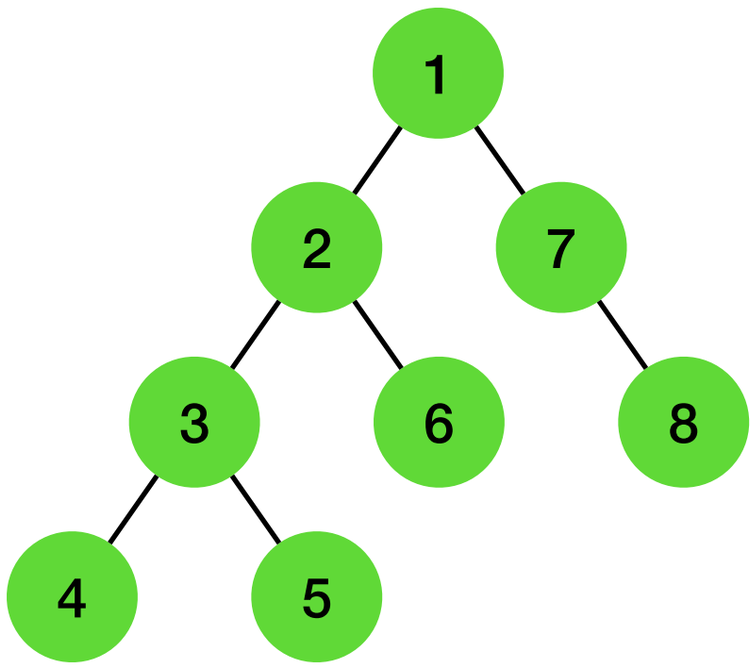


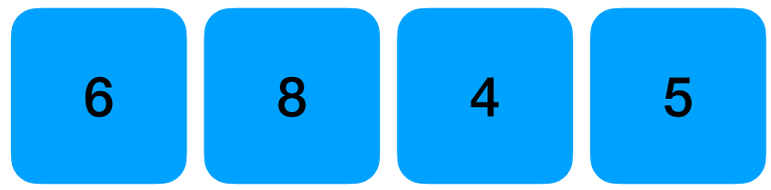


1 2 7 3

🌳Tree

遍歷__BFS

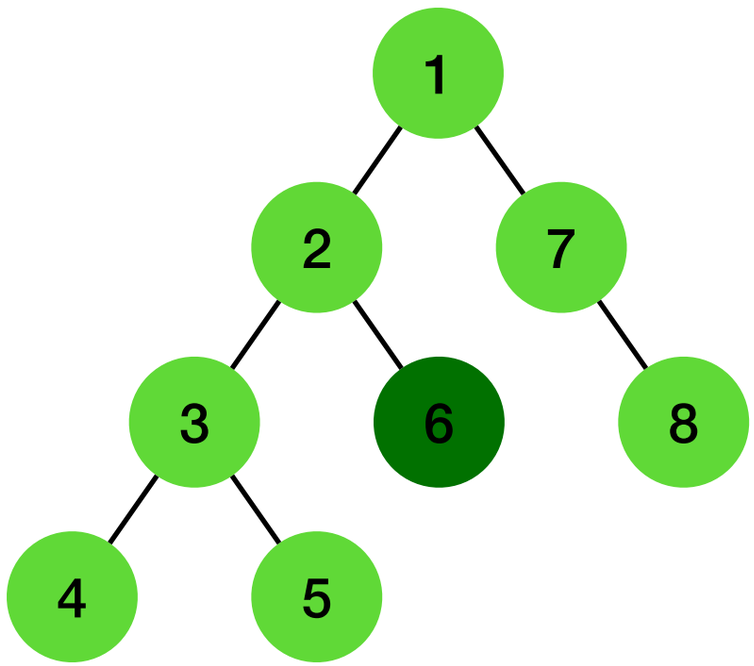


