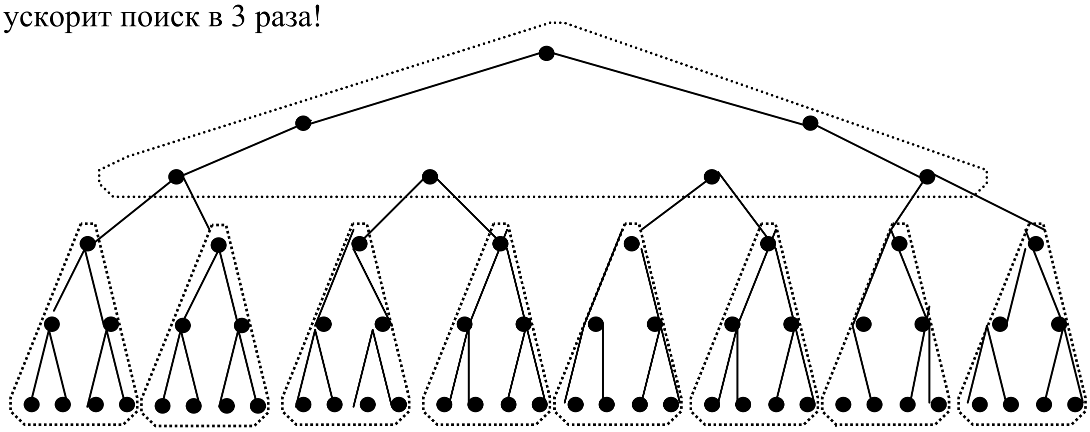
## Б-деревья

До сих пор задачи поиска для деревьев решались в предположении, что все данные хранятся во внутренней памяти. Рассмотрим задачу внешнего поиска, когда данные поступают из внешней памяти. Например, рассмотрим бинарное дерево поиска и предположим, что оно хранится в файле. Тогда изученные алгоритмы становятся малоэффективными, так как потребуют $Log_2N$ обращений к диску. Если же разделить дерево на страницы, например по 7 узлов в каждой, и считывать за обращение одну страницу, то число обращений уменьшится. Например, при N=1млн., число обращений к обычному дереву равно примерно 20, а при постраничном чтении число обращений сократится в 3 раза, что ускорит поиск в 3 раза!



Предположим, что узлы дерева хранятся на внешнем запоминающем устройстве и нам нужно ускорить доступ к узлам дерева. Разобьем дерево на поддеревья, состоящие из n узлов, которые назовем страницами. Как выбрать n, как организовать страницы, чтобы они были эффективно заполнены? Понятно, что n не должно быть слишком большим, так как размеры внутренней памяти ограничены, и чтение будет занимать длительное время. Кнут показал, что значения близкие к оптимальным лежат в диапазоне от 200 до 500.

В 1970 году Р. Байер предложил структуру данных, названную Б-деревьями, позволяющую производить поиск по большому дереву с гарантированной эффективностью довольно простыми методами.

**Определение:** Б-деревом порядка n называется динамическая структура со следующими свойствами

1) Каждая страница имеет не более 2n элементов.

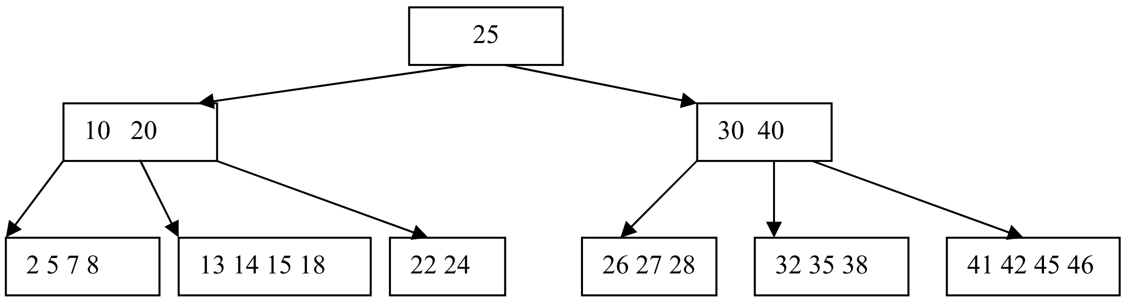
2) Каждая страница, кроме корневой, имеет не менее n элементов. Корневая страница может иметь 1 элемент.

3) Каждая страница является либо листом, либо содержит m+1 потомков, где m - число элементов на этой странице.

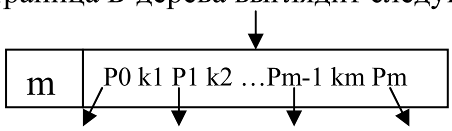
4) Все листья находятся на одном уровне.

