АА-деревья

Работу выполнила студентка группы ИМ15-05Б Просяник Юлия
Руководитель Олейников Борис Васильевич

Содержание

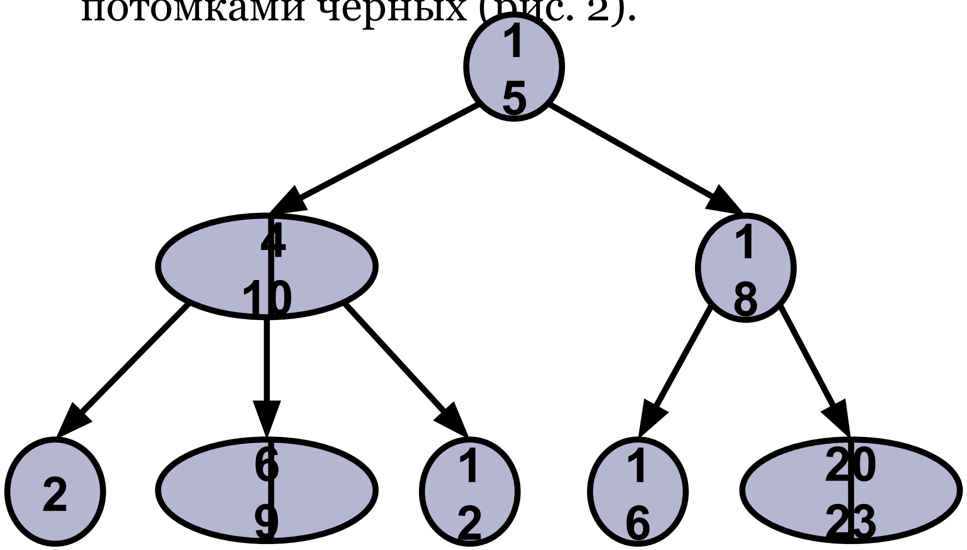
- Определение структуры
- Связь АА-деревьев с другими деревьями поиска
- История структуры
- Свойства АА-деревьев
- Основные операции для работы с АА-деревом и алгоритмы их реализации :
  - Операция “skew”
  - Операция “split”
- Вставка в АА-дерево
- Удаление из АА-дерева
- Список используемой литературы и источников

Определение структуры

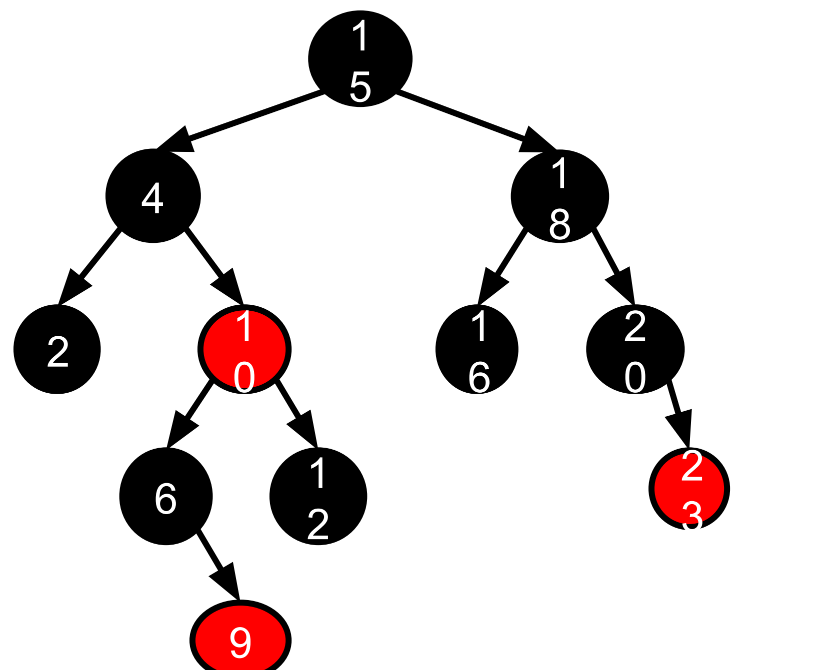
- ***АА-дерево*** – это форма сбалансированного дерева, которое используется для хранения и эффективного извлечения упорядоченных данных. АА-деревья названы по имени их изобретателя *Арне Андерссона (Arne Andersson).*
- АА-дерево является разновидностью красно-черного дерева, но в отличие от красно-черных деревьев, красные узлы на АА-дереве могут быть добавлены только как правый потомок, благодаря этому, АА-деревья обладают повышенной простотой кодирования.

Связь АА-деревьев с красно-черными и 2-3 деревьями

- Физически, красно-черное дерево похоже на 2-3 дерево (обычное бинарное дерево поиска, где некоторые узлы имеют две ссылки, а некоторые узлы группируются в пары и пара содержит три ссылки) (рис. 1) с дополнительными ограничениями. Если представлять узел с двумя ключами в виде двух отдельных узлов, и красить все одиночные узлы и «левые половины» двойных узлов в черный цвет, а «правые половины» в красный, то мы получим обычное красно-черное дерево, в котором все красные вершины являются правыми потомками черных (рис. 2).



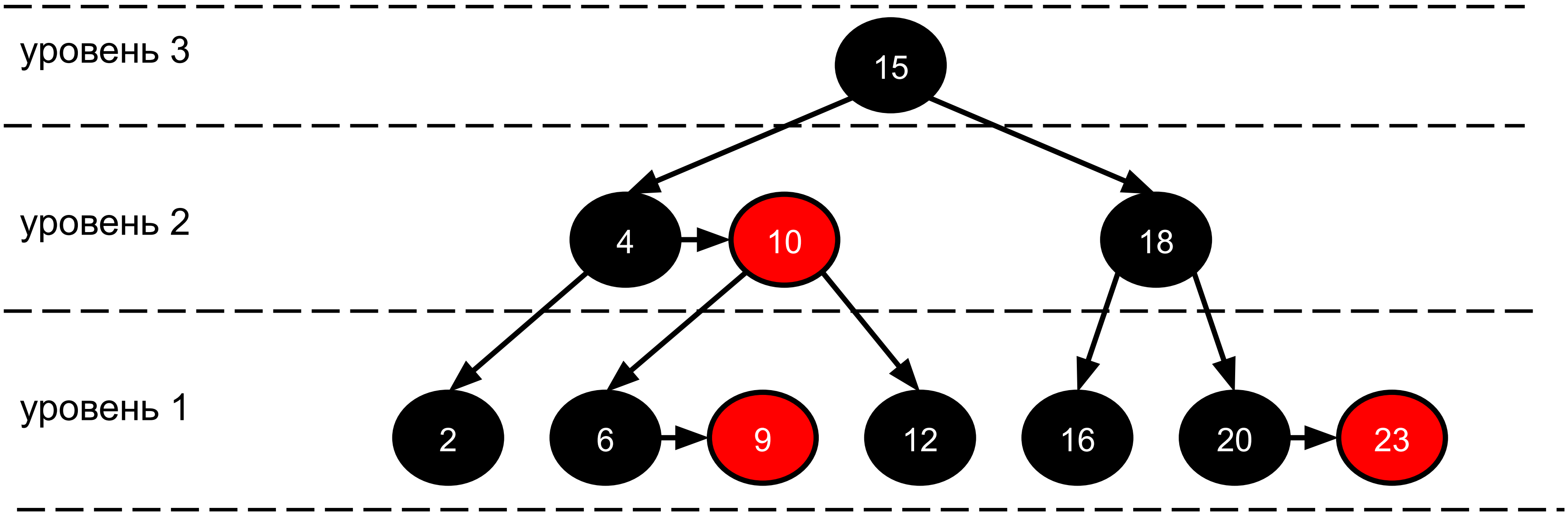
- **Рис. 1.** 2-3 дерево



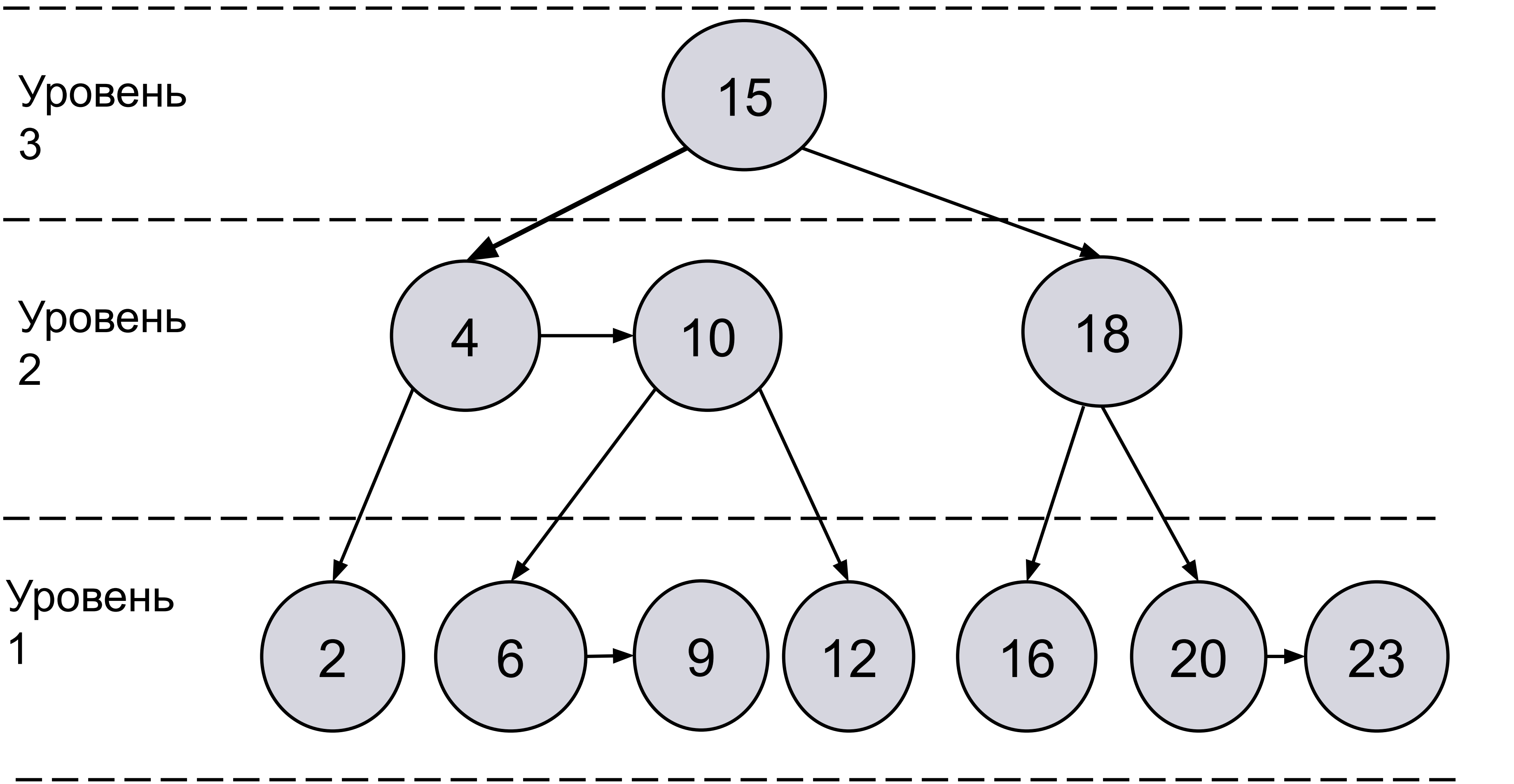
- **Рис. 2**. Красно-черное дерево

Связь АА-деревьев с красно-черными и 2-3 деревьями

- Возвращаясь к АА-деревьям, вместо того, чтобы раскрашивать узлы в красный и черный цвета, введем понятие *уровень узла (level)*. Будем считать, что все листья дерева имеют уровень 1 (единица), а уровень родителя имеет уровень на единицу больший, чем уровень потомка. Красные узлы находятся на уровне своих родителей. То есть, в качестве исключения будем считать, что если потомок является правым потомком, то его уровень может быть равен уровню родительского узла (рис. 3).



Пример АА-дерева



История структуры

- АА-дерево было придумано **Арне Андерссоном в 1993 году**, который первым решил, что для упрощения балансировки дерева нужно ввести понятие **уровень (level) вершины**. Если представить себе дерево растущим сверху вниз от корня (то есть «стоящим на листьях»), то уровень любой листовой вершины будет равен 1. В своей работе Арне Андерссон приводит простое правило, которому должно удовлетворять АА-дерево:

К одной вершине можно присоединить другую вершину того же уровня, но только одну и только справа (одна правая одноуровневая связь).

Свойства АА-дерева

1. Уровень листа равен 1.
2. Уровень левого потомка строго меньше уровня узла.
3. Уровень правого потомка не больше уровня узла.
4. Уровень потомков правого потомка строго меньше уровня узла.
5. Каждый узел с уровнем больше 1 имеет двух потомков.