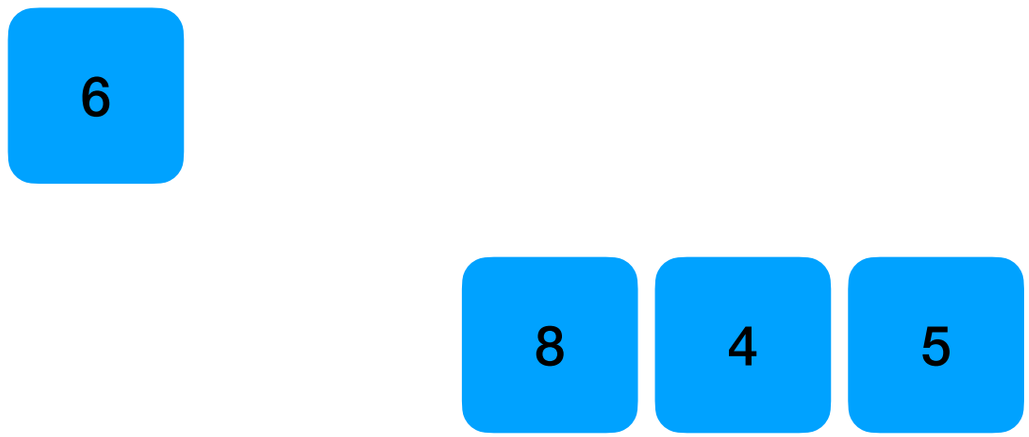


1 2 7 3

🌳Tree

遍歷__BFS

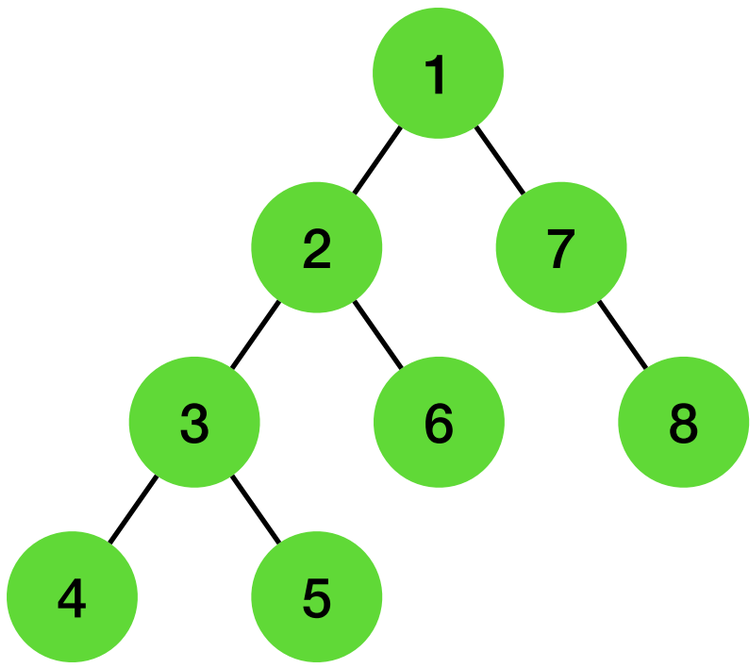




1 2 7 3 6

🌳Tree

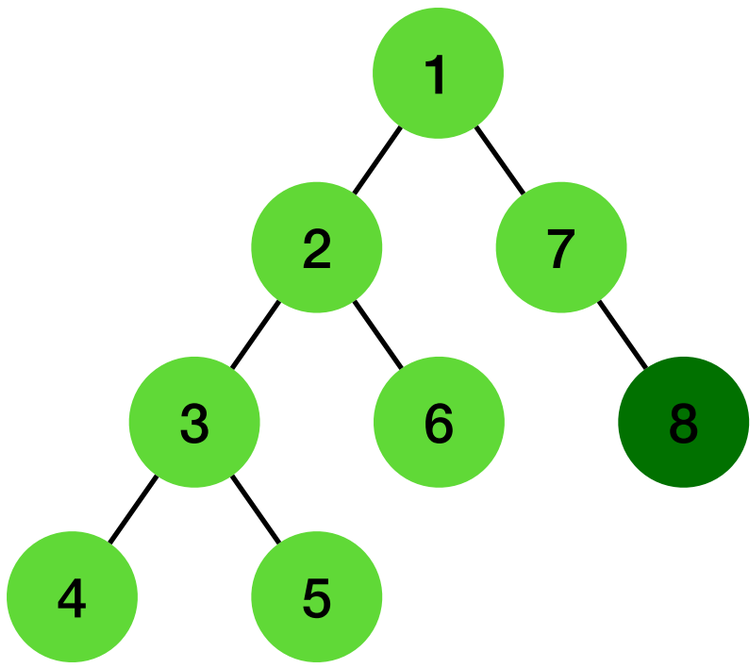
遍歷__BFS

