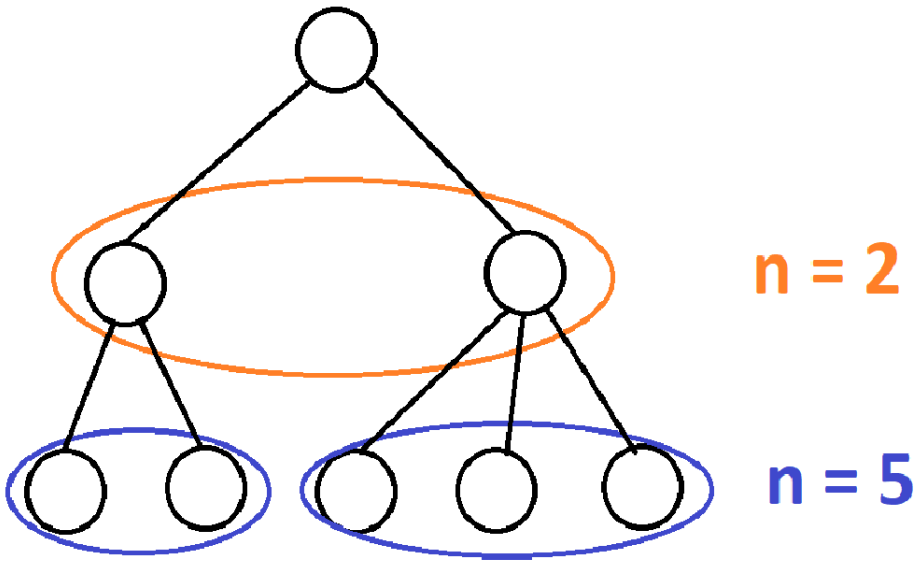
ДЗ3

Наумов Иван М34391

10.03.2022

Построение снизу

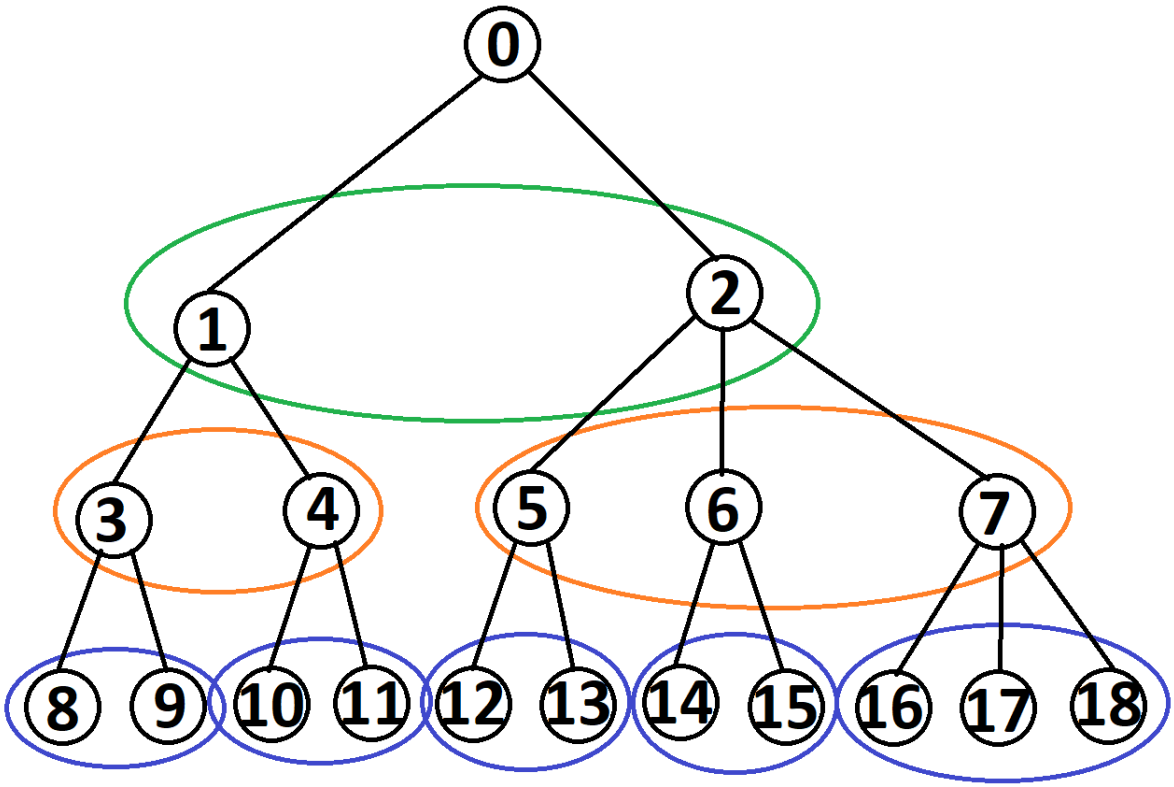
Поскольку в 2-3 дереве все листья на одной глубине, можно легко узнать количество вершин на каждом слое. Будем строить дерево по слоям - на каждом слое присоединяем соседние пары вершин к одному родителю. Если на слое нечетное число вершин - объединим последнюю к двум предпоследним.



