

9章

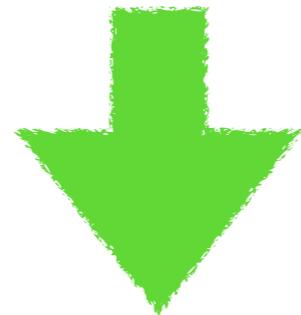
二分探索木

M1 mokky



9.1 問題にチャレンジする前に

探索木は、挿入、検索、削除などの操作が行えるデータ構造で、辞書あるいは優先度付きキューとして用いることができる



探索木の中でも最も基本的なものが二分探索木

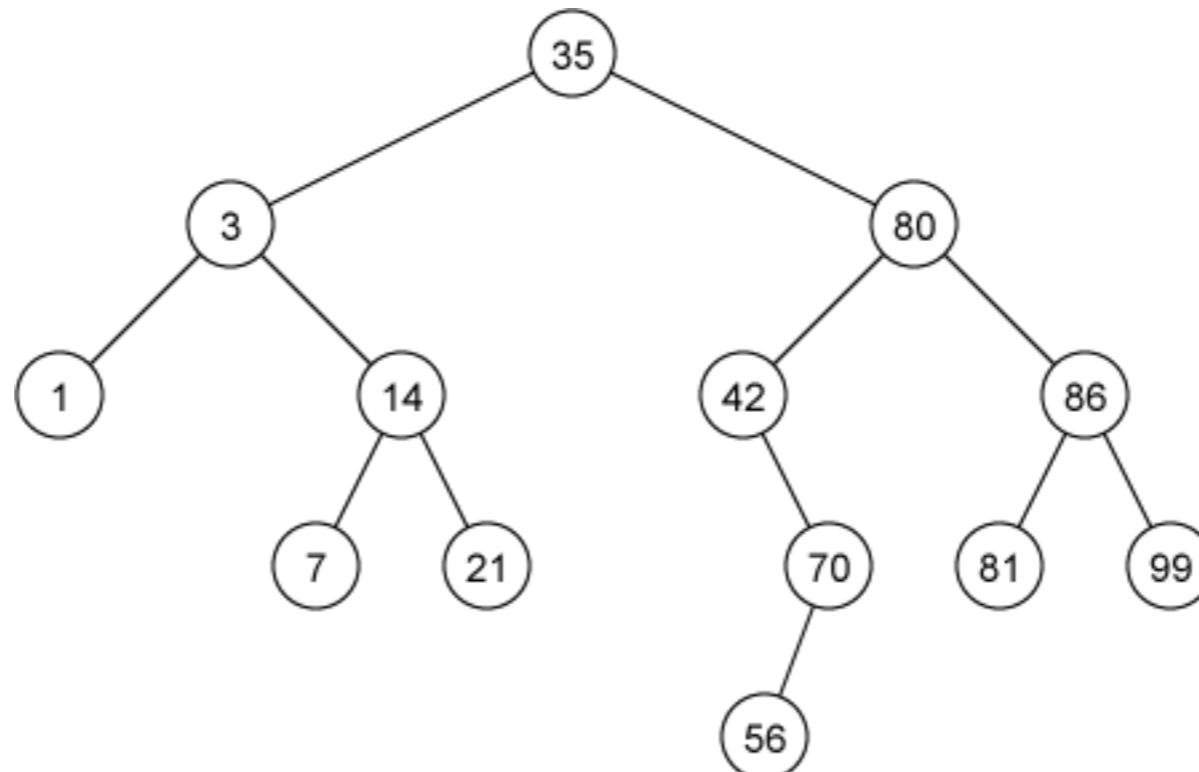
9.1 問題にチャレンジする前に

二分探索木条件(Binary search tree property)

x を二分探索木に属するある節点とする。

y を x の左部分木に属する節点とすると、 y のキー $\leq x$ のキーである。

また、 z を x の右部分木に属する節点とすると、 x のキー $\leq z$ のキーである。



9.2挿入

入力 入力の最初の行に、命令の数 m が与えられます。続く m 行に、`insert k` または `print` の形式で各命令が 1 行に与えられます。

出力 `print` 命令ごとに、中間順巡回アルゴリズム、先行順巡回アルゴリズムによって得られるキーの列をそれぞれ 1 行に出力してください。各キーの前に 1 つの空白を出力してください。