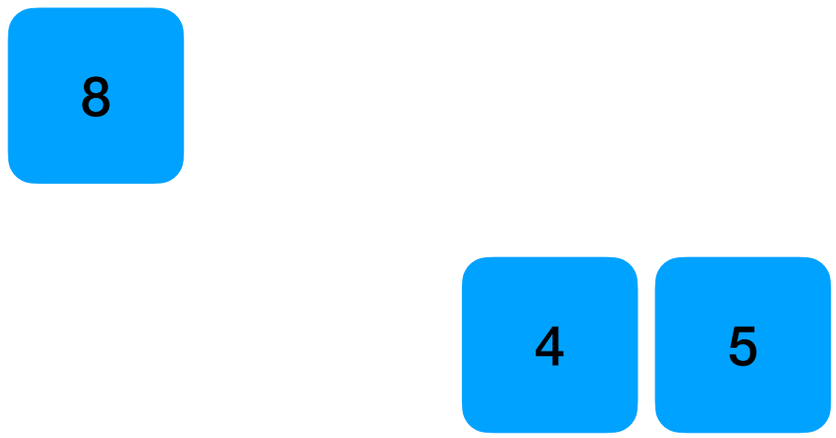


1 2 7 3 6

🌳Tree

遍歷__BFS

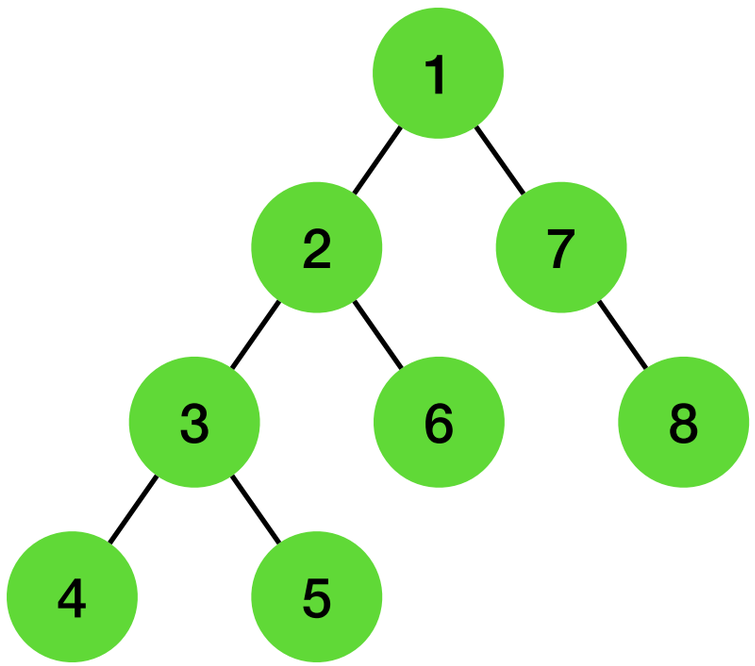


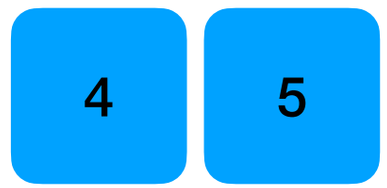


1 2 7 3 6 8

