ИСДП

+ Обеспечивает минимальное среднее время поиска.

- Перестройка дерева после случайного включения вершины – довольно сложная операция.

СДП

+ Процедура построения достаточно проста.

- Среднее время поиска на 39% больше, чем у ИСДП

(в худшем случае может выродиться в список).

АВЛ-деревья

Возможное промежуточное решение - ввести менее строгий критерий сбалансированности.

Определение предложено в 1962 году **Г.М. Адельсон-Вельским** и **Е.М. Ландисом**.

Они предложили балансировать дерево по ***высоте***, а не по размеру.

Определение. Дерево поиска называется АВЛ-деревом, если для каждой его вершины высоты левого и правого поддеревьев отличается не больше, чем на единицу.

Замечание:

1) ИСДП является также и АВЛ-деревом

верно

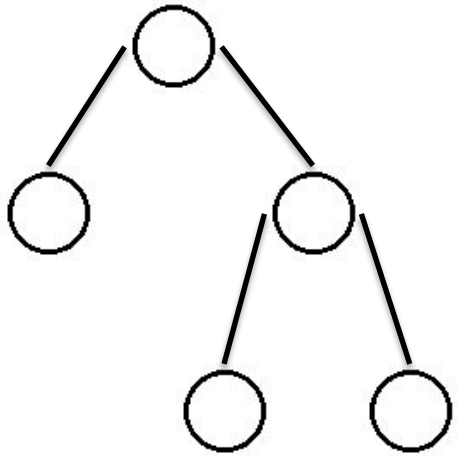
2) АВЛ-дерево является также и ИСДП

не верно

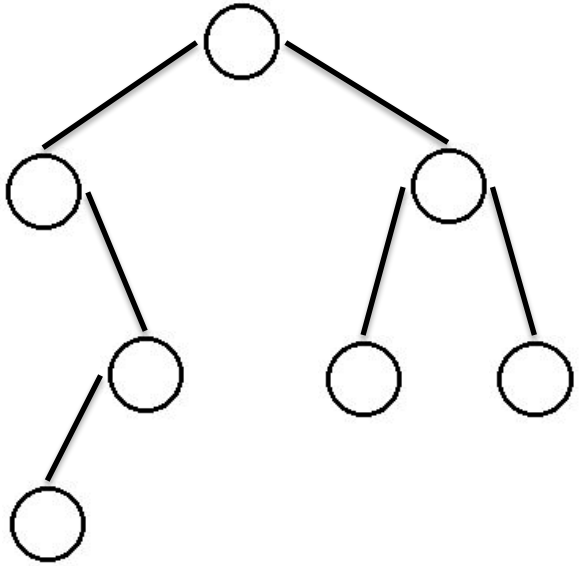
Преимущества:

1) Определение сбалансированности простое;
2) Приводит к простой процедуре перестройки дерева;
3) Среднее время поиска близко к ИСДП.

Какое дерево является АВЛ-деревом?



АВЛ-дерево



не АВЛ-дерево

«Плохие» АВЛ-деревья

Адельсон-Вельский и Ландис доказали **теорему** которая, гарантирует, что

АВЛ-дерево никогда не будет по высоте превышать ИСДП больше, чем на 45% независимо от количества вершин:

$$\log(n+1) \leq h_{АВЛ}(n) < 1{,}44 \log(n+2) - 0{,}328$$

Лучший случай ИСДП

Плохие АВЛ-деревья

Определение.

«***Плохим***» будем называть АВЛ-дерево, которое имеет наименьшее чисто вершин при фиксированной высоте.

Какова структура «плохого» АВЛ-дерева?

Построение: возьмем фиксированную высоту h и построим АВЛ-дерево с минимальным количеством вершин.

