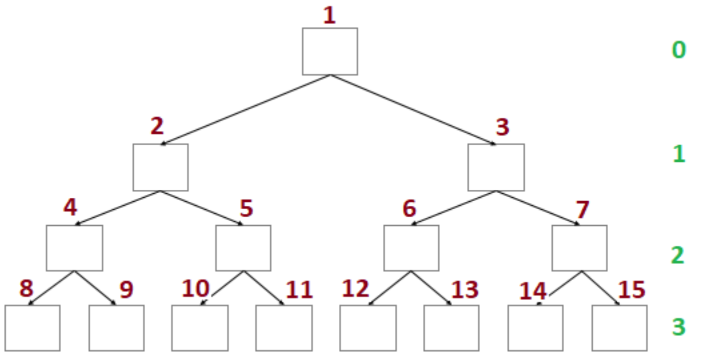
## Сбалансированное двоичное дерево

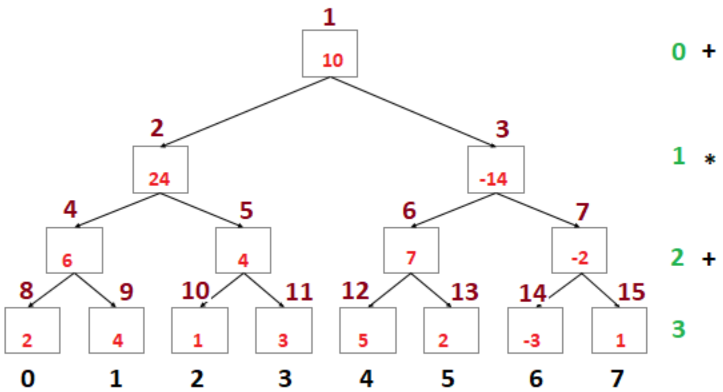
Двоичным сбалансированным деревом называется корнеое дерево такое, что все вершины кроме листьев имеют ровно два потомка, и глубина всех листьев одинакова.




Если пронумеровать все слои дерева, начиная с нуля, то количество вершин на каждом слое будет выражаться формулой $2^k$, где $k$ — номер слоя. Если дерево имеет глубину $k$, то количество вершин будет выражаться формулой $2^{k+1} - 1$.

Если пронумеровать все вершины дерева сверху вниз и слева направо, начиная с единицы, то можно заметить следующую закономерность: левый потомок вершины с номером $s$ имеет номер $2s$, правый потомок имеет номер $2s + 1$, а непосредственный предок имеет номер $[s/2]$. Эта особенность позволяет использовать для хранения такого дерева обыкновенный массив из $2^{k+1}$ элемента. Элемент с номером 0 при этом не используется.

Пусть задан некоторый набор из произвольных двоичных операций и некоторый набор значений. Разместим значения в листьях дерева, а во всех остальных вершинах разместим операции. Произведем вычисления и получим в корне дереав значение некоторого выражения.




На рисунке изображено вычисление выражения с заданными числами. Во всех вершинах, расположенных на четной глубине вычисляется операция сложения, а в вершинах на нечетной глубине операция умножения.

Далее записан примерный листинг функции построения дерева. Для хранения дерева будет использоваться вектор `T`. Вектор `Val` содержит множество значений. На первом этапе определяем глубину дерева `k` так, чтобы количество листьев дерева $2^k$ было не меньшим чем количество значений. Для нахождения $2^k$ используется двоичный сдвиг `1<<k`.

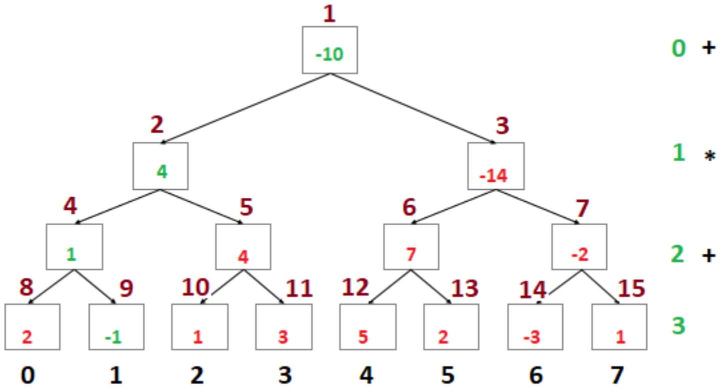
Далее, выделяем память для хранения дерева и инициализируем ее некоторым значением `def`. Это значение должно храниться в листьях, для которых не хватило значений. Вместо `def` требуется подставить нейтральный элемент относительно используемой операции. Например, для операции умножения нейтральным элементом является единица, для сложения нейтральный элемент это 0, для минимума нейтральный элемент это $\infty$.

