Поиск

Алгоритмы
и алгоритмические языки

u.to/72D_Gg

Лекция 9, 2 апреля, 2021

Лектор:

Дмитрий Северов, кафедра информатики 608 КПМ

dseverov@mail.mipt.ru

Обратная связь: **u.to/7Wn7Gg**

Термины

1. Поиск - быстрое извлечение по конкретному признаку части информации, накопленной в компьютере
2. Ключ – специальное поле записи, однозначно характеризующее эту запись.
3. Таблица или файл – набор всех записей.
4. Результаты поиска:
   1. удачный – запись с искомым ключом найдена;
   2. неудачный – запись не найдена.

Задача

- Есть элементы S, **с ключами** K, имеющими отношение порядка.
- Определить отображение H ключей K в адреса A памяти машины так, чтобы поиск элемента с заданным ключом требовал как можно меньше ресурсов?

Линейный поиск O(N)

```
template <typename T> class List {
   Item<T> *front,*back; T rab;
   Item<T>* find(Item<T>* &F, const T& k) {
      if(front==NULL) return (F=NULL);
      Item<T> *ptr=F=front;
      if(front->get_node()==k) return 0;
      while((F=ptr->get_next())!=NULL) {
         if(F->get_node()==k) break; ptr=F; }
    return ptr; }
```

Линейный поиск места включения в упорядоченном списке

```
Item<T>* Find(Item<T>* &F, const T& k) {
   if(front==NULL) return (F=NULL);
    Item<T> *ptr=F=front;
    if(front->get_node()==k) return 0;
    while((F=ptr->get_next())!=NULL)
       if(F->get_node()<k) ptr=F;
       else if(F->get_node()==k) break;
       else { F=NULL; return back; }
  return ptr; }
```

```
38 template <typename T> class List { // список
39 Item<T> *front,*back; T rab; // начало, конец
40 Item<T>* find(Item<T>* &F, const T& k) { // найти с равным ключом
41     if(front==NULL) return (F=NULL);
42     Item<T> *ptr=F=front;
43     if(front->get_node()==k) return 0;
44     while((F=ptr->get_next())!= NULL) {
45         if(F->get_node()==k) break; ptr=F; }
46     return ptr; }
47 Item<T>* Find(Item<T>* &F, const T& k) { // найти последний с не большим ключом
48     if(front==NULL) return (F=NULL);
49     Item<T> *ptr=F=front;
50     if(front->get_node()==k) return 0;
51     while((F=ptr->get_next())!=NULL)
52         if(F->get_node()<k) ptr=F;
53         else if(F->get_node()==k) break;
54         else { F=NULL; return back; }
55     return ptr; }
56 public:
```

6

```
#include <iostream>
#include <fstream>
using namespace std;
#include "My_str.h"
#include "SQL.h"
#include <windows.h>

unsigned long long start, end;
unsigned long long access_counter() { asm("rdtsc"); }

int main(int argn, char** argv) {
   My_string x, pattern("#include 222");
   Queue<My_string> q1, q2;
   List<My_string> a, b;

   ifstream in(argv[1]);

   while(in >> x) { a.push_back(x); b.push_back(x); q1.put(x); q2.put(x); }
   a.revers(); b.sort();

   cout << "---REVERSED---" << a << "---SORTED---" << b << endl;

   start = access_counter();
   cout << "a.remove: " << pattern << (a.remove(pattern)?" - OK\n":" - FAILED\n");
   while( !q1.empty() )
      { x=q1.get(); cout << "a.remove: " << x << (a.remove(x)?" - OK\n":" - FAILED\n");}
   end = access_counter();
   cout << "Elasped time: " << (end - start)/1.4e9 << " s\n\n";

   start = access_counter();
   cout << "b.Remove: " << pattern << (b.Remove(pattern)?" - OK\n":" - FAILED\n");
   while( !q2.empty() ) { x=q2.get();
      cout << "b.Remove: " << x << (b.Remove(x)?" - OK\n":" - FAILED\n"); }
   end = access_counter();
   cout << "Elasped time: " << (end - start)/1.4e9 << " s\n";

   return 0;
}
```

```
---REVERSED---
}
   return 0;

   cout << "Elasped time: " << (end - start)/1.4e9 << " s\n";
   end = access_counter();
      cout << "b.Remove: " << x << (b.Remove(x)?" - OK\n":" - FAILED\n"); }
```

```
#include "My_str.h"
using namespace std;
#include <fstream>
#include <iostream>
```

```
---SORTED---









      cout << "b.Remove: " << x << (b.Remove(x)?" - OK\n":" - FAILED\n"); }
      { x=q1.get(); cout << "a.remove: " << x << (a.remove(x)?" - OK\n":" - FAILED\n");}
   List<My_string> a, b;
```

```
unsigned long long start, end;
using namespace std;
}
```

```
a.remove: #include 222 - FAILED
a.remove: #include <iostream> - OK
a.remove: #include <fstream> - OK
a.remove: using namespace std; - OK
a.remove: #include "My_str.h" - OK
```

```
a.remove:    end = access_counter(); - OK
a.remove:    cout << "Elasped time: " << (end - start)/1.4e9 << " s\n"; - OK
a.remove:   - OK
a.remove:    return 0; - OK
a.remove: } - OK
Elasped time: 0.0307598 s
```

```
b.Remove: #include 222 - FAILED
b.Remove: #include <iostream> - OK
b.Remove: #include <fstream> - OK
b.Remove: using namespace std; - OK
b.Remove: #include "My_str.h" - OK
b.Remove: #include "SQL.h" - OK
b.Remove: #include <windows.h> - OK
b.Remove:  - OK
b.Remove: unsigned long long start, end; - OK
b.Remove: unsigned long long access_counter() { asm("rdtsc"); } - OK
b.Remove:  - OK
```

```
b.Remove:    start = access_counter(); - OK
b.Remove:    cout << "b.Remove: " << pattern << (b.Remove(pattern)?" - OK\n":" - FAILED\n"); - OK
b.Remove:    while( !q2.empty() ) { x=q2.get();  - OK
b.Remove:       cout << "b.Remove: " << x << (b.Remove(x)?" - OK\n":" - FAILED\n"); } - OK
b.Remove:    end = access_counter(); - OK
b.Remove:    cout << "Elasped time: " << (end - start)/1.4e9 << " s\n"; - OK
b.Remove:   - OK
b.Remove:    return 0; - OK
b.Remove: } - OK
Elasped time: 0.0285805 s
```

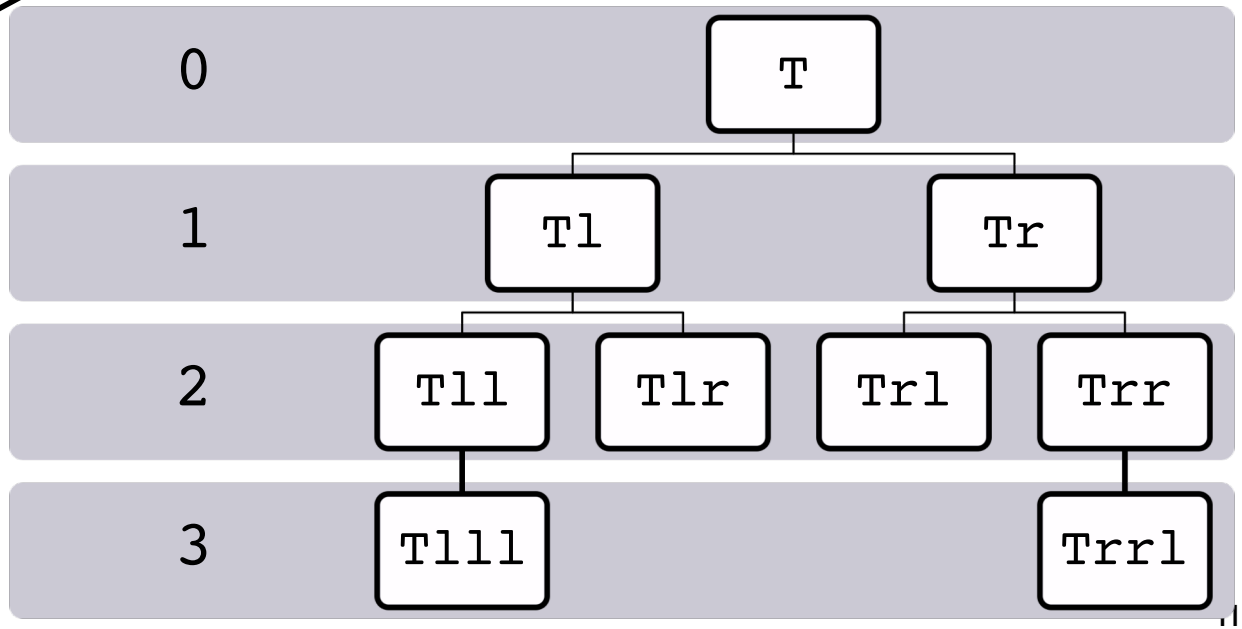
Идеи ускорения с ценой

- Самоорганизующаяся таблица (кеш)
  - обнаруженную запись – в «начало» таблицы и часто используемые записи будут в «начале» таблицы
  - нужно «начало» и переупорядочение
- Бинарный поиск
  - посмотрим ключ в «середине»
  - нужен произвольный доступ и упорядочение
- Правильное дерево
  - ищем за ≈**m** проверок среди ≈$\mathbf{2^m}$ элементов
  - нужна перестройка при записи и удалении

Поиск в дереве O(log N)

- Двоичное дерево поиска
  - Tl < T < Tr,
    например: Tlll < Tll < Tl < T < rl < Trrl < Trr
- Вероятность найти среди n равновероятных ключей: 2*(ln n + g -1)

Константа Эйлера 0,577…

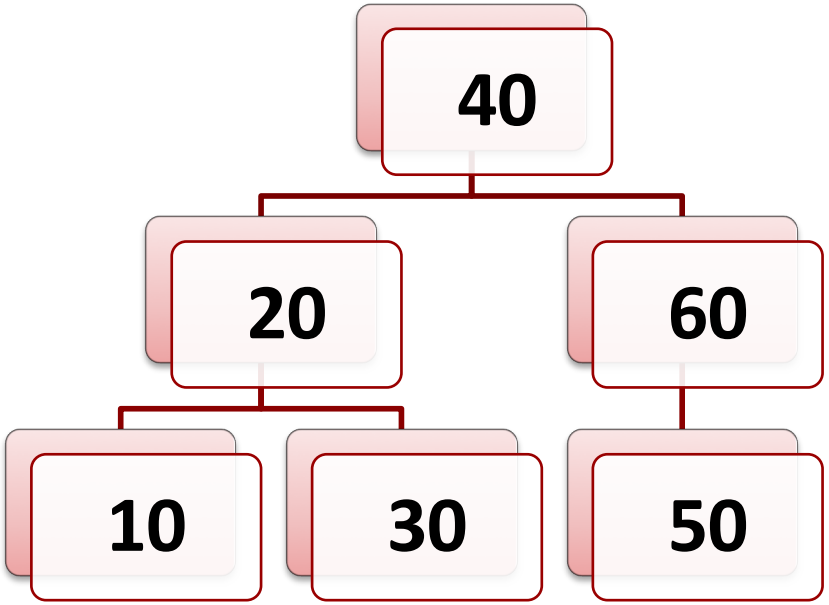


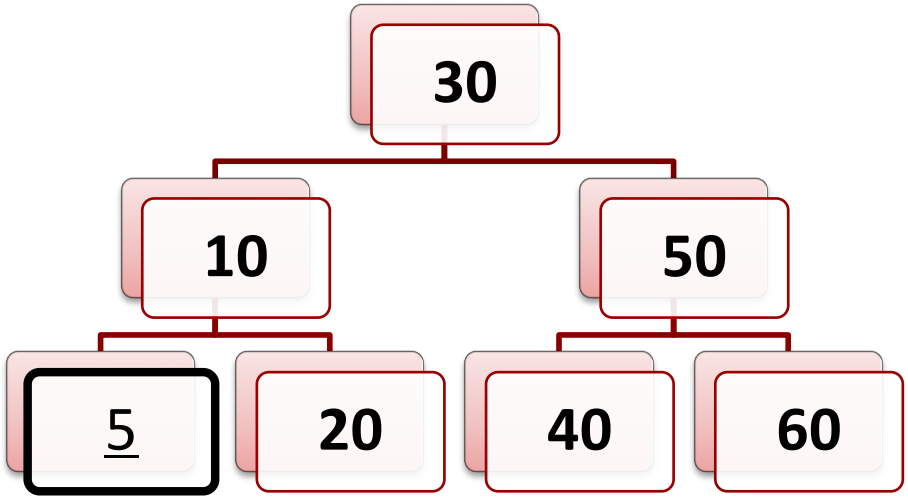
Идеально сбалансированное дерево

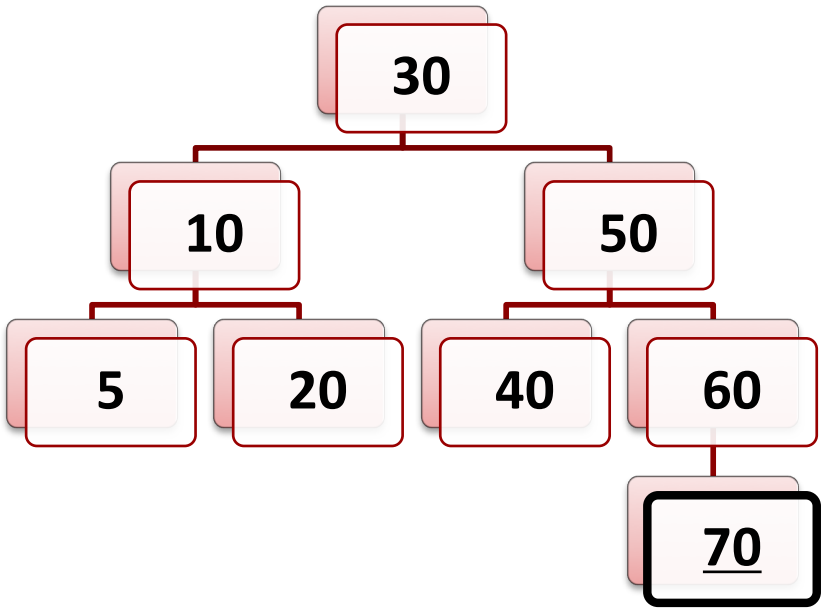
- Двоичное дерево поиска + $| n_r - n_l | \leq 1$
- Вероятность найти среди n равновероятных ключей: $\log_2 - 1$, что лучше любого двоичного дерева поиска в $2\ln 2 \approx 1{,}39\ldots$

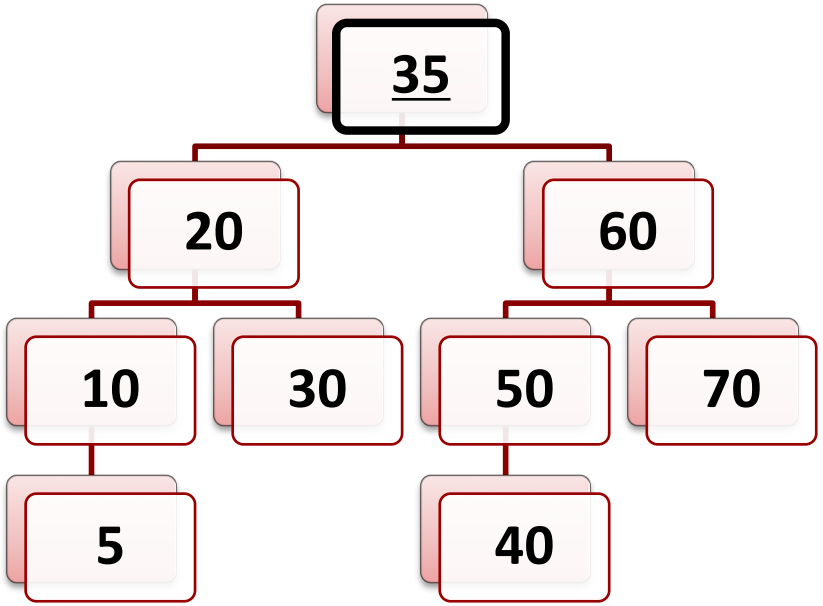
- Выигрыш превышает затраты ?
  - Если много узлов
  - Если много поисков и мало вставок

Идеально сбалансированное дерево









А если не идеализировать?