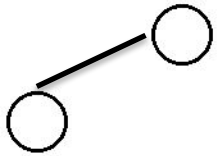
Обозначим такое дерева через T_h

Тогда T_0 – пустое дерево, T_1 - дерево с одной вершиной и т.д.

h=1

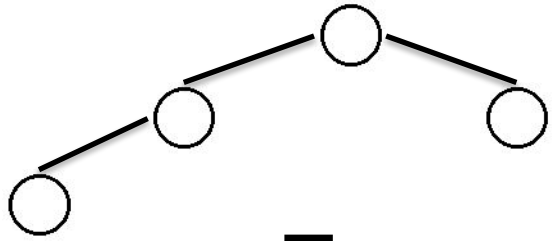
T_1

h=2



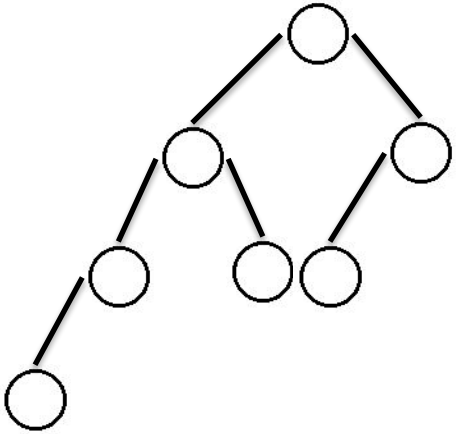
T_2

h=3



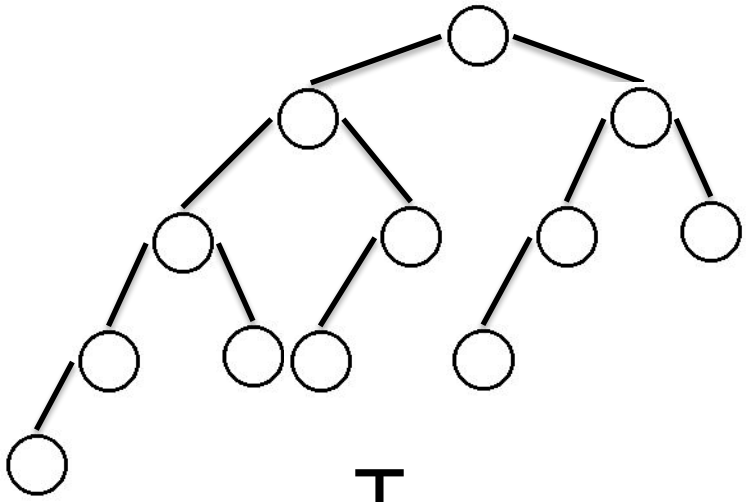
T_3

h=4



T_4

h=5



T_5

Заметим, что

$T_3 = T_2 + T_1 + 1$; $T_4 = T_2 + T_3 + 1$; $T_5 = T_3 + T_4 + 1$.

Для построения T_h для h>1 берем корень и два поддерева с минимальным количеством вершин - высотой T_{h-1} и T_{h-2}

$$T_h = < T_{h-1}, x, T_{h-2} >$$

Алгоритм построения «плохих» АВЛ-деревьев напоминает построение чисел Фибоначчи, поэтому иногда их называют **деревья Фибоначчи**.

Построение АВЛ-дерева

Рассмотрим, что может произойти при включении в сбалансированное по высоте дерево новой вершины.

Пусть добавляется вершина в левое поддерево.

Возможны три случая:

1) Если $h_L < h_R$, то $h_L = h_R$
2) Если $h_L = h_R$, то $h_L > h_R$
3) Если $h_L > h_R$, то $h_L > h_R$ - нарушение баланса и дерево необходимо перестроить.

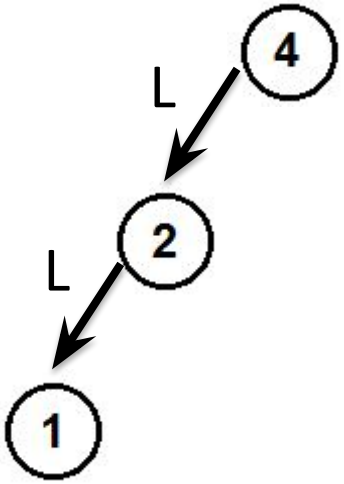
Построение АВЛ-дерева

Пусть добавляется вершина в правое поддерево.