Элементы на страницах Б-деревьев располагаются в возрастающем порядке, что является наследием от деревьев поиска и облегчает алгоритм поиска места для вставки элемента. Эффективность организации страниц заключается в том, что память всегда используется минимум на 50%, так как страницы всегда как минимум наполовину заполнены.

Пример Б-дерева порядка 2. Все страницы кроме корневой могут содержать от 2 до 4 элементов, от 3 до 5 потомков.




В общем виде страница Б-дерева выглядит следующим образом:




Здесь m – число элементов на странице,
k1, …km – элементы, расположенные на странице,
P0 – указатель на страницу-потомка с элементами, меньшими k1
P1 - указатель на страницу-потомка с элементами, большими k1 и меньшими k2
…
Pm – указатель на страницу-потомка с элементами, большими km

```
Const nn= 2*n;
Type
Btree = ^Page;
Item = record
        elem : integer;
        p    : Btree;
       end;
Page = record
        m   :  0..nn;
        p0  :  Btree;
        E   :  array[1..nn] of item;
      end;
```

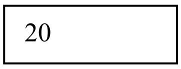
Алгоритм поиска элемента x в Б-дереве сводится к поиску элемента на текущей странице. Это можно сделать известными методами (последовательным или бинарным поиском). Если поиск x на текущей странице неудачен, то в зависимости от значения x возможны три случая:
1) при ki<x<ki+1 (1<=i<m) - продолжим поиск на странице pi
2) при km<x - продолжим поиск на странице pm
3) при x<k1 - продолжим поиск на странице p0

## Алгоритм вставки нового элемента

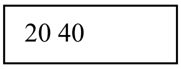
Сначала необходимо найти место для вставки. Задача сводится к поиску подходящей страницы для элемента. Если страница не заполнена, то элемент добавляется в нее с сохранением упорядоченности элементов на странице. Если страница заполнена, то на ней находится 2n элементов. Добавление еще одного элемента приводит к расщеплению страницы на две ровные части, а средний элемент переходит на верхнюю страницу. Две страницы имеют минимальное значение элементов (n). Элемент, перешедший наверх, в свою очередь может вызвать переполнение и расщепление страницы. В конечном итоге некоторый элемент может быть выдавлен на самый верхний уровень, образовав новый корень. Таким образом, Б-деревья растут странным образом – от листьев к корням.

Рассмотрим процедуру последовательного добавления элементов 20, 40, 10, 30, 15, 35, 7, 26, 18, 22, 5, 42, 13, 46, 27, 8, 32, 38, 24, 45, 25 в Б-дерево порядка 2:

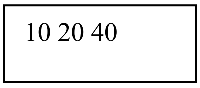
Добавим 20



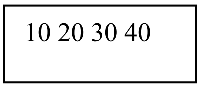

Добавим 40




Добавим 10

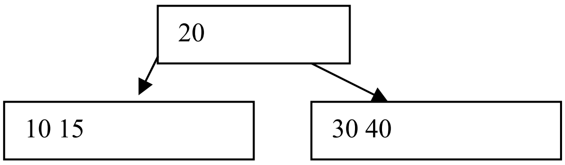



Добавим 30
Страница заполнена

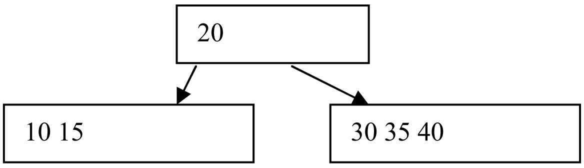



---------------------------------

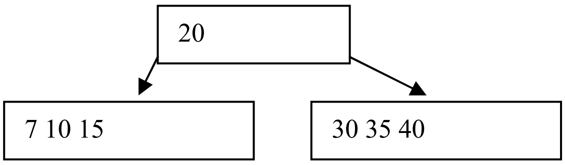
Добавим 15
Происходит расщепление страницы