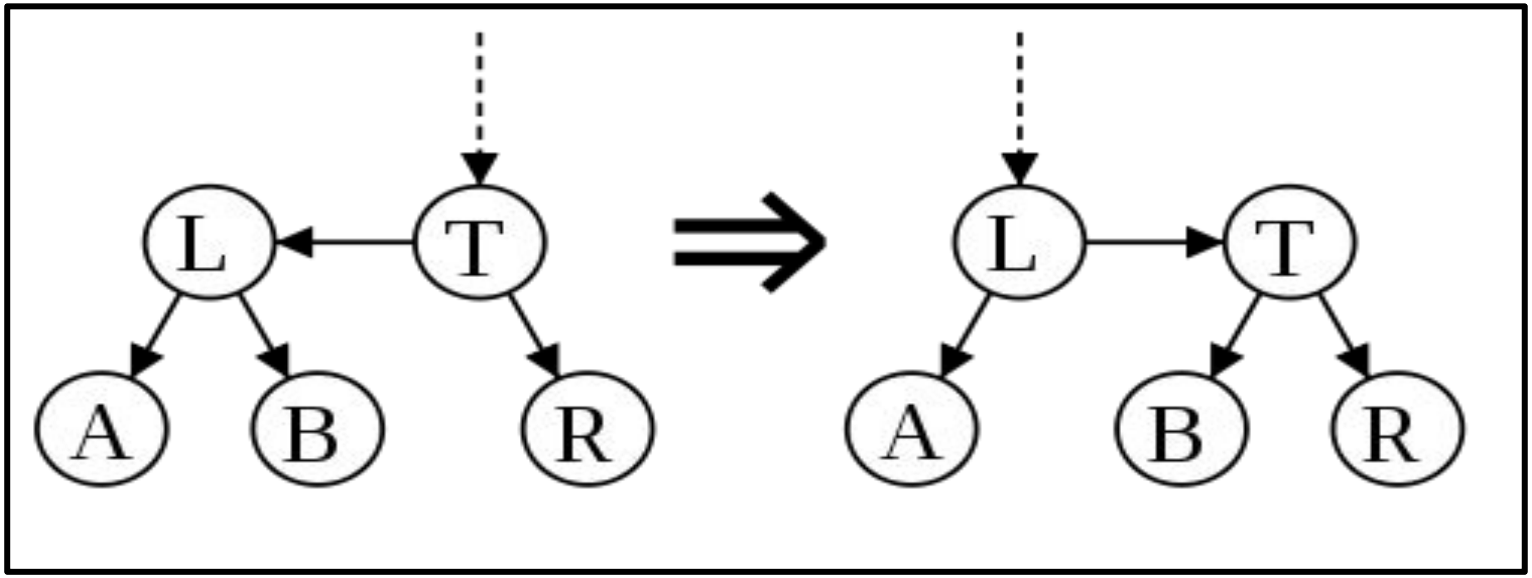
Описание структуры АА-дерева :

```
type
  pl_tree = ^el_tree;
  el_tree = record
   key : integer;
   level : byte;         //уровень вершины (у листьев 1)
   left, right : pl_tree;
  end;
```

ОСНОВНЫЕ ОПЕРАЦИИ ДЛЯ РАБОТЫ С АА-ДЕРЕВОМ И АЛГОРИТМЫ ИХ РЕАЛИЗАЦИИ

- Для балансировки АА-дерева нужно всего 2 (две) различных операции. Нетрудно их понять из правила «одна правая одноуровневая связь» : *это устранение левой связи на одном уровне и устранение двух правых связей на одном уровне.*
- Эти операции в оригинальной работе Арне Андерссона названы *“skew”(«перекос») и “split”(«расщепление»)* соответственно.

“skew” устранение левой связи на одном уровне

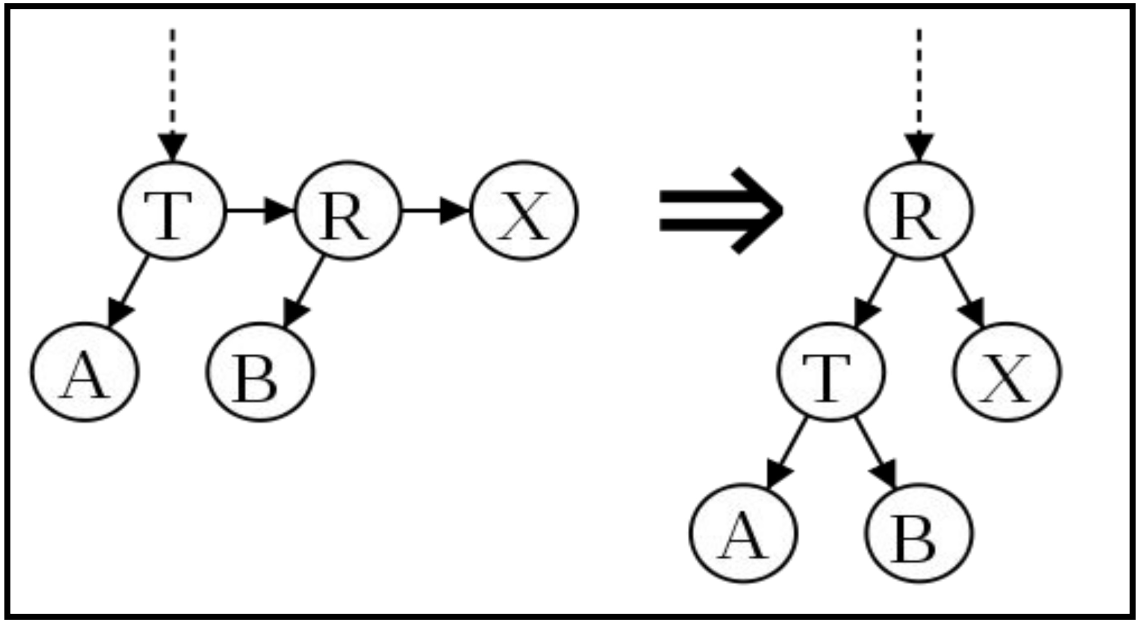


- Данная операция устраняет горизонтальную левую связь при помощи вращения узла вправо каждый раз, когда горизонтальная левая связь найдена.
- На рисунке горизонтальная стрелка обозначает связь между вершинами одного уровня, а наклонная (вертикальная) — между вершинами разного уровня

Код процедуры “skew”

```
procedure Skew_t (var t : pl_tree);
var
   tmp : pl_tree;
begin
   if t <> nil then
   begin
      if  t^.left <> nil then
      begin
         if t^.left^.level = t^.level then
         begin   {rotate right}
               tmp := t;
               t := tmp^.left;
               tmp^.left := t^.right;
               t^.right := tmp;
         end;
      end;
   end;
end;
```

“split” устранение двух правых связей на одном уровне

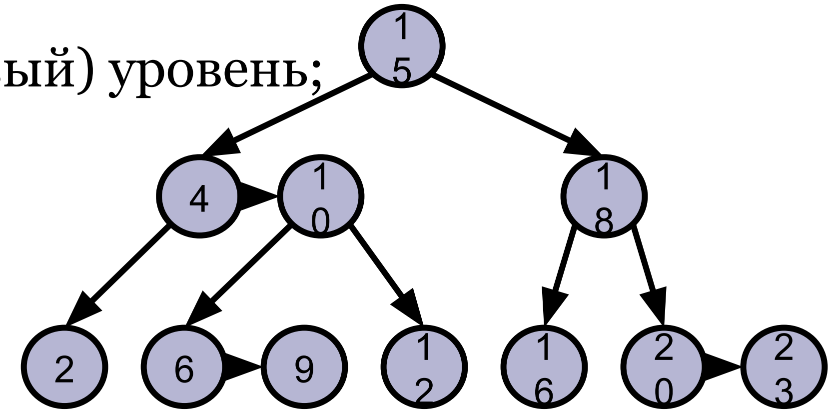


- Данная операция устраняет две последовательные правые горизонтальные связи при помощи вращения узла влево и увеличения уровня среднего узла на единицу

Код процедуры “split”

```
procedure Split_t (var t : pl_tree);
var
    tmp : pl_tree;
begin
    if t <> nil then
    begin
        if t^.right <> nil then
        begin
            if t^.right^.right <> nil then
            begin
                if t^.right^.right^.level = t^.level then
                begin   {rotate left}
                    tmp :=t;
                    t := tmp^.right;
                    tmp^.right := t^.left;
                    t^.left := tmp;
                    Inc (t^.level);
                end;
            end;
        end;
    end;
end;
```

Алгоритм вставки



1. Добавляем новый узел на 1(первый) уровень;
2. Вызываем операцию “skew”;
3. Вызываем операцию “split”.

Вставка узла может:

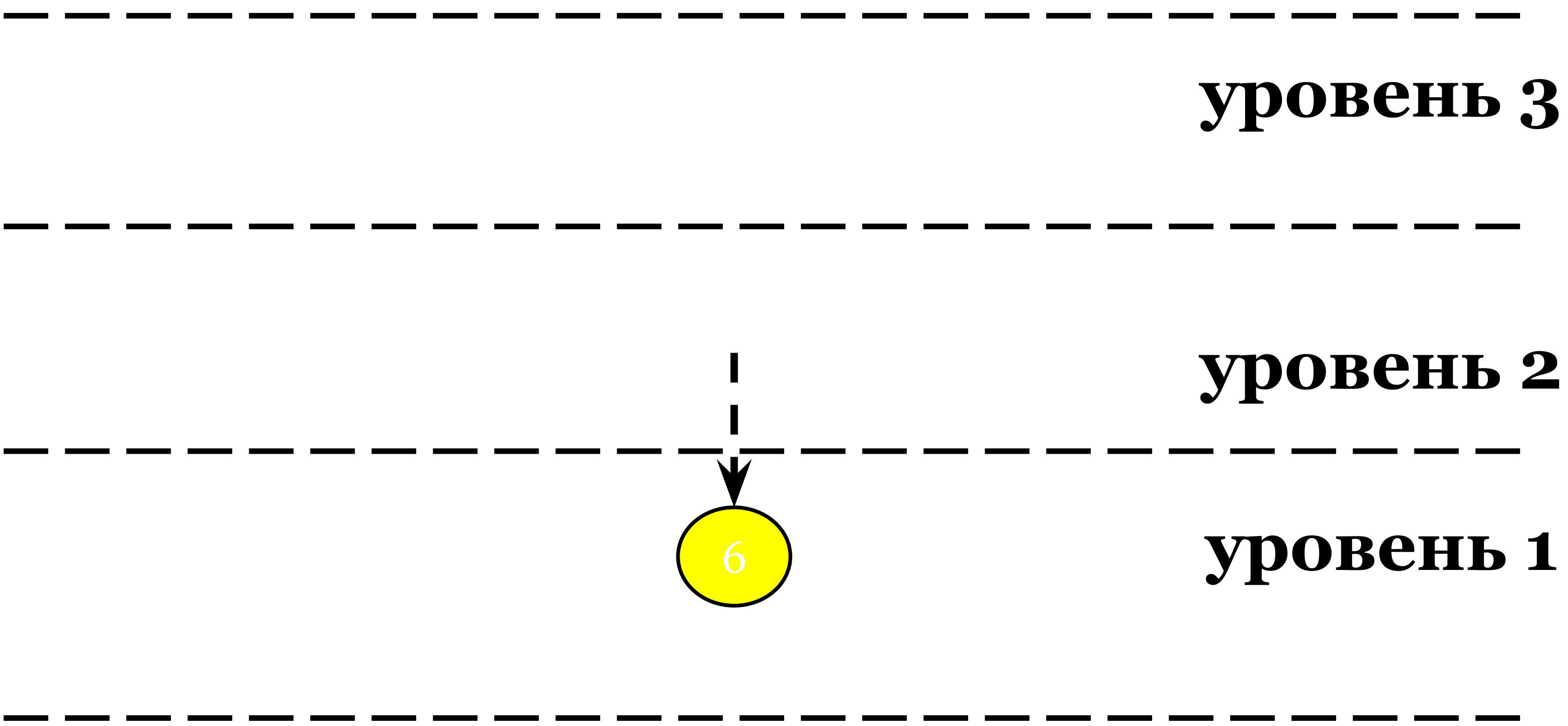
a) Не нарушить правило построения АА-дерева, например 17 ;
b) Нарушить правило «левого потомка», 1 например Исправление может быть сделано простым поворотом “skew”;
c) Нарушить правило «двух правых потомков» , например 25 . Исправление может быть сделано расщеплением “split”;
d) Вставка узла может повлечь за собой серию поворотов и расщеплений, например 5 .

Вставка элемента в дерево

```
procedure Insert_Node (var root_f : pl_tree; x : integer);
begin
   if root_f = nil then
   begin
    new (root_f);
    root_f^.left := nil;
    root_f^.right := nil;
    root_f^.key := x;
    root_f^.level := 1;
   end else
   begin
    if root_f^.key > x then
          Insert_Node (root_f^.left,x)
    else if root_f^.key < x then
          Insert_Node (root_f^.right,x);
   end;
   Skew_t (root_f);
   Split_t (root_f);
end;
```