- Дерево сбалансированное идеально

$$| n_r - n_l | \leq 1$$

- АВЛ-дерево, сбалансированное «неидеально», **но практично** по Адельсону-Вельскому и Ландису

$$| h_r - h_l | \leq 1$$

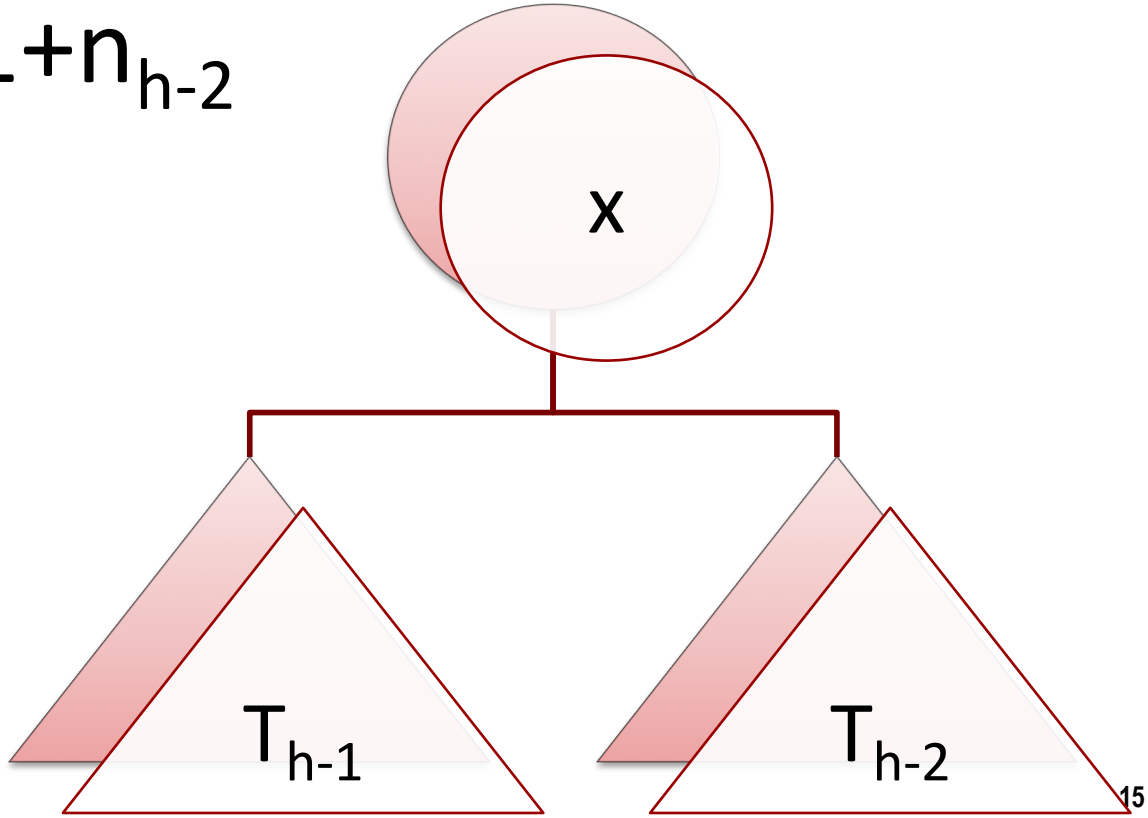
$$\log_2 (n+1) < h_b(n) < 1{,}4404 * \log_2 (n+2) - 0.328$$

- Какое АВЛ-дерево самое «плохое» - минимум узлов при заданной высоте ?

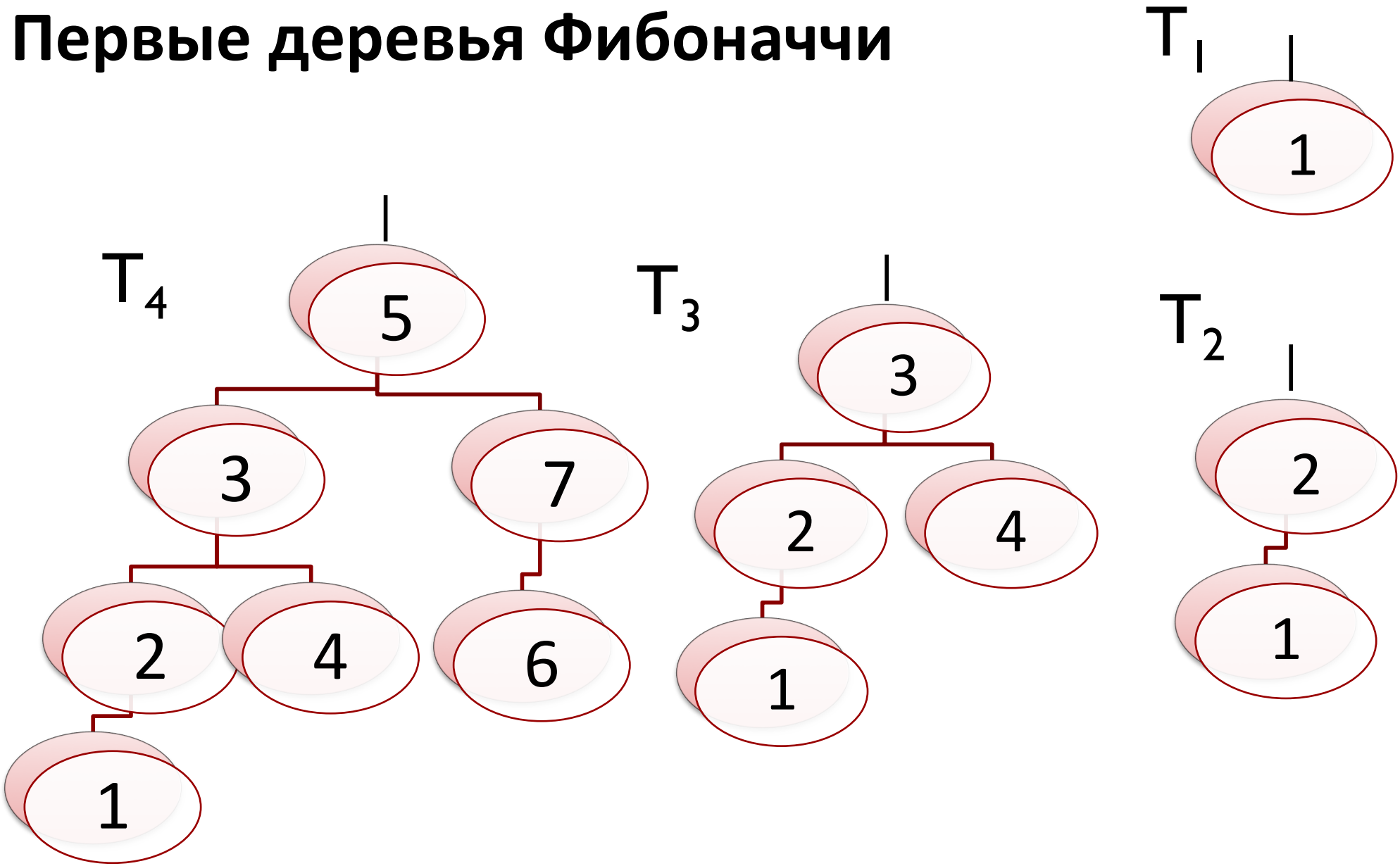
Дерево Фибоначчи

- T_0 – дерево: высота 0, вершин 0
- T_1 – дерево: высота 1 , вершин 1
- T_h – дерево: = < T_{h-1}, x, T_{h-2} >, высота h, вершин $n_h=n_{h-1}+1+n_{h-2}$