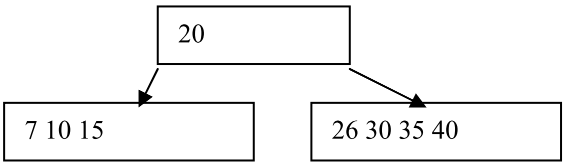

Добавим 35



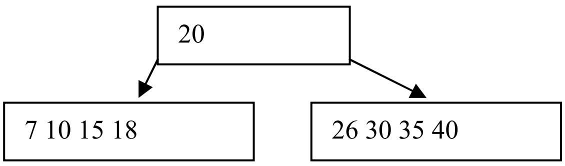

Добавим 7




Добавим 26

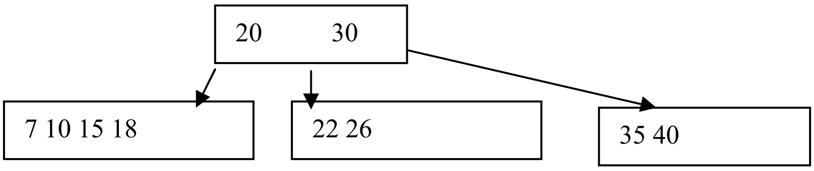



Добавим 18

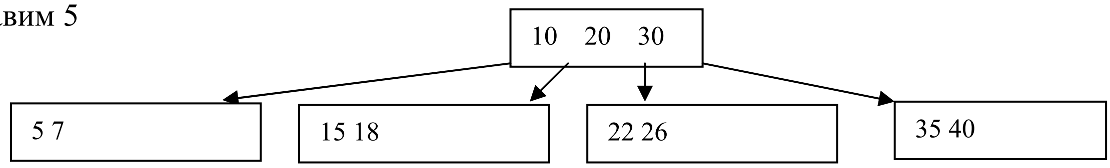



Нижние страницы заполнены
---------------------------------

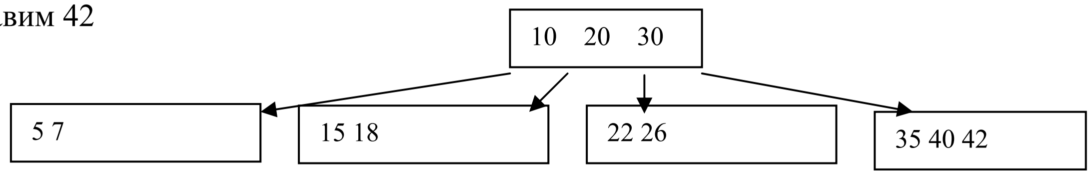
Добавим 22



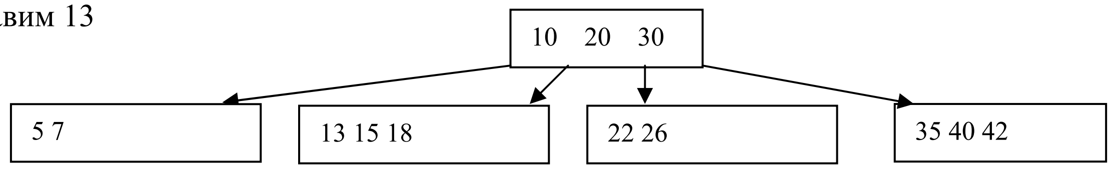

Добавим 5



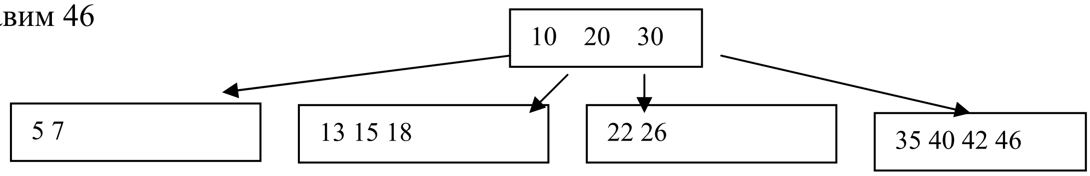

Добавим 42



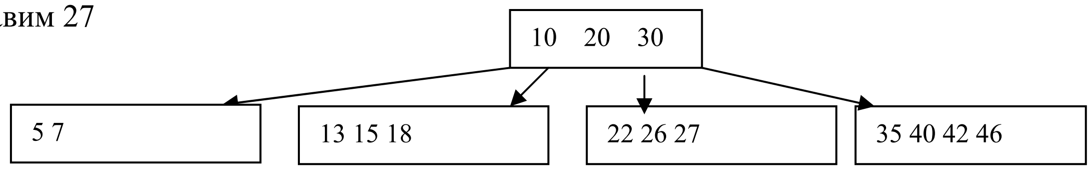

Добавим 13



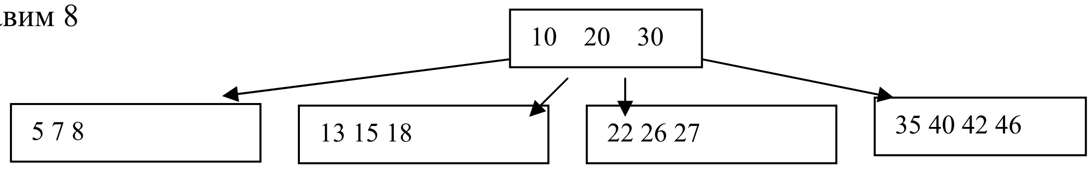

Добавим 46



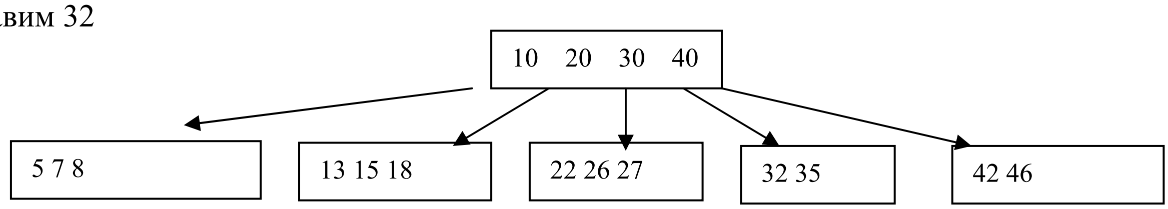

Добавим 27



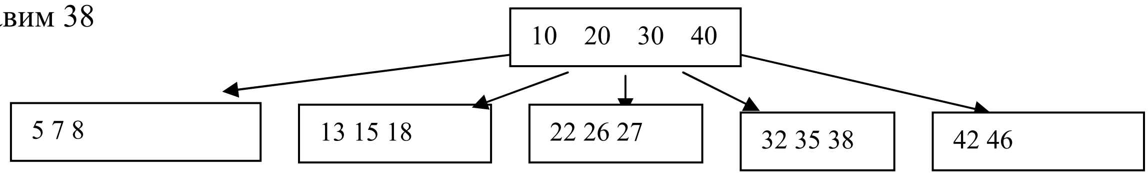

Добавим 8



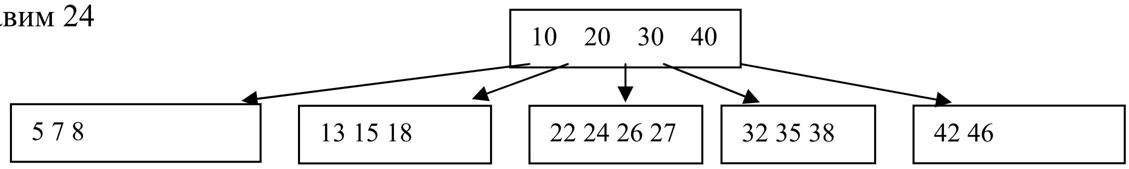