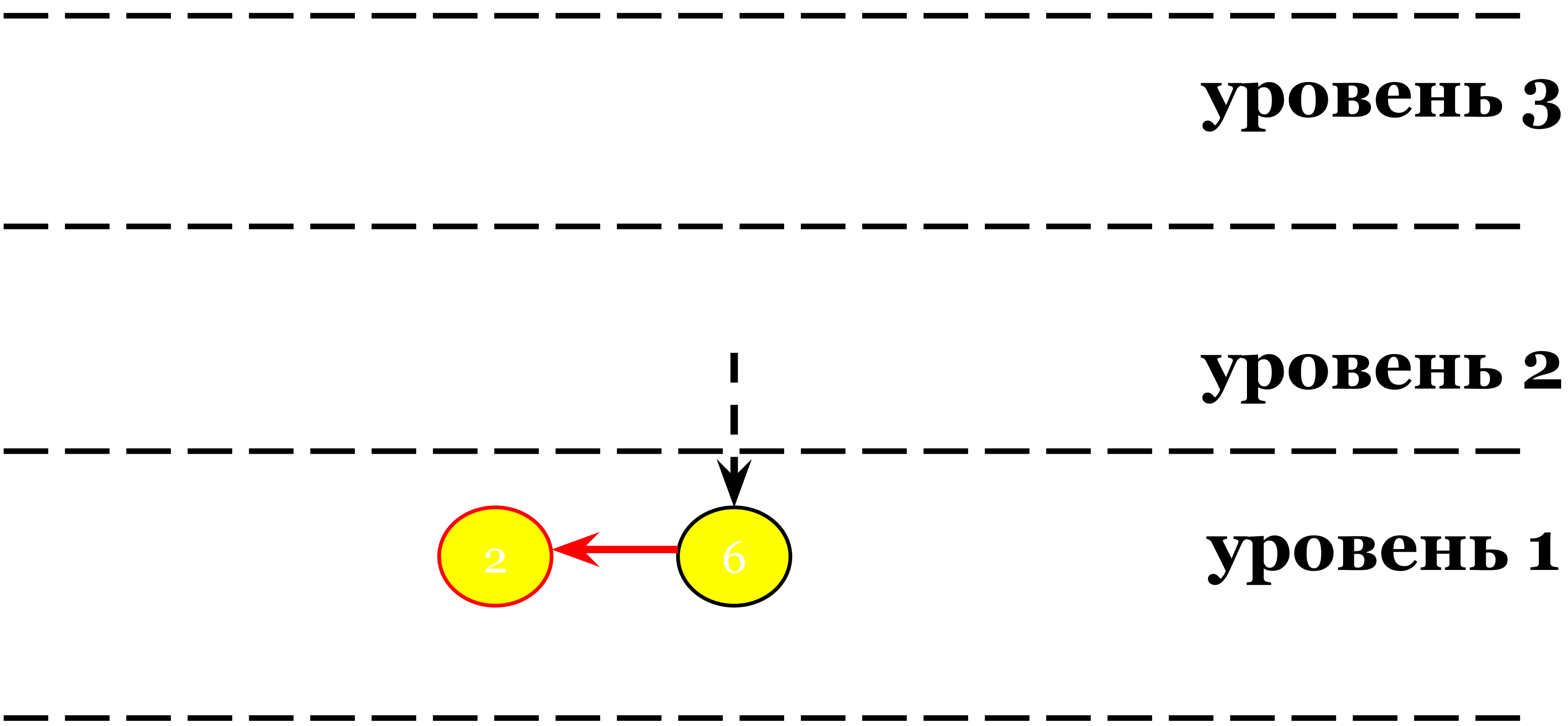
Пример вставки

- Вставляем : **6;**



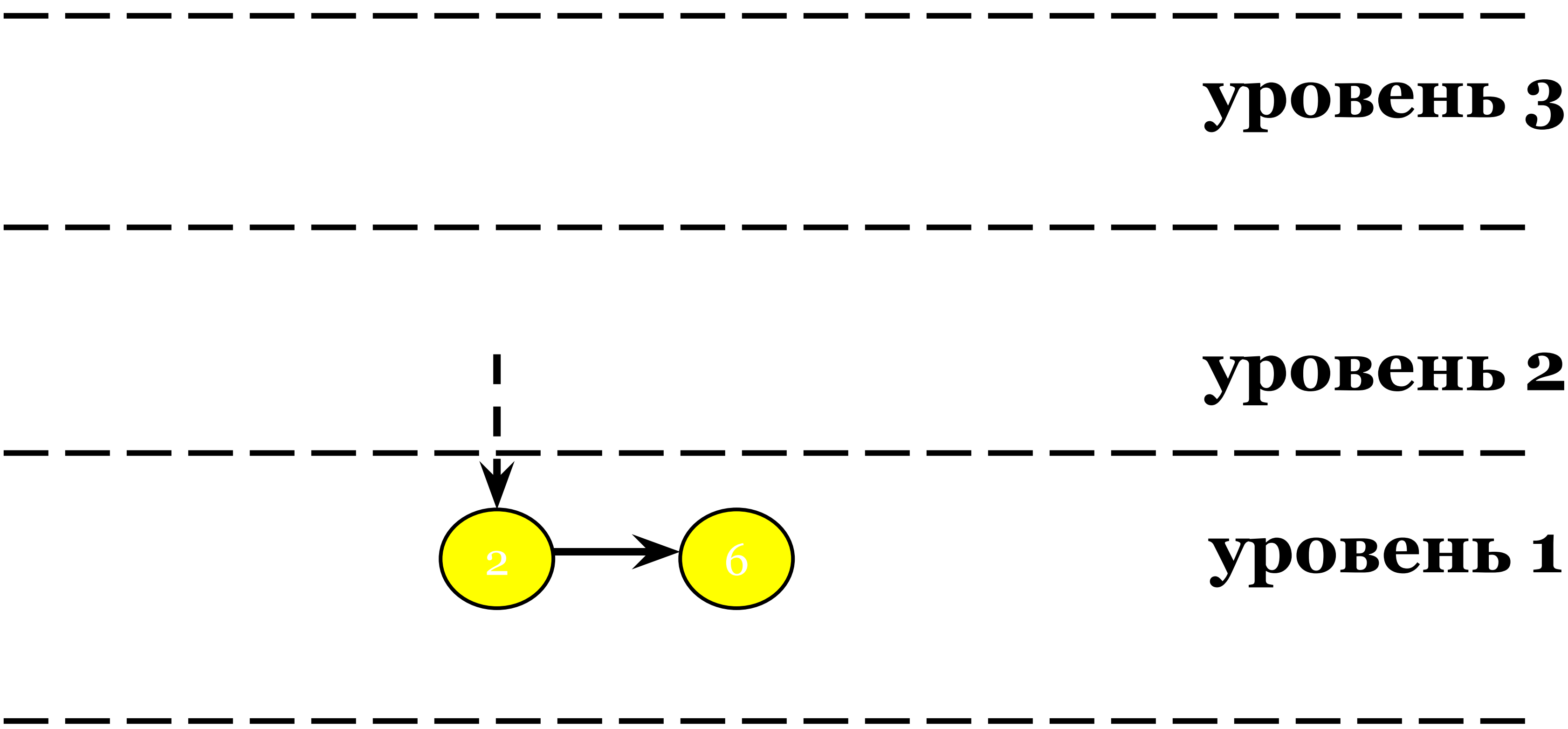
Пример вставки

- Вставляем : **6; 2;**



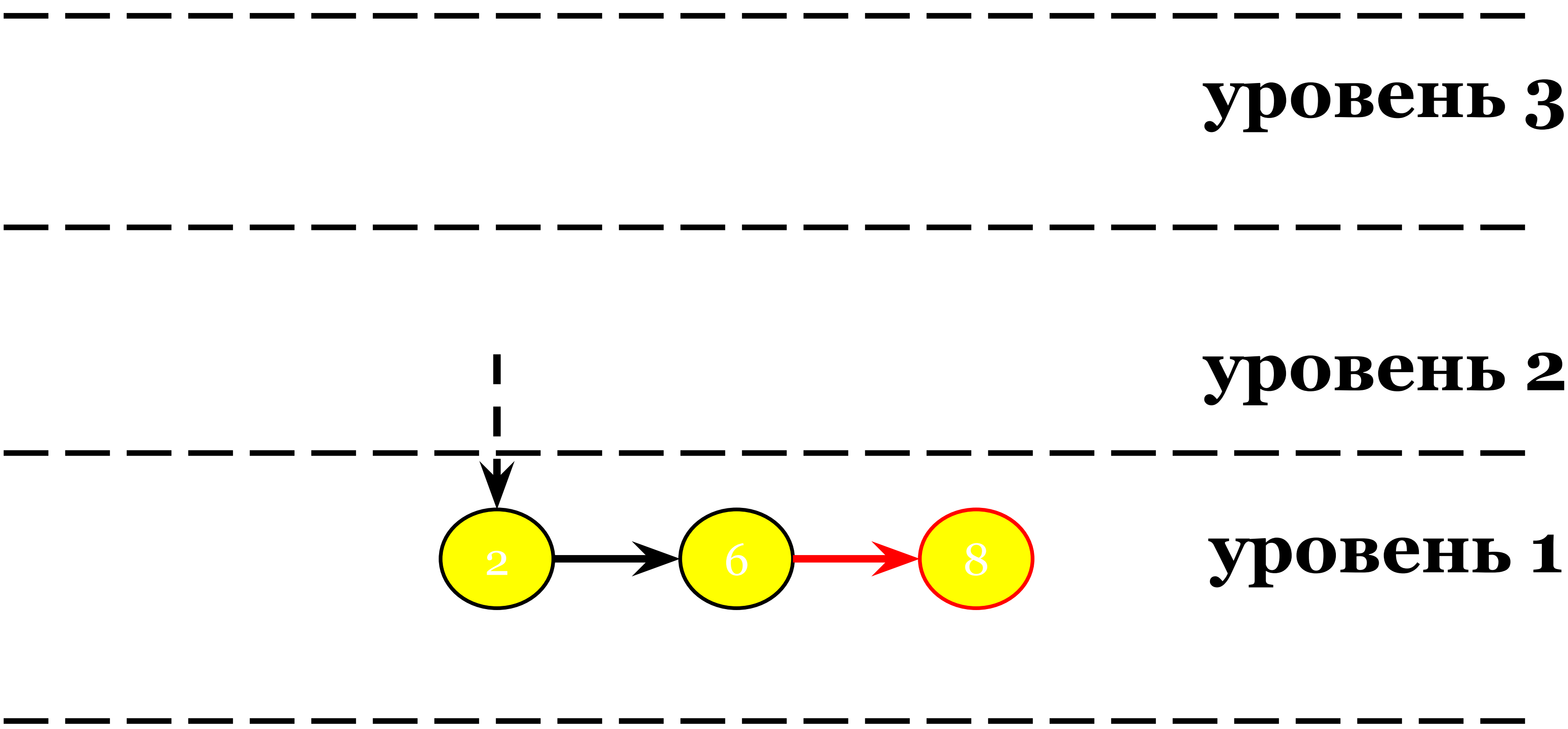
Пример вставки

- Вставляем : **6; 2;**



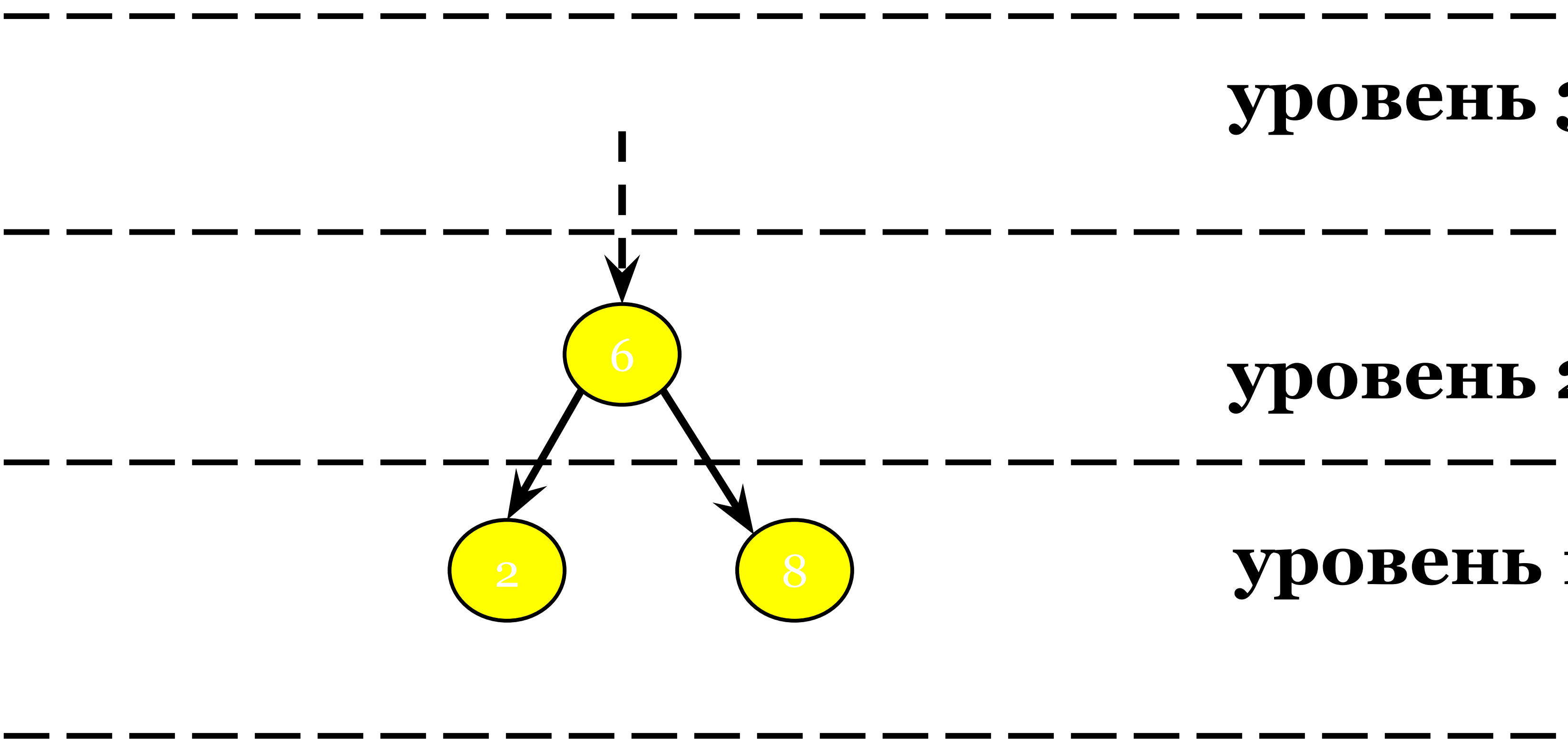
Пример вставки

- Вставляем : **6; 2; 8;**



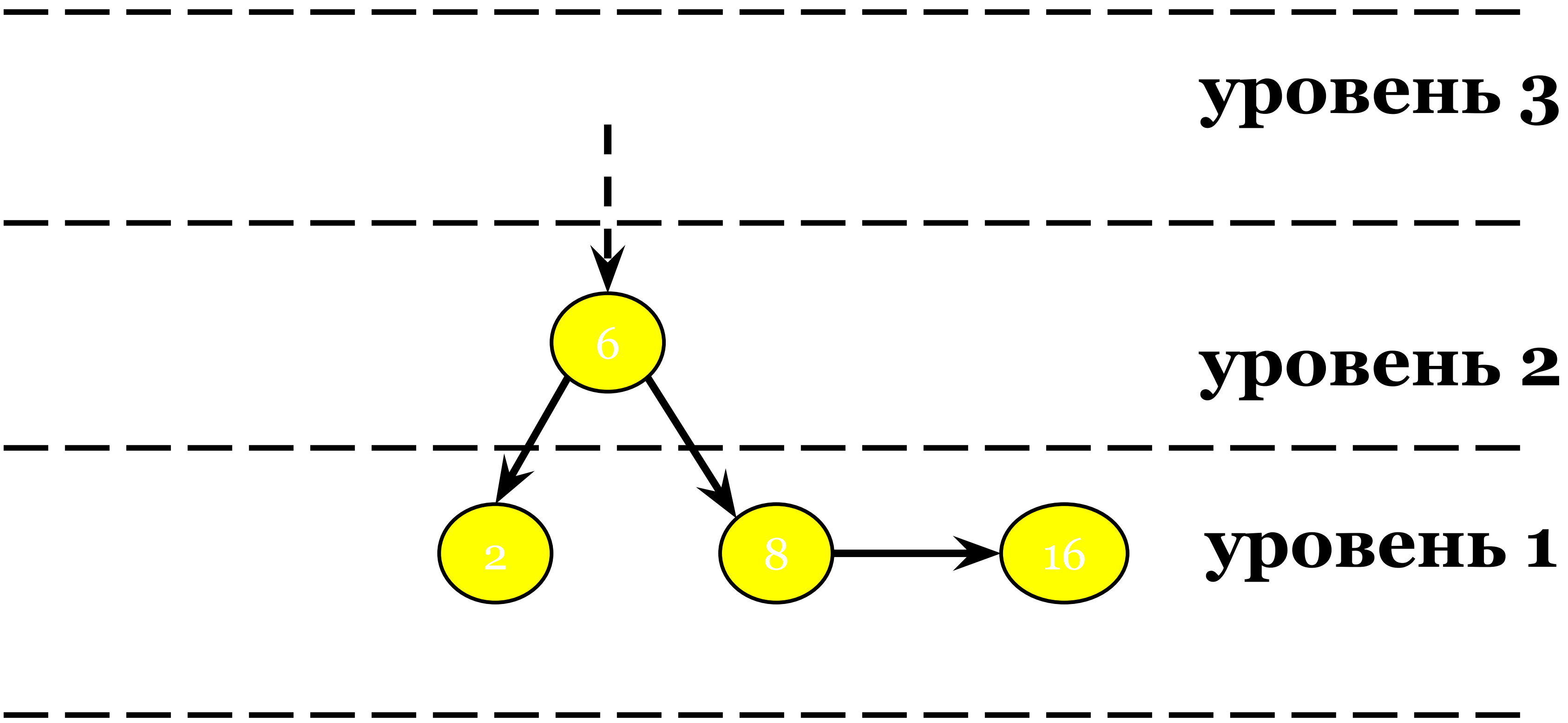
Пример вставки

- Вставляем : **6; 2; 8;**



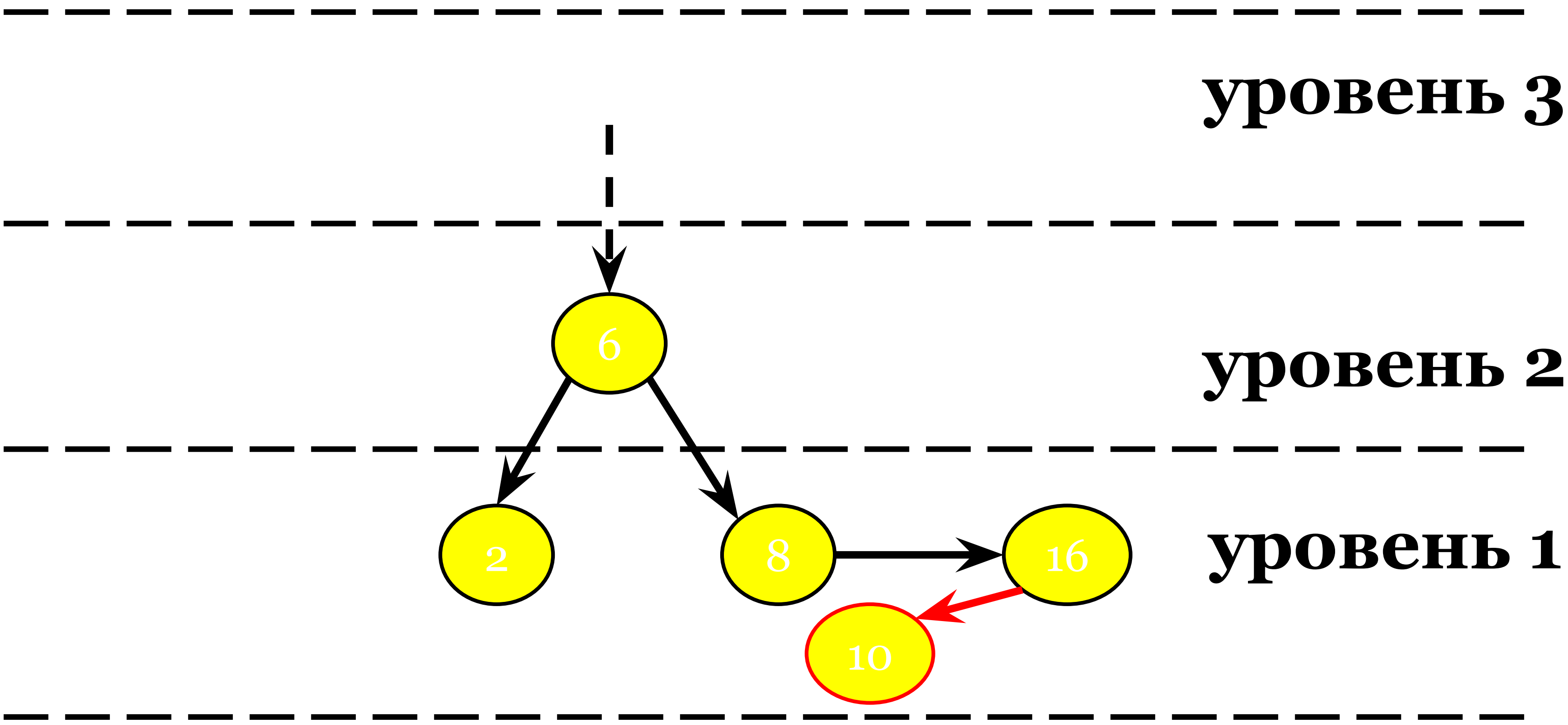
Пример вставки

- Вставляем : **6; 2; 8; 16;**



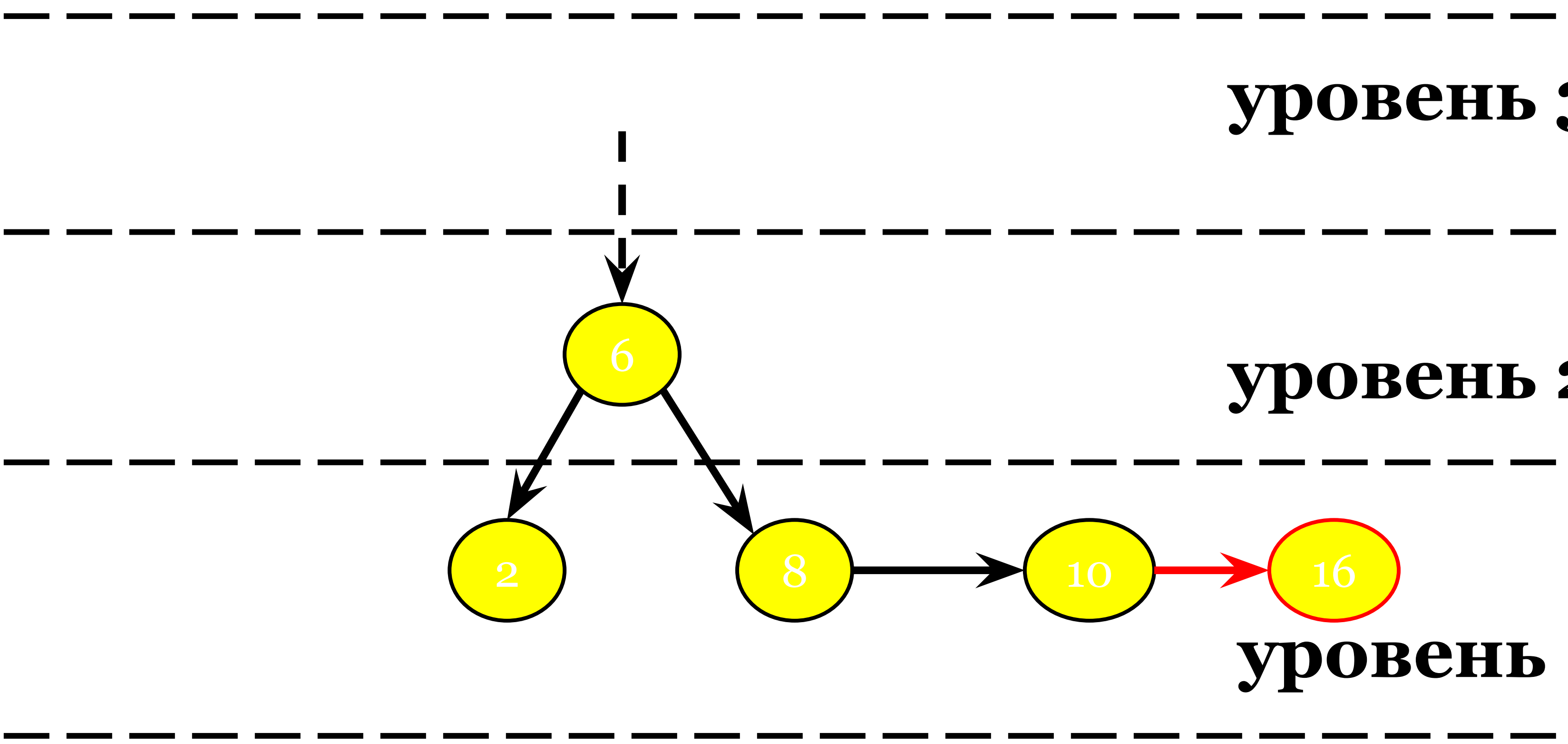
Пример вставки

- Вставляем : **6; 2; 8; 16; 10;**



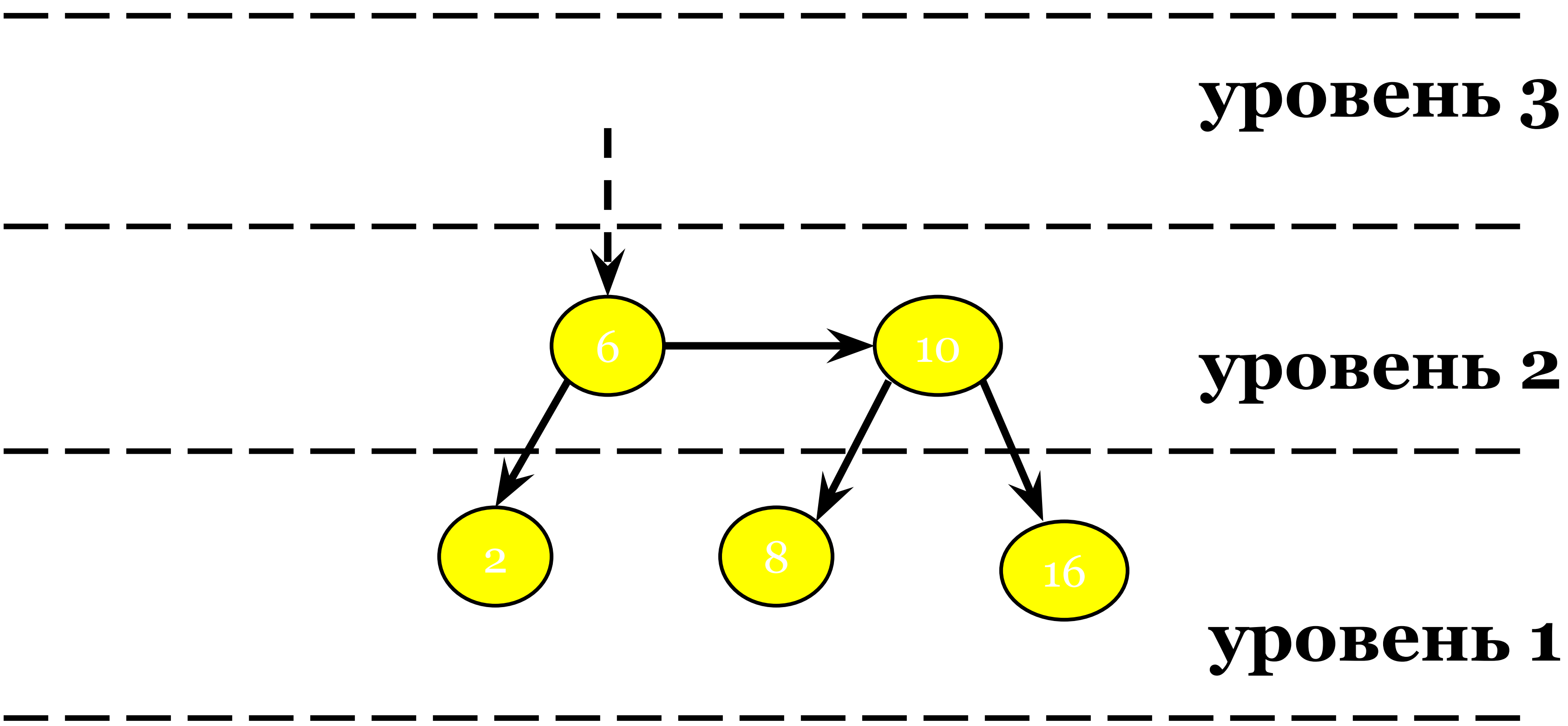
Пример вставки

- Вставляем : **6; 2; 8; 16; 10;**



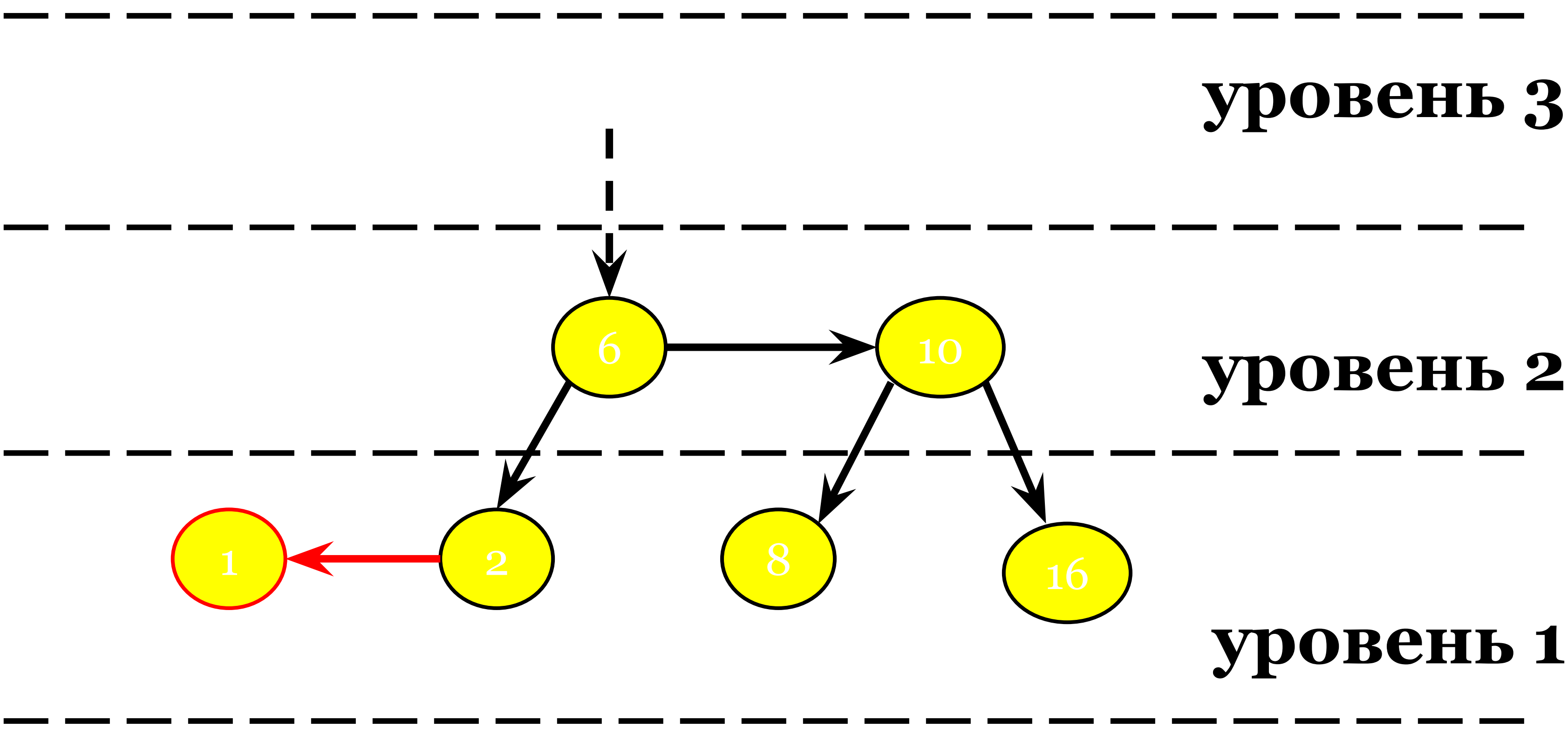
Пример вставки

- Вставляем : **6; 2; 8; 16; 10;**