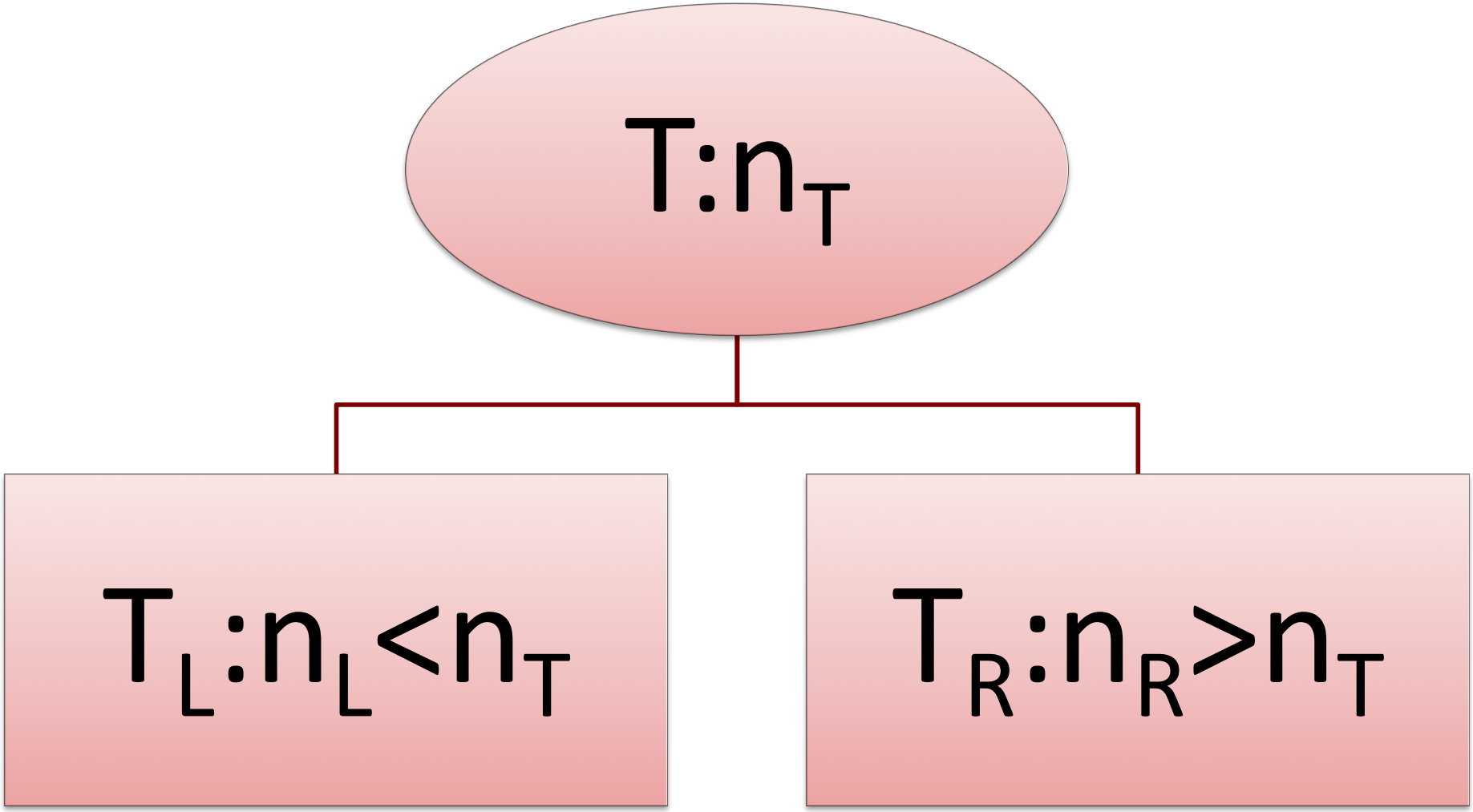
Числа Леонардо



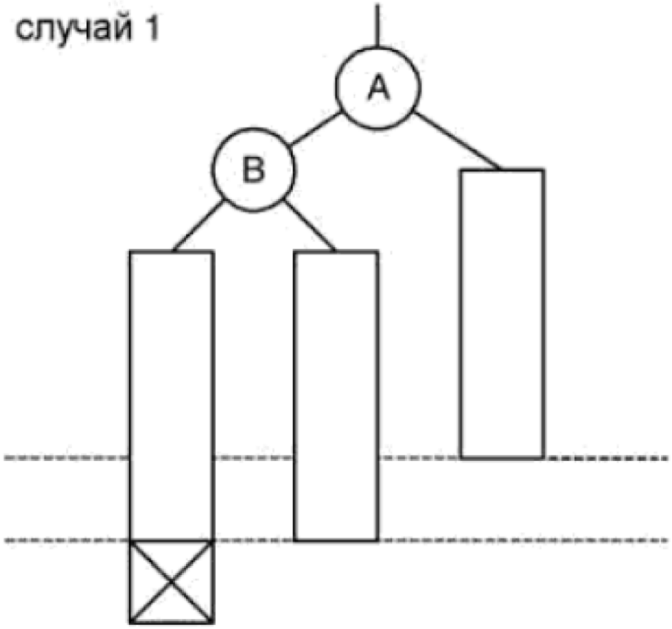
Первые деревья Фибоначчи



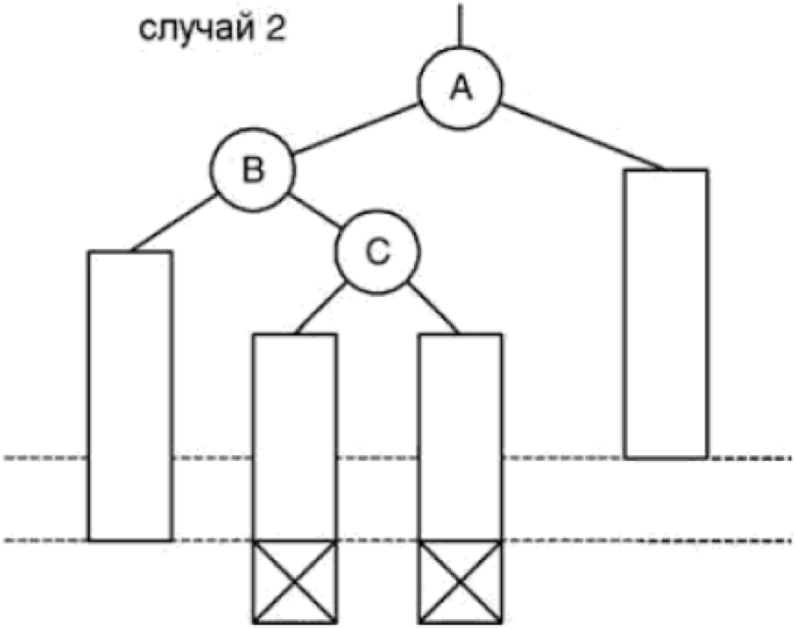
Вставка в АВЛ дерево ключа $k<n_T$



случай 1

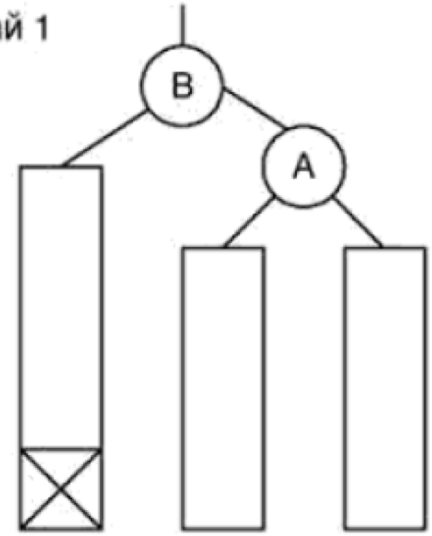


случай 2

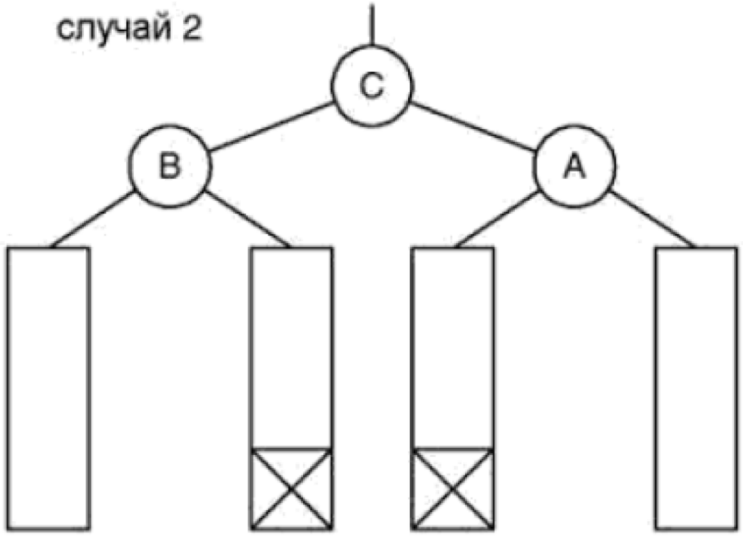


Нарушение баланса в результате вставки

случай 1



случай 2



Восстановление баланса

Идея алгоритма

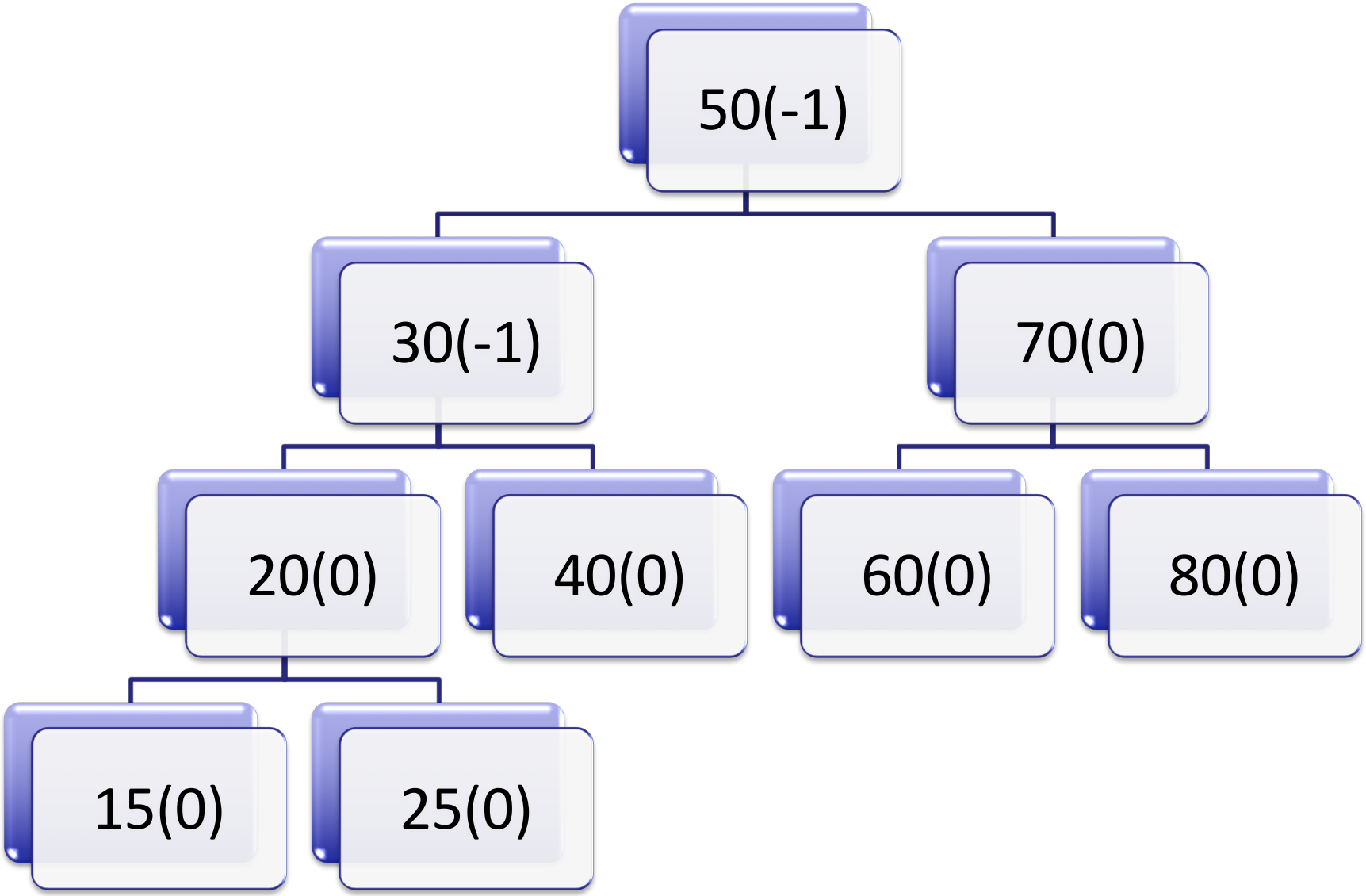
- В узле храним
  - ключ, баланс, указатели: левый, правый, A, B, C
- Вставка
  1. пройти по пути поиска и проверить, что данного ключа в дереве еще нет;
  2. вставить новый узел и определить полученный баланс;
  3. вернуться по пути поиска и, при необходимости, выполнять балансировку
  4. если «высота поддерева со вставкой увеличилась», то варианты
     1. дисбаланс исправляется вставкой
     2. левое поддерево на 1 выше
     3. балансировка (левое поддерево на 2 выше)
        1. если левое подподдерево выше, то случай 1 (A-B ротация указателей)
        2. иначе случай 2 (A-B-C ротация)

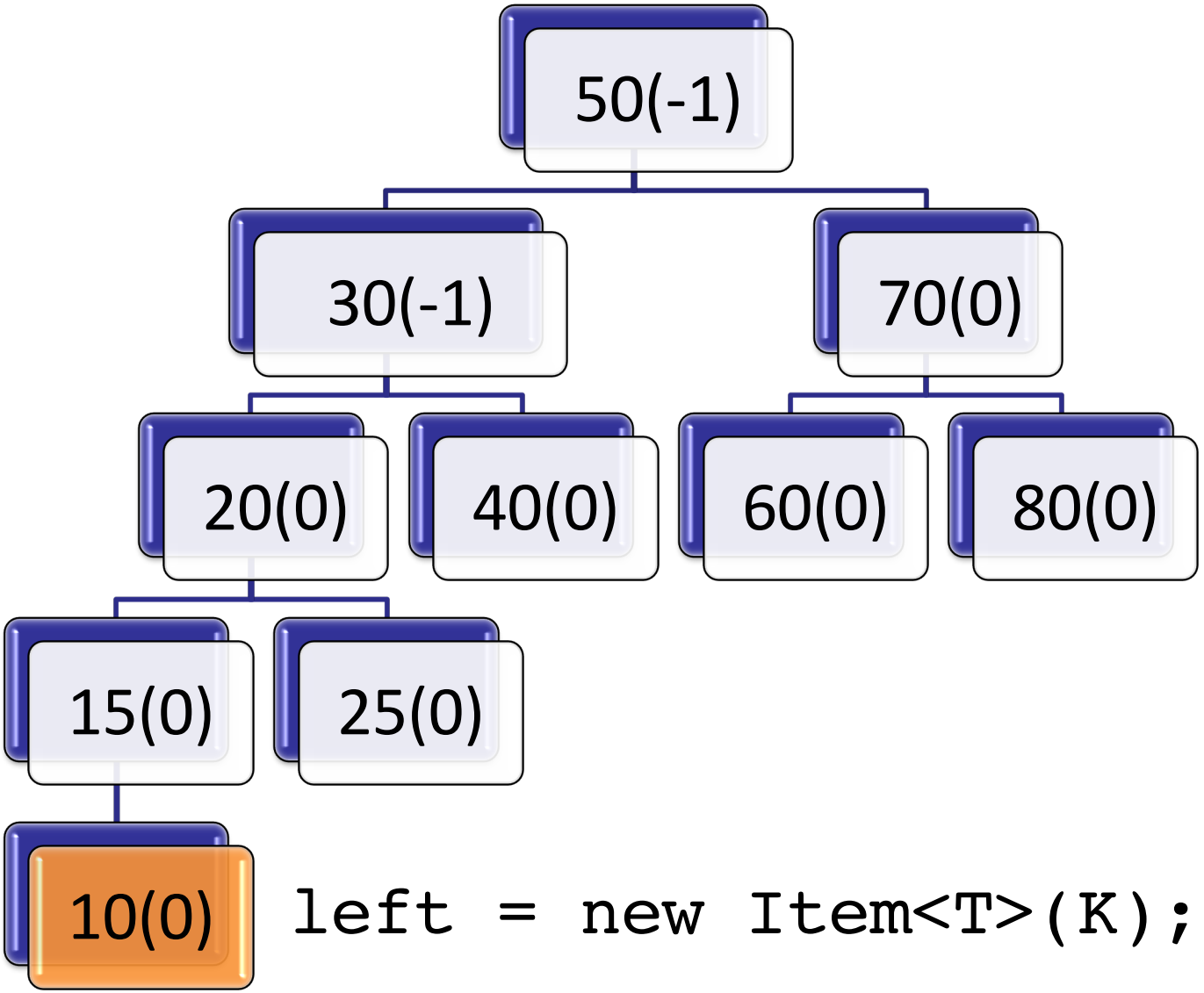
```
#include <iostream>
using namespace std;
#include <windows.h>

HANDLE hStdOut = GetStdHandle(STD_OUTPUT_HANDLE);

#include "Tree.h"

int main() {
int i,z=0;
Tree<int> a;
    while(cin >> i) { a.ins(i);
    switch(z++%3) {
    case 0:
    SetConsoleTextAttribute(hStdOut,FOREGROUND_RED | FOREGROUND_INTENSITY); break;
    case 1:
    SetConsoleTextAttribute(hStdOut, FOREGROUND_GREEN | FOREGROUND_INTENSITY); break;
    case 2:
    SetConsoleTextAttribute(hStdOut, FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_INTENSITY); break; }
    cout << a;
    SetConsoleTextAttribute(hStdOut,FOREGROUND_RED | FOREGROUND_GREEN | FOREGROUND_BLUE | FOREGROUND_INTENSITY);
}
    return 0;
}
```

```
tree.cpp
template<typename T> class Item {
    T key; short b; Item<T> *left; Item<T> *right;
    Item<T> *A,*B,*C;
public:
    Item(const T& n=0, Item* l=0, Item* r=0, short bal=0) { key = n; b=bal; left = l; right = r; }
T&  get_key() { return key; }
short&  get_b() { return b; }
Item<T>*&   get_left() { return left; }
Item<T>*&   get_right() { return right; }
void put_left(const T& k) { delete left; left = new Item<T>(k); }
void put_right(const T& k) { delete right; right = new Item<T>(k); }
void ins(const T& K,bool& h); };
template<typename T> void Item<T>::ins(const T& K,bool& h) {
    if(key == K) { h=false; return; }
    if(K < key)  { if(left) left->ins(K,h); else { left = new Item<T>(K); h=true; }
            if(h) switch (b) {
                case 1: h=false;
                case 0: b--; break;
                case -1: A = left;
                    if( A->b == -1) { B=new Item<T>; *B=*this; B->b=0; right=B; key=A->key;
                            B->left = A->right; left = A->left; delete A; }
                    else { C = new Item<T>; *C = *this; B = A->right; A->right = B->left;
                        right = C; key=B->key; C->left = B->right;
                        C->b = (B->b == -1)?1:0; A->b = (B->b == 1)?-1:0; delete B;}
                    b = 0; h = false;
                }       // switch
            }           // K < key
```

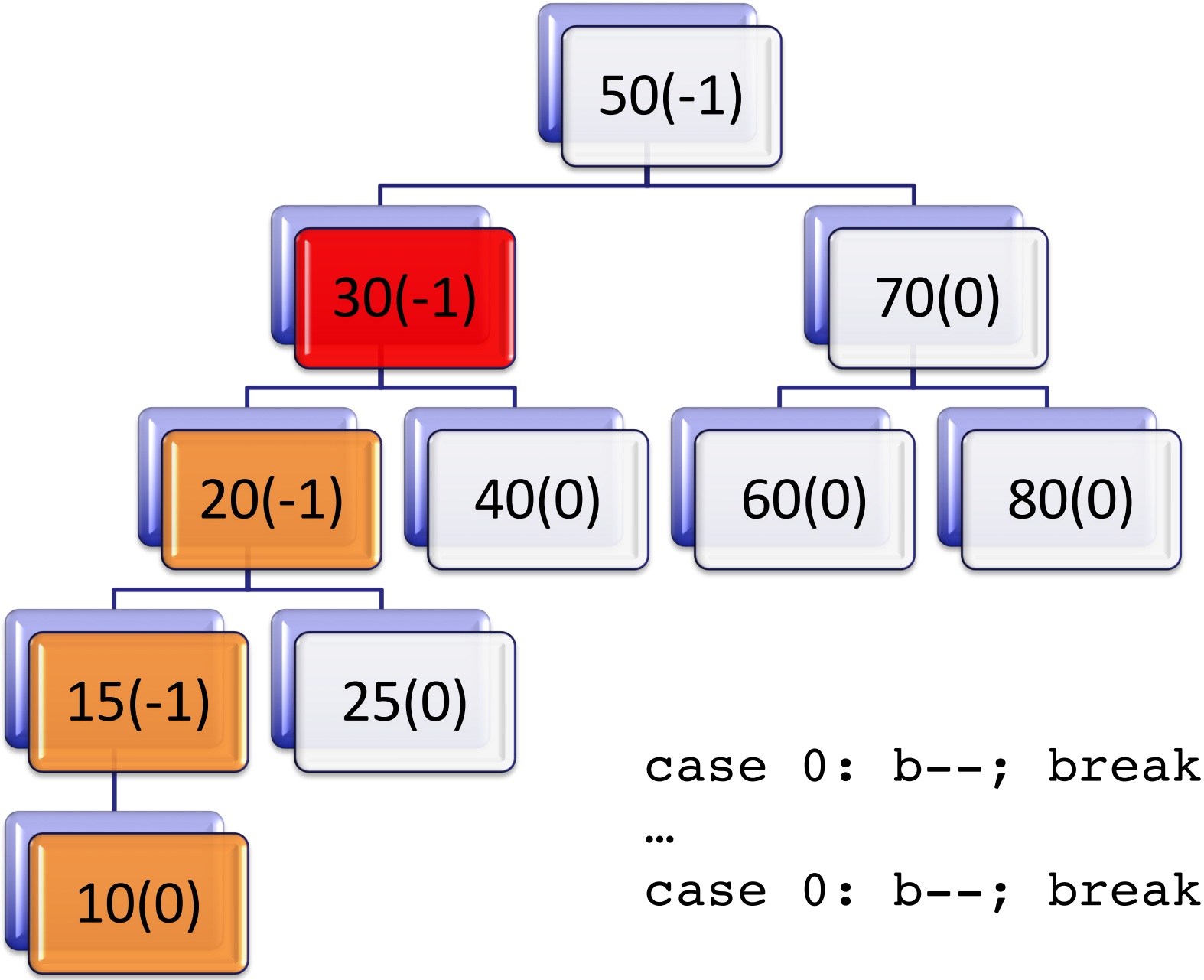
```
    else          { if(right) right->ins(K,h); else { right = new Item<T>(K); h=true; }
            if(h) switch (b) {
                case -1: h=false;
                case 0: b++; break;
                case 1: A = right;
                    if( A->b == 1) { B = new Item<T>; *B = *this; B->b=0; left = B; key = A->key;
                            B->right = A->left; right = A->right; delete A; }
                    else {  C = new Item<T>; *C = *this; B = A->left; A->left = B->right;
                        left = C; key = B->key; C->right = B->left;
                        C->b = (B->b == 1)?-1:0;
                        A->b = (B->b == -1)?1:0; delete B; }
                b = 0; h = false;
                }       // end switch
            }           // end K > key
}

template<typename T> class Tree;
template<typename T> ostream& operator<<(ostream& a, const Tree<T>& b);
template<typename T> class Tree {
    Item<T> *top;
public:
Tree(Item<T>* I=NULL) { top = I; }
void insert(const T& k);
Item<T>* insert(T a[], int n);
void ins(const T& k) {bool h=false; if(top) top->ins(k,h); else top=new Item<T>(k); }
friend ostream& operator<<(ostream& a,const Tree<T>& b);
};
```

```
template<typename T> void Tree<T>::insert(const T& k) { if(top==NULL) { top=new Item<T>(k); return; }
Item<T> *p=top;
for(;;)  if( p->get_key() < k) if(p->get_right()) p=p->get_right(); else { p->put_right(k); break; }
         else  if(p->get_left()) p=p->get_left(); else { p->put_left(k); break; } }

template<typename T> Item<T>* Tree<T>::insert(T a[], int n) { int nl=n/2, nr=n-nl-1;
         if(n) insert(a[n/2]); else return NULL;
         top->get_left() = Tree(top->get_left()).insert(a,nl);
         top->get_right() = Tree(top->get_right()).insert(&a[n/2+1],nr);
         return top; }

template<typename T> ostream& operator<<(ostream& a, const Tree<T>& b) {
static int k=0; if(b.top==NULL) return a;
    k++; if(b.top->get_right()!=NULL) a << Tree<T>(b.top->get_right()); k--;
    for(int i=0;i<k;i++) a << '\t'; a << b.top->get_key();
    short s=b.top->get_b(); if(s) cout << '(' << s << ")\n"; else cout << endl;
    k++; if(b.top->get_left()!=NULL) a << Tree<T>(b.top->get_left()); k--;
    return a; }
```

