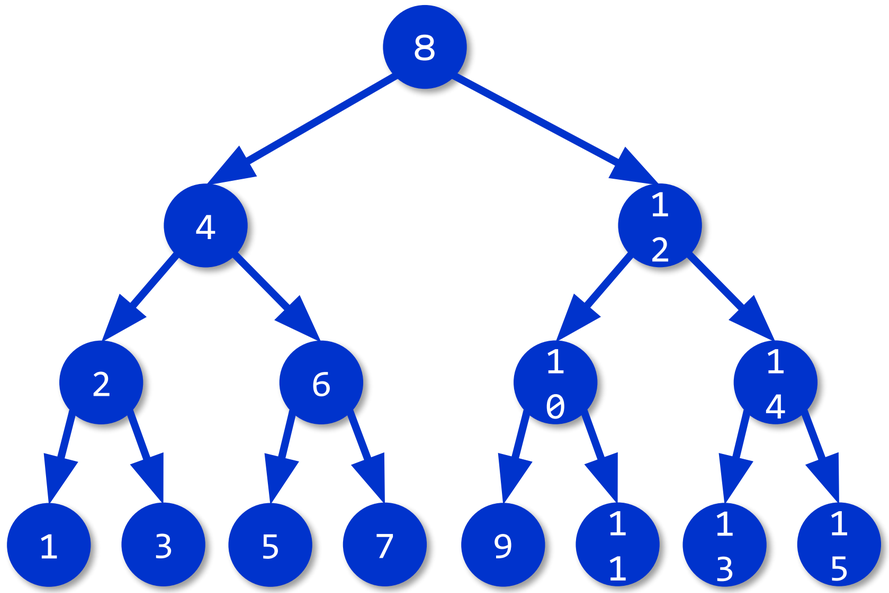
③ ハッシュテーブルとの比較

集合の実現方法をハッシュテーブルと比べてみます。

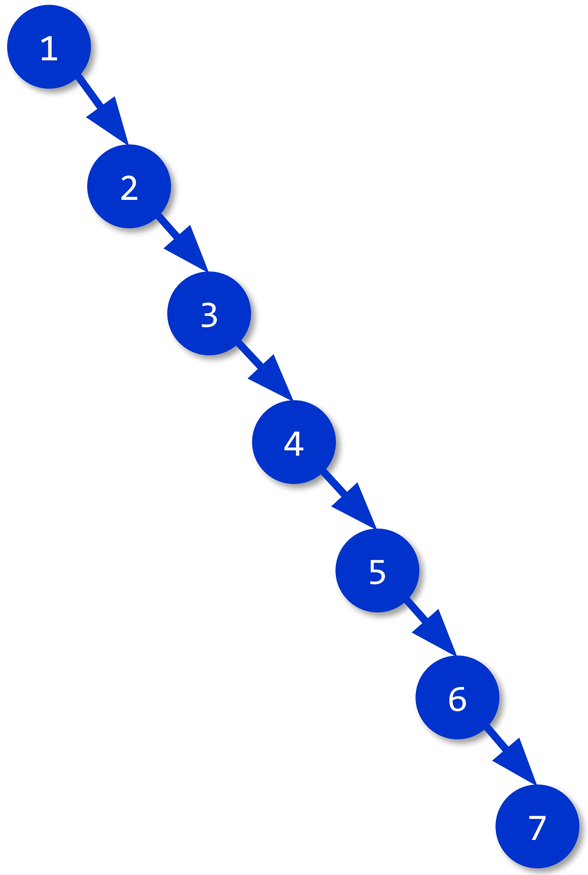
二分探索木の計算量

- 二分探索木に要素が均等に追加されていくとすると、n 個の要素を含む集合を表す木の高さは、約 $\log_2 n$ になります
- 従って、理想的な二分探索木における各種操作の計算量は以下のようになります
  - 追加　$O(\log n)$
  - 削除　$O(\log n)$
  - 検索　$O(\log n)$
  - 最小値の探索　$O(\log n)$
  - 最大値の探索　$O(\log n)$
  - ※平均的な場合も、同様になります



二分探索木の計算量（最悪の場合）

- 一方で、偏った順に要素の追加がなされると、一方向にのみ枝が伸びてしまう可能性があります
  - 具体的には、ソートされたリストを順に追加した場合です



- このような場合の、各種操作の計算量は：　$O(n)$

ハッシュテーブルとの比較

- 計算量の観点で比較すると、次のようになります（nは要素数）
  - ハッシュテーブルのサイズは適切に（十分に大きく）設定されている前提です

	追加	削除	検索	最小値	最大値
配列	$O(n)$	$O(n)$	$O(n)$	$O(n)$	$O(n)$
ハッシュテーブル	$O(1)$	$O(1)$	$O(1)$	$O(n+N)$	$O(n+N)$
二分探索木	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$	$O(\log n)$

※ハッシュテーブルの場合、ハッシュテーブルの配列長を N とします。（要素数と配列長の大きい方に比例します。）

追加、削除、検索のみが求められる場合には、一般にハッシュテーブルを用いた方が良いことが多い

要素の大小関係を考慮した処理（最小値・最大値の探索、範囲内の要素の探索など）の場合は、二分探索木を用いた方が良い