制約 命令の数は 500,000 を超えない。また、`print` 命令の数は 10 を超えない。

$$-2,000,000,000 \leq k \leq 2,000,000,000$$

上記の疑似コードのアルゴリズムに従う場合、木の高さは 100 を超えない。

二分探索木中のキーに重複は発生しない。

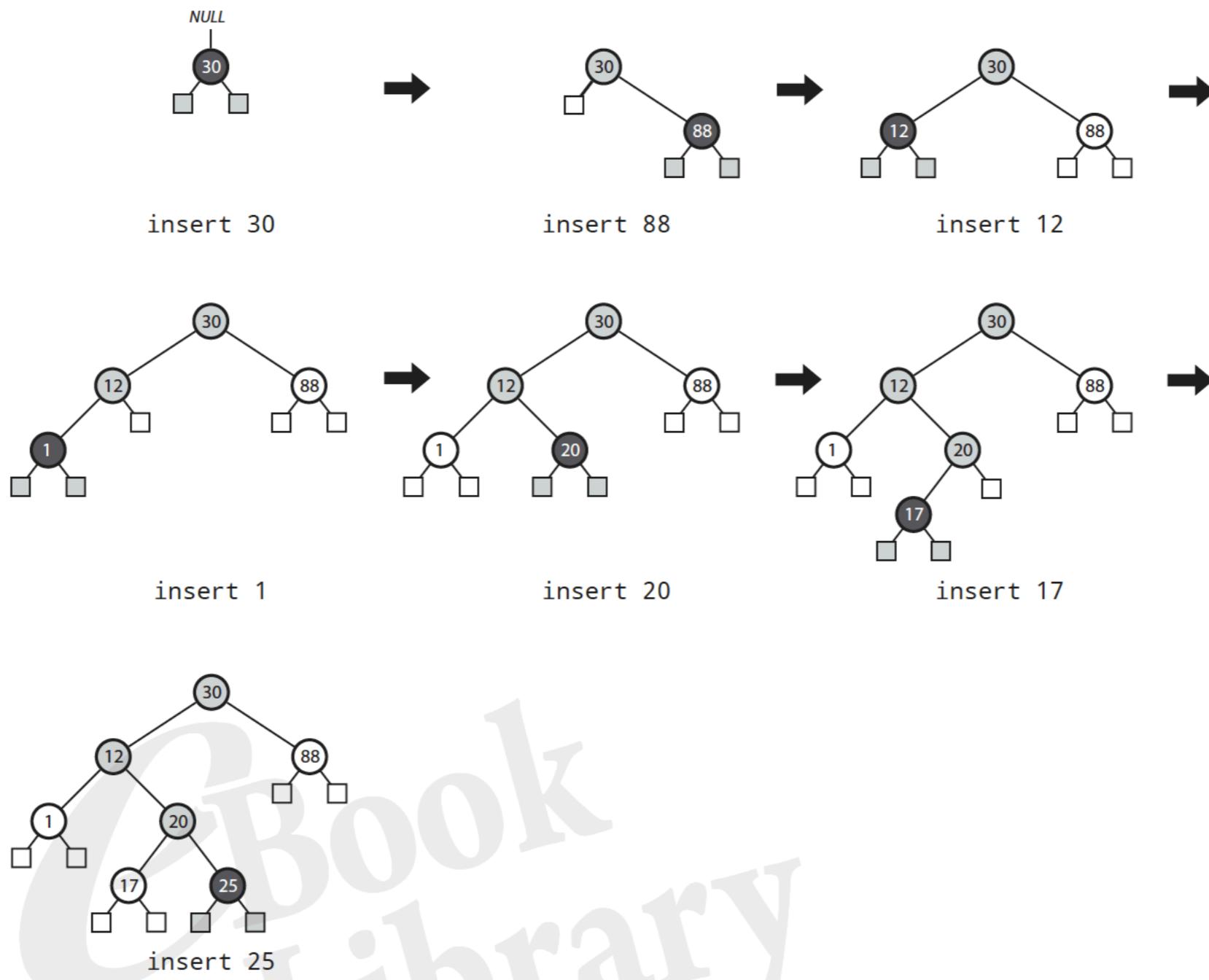
入力例

```
8
insert 30
insert 88
insert 12
insert 1
insert 20
insert 17
insert 25
print
```