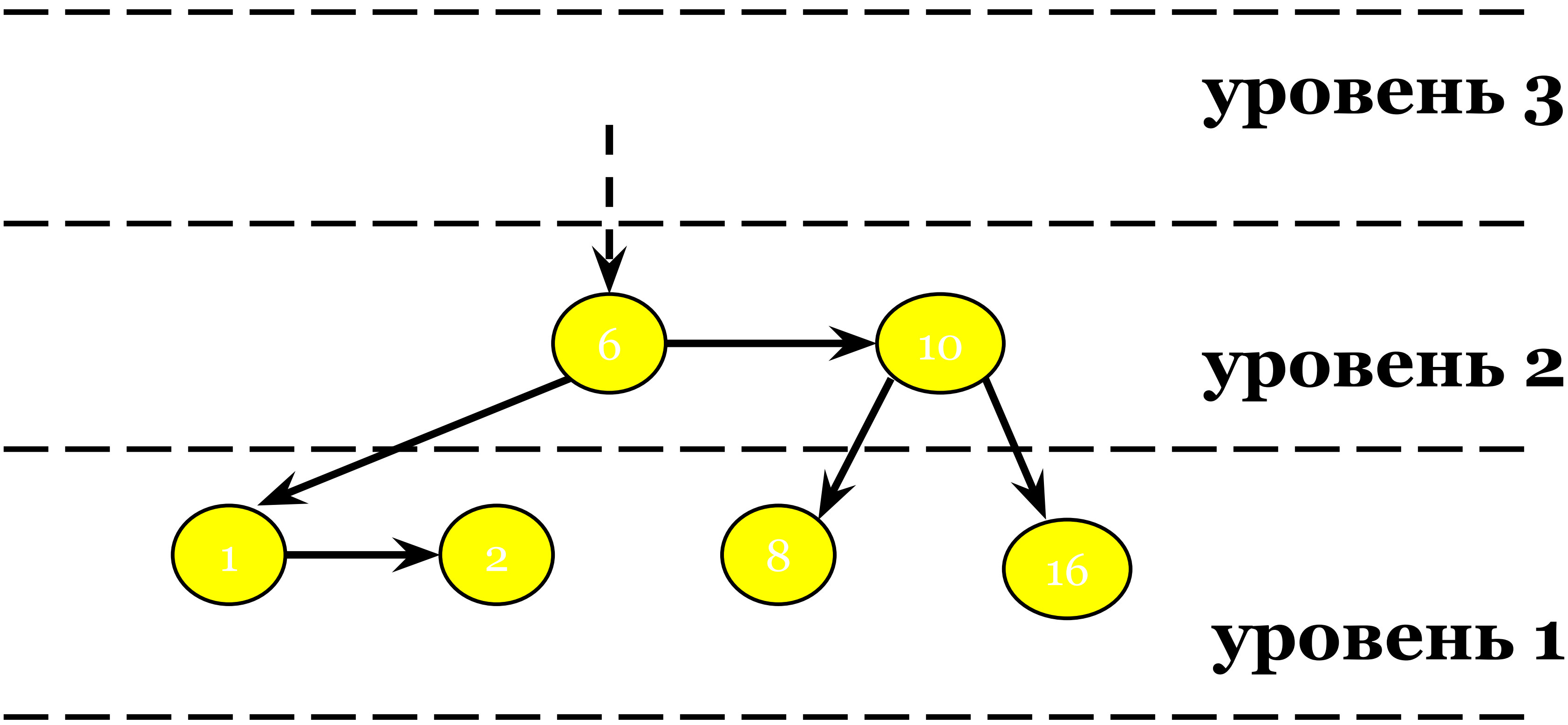
Пример вставки

- Вставляем : **6; 2; 8; 16; 10; 1.**



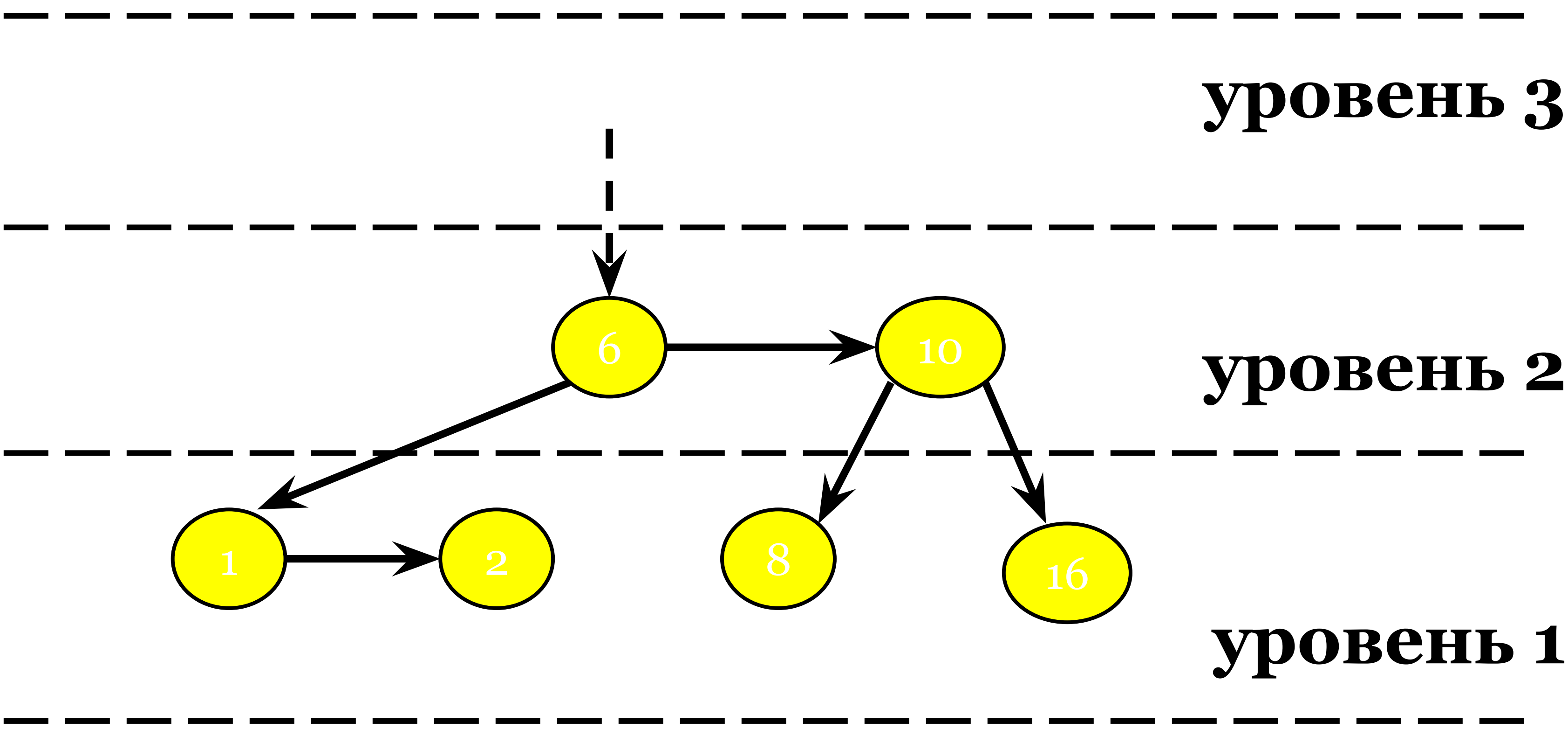
Пример вставки

- Вставляем : **6; 2; 8; 16; 10; 1.**



Пример вставки

- *Дерево полностью сбалансировано!*



Алгоритм удаления

- Удаление элемента также производится по правилам удаления из обычного двоичного дерева поиска с последующей балансировкой.
- Как и в случае вставки элемента, балансировка производится с помощью только двух тех же преобразований – поворота и расщепления.

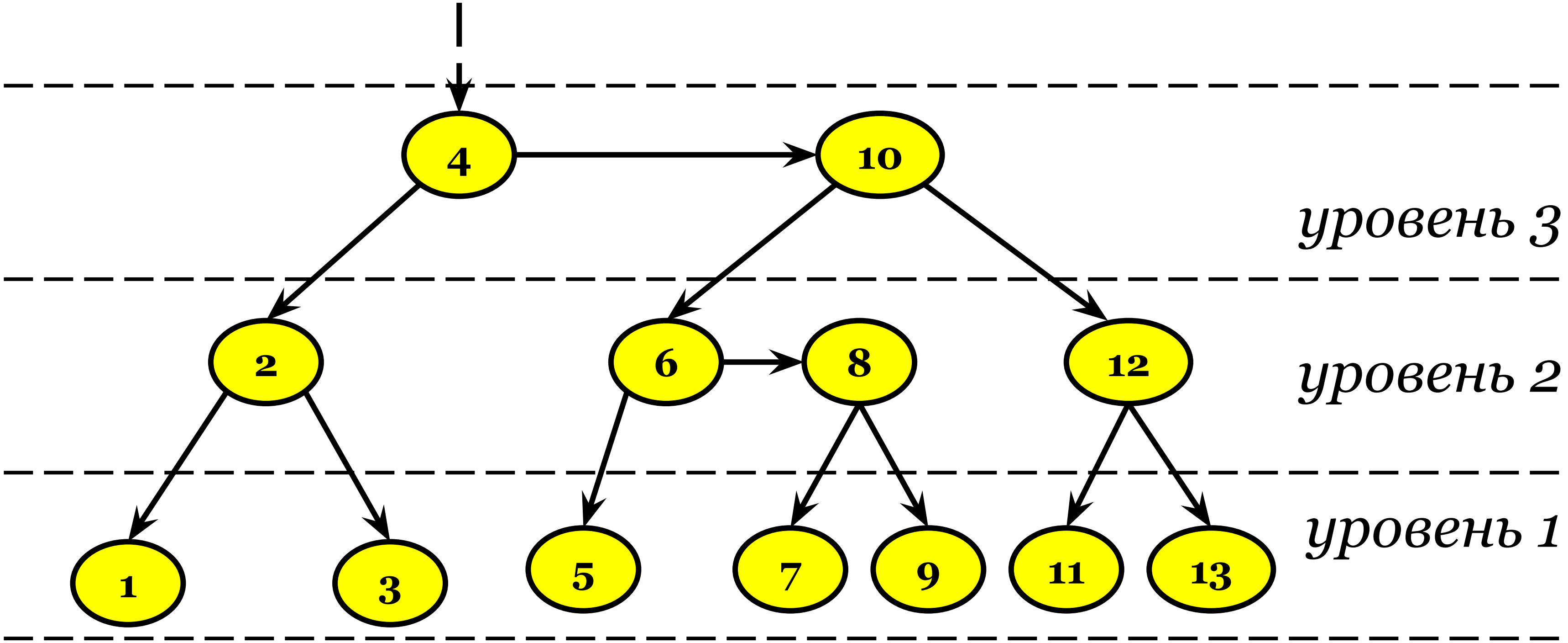
Удаление элемента

```
procedure Delete_Node (var root_f : pl_tree;                    newkey : integer);
begin
     if root_f <> nil then
     begin
     // 1. спускаемся вниз и запоминаем last и deleted
       last := root_f;
       if newkey < root_f^.key then