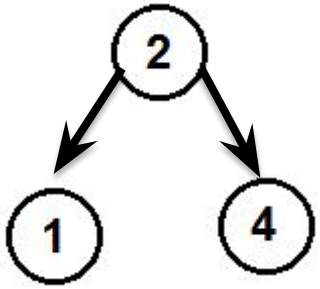
Возможны три случая:

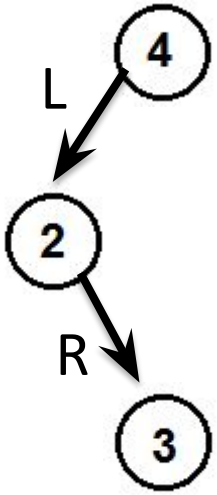
1) Если $h_L > h_R$, то $h_L = h_R$
2) Если $h_L = h_R$, то $h_L < h_R$
3) Если $h_L < h_R$, то $h_L < h_R$ - нарушение баланса и дерево необходимо перестроить.

Рассмотрим перестроение АВЛ-дерева на простых примерах

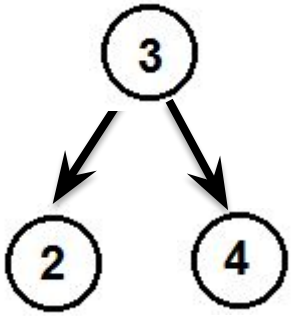


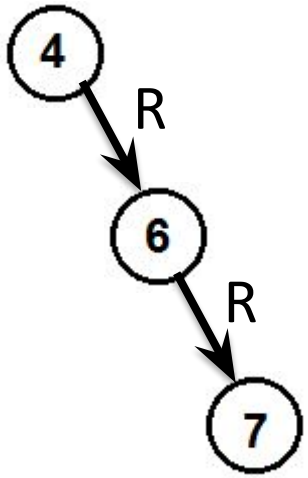
LL - поворот



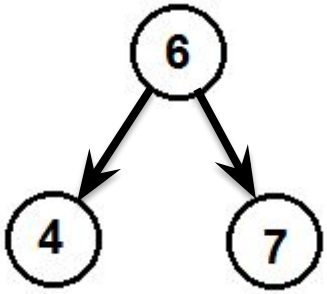


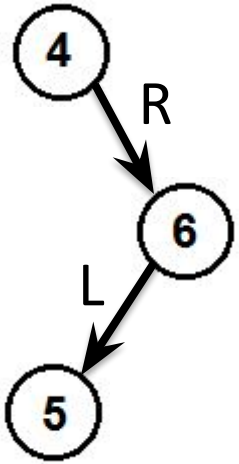
LR - поворот



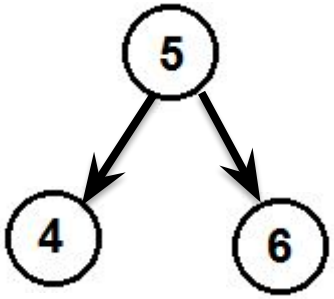


RR - поворот





RL - поворот



Идея алгоритма построения АВЛ-дерева

Вначале ***добавим новую вершину*** в дерево так же ***как в случайное***, т.е. проходим по пути поиска до нужного места включения в качестве листовой вершины.

Двигаясь назад по пути поиска будем искать вершину, в которой нарушился баланс, т.е. высота левого и правого поддерева стала отличаться больше, чем на единицу.

Если такая вершина найдена, то ***изменим структуру дерева*** для восстановления

Задачи при перестроении АВЛ-дерева

1) Как осуществить движение назад по пути поиска?