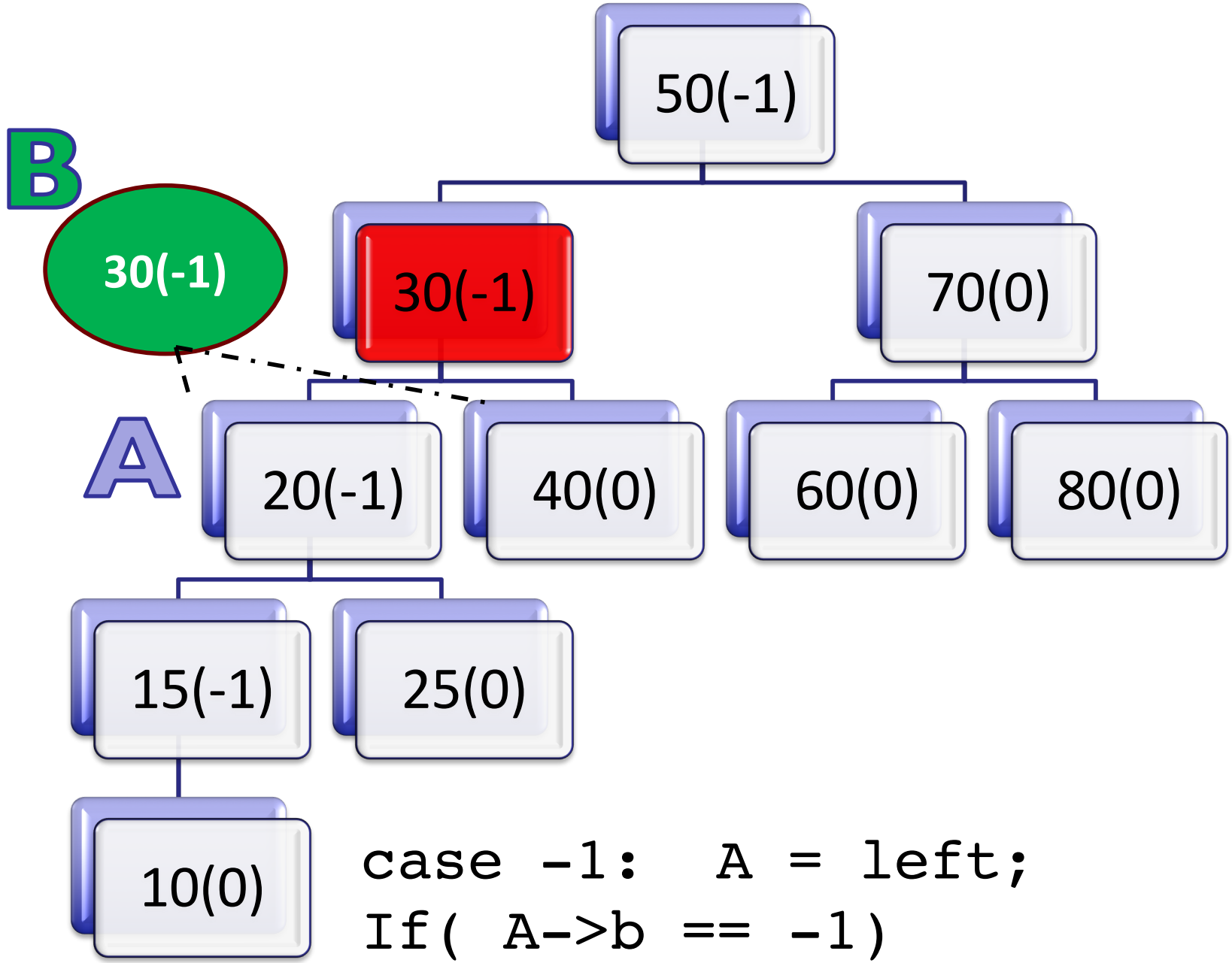

50(-1)
30(-1)
70(0)
20(0)
40(0)
60(0)
80(0)
15(0)
25(0)



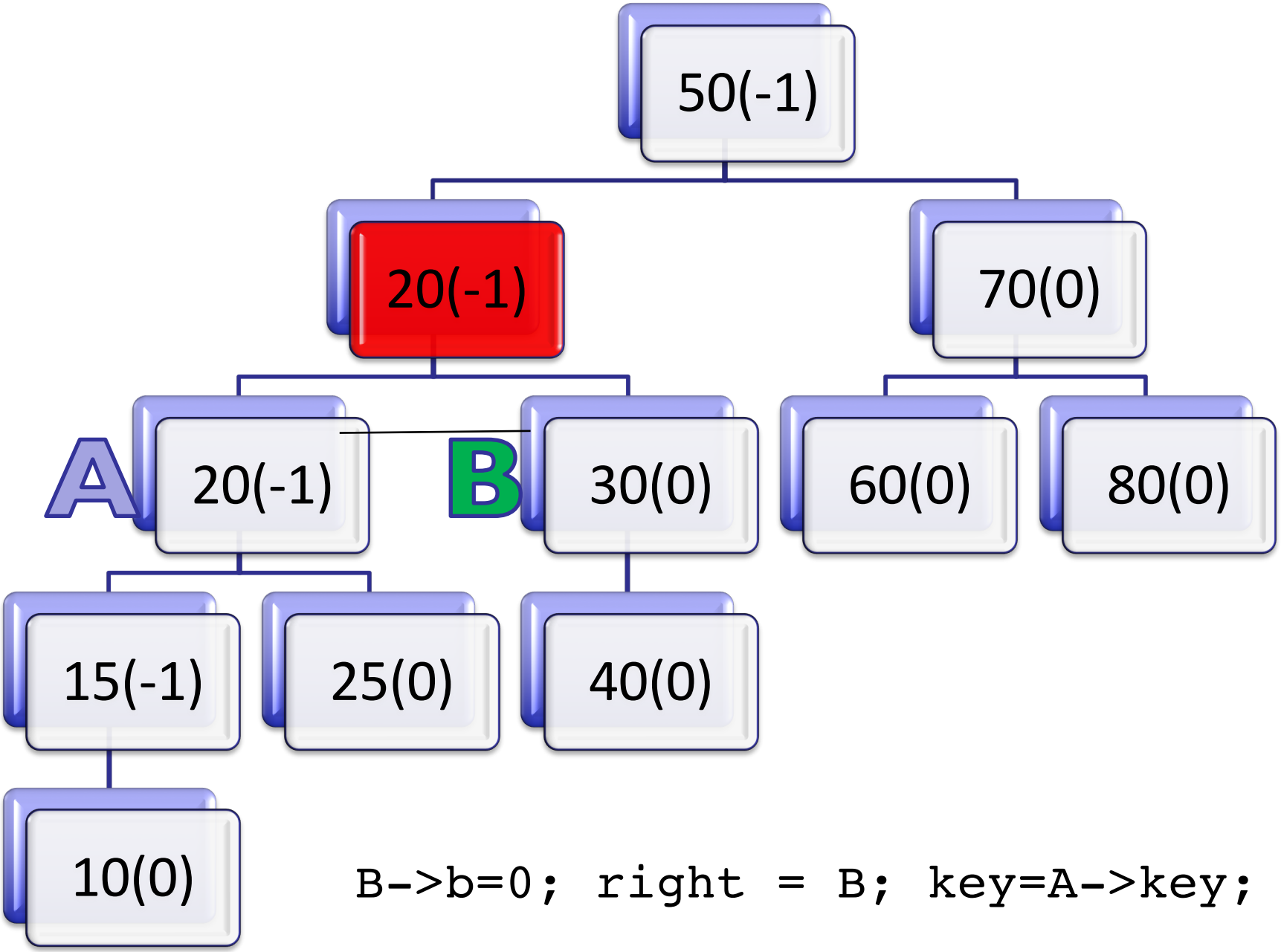

```
left = new Item<T>(K); h++;
```



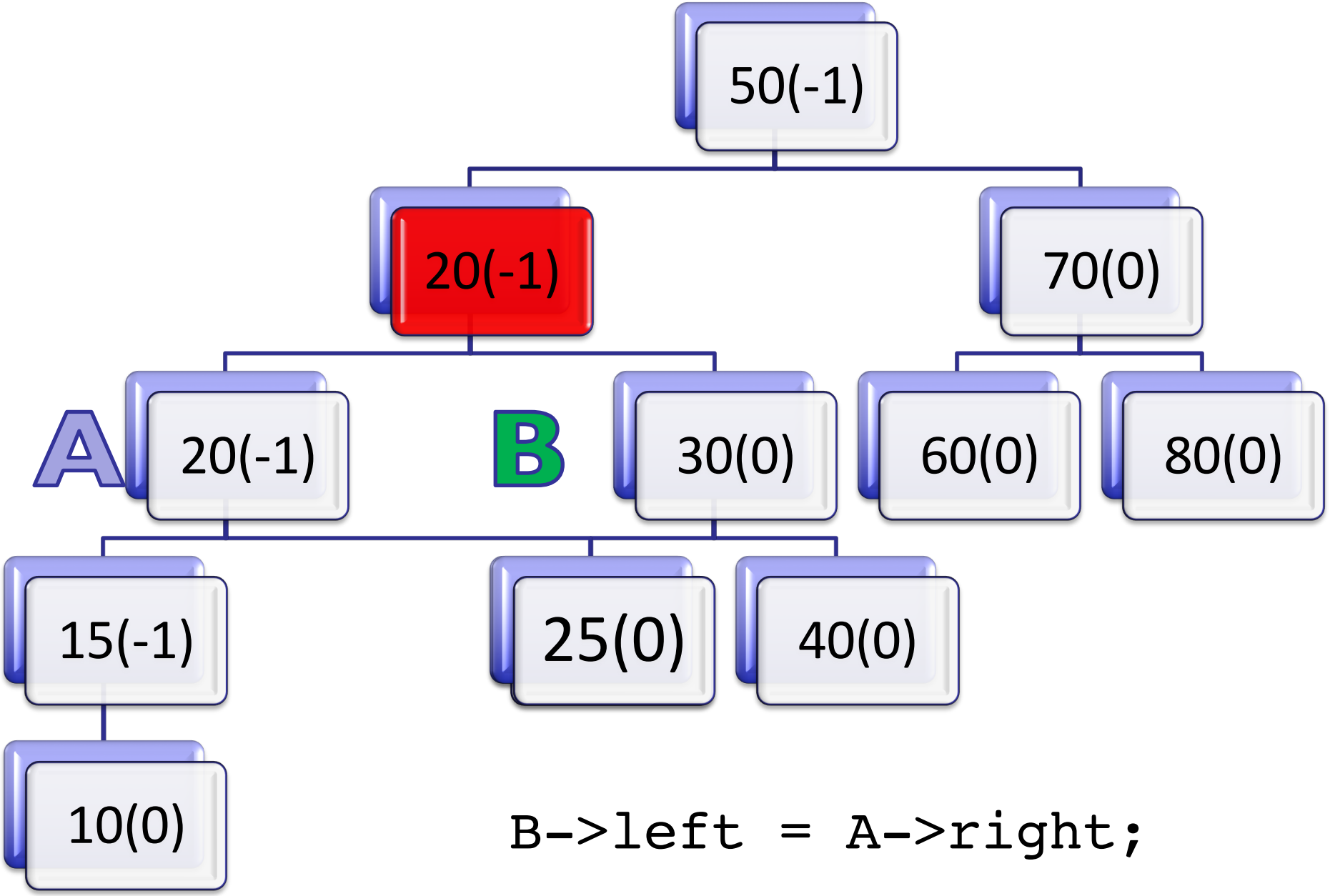

```
case 0: b--; break;
…
case 0: b--; break;
```



