Структура данных

Будем использовать дополненное самобалансирующееся дерево поиска. Правое поддерево содержит элементы больше корня, а левое меньше корня.

Для каждого узла дерева будем хранить:

* min – минимальное значение в поддереве, включая вершину.
* max – максимальное значение в поддереве, включая вершину.
* size –количество элементов в поддереве, включая вершину.
* Указатель на правого ребенка
* Указатель на левого ребенка
* Указатель на родителя

В дальнейшем максимум/минимум/размер правого поддерева для узла x – x->right.min/max/size соответственно, для левого x->left.min/max/size,

Эти свойства можно сохранять, притом insert и delete будут работать за . Операции insert и delete делаем по общеизвестным алгоритмам, затем поднимаемся вверх по дереву до корня, обновляя значения min, max, size для каждого встретившегося элемента. В итоге будет совершено действий.

Свойство 1 такого дерева: Если для какого-то узла node выполняется node.max – node.min + 1 = node.size, то в правом поддереве содержится size последовательных элементов.

Алгоритм операции find (3й):

1. Находим нужный нам элемент x.
   1. Если x->right = NIL (там нет элементов), то идем вверх.
2. X->right.min > x + 1, то искомый ответ x+1.
3. 1. Если 2 не выполнено идем от x вверх к корню.
4. 2. Если мы пришли в вершину слева, то сравниваем x и значение x.parent, если разница более 1, то ответ x + 1.
5. Пусть node = x.parent – узел, в который мы пришли, проверяем для node свойство 1.
6. 1. Если оно не выполнено, то ответ, который мы ищем находится в правом поддереве. Тогда его можно найти за.
   1. Если 2 выполняется, то ответ точно не в этом поддереве и нужно продолжать движение вверх. При этом x увеличить x = node.max.
7. Когда дошли до корня, если ответ все еще не найден, то он либо в правом поддереве, либо root.max + 1(все так же по свойству).

Асимптотика алгоритма:

1. Поиск x – .
2. Поиск 4.1. (в правом поддереве) тоже .
3. Подъем до вверх (максимум до корня) не более .
4. В итоге find(x) работает за .