            Delete_Node (root_f^.left, newkey)
       else
       begin
            deleted := root_f;
            Delete_Node (root_f^.right,  newkey);
       end;

       // 2. удаляем элемент, если найден
       if (root_f = last) and (deleted <> nil) and    (newkey = deleted^.key)
       then
       begin
            deleted^.key := root_f^.key;
            deleted := nil;
            root_f := root_f^.right;
            dispose (last);
       end
       else if (Get_Level(root_f^.left) <        (Get_Level(root_f) - 1)) or
       (Get_Level(root_f^.right) <     (Get_Level(root_f) - 1)) then
       begin
       // 3. выполняем балансировку при движении вверх
              Dec (root_f^.level);
```

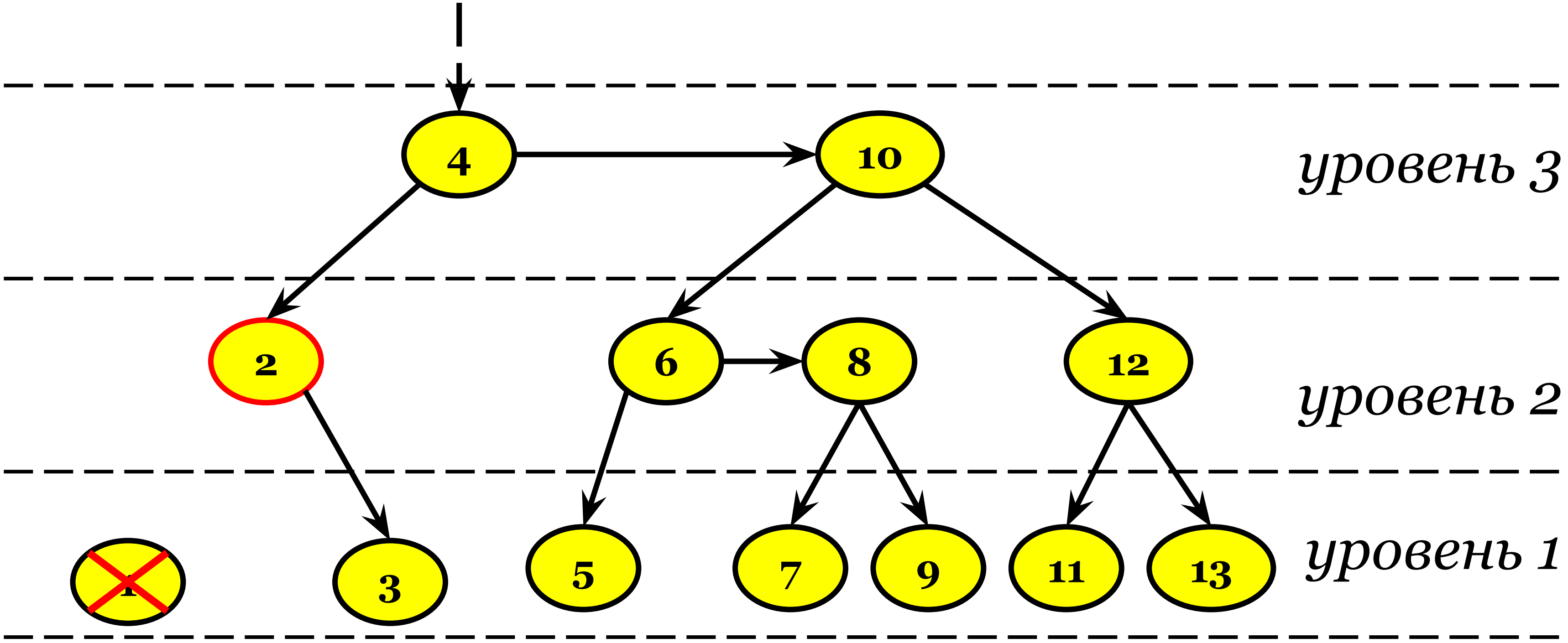
Пример удаления

- Удаляем **узел 1**.