2) Как определить нарушение баланса?

3) Как восстанавливать баланс?

Решение:

а) Использовать рекурсию, которая позволит хранить адреса всех пройденных вершин по пути поиска и автоматически в них возвращаться на обратном пути рекурсии.

б) введем в каждую вершину дополнительный показатель баланса Balance:

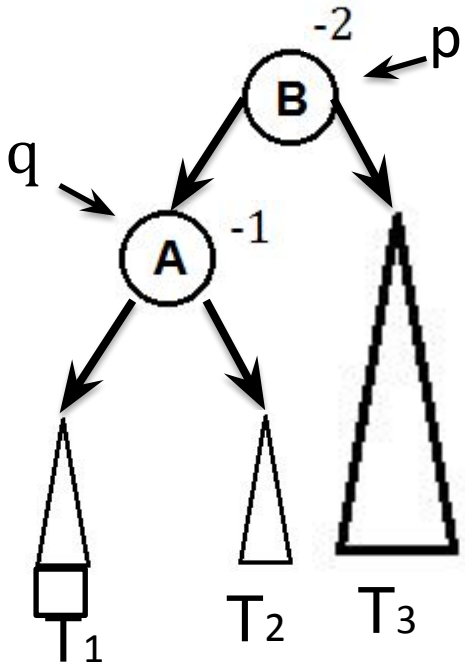
$$\begin{cases} -1 & h_L > h_R \\ 0 & h_L = h_R \\ 1 & h_L < h_R \end{cases}$$

При включении новой вершины её баланс равен нулю. При движении назад по пути поиска *показатель баланса для всех вершин пересчитывается*, причем не нужно просматривать все поддеревья, только путь поиска.

Нарушение баланса возможно только в одной вершине и один поворот полностью восстанавливает структуру АВЛ-дерева.

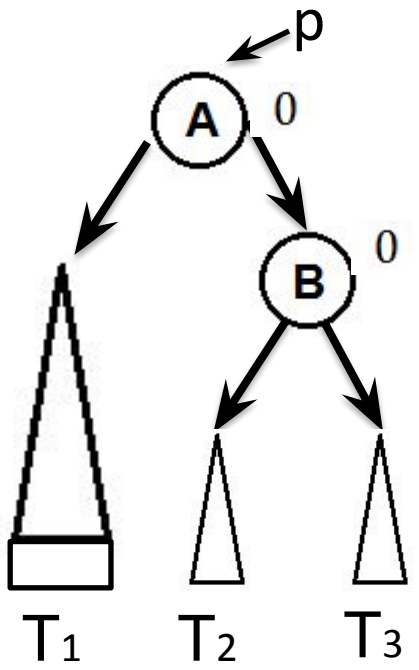
Балансировка выполняется с помощью поворотов дерева: ***LL, LR, RL, RR***.