出力例

```
1 12 17 20 25 30 88
30 12 1 20 17 25 88
```

9.2.1 解説



9.2.2 考察

二分探索木への挿入操作の計算量は、木の高さを h とすると $O(h)$ となる。
つまり、節点の個数を n とすると、入力に偏りがなければ $O(\log n)$ になる。
一般的には、挿入されるキーとそれらの順番によっては木の高さが高くなる
可能性があり、最悪の場合は節点の数 n に近づき計算量が $O(n)$ となってしまう。

9.3 探索

入力 入力の最初の行に、命令の数 m が与えられます。続く m 行に、insert k 、find k または print の形式で命令が 1 行に与えられます。

出力 find k 命令ごとに、 T に k が含まれる場合 yes と、含まれない場合 no と 1 行に出力してください。

さらに print 命令ごとに、中間順巡回アルゴリズム、先行順巡回アルゴリズムによって得られるキーの列をそれぞれ 1 行に出力してください。各キーの前に 1 つの空白を出力してください。

制約 命令の数は 500,000 を超えない。また、print 命令の数は 10 を超えない。

$$-2,000,000,000 \leq k \leq 2,000,000,000$$

上記の疑似コードのアルゴリズムに従う場合、木の高さは 100 を超えない。

二分探索木中のキーに重複は発生しない。

入力例

```
10
insert 30
insert 88
insert 12
insert 1
insert 20
find 12
insert 17
insert 25
find 16
print
```

出力例

```
yes
no
1 12 17 20 25 30 88
30 12 1 20 17 25 88
```

9.3.1 考察

find操作もinsertと同様に木の高さを h とすると、 $O(h)$ のアルゴリズムとなる。

9.4 削除

二分探索木 T から与えられたキー k を持つ節点 z を削除する $\text{delete } k$ は以下の 3 つの場合を検討したアルゴリズムに従い、二分探索木条件を保ちつつ親子のリンク（ポインタ）を更新します：

1. z が子を持たない場合、 z の親 p の子（つまり z ）を削除する。
2. z がちょうど 1 つの子を持つ場合、 z の親の子を z の子に変更、 z の子の親を z の親に変更し、 z を木から削除する。
3. z が子を 2 つ持つ場合、 z の次節点 y のキーを z のキーへコピーし、 y を削除する。
 y の削除では 1. または 2. を適用する。ここで、 z の次節点とは、中間順巡回で z の次に得られる節点である。

入力 入力の最初の行に、命令の数 m が与えられます。続く m 行目に、`insert k`、`find k`、`delete k` または `print` の形式で命令が 1 行に与えられます。

出力 `find k` 命令ごとに、 T に k が含まれる場合 `yes` と、含まれない場合 `no` と 1 行に出力してください。

さらに `print` 命令ごとに、中間順巡回アルゴリズム、先行順巡回アルゴリズムによって得られるキーの列をそれぞれ 1 行に出力してください。各キーの前に 1 つの空白を出力してください。

制約 命令の数は 500,000 を超えない。また、`print` 命令の数は 10 を超えない。

$$- 2,000,000,000 \leq k \leq 2,000,000,000$$

上記の疑似コードのアルゴリズムに従う場合、木の高さは 100 を超えない。
二分探索木中のキーに重複は発生しない。

9.4 削除

入力例

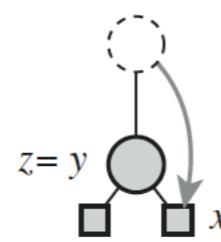
```
18
insert 8
insert 2
insert 3
insert 7
insert 22
insert 1
find 1
find 2
find 3
find 4
find 5
find 6
find 7
find 8
print
delete 3
delete 7
print
```