Добавим 32



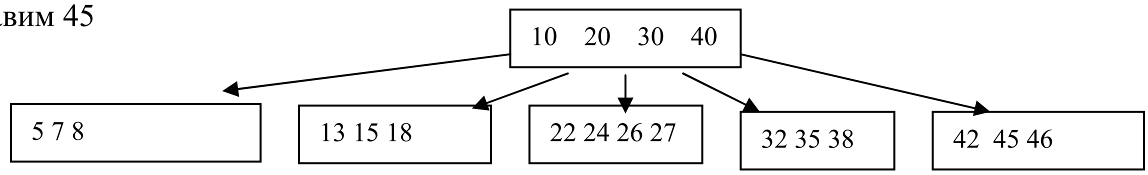
Добавим 38



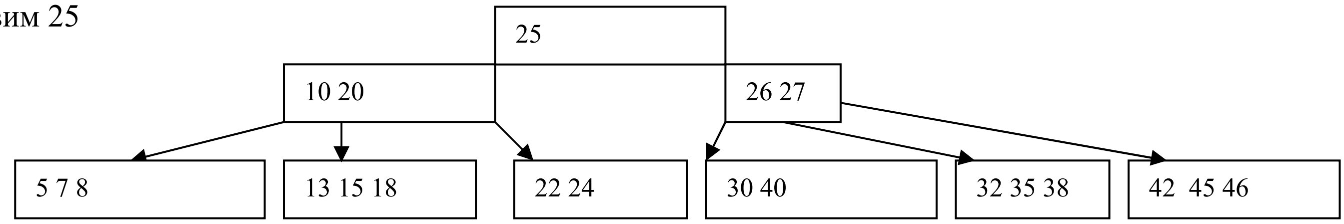

Добавим 24




Добавим 45




Добавим 25




```
Const n= 2;
      nn= 2*n;
Type
Btree = ^Page;
Item = record
          elem : integer;
          p    : Btree;
          count: integer;
       end;
Page = record
          m    :  0..nn;
          p0   :  Btree;
          E    :  array[1..nn] of item;
        end;
Procedure Search(x: integer; T: Btree; var h: boolean; var v : item);
var k,l,r: integer;
    q    : Btree;
    u    : Item;
Procedure Insert;
var i : integer;
    TT: Btree;
begin
       if T^.m < nn then {}
          begin
             inc(T^.m);
```