Построение сверху

При использовании рекурсии гораздо удобнее строить дерево сверху, чтобы родитель заранее мог определить своих детей. В целом, построение 2-3 дерева таким образом очень похоже на построение, скажем, дерева отрезков, однако в данном случае необходимо более аккуратно пересчитывать индексы вершин, и знать, когда создать 3 детей. Для этого будет достаточно для слоя высоты h ($h_{root} = 0$) уметь определять количество вершин на нем.
Для начала заметим, что если на слое высоты h расположено n вершин, то на слое глубиной $h + 1$ (родительский) вершин будет $\lfloor \frac{n}{2} \rfloor$. Таким образом, вычислив максимальную глубину рекурсии (which is $\lfloor \log n \rfloor$) мы можем передавать параметр высоты и быстро считать число вершин на текущем уровне.

Касаемо индексации вершин - будем нумеровать их сверху вниз, слева направо.



Если индекс текущей вершины v, высота слоя h и слева от нее на текущем слое vl вершин, то индекс самого левого сына будет:

$$v + (\left\lfloor \frac{n}{2^h} \right\rfloor - vl) + (2 * vl) =$$

$$= v + \left\lfloor \frac{n}{2^h} \right\rfloor + vl$$

Здесь мы пользуемся фактом, что все левые вершины будут "двоиться но не "троиться". Таким образом, будем еще передавать в рекурсию число вершин слева от нашей.

Как определить окно для родителя с 3мя сыновьями?

Если у родителя 3 сына, то поддеревья двух левых - полные бинарные деревья, значит их размер 2^{h-1}, исходя из чего можно легко пересчитать окна для сыновей:
$[l, l + 2^{h-1})$
$[l + 2^{h-1}, l + 2^h)$
$[l + 2^h, r)$
Где $[l, r)$ - окно родителя.

```
# a - sorted array
# [l, r) - range in array a
# tree - array with tree nodes
# v - index of current vertex in tree array
# h - height of vertex tree[v]
# vl - amount of nodes to the left of tree[v] in current layer
fun build1(a, l, r, tree, v, h, vl):
    if h == 0:
        tree[v].keys[0] = tree[v].keys[1] = a[l]
        return a[l]
    w = len(a) // (2 ** h) # amount of nodes in current layer
    ch = v + w + vl
    if w % 2 == 1 and r == len(a): # tree[v] has 3 children
        tree[v].length = 3
        tree[v].sons = [ch, ch + 1, ch + 2]
        tree[ch].parent = tree[ch + 1].parent = tree[ch + 2].parent = v
        fork2join({
        for2join(
        m1 = { build1(a, l, l + 2 ** (h - 1), tree, ch, h - 1, 2 * vl) },
        m2 = { build1(a, l + 2 ** (h - 1), l + 2 ** h, tree, ch + 1, h - 1, 2 * vl + 1) })},
        m3 = { build1(a, l + 2 ** h, r, tree, ch + 2, h - 1, 2 * vl + 2) }
        )
        tree[v].keys = [m1, m2]
        return max(m1, m2, m3)
    else:
        tree[v].length = 2
        tree[v].sons = [ch, ch + 1, -1]
        tree[ch].parent = tree[ch + 1].parent = v
        fork2join(
        m1 = { build1(a, l, l + 2 ** (h - 1), tree, ch, h - 1, 2 * vl) },
        m2 = { build1(a, l + 2 ** (h - 1), r, tree, ch + 1, h - 1, 2 * vl + 1) }
        )
        tree[v].keys = [m1]
        return max(m1, m2)

fun build(a):
    n = len(a)
    tree = [node()] * (2 * n)
    build1(a, 0, len(a), tree, 0, int(log(n, 2)), 0)
    return tree
```