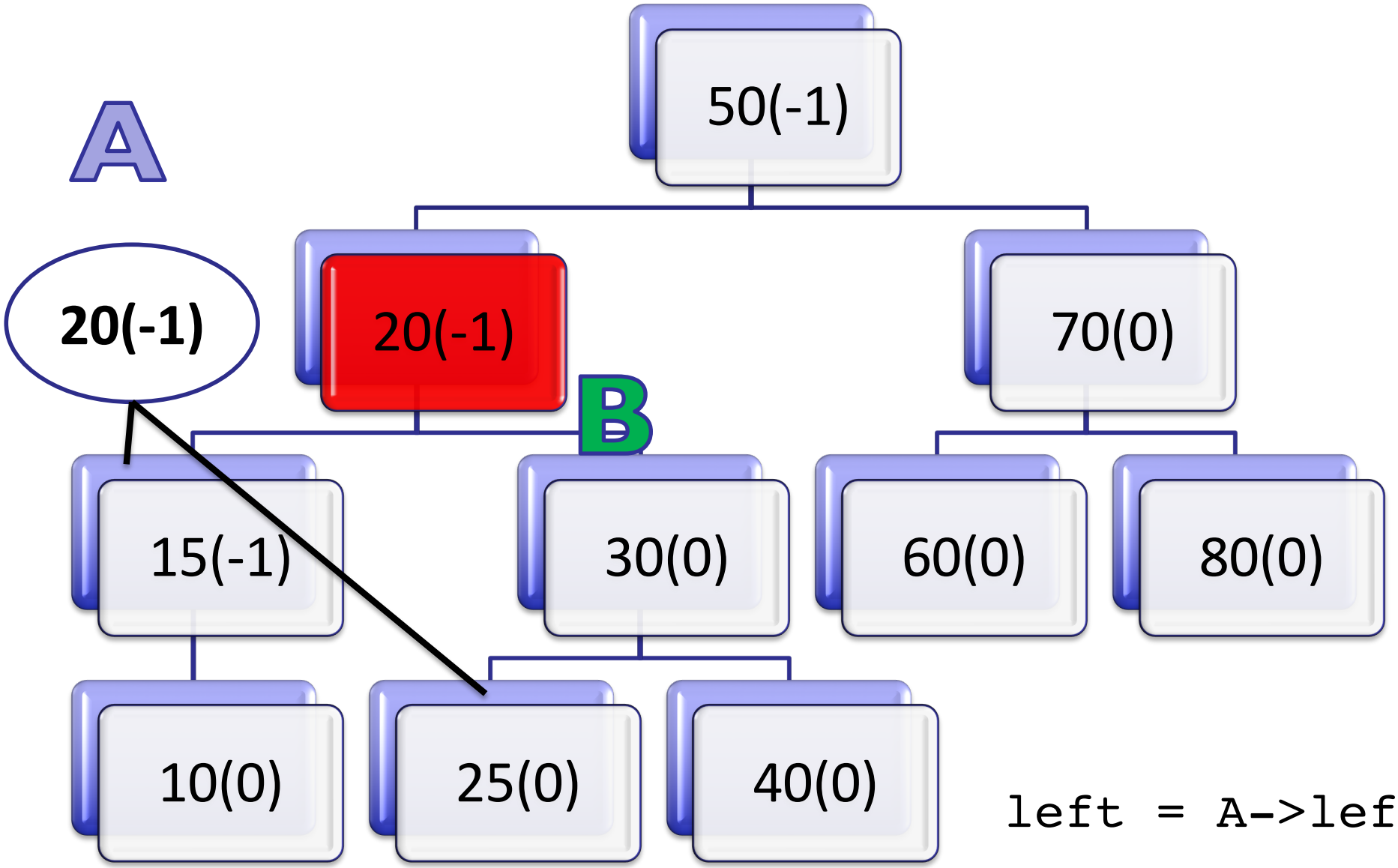

```
case -1:  A = left;
If( A->b == -1)
{ B = new Item<T>; *B=*this;
```