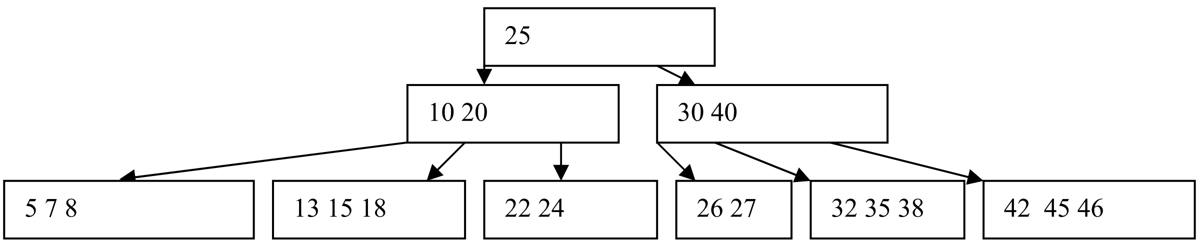
```
            h:=false ;
            for i:= T^.m downto r+2 do T^.E[i]:=T^.E[i-1];
            T^.E[r+1] :=u
          end
        else
            begin
             new(TT);
             if r<=n then
               begin
                 if r = n then v:=u
                 else
                  begin v:= T^.E[n];
                    for i:= n downto r+2 do T^.E[i]:= T^.E[i-1];
                     T^.E[r+1] := u
                  end;
                  for i:=1 to n do TT^.E[i]:=T^.E[i+n]
               end
             else
             begin
                 r:=r-n;
                 v:=T^.E[n+1];
                 for i:= 1 to r-1 do TT^.E[i]:=T^.E[i+n+1];
                  TT^.E[r]:=u;
                 for i:= r+1 to n do TT^.E[i]:=T^.E[i+n];
             end;
             T^.m:=n; TT^.m:=n; TT^.p0:=v.p; v.p:=TT
            end
end;

begin
   if T = nil then {элемента нет,заполним v значением x,дерево увеличилось}
   begin
      v.elem  :=x;
      v.p     :=nil;
      v.count :=1;
      h       := true;
   end
   else       {бинарным поиском в упорядоченном массиве найдем x}
   begin
      l:=1; r:=T^.m;
      repeat
          k:=(l+r) div  2;
          if x<=T^.E[k].elem then r:=k-1;
          if x>=T^.E[k].elem then l:=k+1;
      until r < l;
      if l - r > 1 then  {если нашли увеличим счетчик}
                  begin inc(T^.E[k].count); h:= false end
      else begin       {если не нашли, перейдем на нужную подстраницу}
          if r = 0 then q:=T^.p0 else q:=T^.E[r].p;
           search(x,q,h,u);   {рекурсивно обработаем подстраницу}
```

```
            if h then insert
                                   {если дерево увеличилось вставим элемент
                                   либо на страницу, либо расщепим ее на две }
            end;
        end;
    end;
```

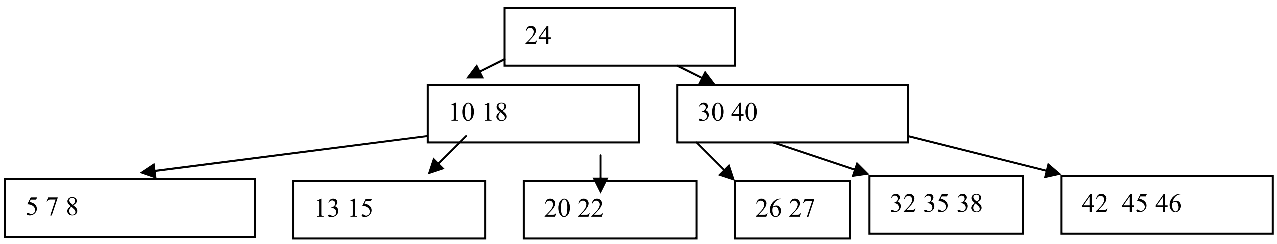
Рассмотрим задачу удаления узлов из Б-дерева. Различаются два случая:

1. Элемент находится на листе. Удаление просто.
2. Элемент находится не на листе. Тогда он заменяется на самый правый элемент самого правого листа, а у листа удаляется элемент.