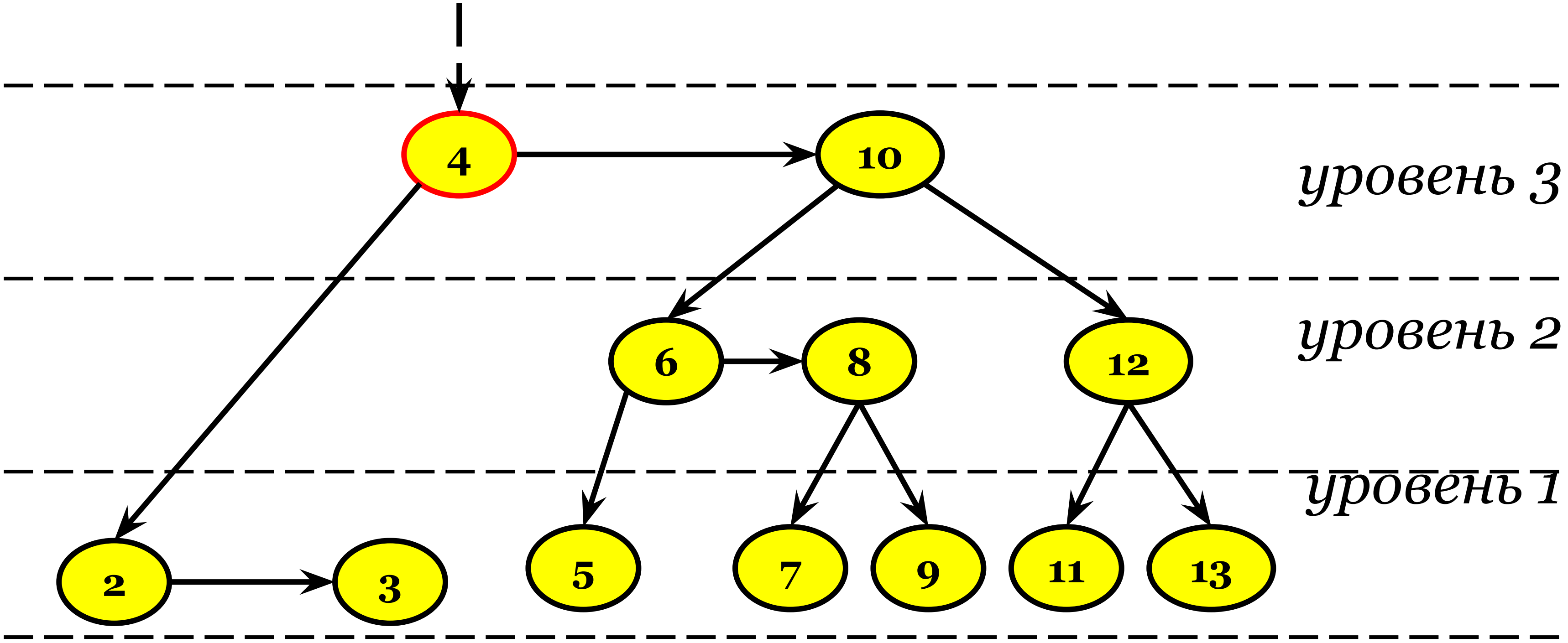
Пример удаления

- Удаляем **узел 1**.
- **Узел 2**, теперь нарушает свойство №5 (Каждый узел с уровнем больше, чем единица имеет двух потомков).



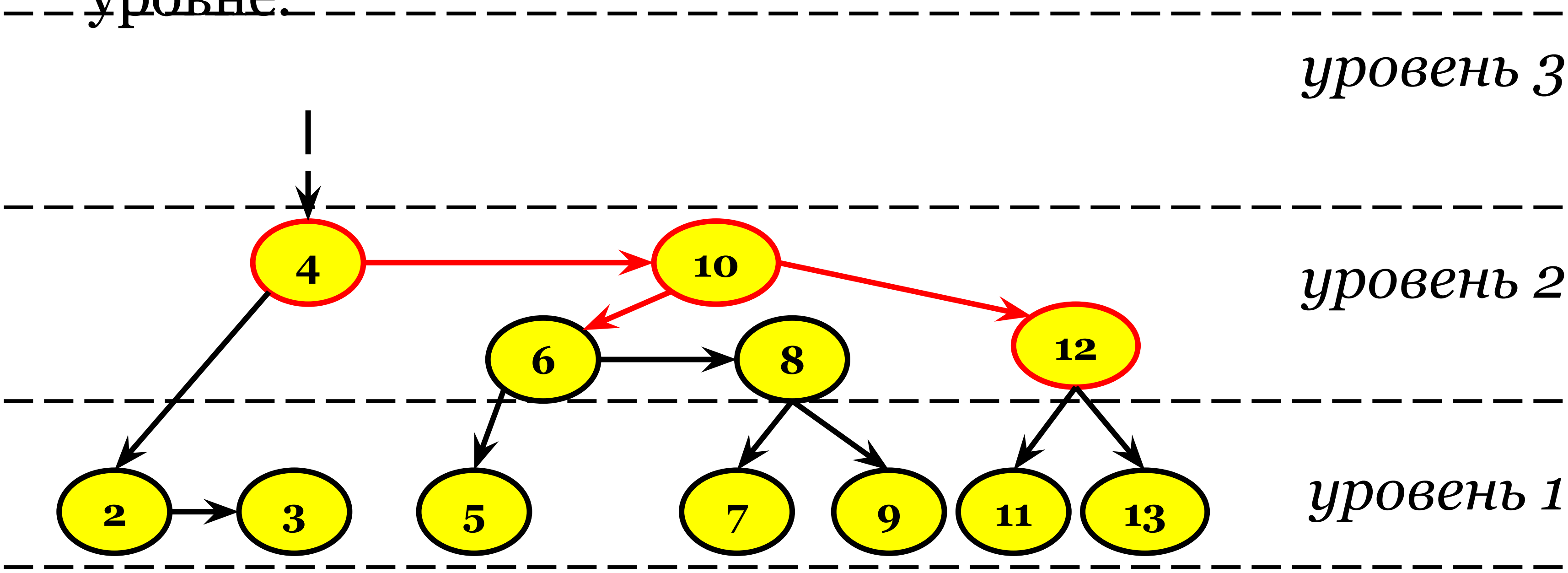
Пример удаления

- Уменьшаем уровень **узла 2**.
- Теперь уровень **узла 4** отличается от уровня его левого потомка (узла 2) больше, чем на единицу.



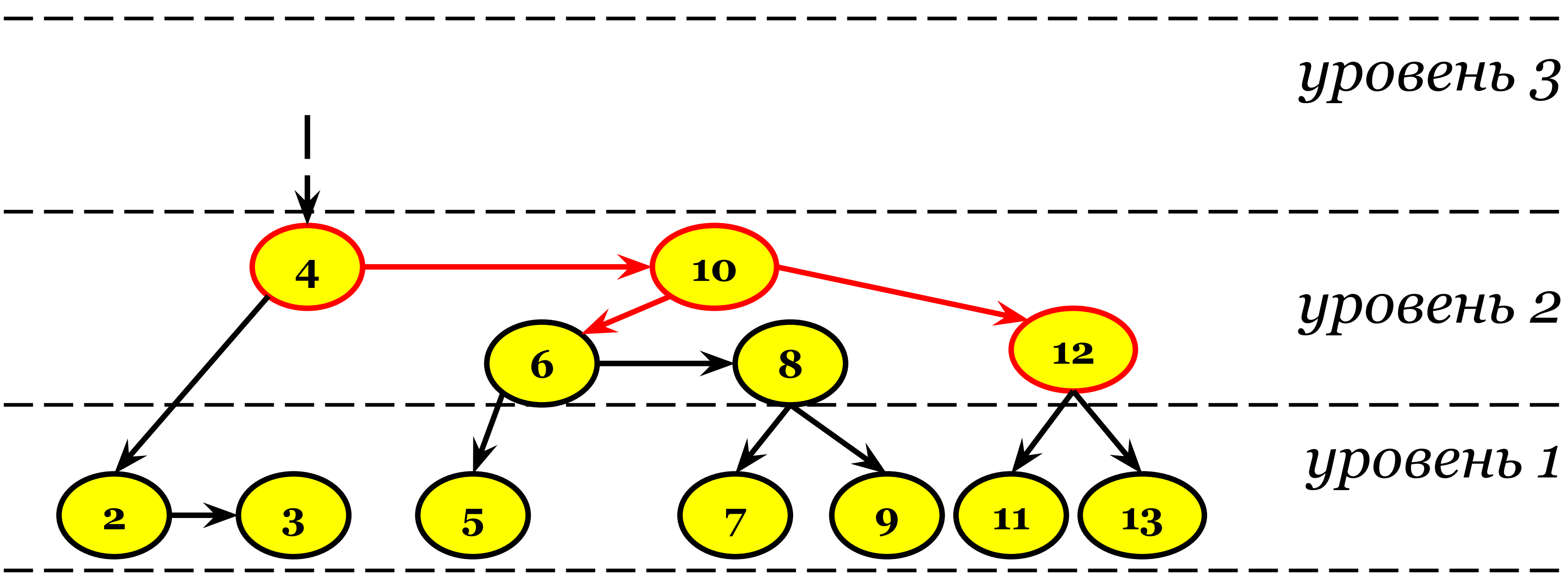
Пример удаления

- Уменьшаем уровень **узлов 4 и 10**.
- У **узла 4** теперь есть две последовательные правые связи.
- У **узла 10** теперь появилась левая связь на одном уровне.



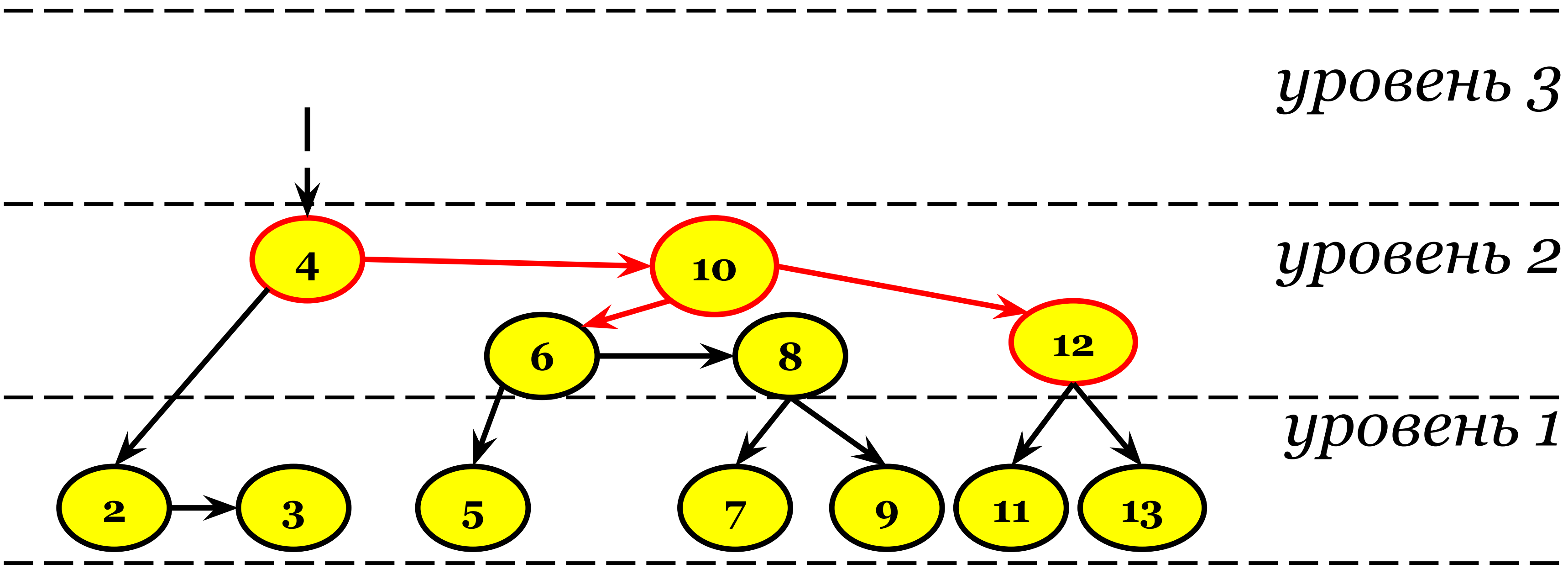
Пример удаления

- Необходимо вызвать *<u>три раза</u>* операцию *<u>“skew”</u>* и *<u>два раза</u>* операцию *<u>“split”</u>*.



