탐색 알고리즘

주어진 데이터에서 원하는 값을 찾는 알고리즘이다.

1. Linear Search Algorithm (선형 탐색 알고리즘)

⇒ 맨 앞 or 맨 뒤에서 순서대로 하나씩 찾아보는 알고리즘이다.

2. Binary Search Algorithm (이진 탐색 알고리즘)

⇒ 중간 지점을 기준으로 데이터를 분석 나눠서 탐색하는 알고리즘이다.

3. Hash Search Algorithm (해시 탐색 알고리즘)

⇒ value와 key를 미리 연결함으로 짧은 시간내에 탐색할 수 있는 알고리즘.

- 해시함수로 데이터를 저장하는 알고리즘 ····· ①
- 해시함수로 데이터를 검색하는 알고리즘 ····· ②

① 데이터를 저장할때, 배열크기를 데이터의 1.5 배 ~ 2배 정도의 크기로 관리한다.

"값 % 길이 ⇒ 저장" 이런식으로 저장되므로 다른 데이터와 동일한 hash 값이 나오면 conflict가 생기므로 크기를 키워서 index+1 을 하거나 linked list를 사용한다.

② 저장할 때 처음 데이터를 해시함수로 찾는다. 나올때까지 +1 시켜주고 0을 만나면 없다는 의미로 해석하면된다.

↳ conflict 한 경우를 생각!

4. Binary Search Tree (이진 탐색 트리)

⇒ tree 자료구조에 의거해 탐색하는 자료구조이다.

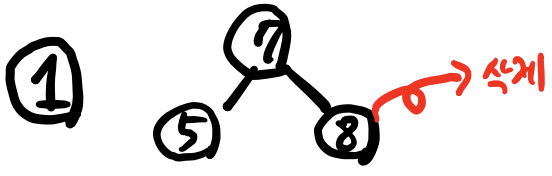
root를 기준으로 왼쪽 subtree들은 작은 값을 가지고 오른쪽은 큰 값을 가진다

이를 활용한 탐색, 삽입 알고리즘이 가능하다! (삭제도 마찬가지)

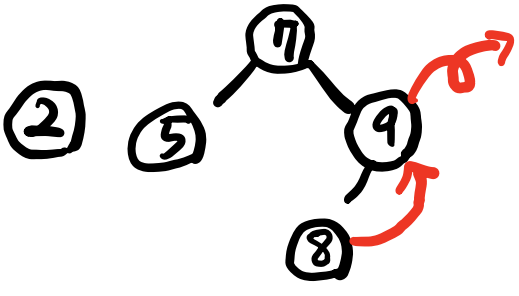
① 탐색 알고리즘 :
- root 보다 큰 값이면 → 오른쪽 subtree로 이동
- root 보다 작은 값이면 → 왼쪽 subtree로 이동
- root 값과 같은 값이면 → 탐색 종료, 값 return
- 없으면 None return! (NULL, 이건 사용자 마음)

② 삽입 알고리즘 :
- ①, 탐색 수행
- 탐색 실패하면, 탐색이 종료된 위치에 노드를 삽입한다.
- 탐색에 성공하면, 이미 있는 값이므로 넣지않고 넘어간다.

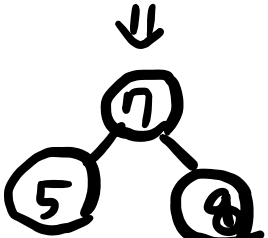
③ 삭제 알고리즘 :
- ①, 탐색수행
- 삭제하려는 노드에 따라 달라진다.