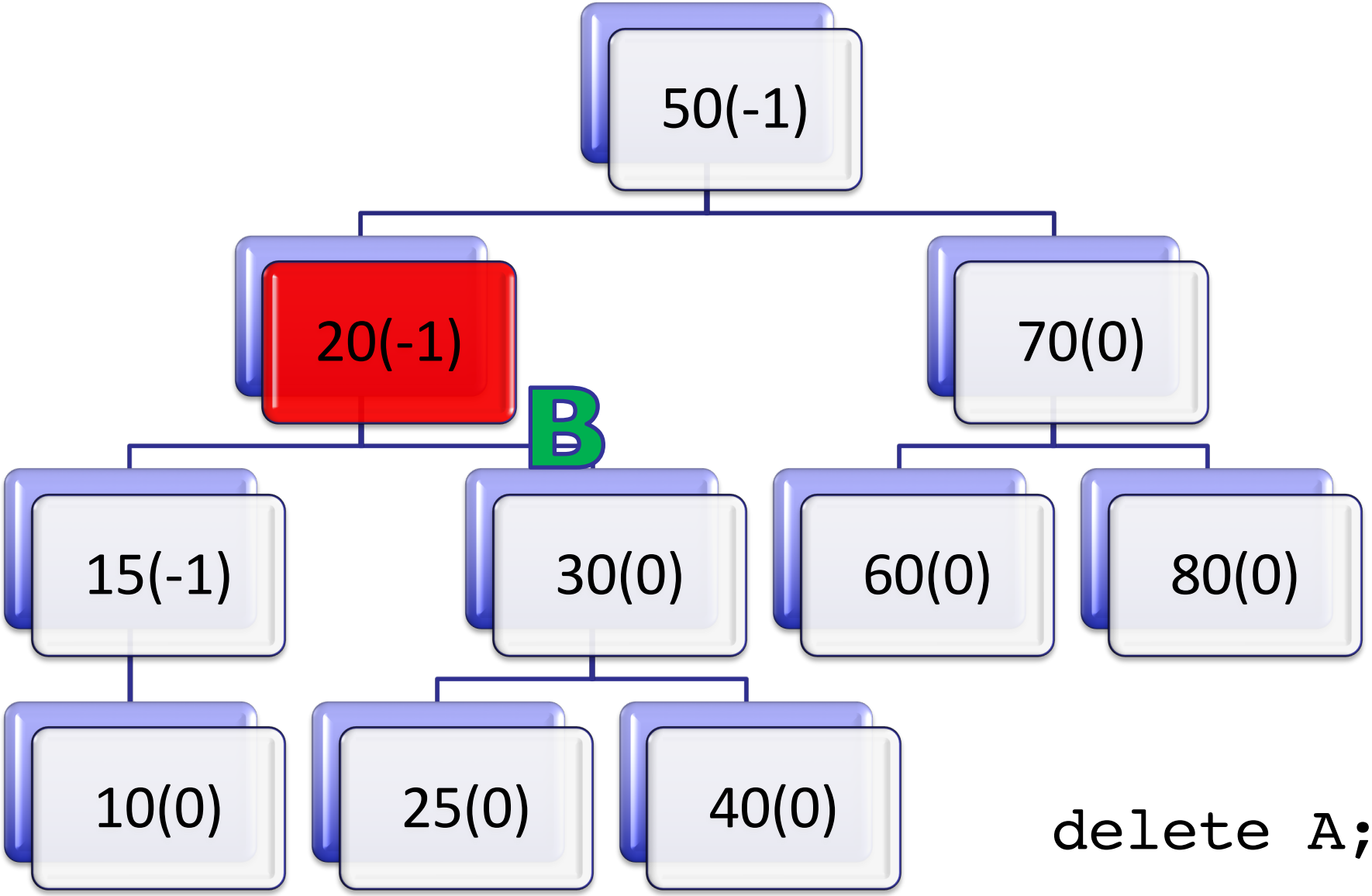

```
B->b=0; right = B; key=A->key;
```



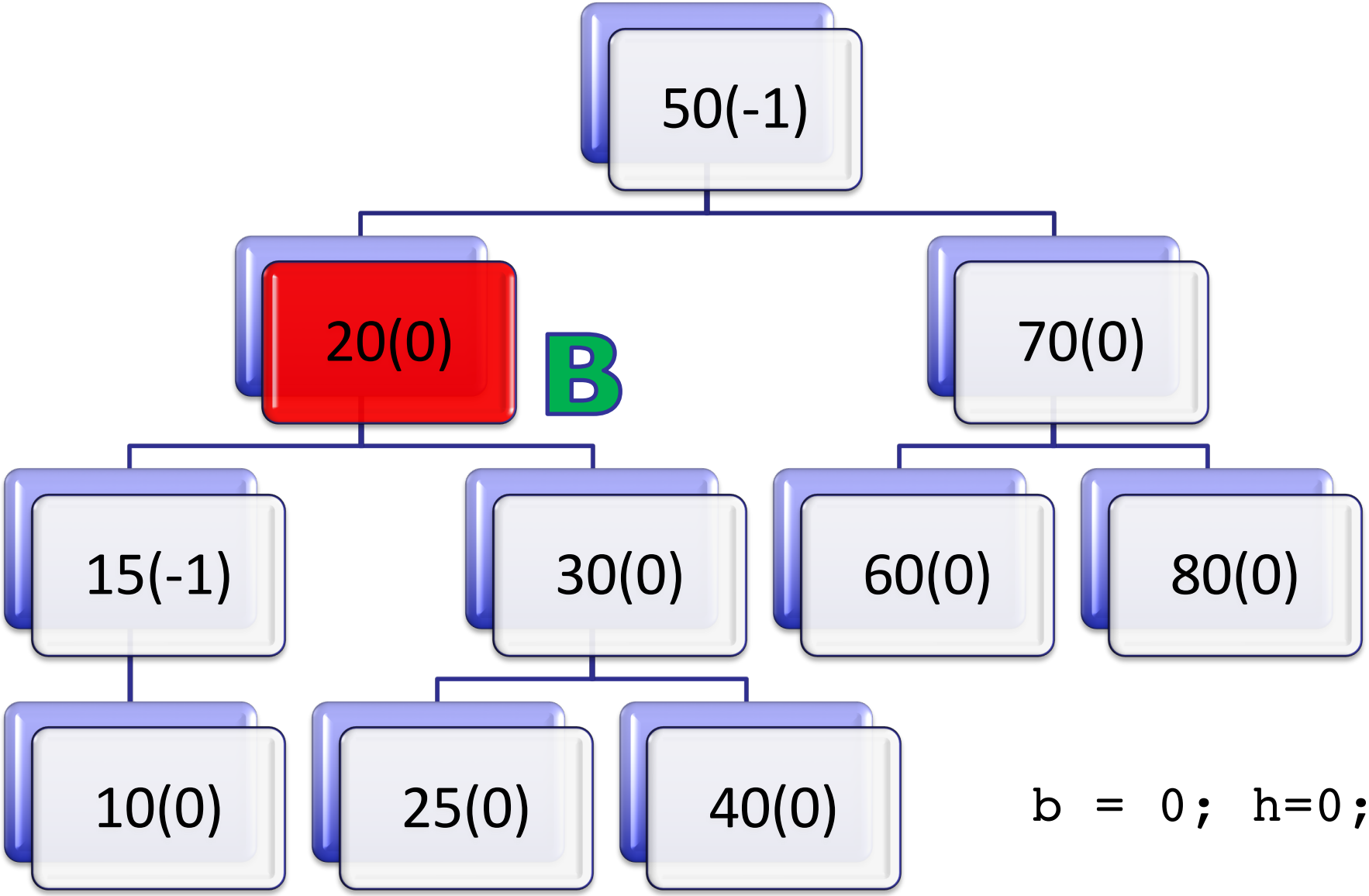

```
B->left = A->right;
```



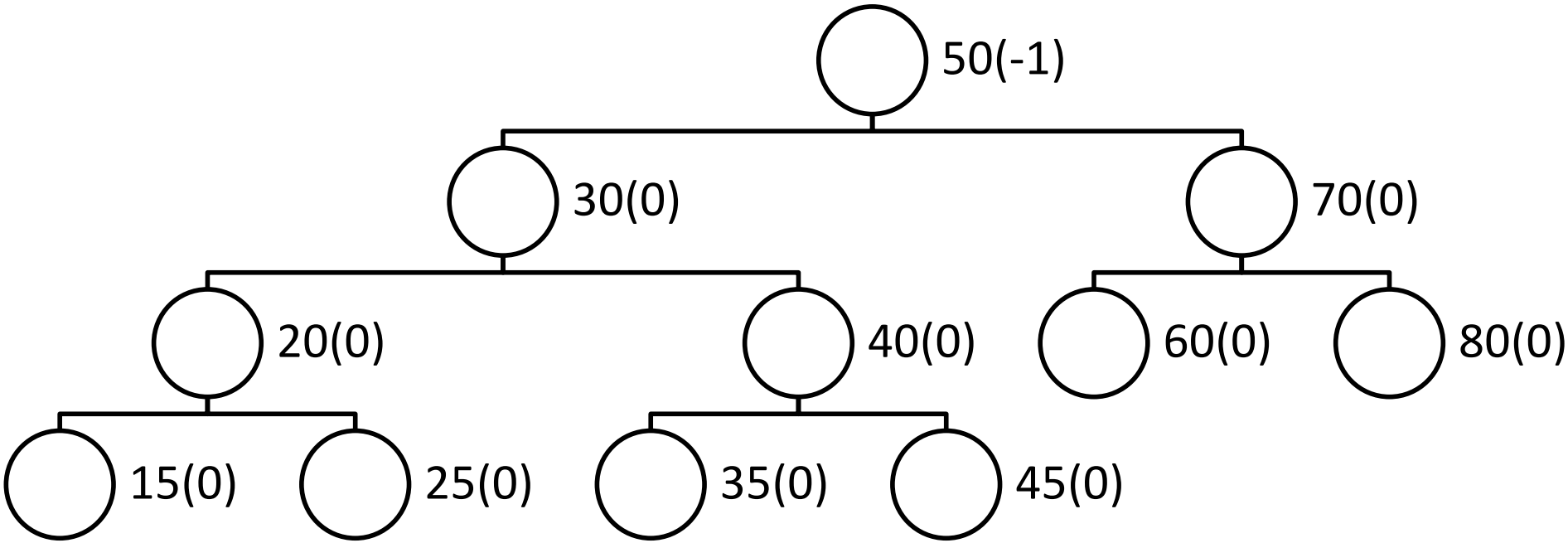

```
left = A->left;
```

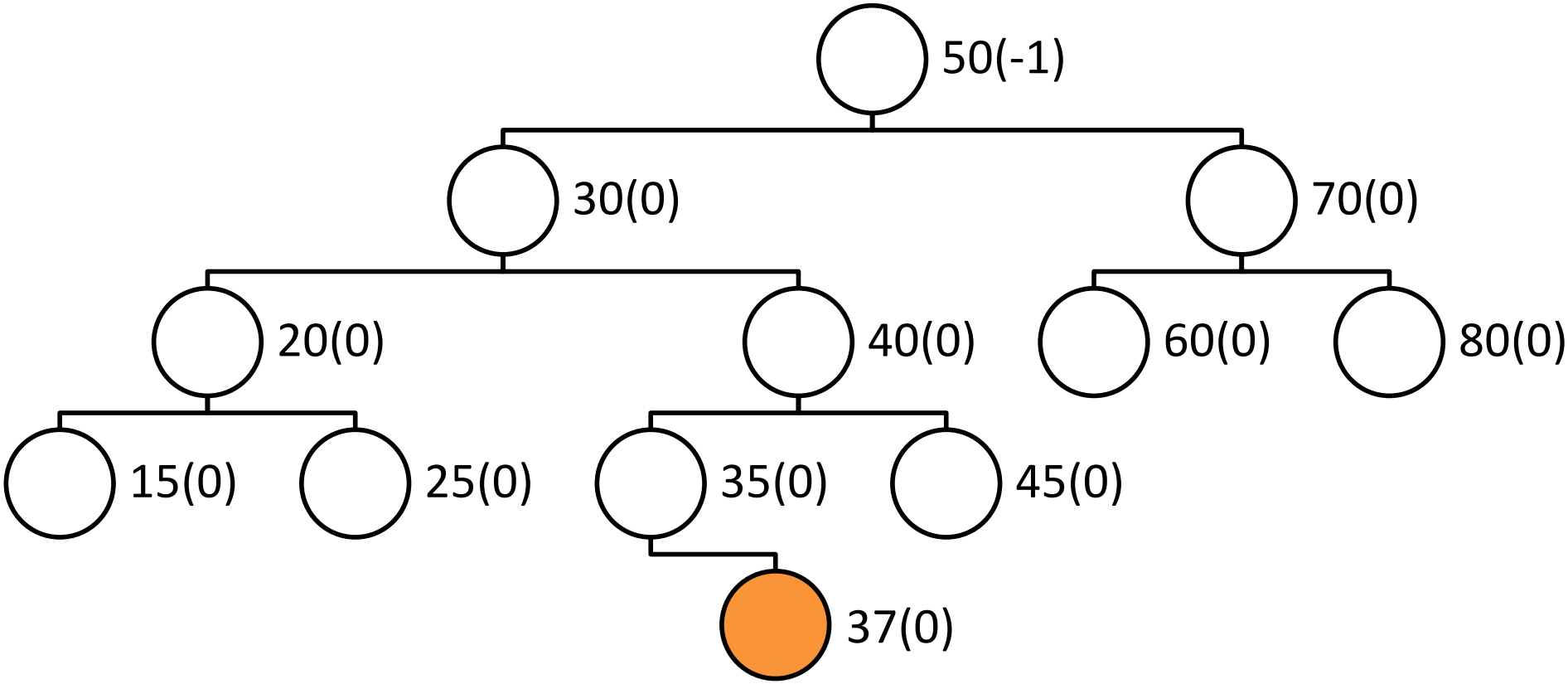



```
delete A;
```

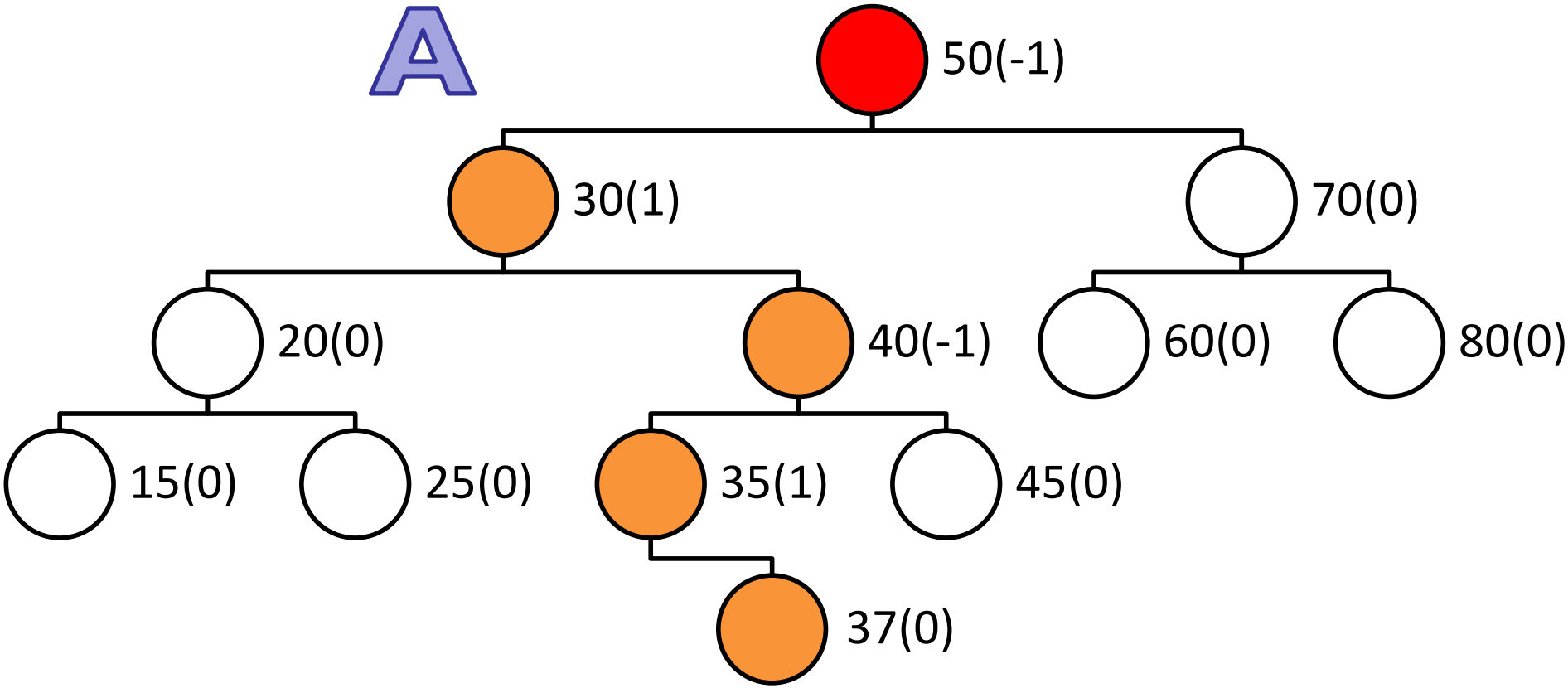



```
b = 0; h=0;
```

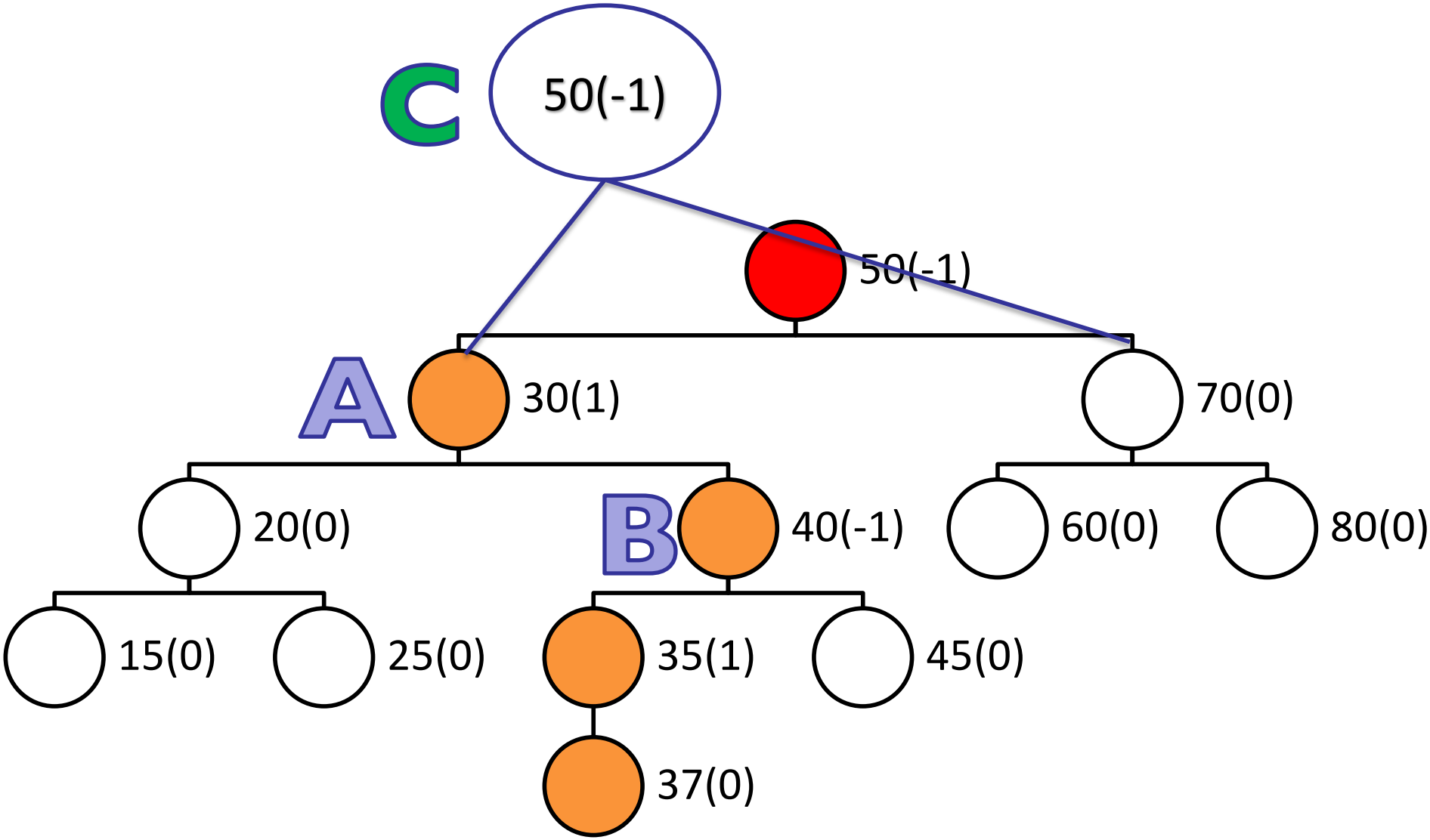

50(-1)
30(0)
70(0)
20(0)
40(0)
60(0)
80(0)
15(0)
25(0)
35(0)
45(0)



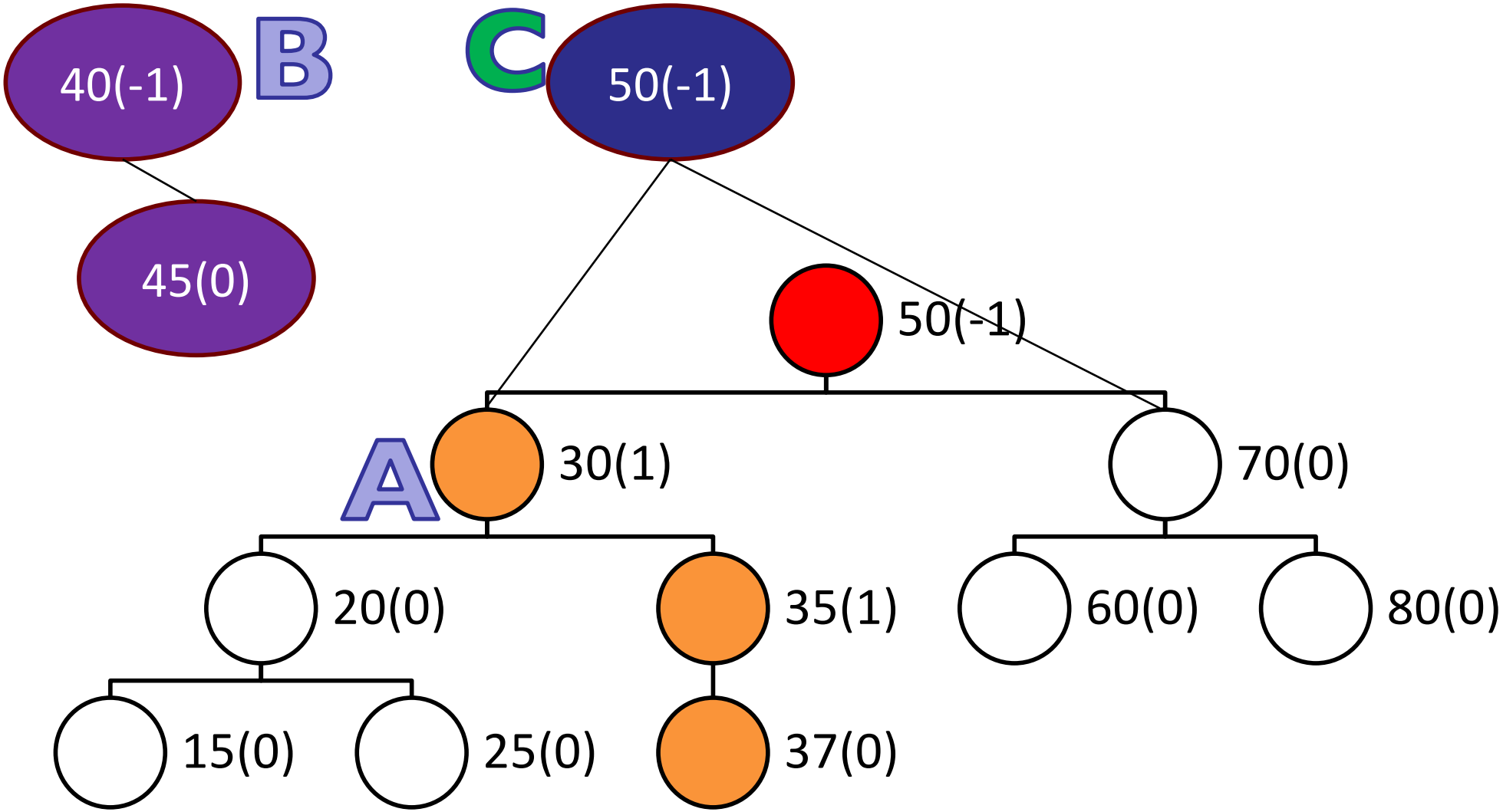

```
else { if(right) right->ins(K); else { right = new Item<T>(K); h++;}
```



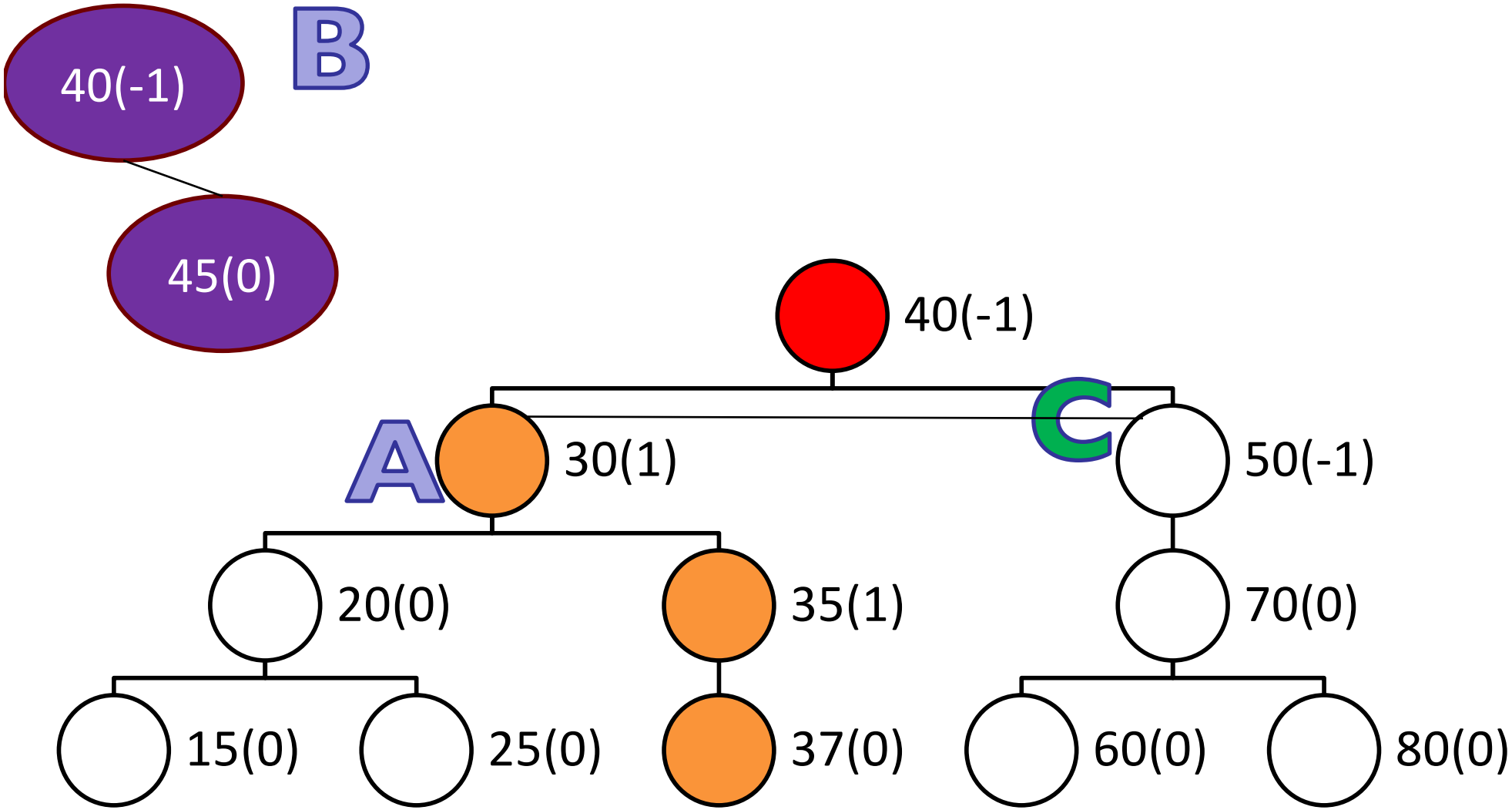

```
else          { if(right) right->ins(K);
…
if(K < key)   { if(left) left->ins(K);
```