Далее находим значения во всех узлах дерева. Символы `op` следует заменить на символ требуемой операции.

```
int k;
```

```
vector<int> T;
void build(vector<int> &Val) {
   for (k=1;(1<<k)<Val.size();k++) {}
   T.resize(2*(1<<k),def);
   for (int i=0;i<Val.size();i++)
       T[(1<<k)+i]=Val[i];
   for (int i=(1<<k)-1;i>0;--i)
       T[i]=T[2*i] op T[2*i+1];
}
```

Теперь, пусть один из аргументов выражения изменился. Чтобы пересчитать значение всего выражения достаточно лишь пройти по цепочке от листа к корню и пересчитать значения только в этих вершинах.

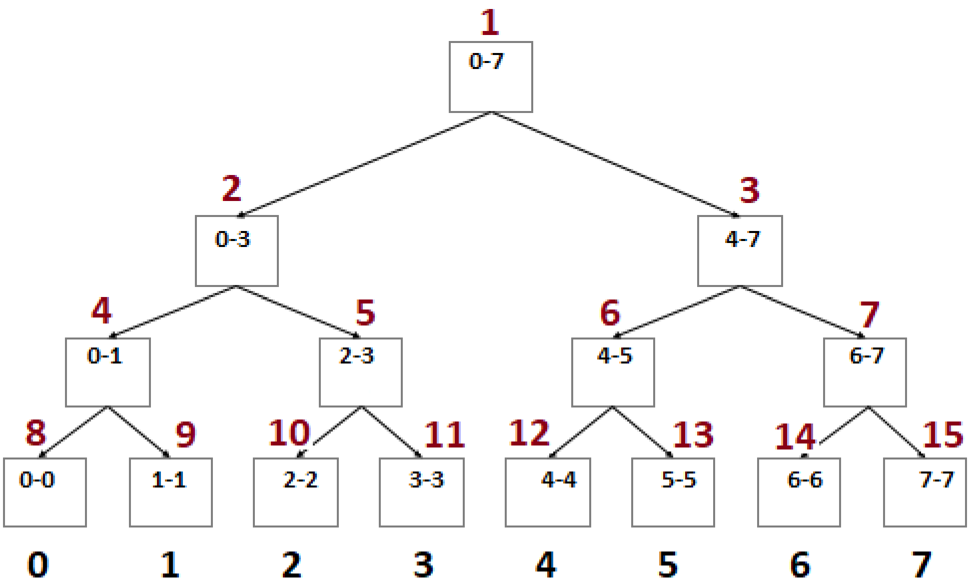



Далее записан листинг функции для изменения элемента дерева. Предполагается, что дерево хранится в массиве `T`, глубина дерева равна `k`, параметр `pos` задает порядковый номер изменяемого значения (номера начинаются с нуля), параметр `val` задает само значение. Для нахождения $2^k$ используется двоичный сдвиг `1<<k`, символы `op` следует заменить на символ требуемой операции.