LL – поворот //RR – поворот симметричен

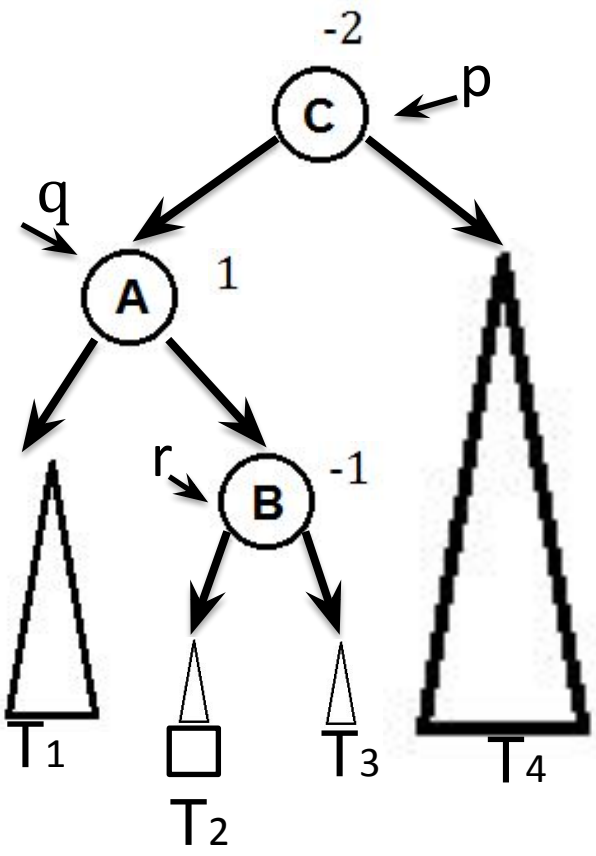


LL-поворот

q = p-->Left
p-->Bal = 0
q-->Bal = 0
p-->Left = q-->Right
q-->Right = p
p = q

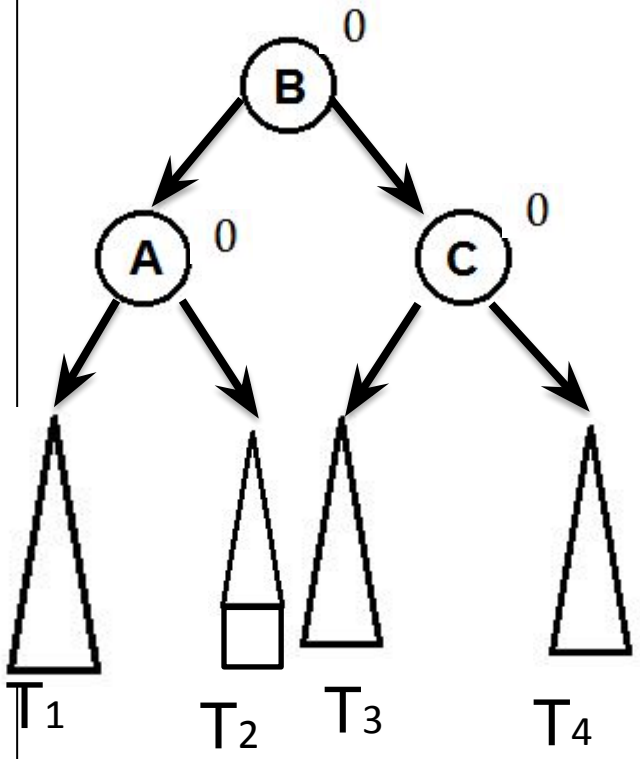


LR – поворот //RL – поворот симметричен



LR-поворот

```
q = p-->Left
r = q-->Right
IF (r-->Bal < 0)
p-->Bal = 1
ELSE  p-->Bal = 0
 FI
IF (r-->Bal > 0)
q-->Bal = -1
ELSE q-->Bal = 0
FI
q-->Right = r-->Left
p-->Left = r-->Right
r-->Left = q;
r-->Right = p;
p = r;
```