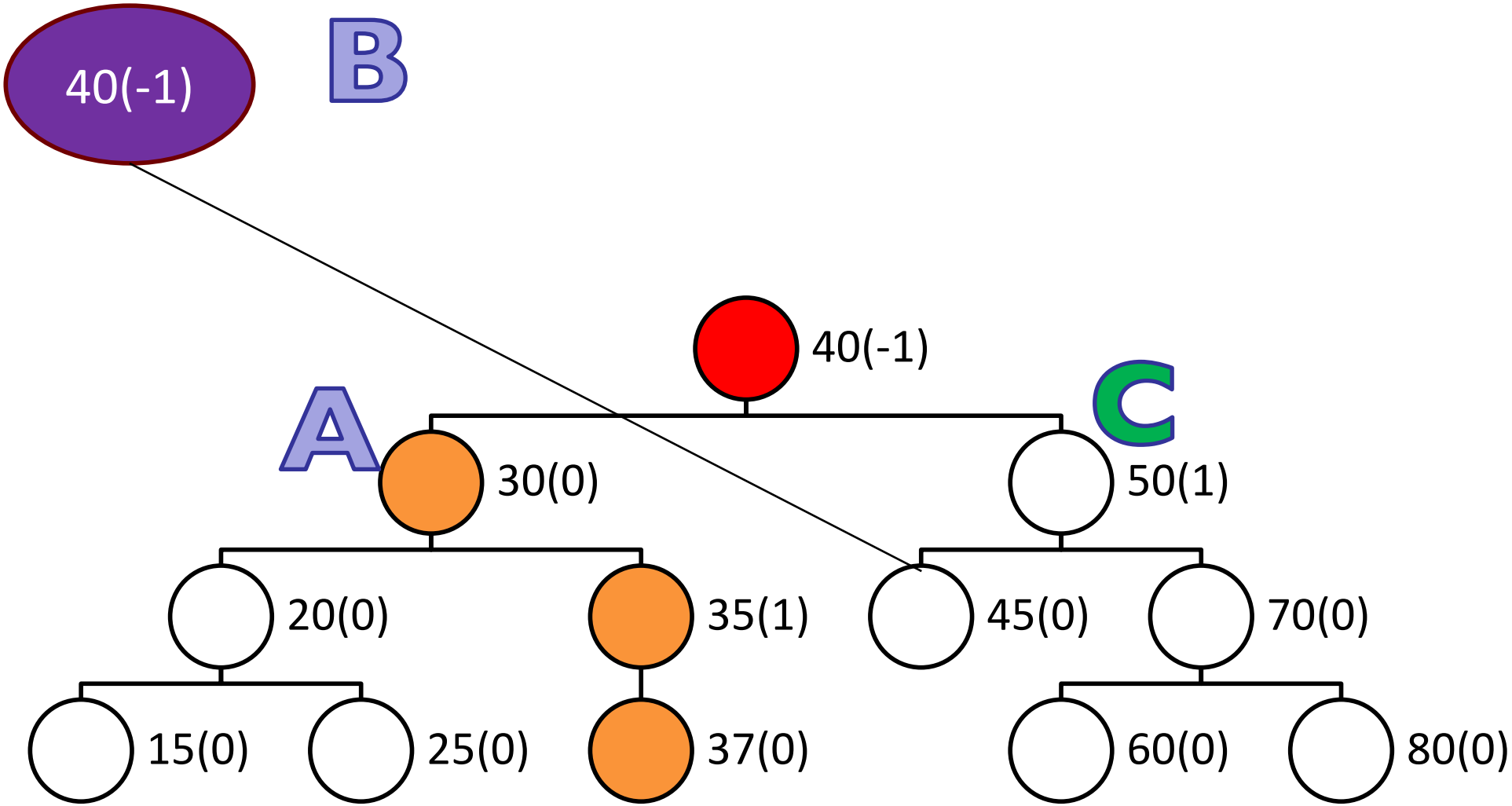

```
C = new Item<T>; *C = *this; B = A ->right;
```



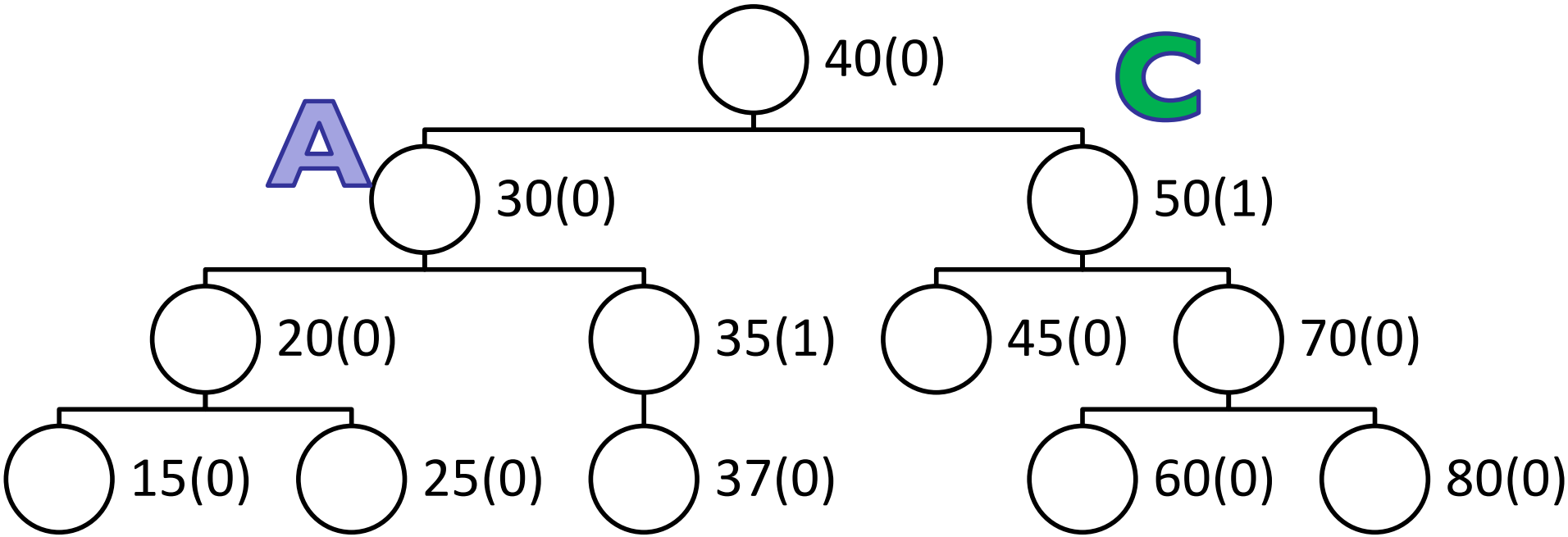

```
A ->right = B->left;
```




```
right = C; key = B->key;
```




```
C->left = B->right;
C->b = (B->b==-1)1:0);
A->b = (B->b==1)-1:0);
```

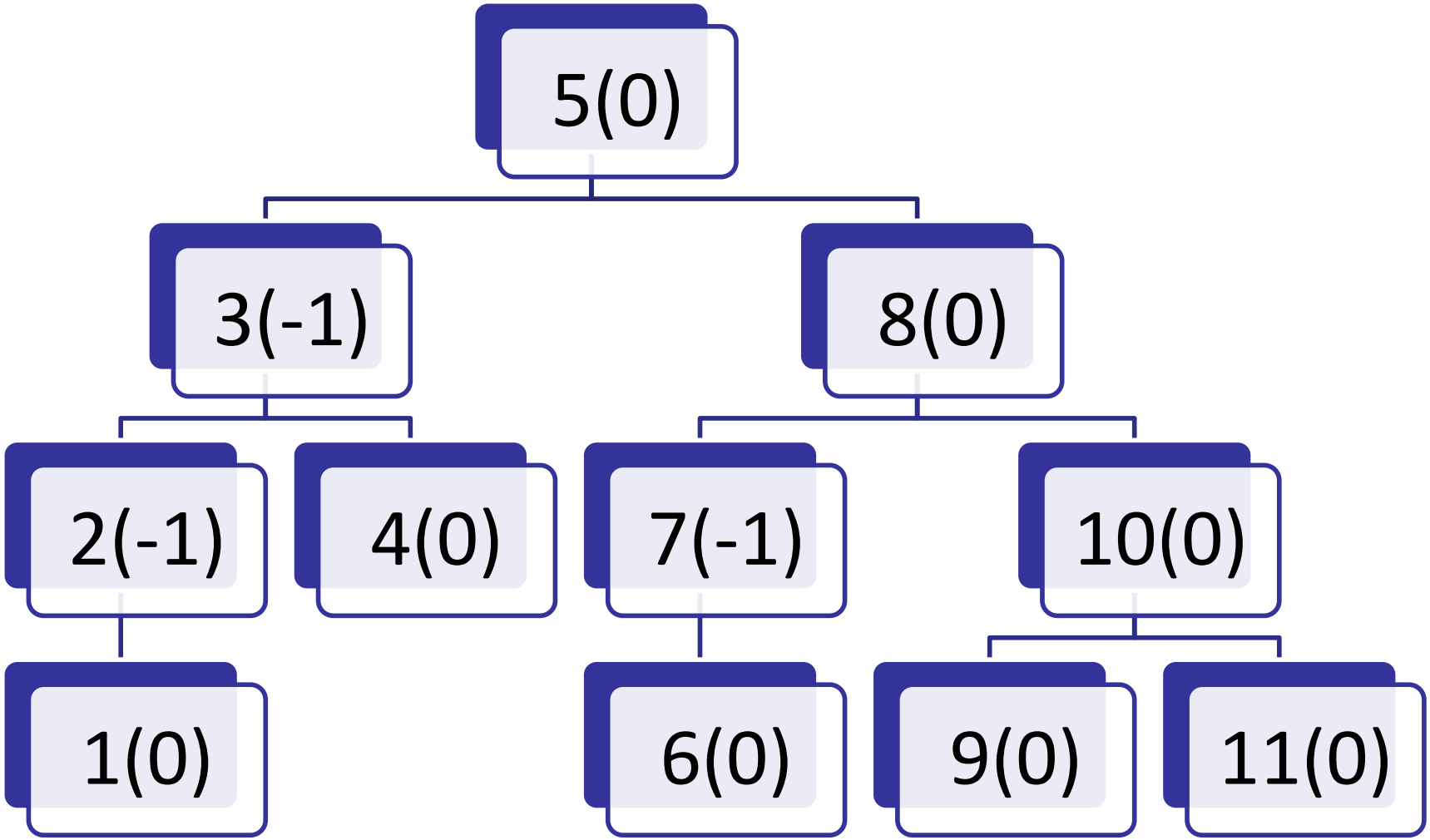



```
delete B; b=0; h=0;
```

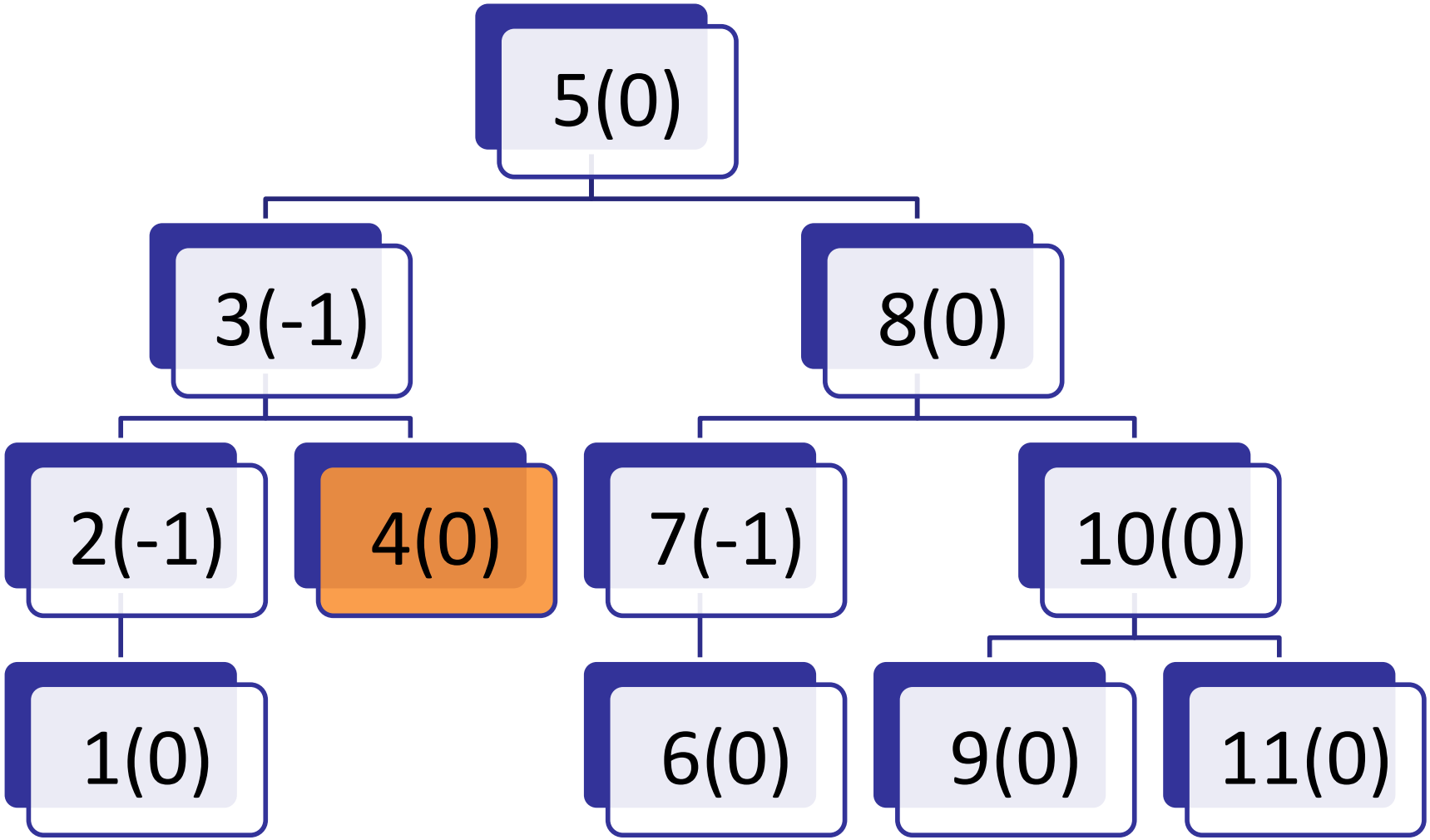
Эффективность ?