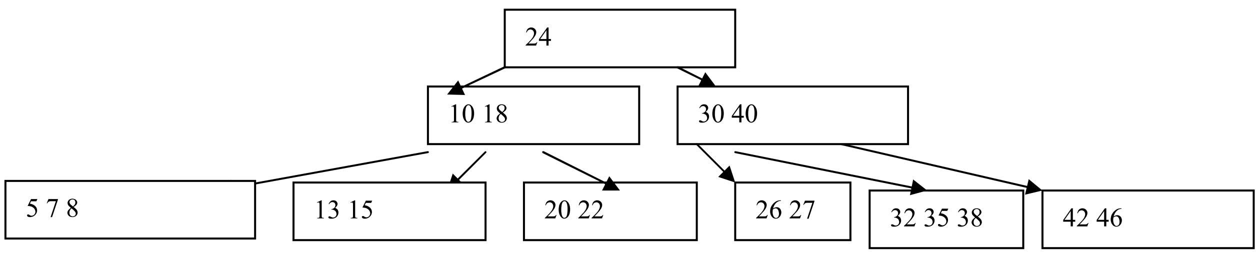
После удаления необходимо проверить, что на листе осталось элементов не меньше n, иначе будет нарушено основное условие Б-деревьев. В этом случае забираются элементы с одной из соседних страниц. Если это невозможно (в случае, когда соседняя страница тоже достигла своего минимального уровня), то страницы сливают, добавляя средний элемент со страницы-предка.
Будем удалять у построенного дерева узлы в порядке обратном их поступлению: 25, 45, 24, 38, 32, 8, 27, 46, 13, 42, 5, 22, 18, 26, 7, 35, 15.