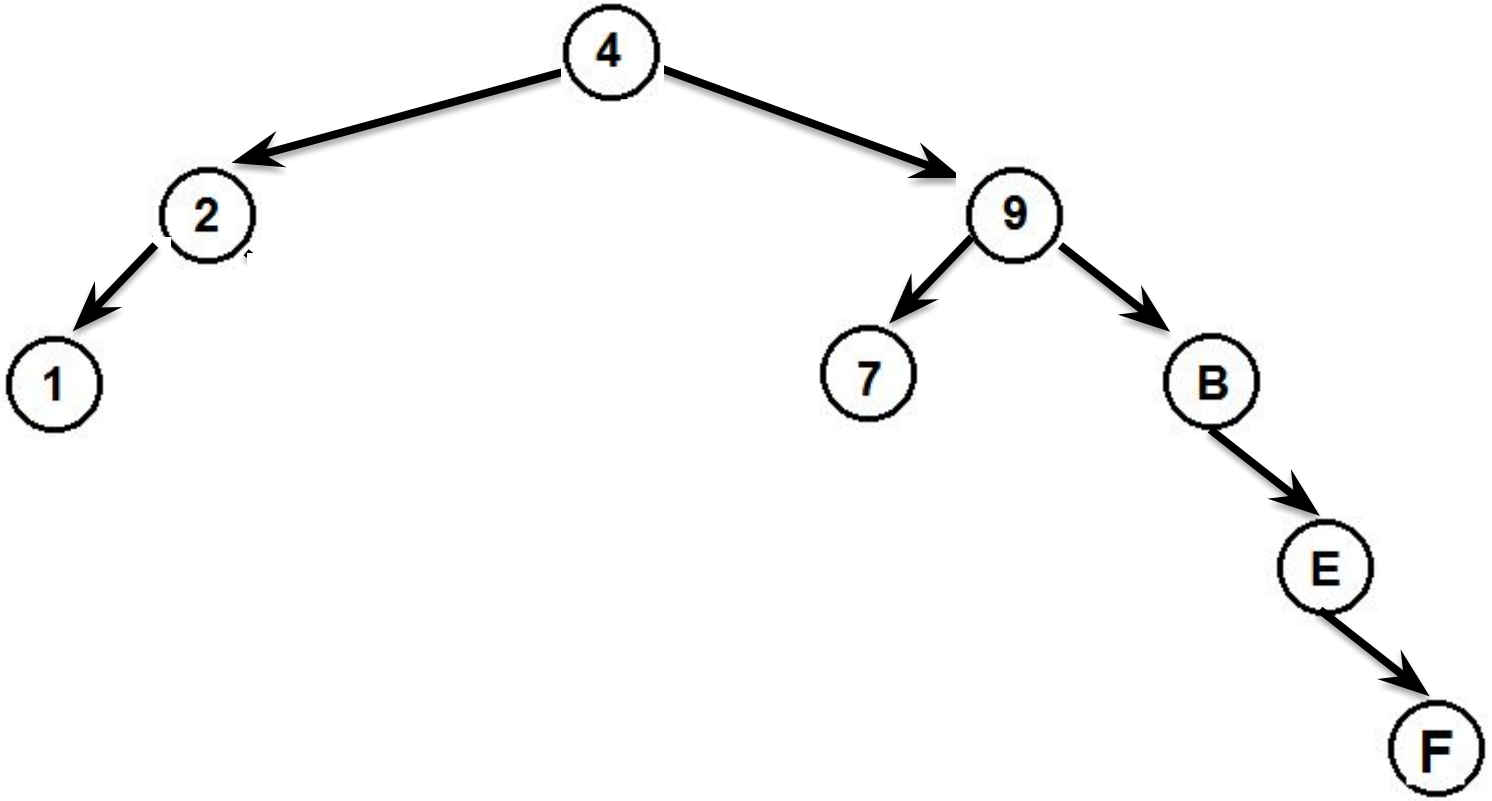
Введем логическую переменную Rost которая будет показывать, что дерево увеличилось (Rost =да)

Добавить АВЛ *(D - данные, vertex*&p)*

```
IF (p = NULL) память (p);
        p-->Data = D;    p-->Left = NULL;
        p-->Right = NULL;  p-->Bal = 0;  Rost = да;
ELSE  IF (p-->Data  > D) Добавить АВЛ (D, p-->Left)
            IF (Rost = да)
              IF (p-->Bal > 0) p-->Bal = 0; Rost = нет
              ELSE  IF (p-->Bal = 0)  p-->Bal = 1
                      ELSE  IF (p-->Left-->Bal < 0) <LL-поворот>
Rost=нет
                                              ELSE          <LR-поворот>  Rost=нет
                                  FI
                      FI
              FI
```