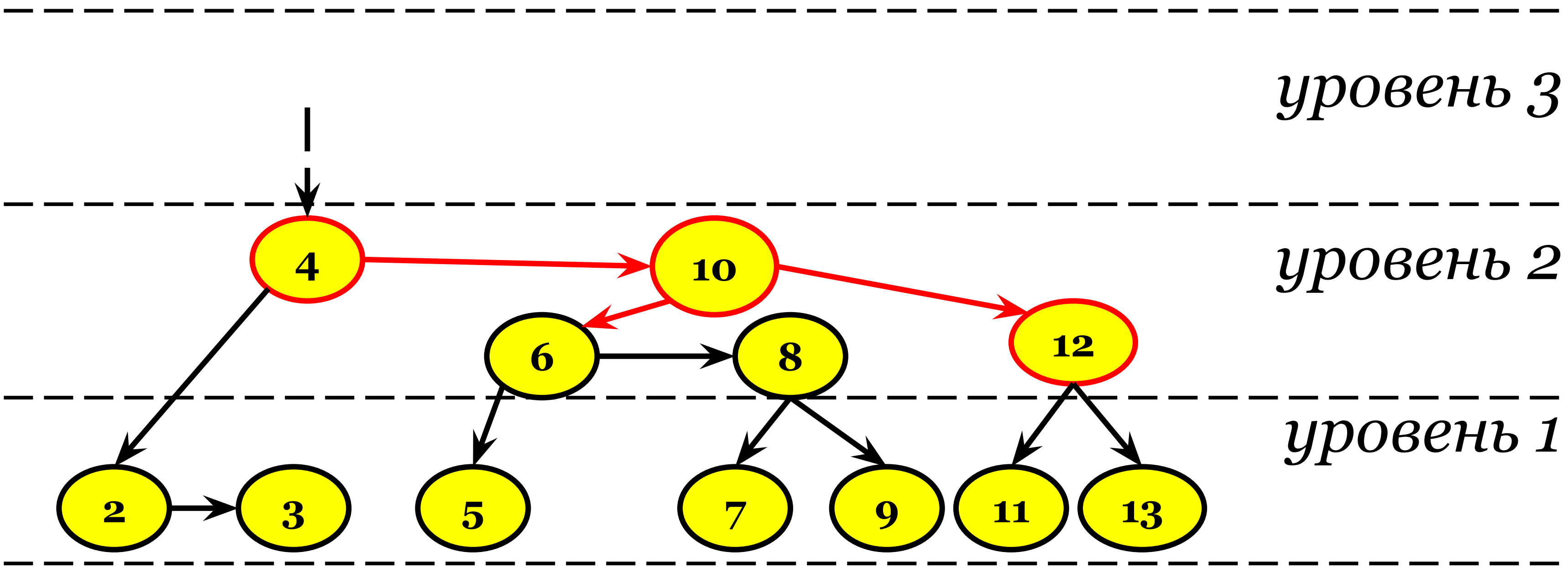
Пример удаления

- Skew (node 4); *//ничего не происходит*



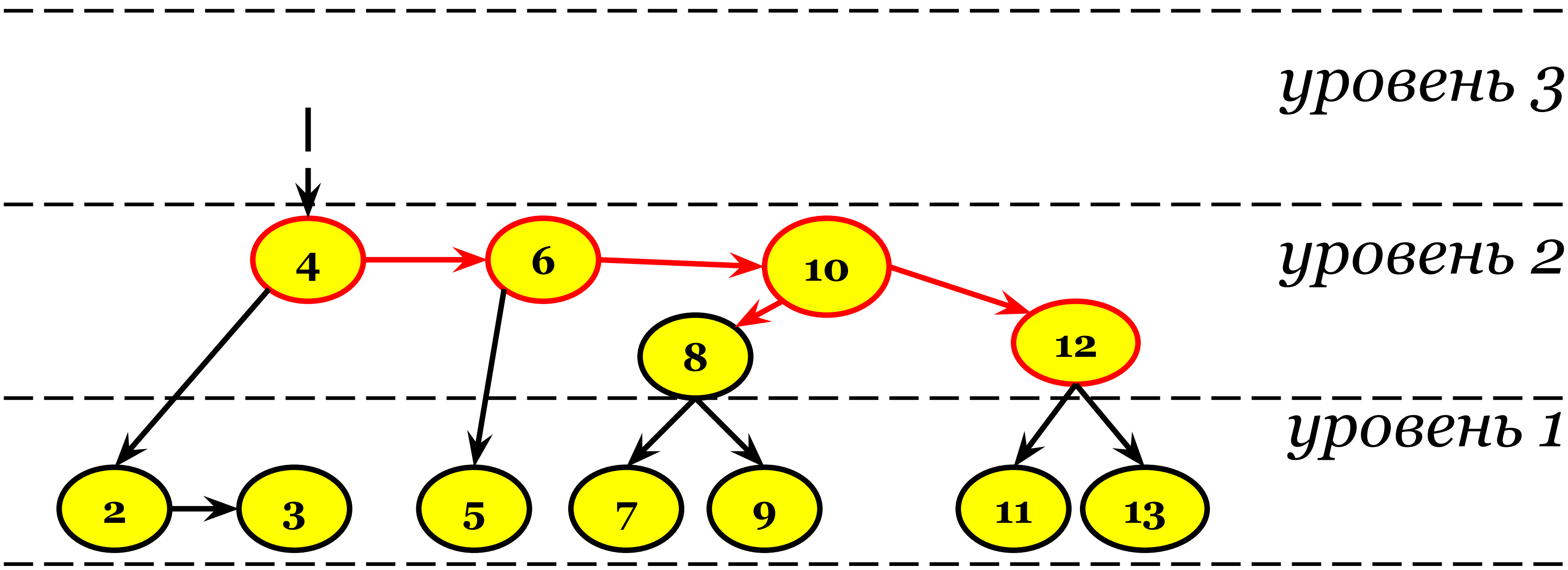
Пример удаления

- Skew (node 4); *//ничего не происходит*
- Skew (node 4^.right); //узел 10



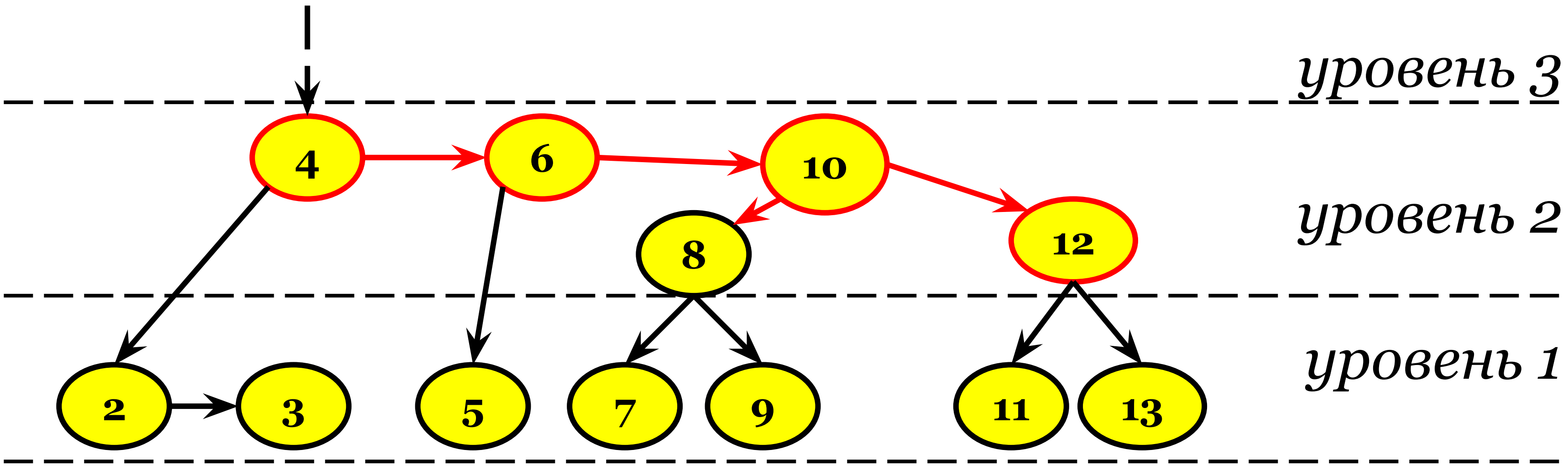
Пример удаления

- Skew (node 4); //ничего не происходит
- Skew (node 4^.right); //узел 10



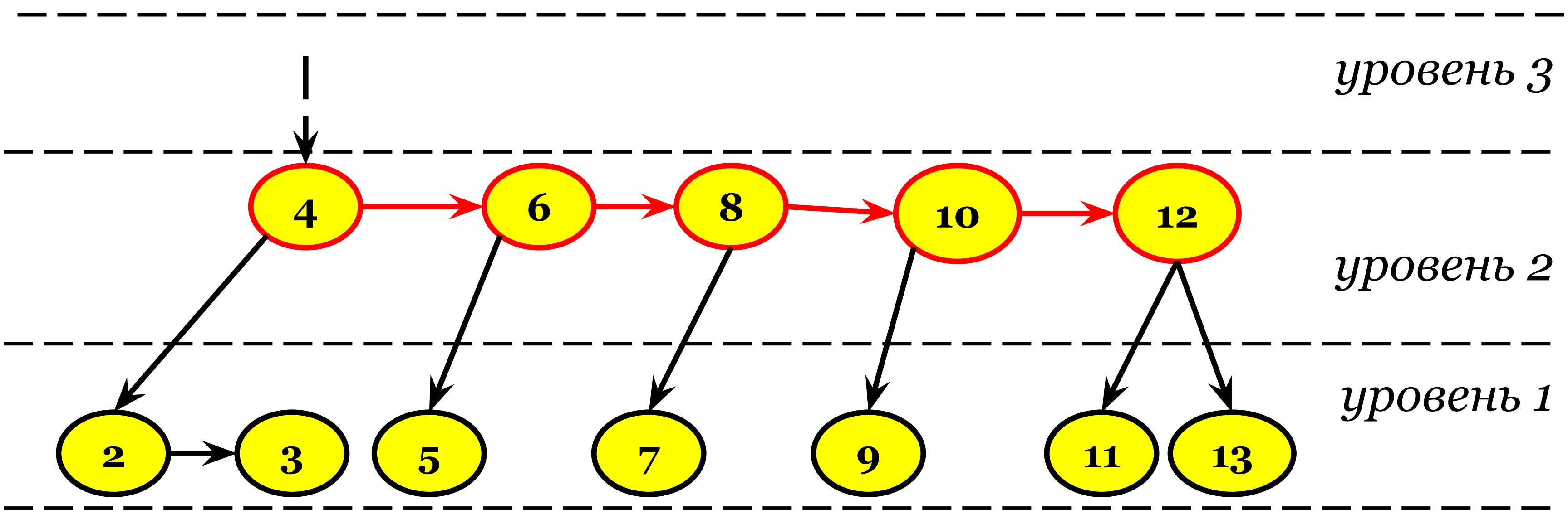
Пример удаления

- Skew (node 4); //ничего не происходит
- Skew (node 4^.right); //узел 10
- Skew (node 4^.right^.right); //снова узел 10



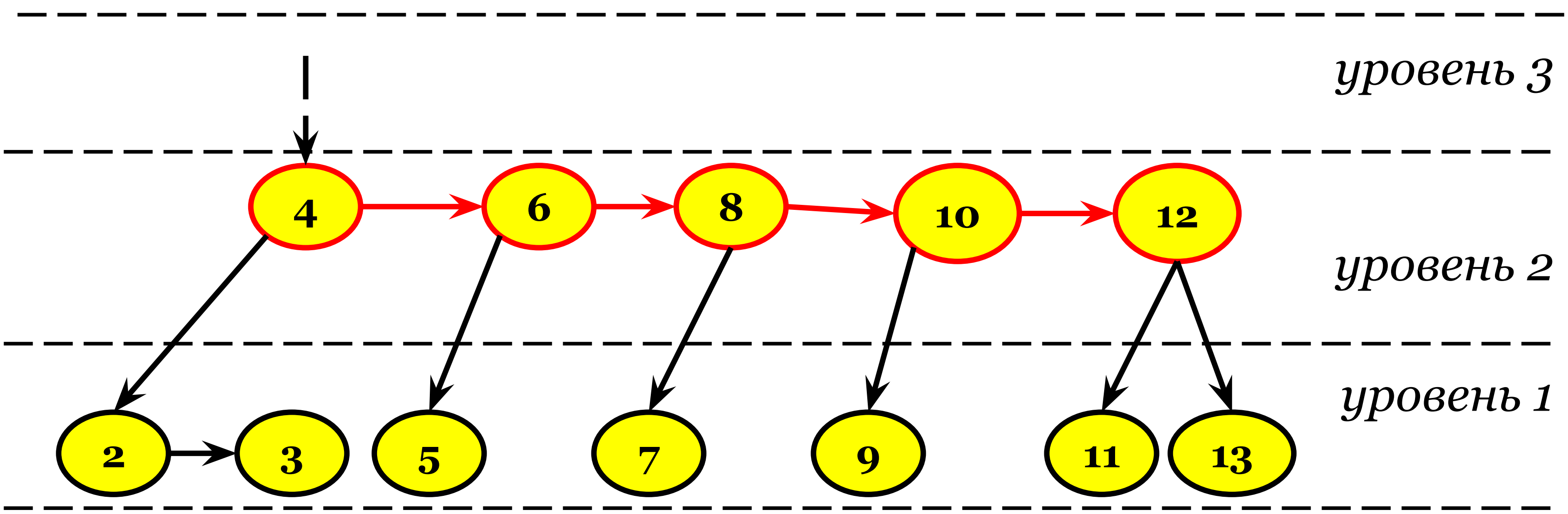
Пример удаления

- Skew (node 4); //ничего не происходит
- Skew (node 4^.right); //узел 10
- Skew (node 4^.right^.right); //снова узел 10