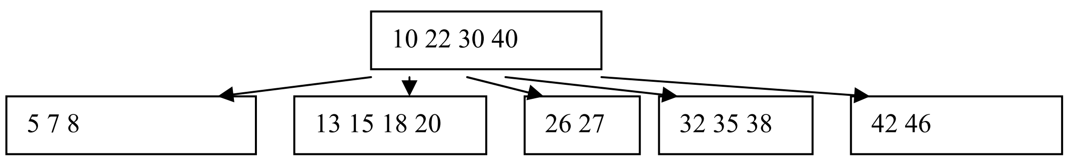

Удалим 25



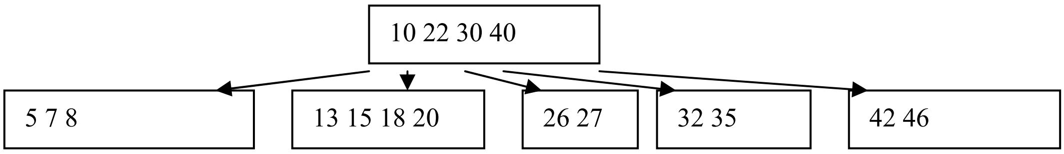

Удалим 45



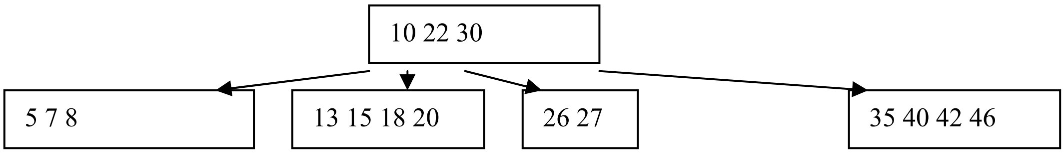

Удалим 24



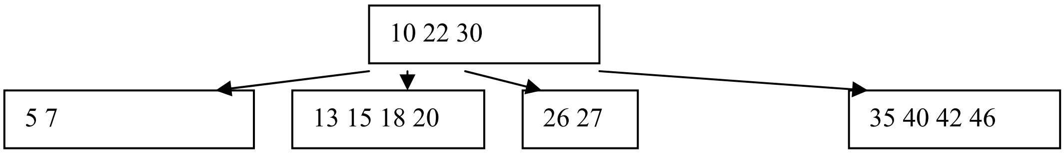
Удалим 38



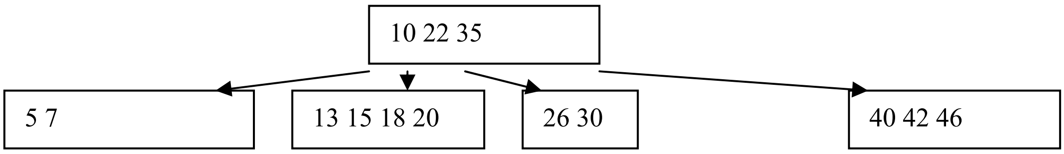

Удалим 32



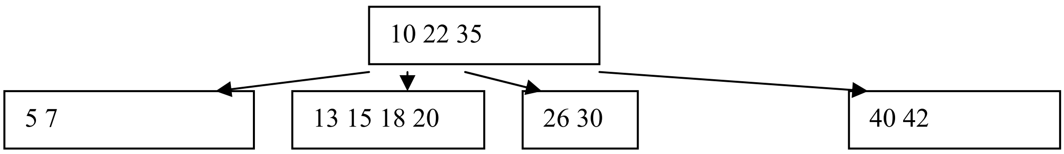

Удалим 8



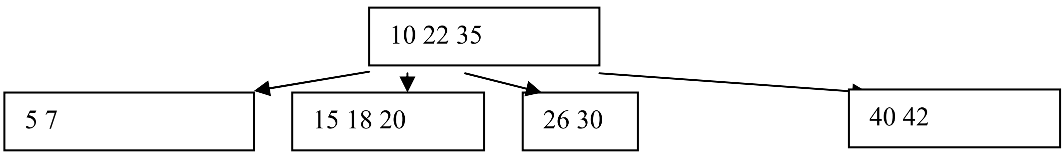

Удалим 27



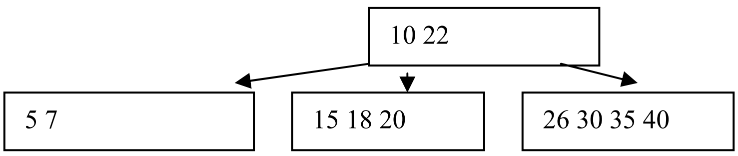

Удалим 46



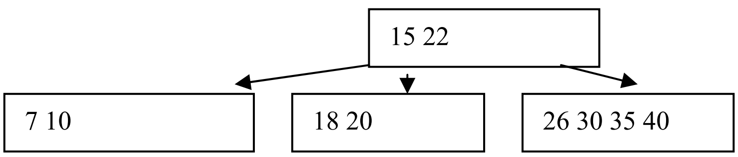

Удалим 13



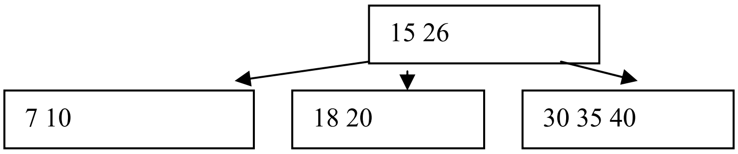

Удалим 42



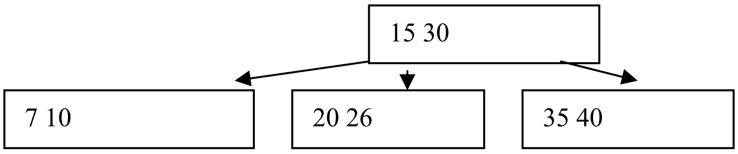

Удалим 5



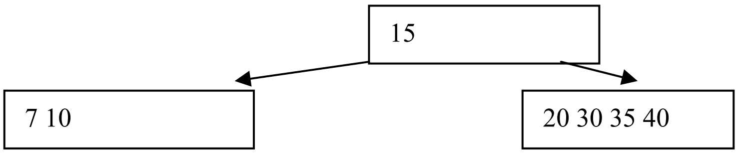
Удалим 22



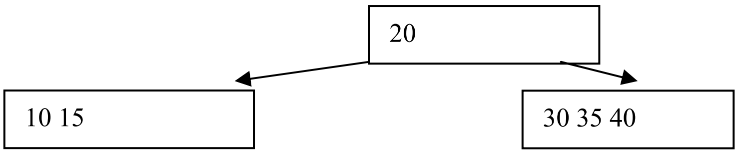

Удалим 18



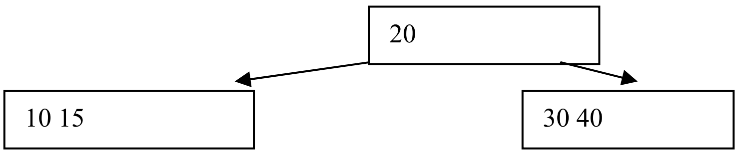

Удалим 26



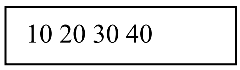

Удалим 7




Удалим 35




Удалим 15




```
procedure Delete(x: integer; T: Btree; var h: boolean);
var i, l, r, k : integer;
   Q         : Btree;
procedure UnderFlow(C, A: Btree; s: integer; var h : boolean);
{A - страница с нехваткой, C - предок,  B-правая или левая страница от A}
var B: Btree;
   i, k, mb, mc : integer;
begin
    mc:=C^.m;
    if s< mc then
    begin
    {B - правая страница от A}
        s:=s+1;
        B:=C^.E[s].p;
        mb:=B^.m;
        k:=(mb-n+1) div 2;
```

```
A^.E[n]:=C^.E[s]; A^.E[n].p:=B^.p0;
if k>0 then
  begin
   for i:=1 to k-1 do A^.E[i+n]:=B^.E[i];
           C^.E[s]:=B^.E[k];
           C^.E[s].p:=B;
           B^.p0:=B^.E[k].p;
           mb:=mb-k;
           for i:=1 to mb do B^.E[i]:=B^.E[i+k];
           B^.m:=mb; A^.m:=n-1+k;
           h:=false
         end
         else
           begin
           for i:=1 to n do A^.E[i+n] :=B^.E[i];
           for i:= s to mc-1 do C^.E[i]:=C^.E[i+1];
           A^.m:=nn; C^.m:=mc-1;
           h:= C^.m<n
           end
    end
    else
    {s - последний правый элемент на C ->
    B - левая страница от A}
    begin
         if s =1 then B:=C^.p0 else B:=C^.E[s-1].p;
         mb:=B^.m+1;
         k:=(mb-n) div 2; {0<=k<=n, k=0 при B^.m=n и k>0 при
B^.m>n}
         if k>0 then
         {пересылка на A 1 элемента из C и k элементов  со
страницы B
         на A станет n-1+k, B ---  mb}
         begin
           for i:=n-1 downto 1 do A^.E[i+k]:=A^.E[i];
           A^.E[k]:=C^.E[s]; A^.E[k].p:= A^.p0; mb:=mb-k;
           for i:=k-1 downto 1 do A^.E[i]:=B^.E[i+mb];
           A^.p0:=B^.E[mb].p;
           C^.E[s]:=B^.E[mb];
           C^.E[s].p:=A;
           B^.m:=mb-1; A^.m:=n-1+k;
           h:=false
         end
         else
         {k=0 ->
         размер B = n, A = n-1  -> A+B=2n-1 + 1 элемент из C