🌳Tree

遍歷__BFS

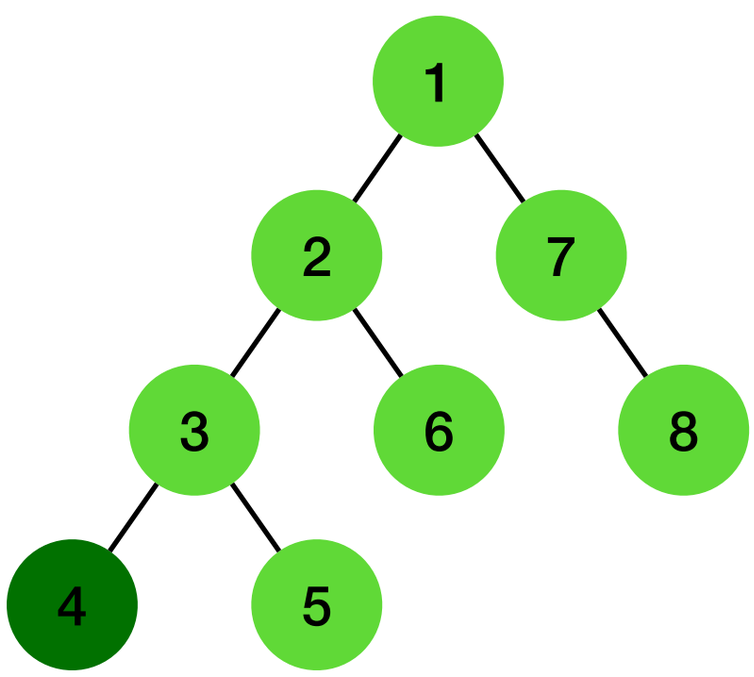




1 2 7 3 6 8

🌳Tree

遍歷__BFS



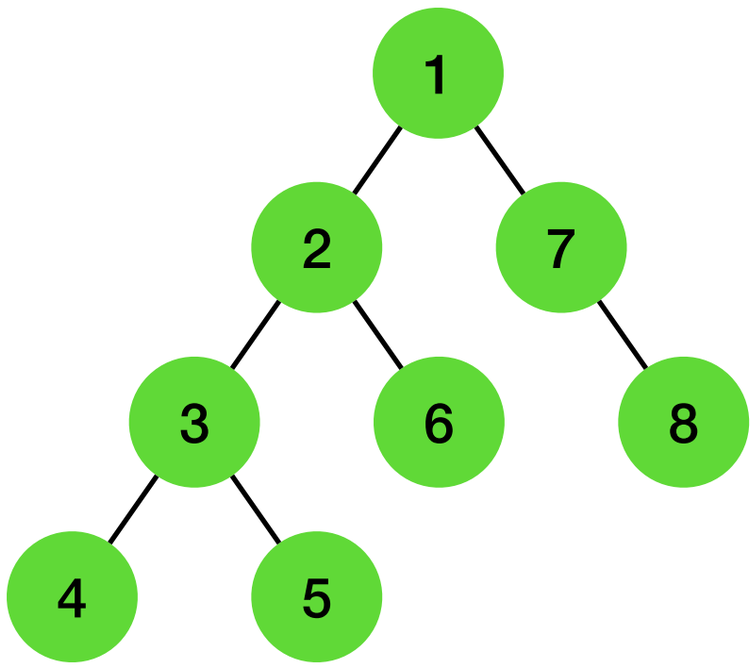
4

5

1 2 7 3 6 8 4

🌳Tree