出力例

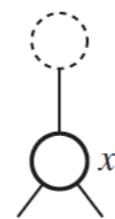
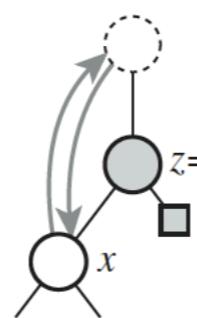
```
yes
yes
yes
no
no
no
yes
yes
1 2 3 7 8 22
8 2 1 3 7 22
1 2 8 22
8 2 1 22
```

9.4.1 解説

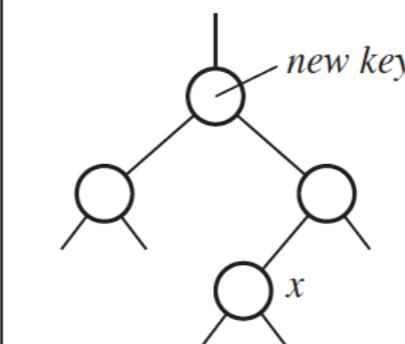
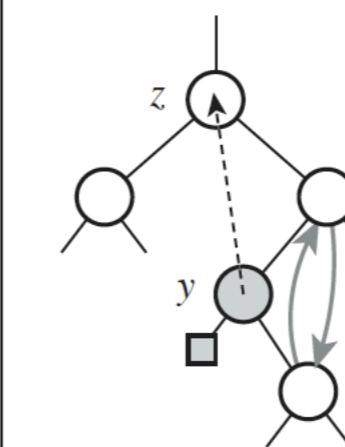
case 1



case 2



case 3



9.4.2 考察

ここで示した二分探索木からの要素の削除は、木の高さを h とすると、まず与えられたキーを持つノードを探索するために $O(h)$ 、次節点を求めるためにも $O(h)$ の計算を必要とするため、 $O(h)$ のアルゴリズムとなる。

一般的に、挿入、探索、削除を行うことができる二分探索木は、それらの計算量が $O(\log n)$ となるように実装される（ n は節点の数）。

そのためには、常に木の高さをできるだけ低く維持する必要がある。
このようなバランスのとれた木を、平衡二分探索木と呼ぶ。

9.5 標準ライブラリによる集合の管理

シーケンスコンテナ

追加される要素はその値に関係なく特定の位置に配置され、
その位置は挿入した時間と場所に依存する。

例 : list, vector

連想コンテナ

追加される要素の位置は何らかのソート基準に従って決められる。
連想コンテナの特長は、常に二分探索が行えることであり、
要素の探索を高速に行うことができる。

9.5.1 set

set は、要素が値によってソートされている集合で、挿入された要素は集合の中にただ 1 つ存在し、要素の重複がない。

関数名	機能	計算量
size()	セットの中の要素数を返します。	$O(1)$
clear()	セットの中を空にします。	$O(n)$
begin()	セットの先頭を指すイテレータを返します。	$O(1)$
end()	セットの末尾を指すイテレータを返します。	$O(1)$
insert(key)	セットに要素 key を挿入します。	$O(\log n)$
erase(key)	key を持つ要素を削除します。	$O(\log n)$
find(key)	key に一致する要素を探索し、その要素へのイテレータを返します。 key に一致する要素が無ければ末尾 end() を返します。	$O(\log n)$

二分探索木で実装されているが、平衡化されているから常に $O(\log n)$

9.5.2 map

mapはキーと値の組を要素とする集合で、各要素は1つのキーと1つの値を持ち、キーをソートの基準とする。

集合の中で各キーは必ずひとつだけ存在し、重複はない。

map は任意の型の添え字を使用することができる連想配列として使用することができる。

関数名	機能	計算量
size()	マップの中の要素数を返します。	$O(1)$
clear()	マップの中を空にします。	$O(n)$
begin()	マップの先頭を指すイテレータを返します。	$O(1)$
end()	マップの末尾を指すイテレータを返します。	$O(1)$
insert((key, val))	マップに要素(key, val)を挿入します。	$O(\log n)$
erase(key)	keyを持つ要素を削除します。	$O(\log n)$
find(key)	keyに一致する要素を探索し、その要素へのイテレータを返します。 keyに一致する要素が無ければ末尾end()を返します。	$O(\log n)$

平衡な二分探索木で実装されているから常に $O(\log n)$