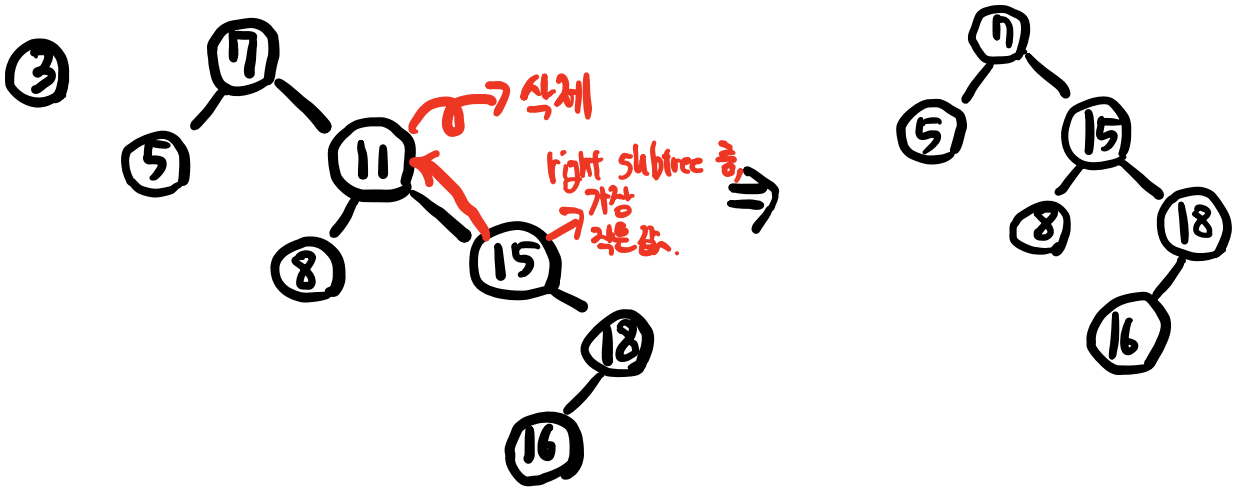
1. leaf node 일 경우. (자식이 없는 node)
→ 그냥 삭제한다



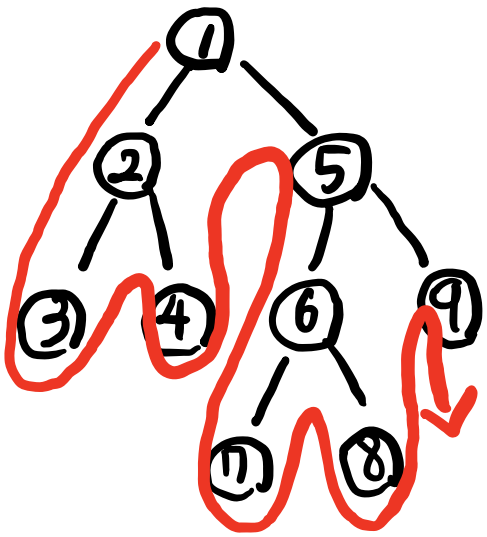
2. 자식이 한 개 있는 node 일 경우
⇒ 자신이 삭제된 node의 위치에 자식을 붙여넣는다.



3. 자식이 두 개 있는 node 일 경우
⇒ 오른쪽 subtree 중, 가장 작은 값을 선택해
삭제된 node 위치에 붙인다.
⇒ 가장 작은 것이 들어가야 BST의 성질을 유지할 수 있다!



5. Depth First Search (깊이 우선 탐색)



옆 그림의 node 번호 순서대로 → 그래프 구조 확인에 좋음

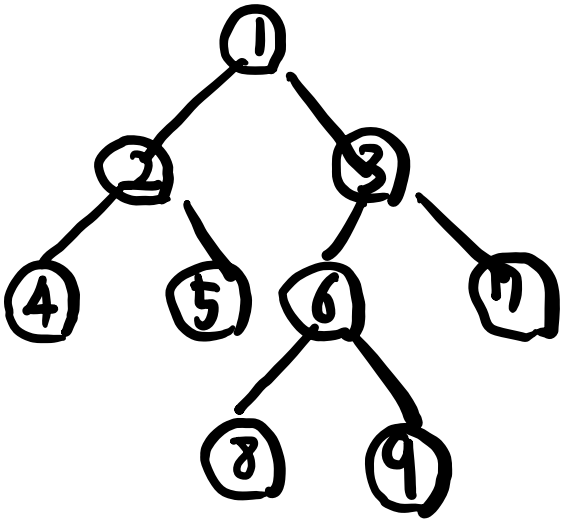
탐색하는 알고리즘이다.

방문을 따로 가지며, 방문한 node를 다시 돌지 않게 한다

ex) 1 → 2 → 3 → 4 (3은 이미 방문 했었다!)

다시 back

6. Breadth First Search (너비 우선 탐색)



옆그림의 node 번호 순서대로 → 최단 경로 찾는데 좋음

탐색하는 알고리즘이다.

방문을 따로 가지며, 방문한 node를 기록하고 인접한 node를 queue에 저장한다. queue에서 node를 꺼내어 해당 node와 연결된 인접 node들에서 방문하지 않은 node를 queue에 추가한다.

ex) ①→②③을 queue에 저장

queue의 FIFO 성질에 의해 ②가 pop 되고,

②의 인접 node를 queue에 추가, 그다음 ③이 pop

⋮