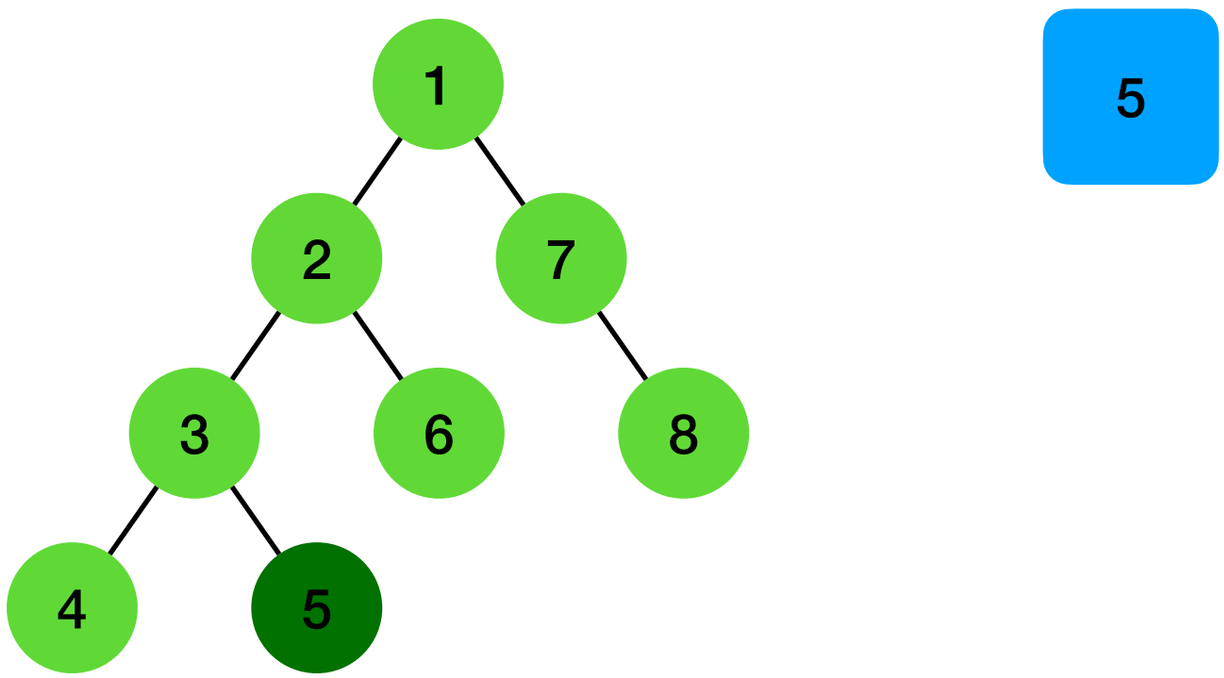
遍歷__BFS



1 2 7 3 6 8 4

🌳Tree

遍歷__BFS

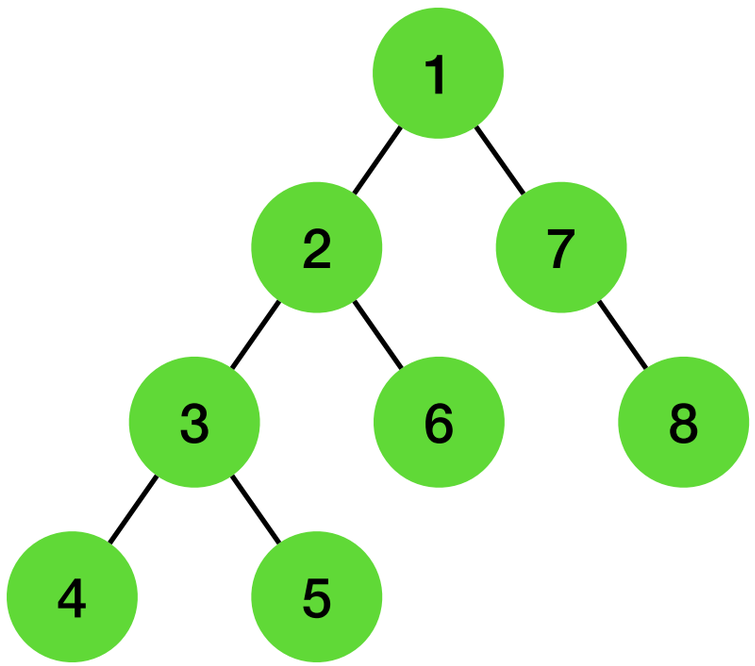


5

1 2 7 3 6 8 4 5

🌳Tree

遍歷__BFS



1 2 7 3 6 8 4 5