         слияние страниц A C и B -
          общая страница B заполняется полностью nn элементами
          на странице C на 1 элемент меньше}
         begin
           B^.E[mb]:=C^.E[s];
           B^.E[mb].p:=A^.p0;
```

```
                for i:=1 to n-1 do B^.E[i+mb]:=A^.E[i];
                B^.m:=nn; C^.m:=mc-1;
                h:=C^.m<n
              end
            end
end;

procedure Del(P: Btree; var h: boolean);
var P1: Btree;
begin
    with P^ do
    begin
      P1:=E[m].p {спустимся к самой правой подстранице P};
      if P1 <> nil then
             begin Del(P1,h); if h then underflow(P, P1, m, h)
end
      else begin
          P^.E[m].p:=T^.E[k].p;
          T^.E[k]:=P^.E[m];  {заменяем удаляемый элемент на самый
правый}
          dec(m);
          h:=m<n
            end
     end;
end;
begin
           if T = nil then {элемента нет}
     begin
          writeln('Элемента нет в дереве');
          h       := false;
     end
     else       {бинарным поиском в упорядоченном массиве найдем x
                   и его номер к}
     begin
          l:=1; r:=T^.m;
          repeat
                k:=(l+r) div  2;
                if x<=T^.E[k].elem then r:=k-1;
                if x>=T^.E[k].elem then l:=k+1;
          until r < l;
          if r = 0 then q:=T^.p0 else q:=T^.E[r].p;
             {определим потомка-нужную подстраницу q  от x}
          if l - r > 1 then  {если нашли- удалим элемент}
              begin
              if Q = nil then
              {если потомок nil, то
               элемент находится на листе - удалим его
               Уменьшим размер и сдвинем элементы после k влево на
1}
                   begin
                     dec(T^.m);
```

```
                    h:=T^.m < n;
                    for i:= k to T^.m do T^.E[i]:=T^.E[i+1]
                   end
              else
              {если потомок не nil, то
               страница имеет потомков - вызовем процедуру
удаление
               и если была нехватка то процедуру пересылки
элементов}
                  begin
                  del(q,h);
                  if h then underflow(T, Q, r, h)
                  end
             end
          else begin
              {не нашли элемент}
                delete(x,Q,h);   {рекурсивно обработаем
подстраницу
                и если была нехватка то вызовем процедуру
пересылки}
                if h then underflow(T,Q,r,h)
               end;
      end;
end;
```

Упражнение

1. Предположим, что ключи 1, 2, 3,… в возрастающем порядке вставляются в первоначально пустое Б-дерево порядка 101. Какой ключ впервые приведет к появлению листьев на уровне 4?