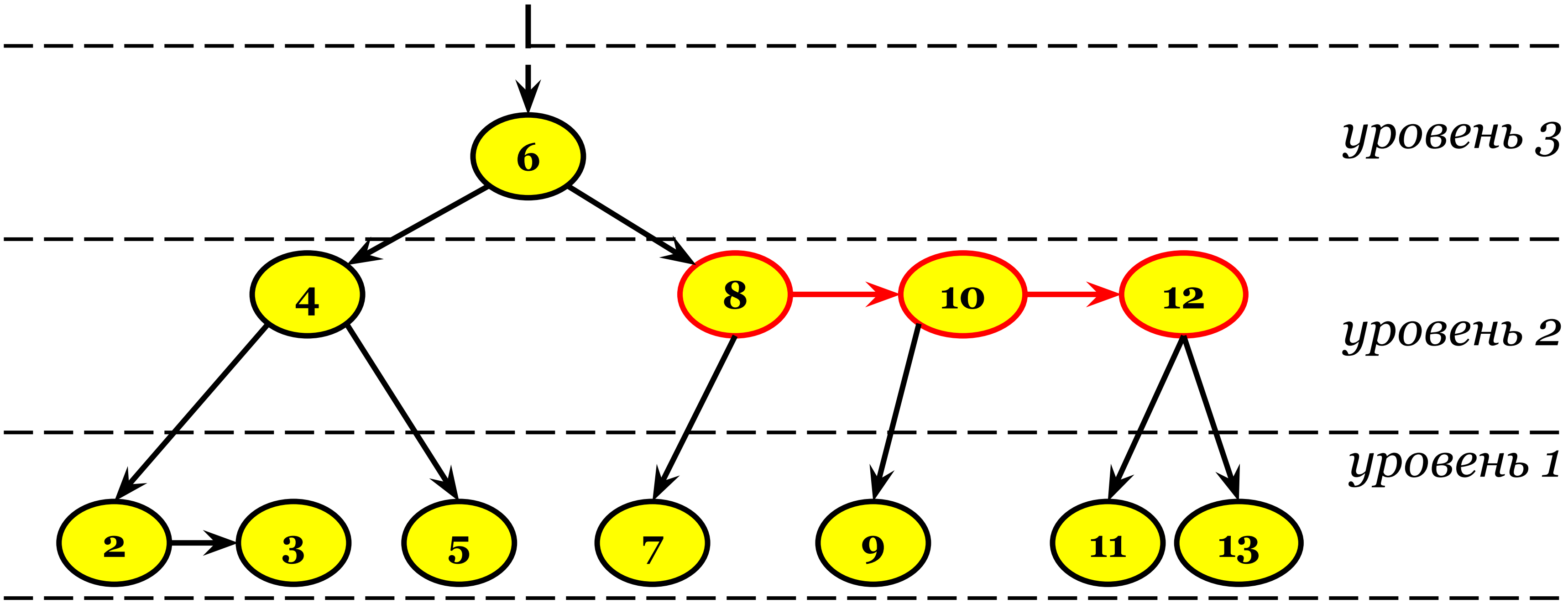
Пример удаления

- Skew (node 4); //*ничего не происходит*
- Skew (node 4^.right); //*узел 10*
- Skew (node 4^.right^.right); //*снова узел 10*
- Split (node 4); //*появится новый корень поддерева*



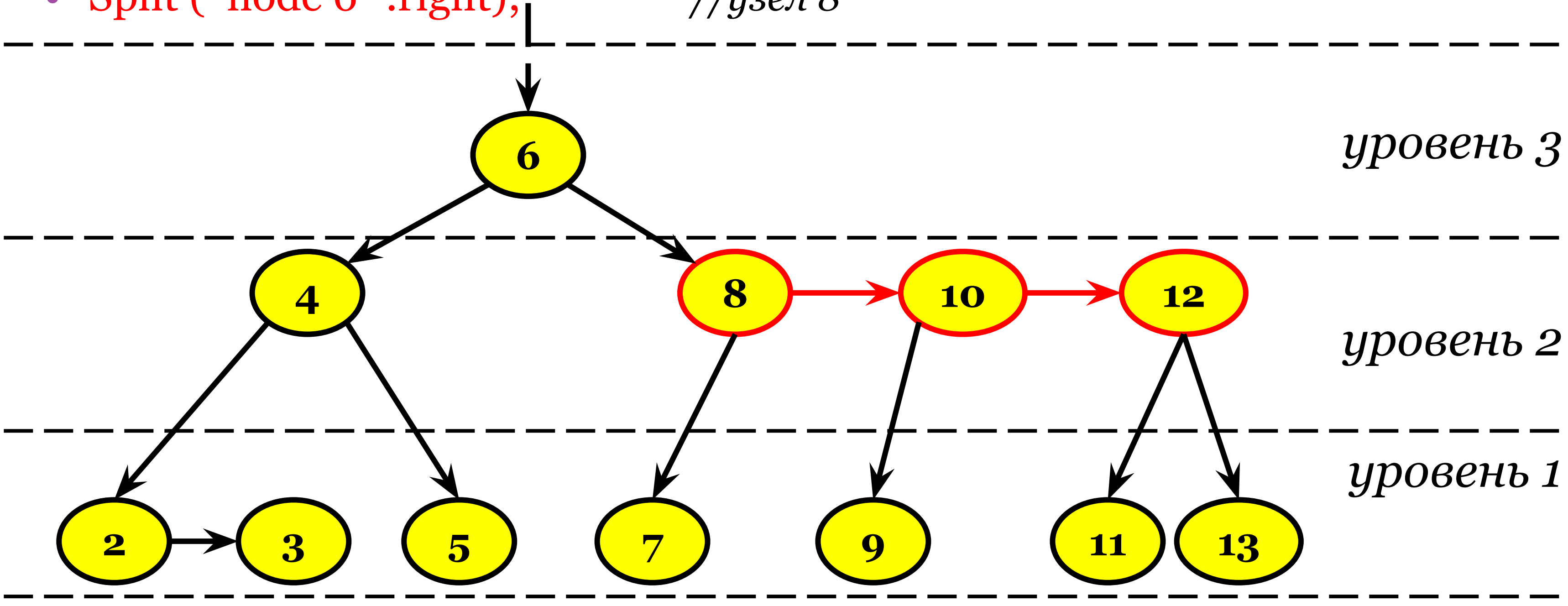
Пример удаления

- Skew (node 4); *//ничего не происходит*
- Skew (node 4^.right); *//узел 10*
- Skew (node 4^.right^.right); *//снова узел 10*
- Split (node 4); *//появится новый корень поддерева*



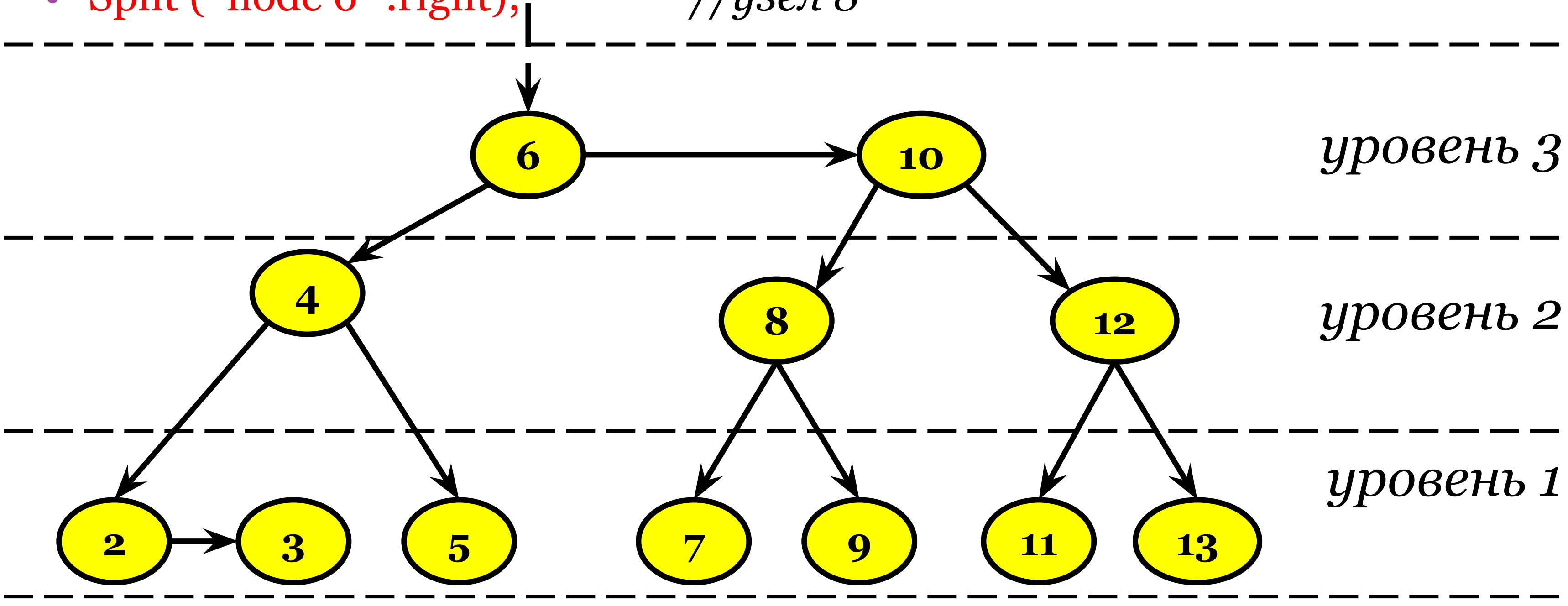
Пример удаления

- Skew (node 4); *//ничего не происходит*
- Skew (node 4^.right); *//узел 10*
- Skew (node 4^.right^.right); *//снова узел 10*
- Split (node 4); *//появится новый корень поддерева*
- Split (node 6^.right); *//узел 8*



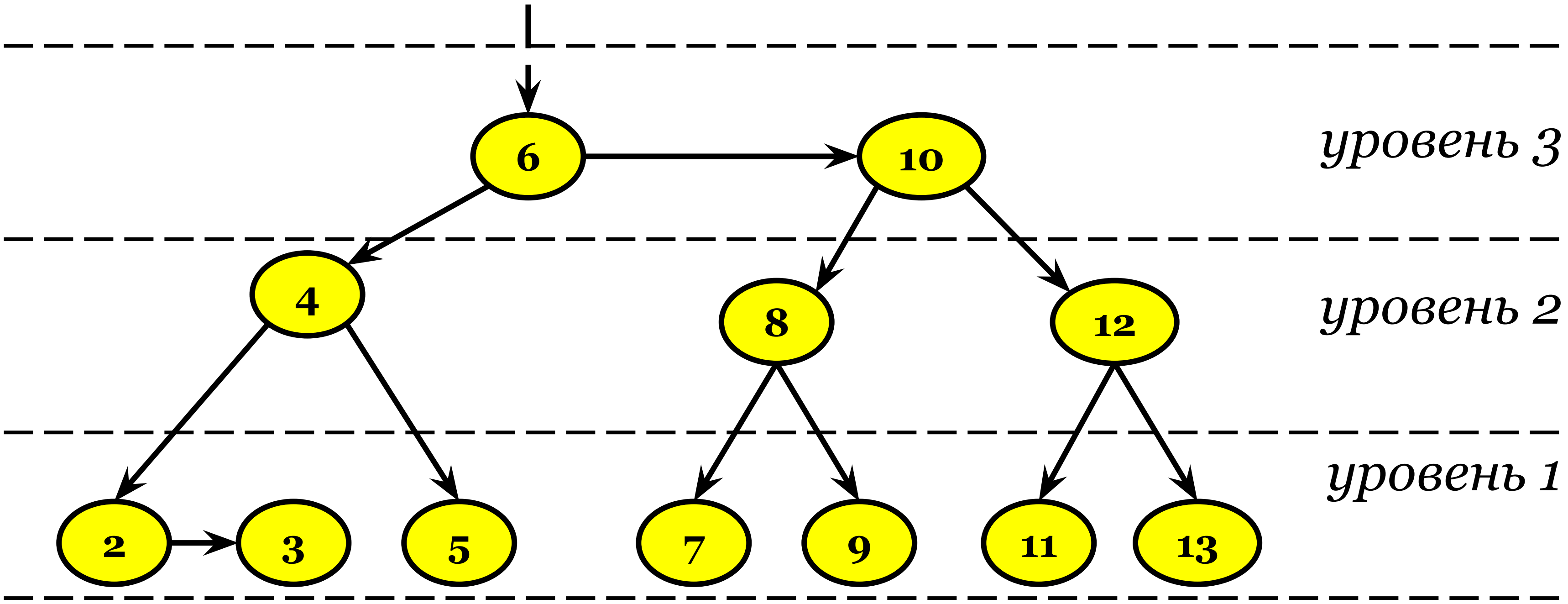
Пример удаления

- Skew (node 4); *//ничего не происходит*
- Skew (node 4^.right); *//узел 10*
- Skew (node 4^.right^.right); *//снова узел 10*
- Split (node 4); *//появится новый корень поддерева*
- Split (node 6^.right); *//узел 8*



Пример удаления

- Дерево полностью сбалансировано!



Заключение

- В своей работе Арне Андерссон делает вывод, что если сравнивать по производительности четыре типа двоичных деревьев поиска, а именно:
  - ✔ АВЛ-дерево;
  - ✔ красно-черное дерево;
  - ✔ 2-3-дерево;
  - ✔ АА-дерево,

  то можно сделать вывод, что сбалансированность (и скорость поиска) лучше всего у АВЛ-дерева, чуть хуже у красно-черного дерева, и еще чуть хуже у 2-3-дерева и эквивалентного ему по структуре АА-дерева.
- Алгоритмы балансировки очень сложны для АВЛ-дерева и 2-3-дерева, поэтому на практике предпочитают использовать красно-черные и АА-деревья. Самые простые алгоритмы вставки и удаления узлов у АА-дерева, однако, если вставка и удаление элементов встречаются гораздо реже, чем поиск, то красно-черные деревья будут предпочтительнее .
- Преимуществом АА-дерева по сравнению с красно-черным деревом является то, что алгоритмы, используемые при вставке и удалении узла в АА-дереве, а также балансировка дерева существенно проще, чем в красно-черном дереве.

СПИСОК ИСПОЛЬЗУЕМОЙ ЛИТЕРАТУРЫ И ИСТОЧНИКОВ

- AA-Tree – http://en.wikipedia.org/wiki/AA_tree.
- AA-Tree или простое бинарное дерево – http://habrahabr.ru/post/110212 .
- AA-дерево – http://www.proteus2001.narod.ru/gen/txt/8/aa.html.
- A. Andersson. Balanced search trees made simple. Algorithms and Data Structures, pages 60-71, 1993 .
- Сайт А.А.Кубенского для студентов ИТМО. Алгоритмы и структуры данных. Презентация лекции по 2-3 деревьям и AA-деревьям https://drive.google.com/file/d/0BFHfoLzonFRMV0xb1d1RXBSblU/view?pref=2&pli=1 .
- David Babcock. York College of Pennsylvania. CS 350 : Data Structures AA Trees .
- The European Journal for the Informatics Professional UPGRADE http://www.upgrade-cepis.org Vol. V, No. 5, October 2004 .

Спасибо за внимание!