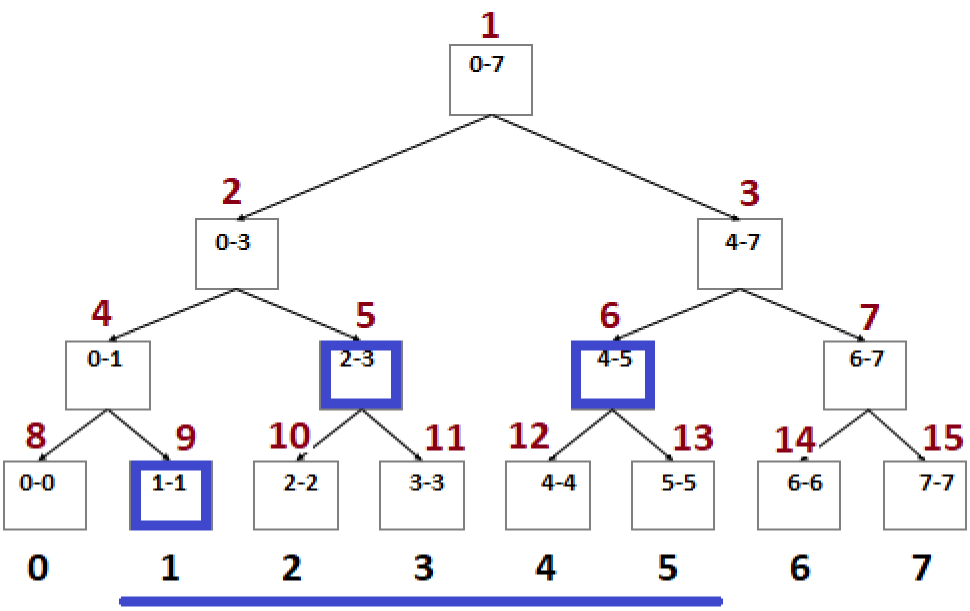
```
void update(int pos,int val) {
    int tree_pos=(1<<k)+pos;
    T[tree_pos]=val;
    for (tree_pos/=2;tree_pos>0;tree_pos/=2)
        T[tree_pos]=T[2*tree_pos] op T[2*tree_pos+1]
}
```

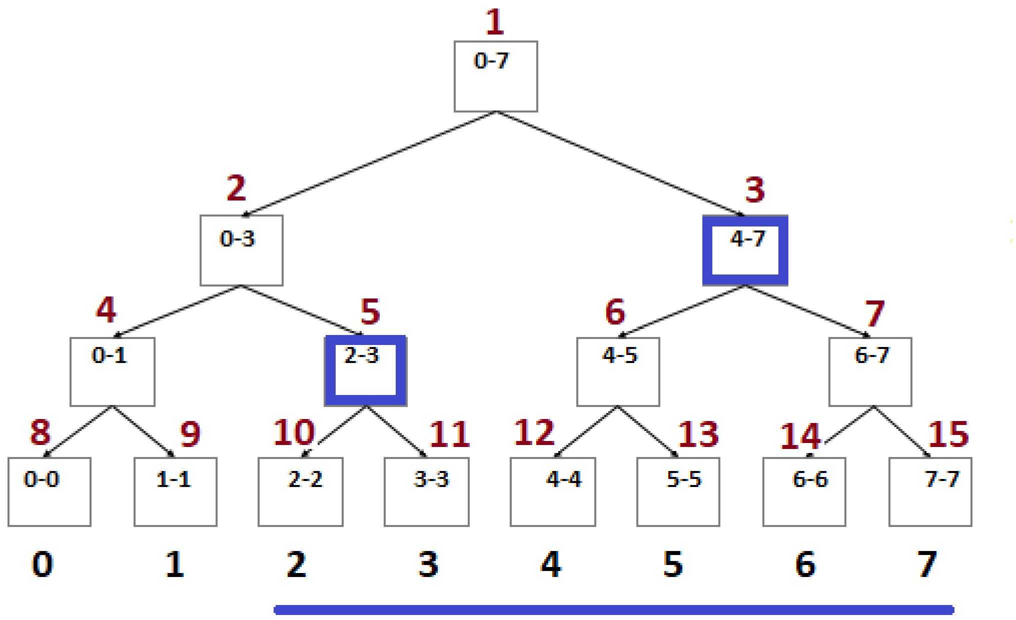
## Дерево отрезков

Теперь, пусть все узлы дерева вычисляют одну и ту же операцию и эта операция ассоциативна, то есть для нее выполняется свойство $(a \circ b) \circ c = a \circ (b \circ c) = a \circ b \circ c$. Тогда, каждая вершина хранит результат применения операции к некоторому отрезку значений.




Например, если в качестве операции используется сложение, то можно найти сумму чисел на любом отрезке, складывая значения в некоторых вершинах дерева.






Далее записан листинг функции для подсчета результата применения операции к заданному отрезку дерева. Закрытый с двух сторон интервал `[l;r]` передается в функцию. Предполагается, что дерево хранится в массиве `T`, глубина дерева равна `k`. Операция задана символами `op`. Предполагается, что для операции дополнительно выполняется свойство коммутативности, то есть $a \circ b = b \circ a$.

```
int get(int l,int r) {
  int val=def;
  for (l+=(1<<k),r+=(1<<k)+1;l<r;l/=2,r/=2) {
      if (l&1) val=val op T[l++];
      if (r&1) val=val op T[--r];
  }
  return val;
}
```