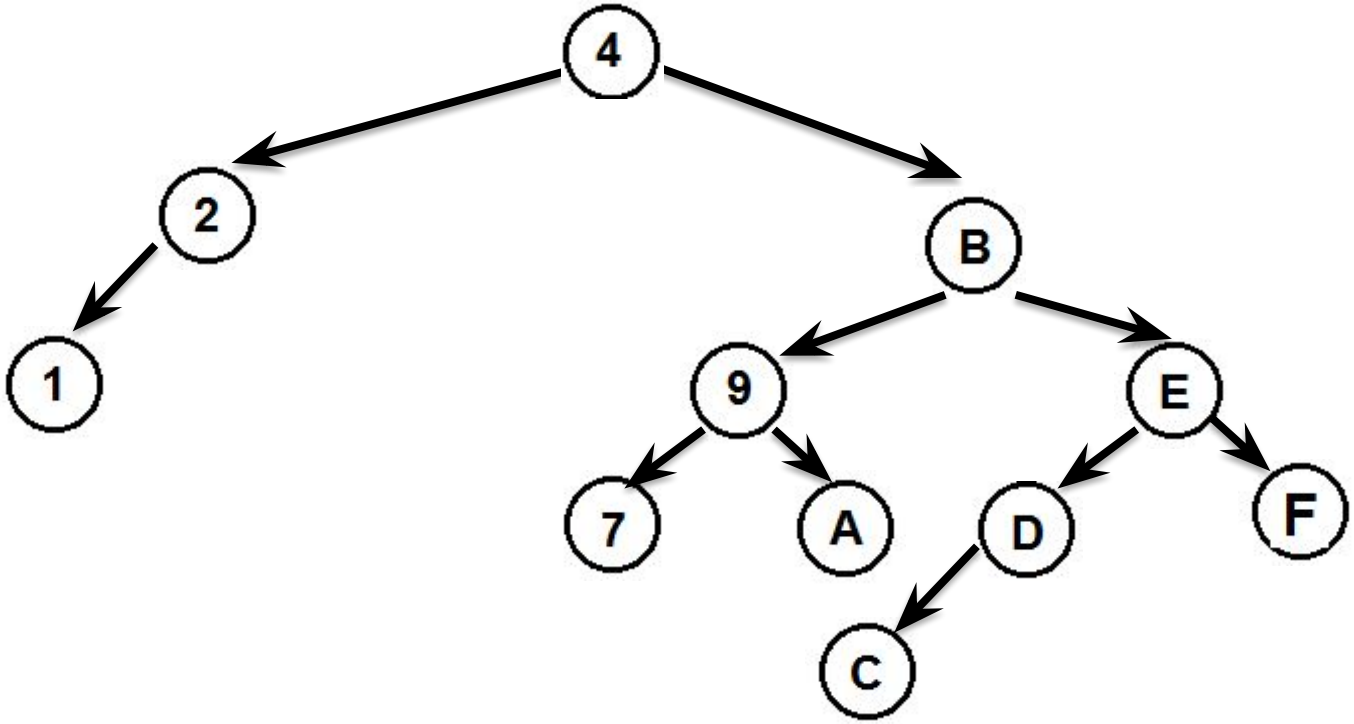
```
ELSE  IF (p-->Data < D) Добавить АВЛ (D, p-->Right)
          IF (Rost = да)
             IF (p-->Bal < 0) p-->Bal = 0;  Rost = нет
             ELSE  IF (p-->Bal = 0) p-->Bal = 1;  Rost = да
                      ELSE IF (p-->Right-->Bal > 0) <RR-поворот>  Rost = нет
                                                ELSE          <RL-поворот>  Rost =
нет
                                   FI
                      FI
             FI
          FI
       ELSE   < Вершина есть в дереве >
       FI
FI
```

B 9 2 4 1 7 E F A D C 3 5 8 6



RR

B 9 2 4 1 7 E F A D C 3 5 8 6



RR

B 9 2 4 1 7 E F A D C 3 5 8 6