- Вопросы
  - Какова средняя высота АВЛ-деревьев $n!$ равновероятных перестановок n ключей?
  - Какова вероятность балансировки вставки?
- Ответы
  - Математического исследования нет, а на практике
    - $h = \log(n) + 0.25$
    - Вероятность балансировки 0.5
- Вывод
  - Трудно превзойти простейший алгоритм вставки в дерево
  - Эффективность балансировки - условна

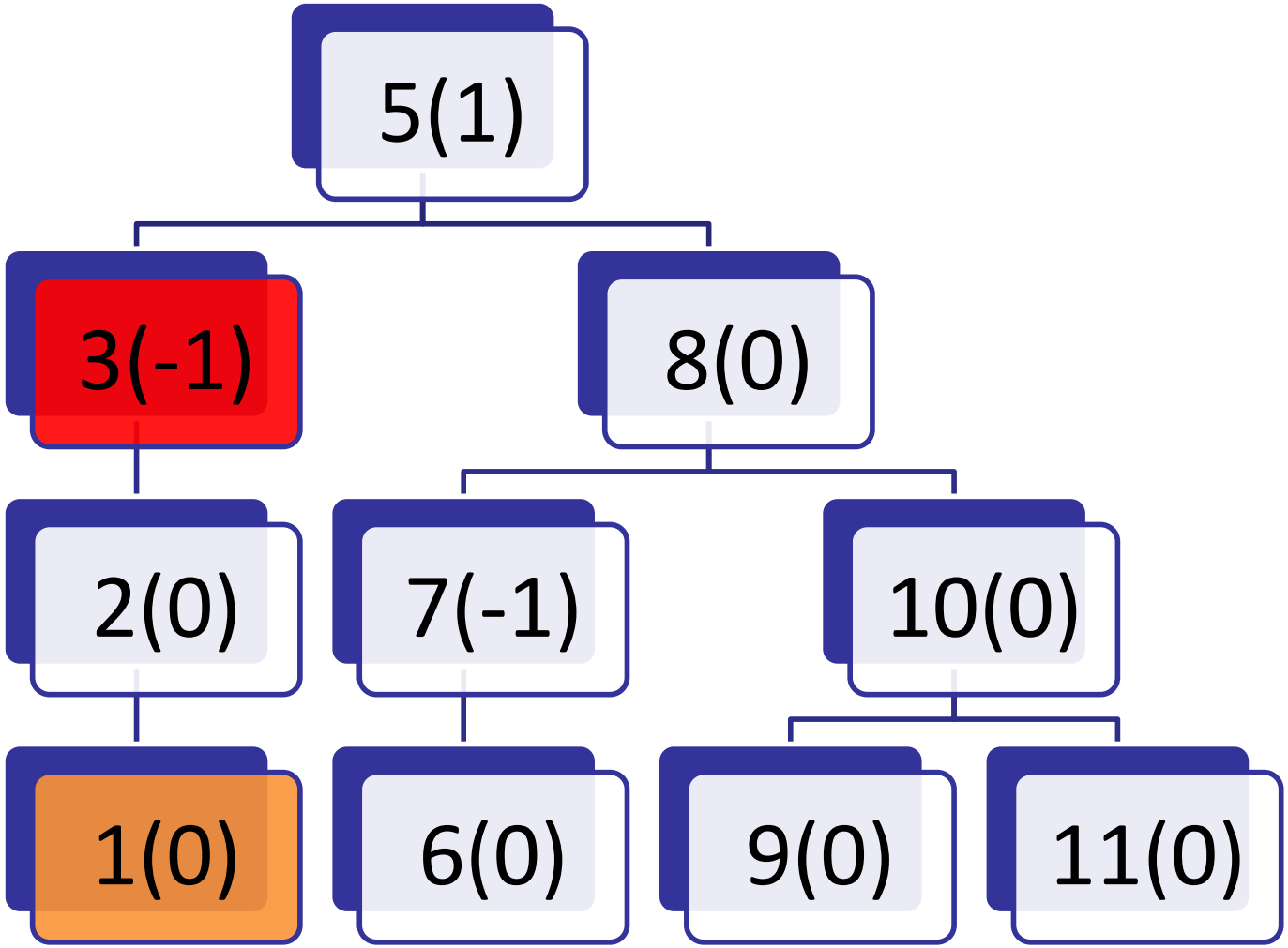
Удаление элемента из АВЛ дерева - 1



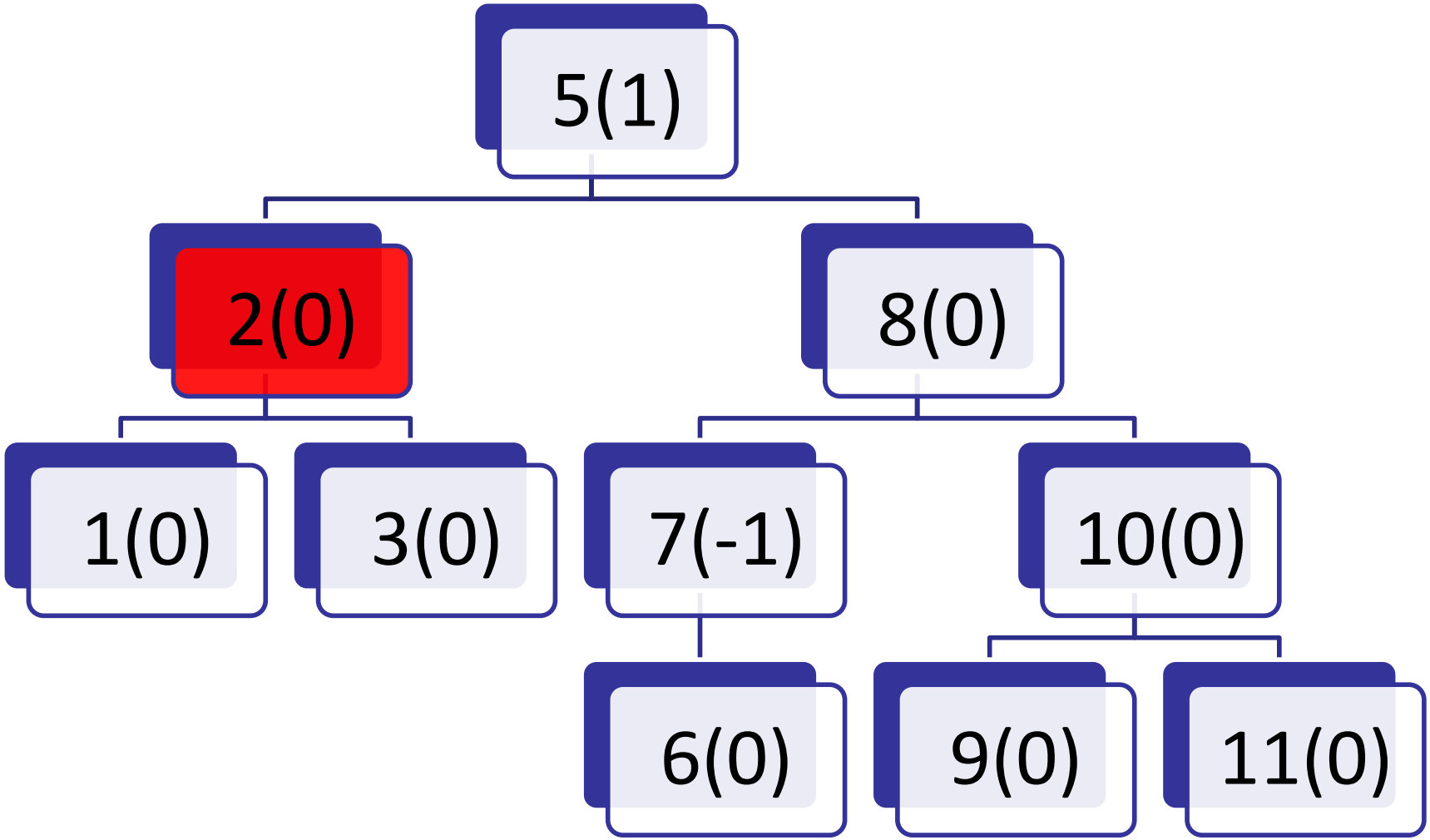
Удаление элемента из АВЛ дерева - 2



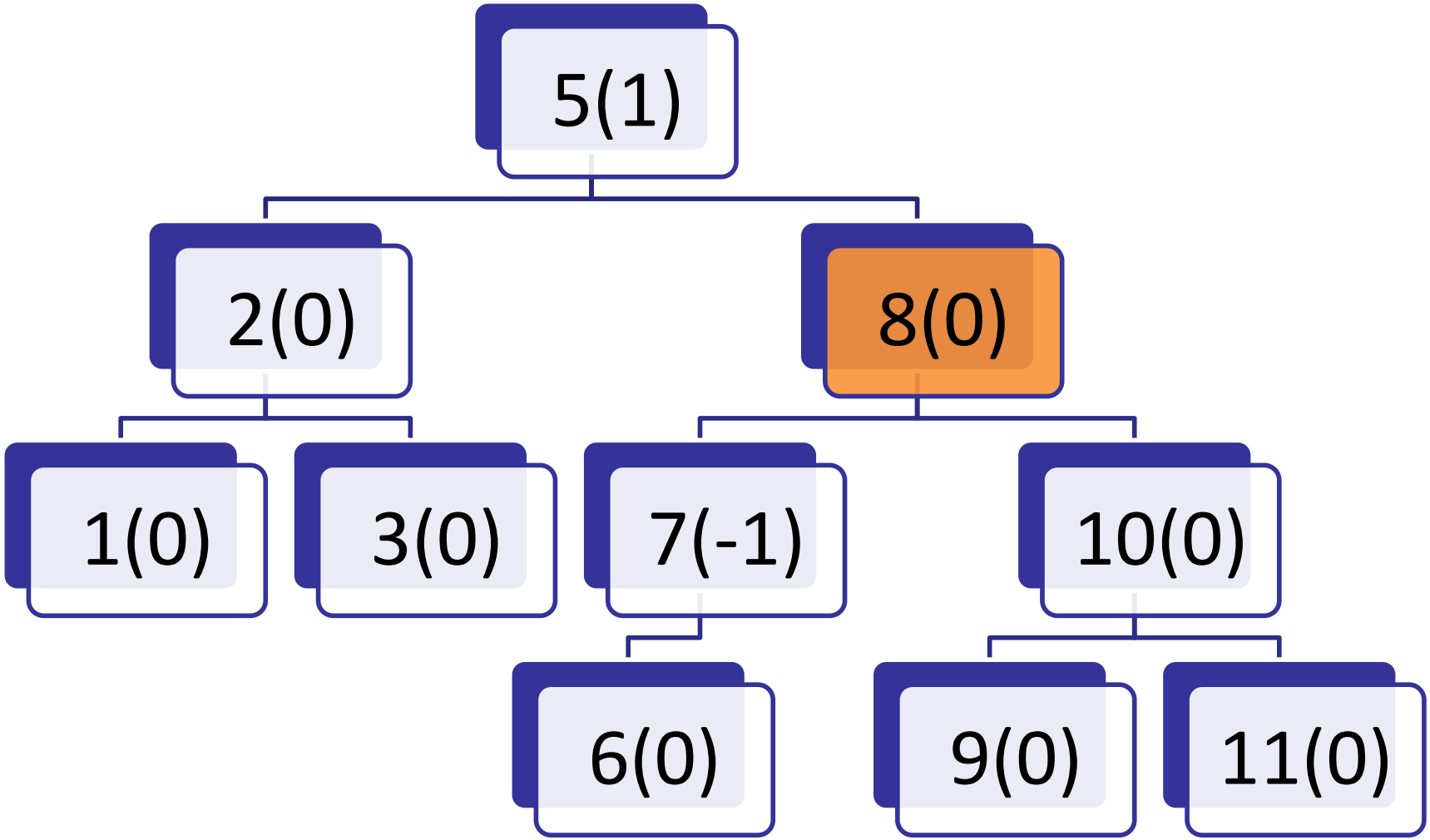
Удаление элемента из АВЛ дерева - 3



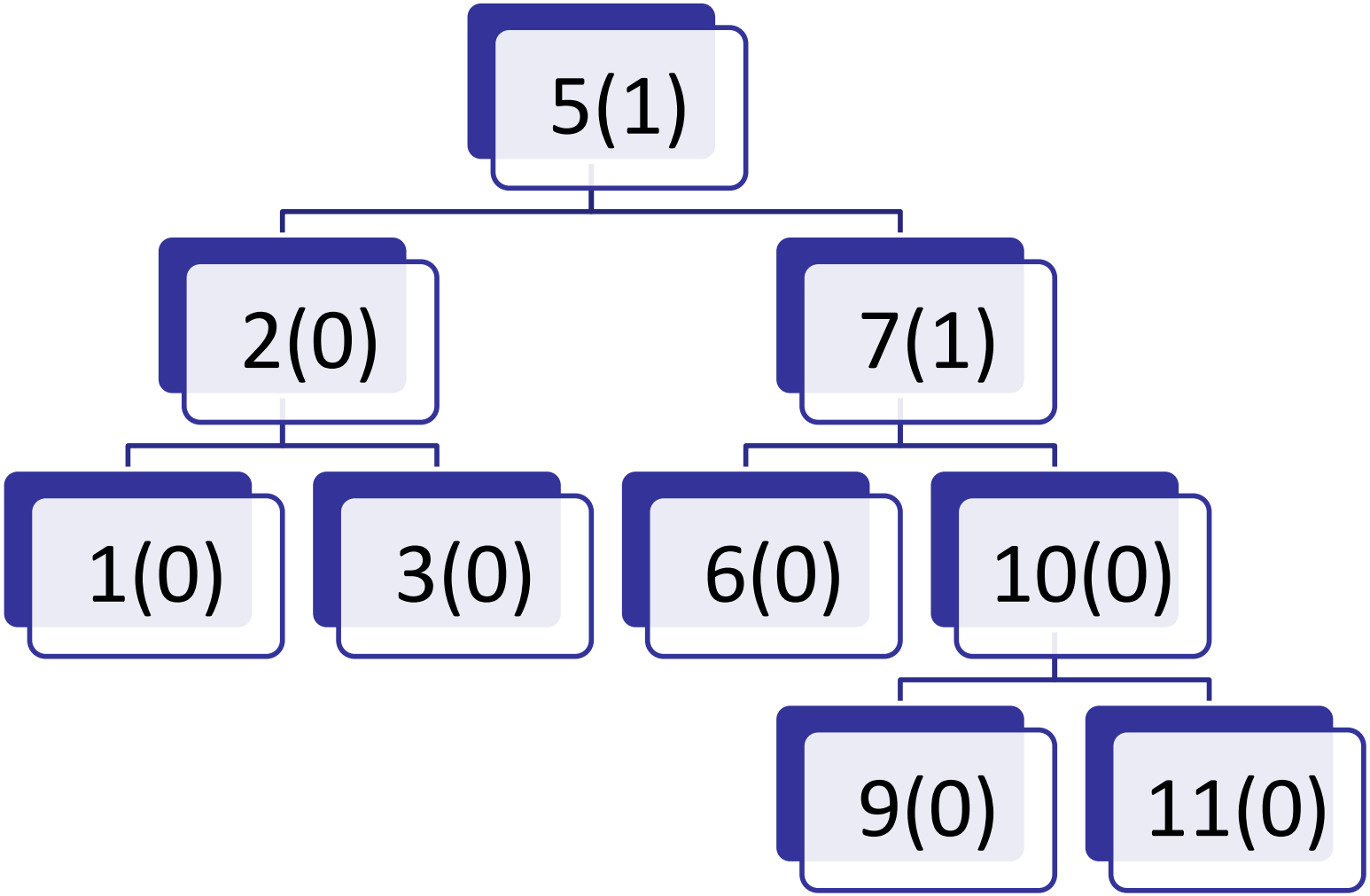
Удаление элемента из АВЛ дерева - 4



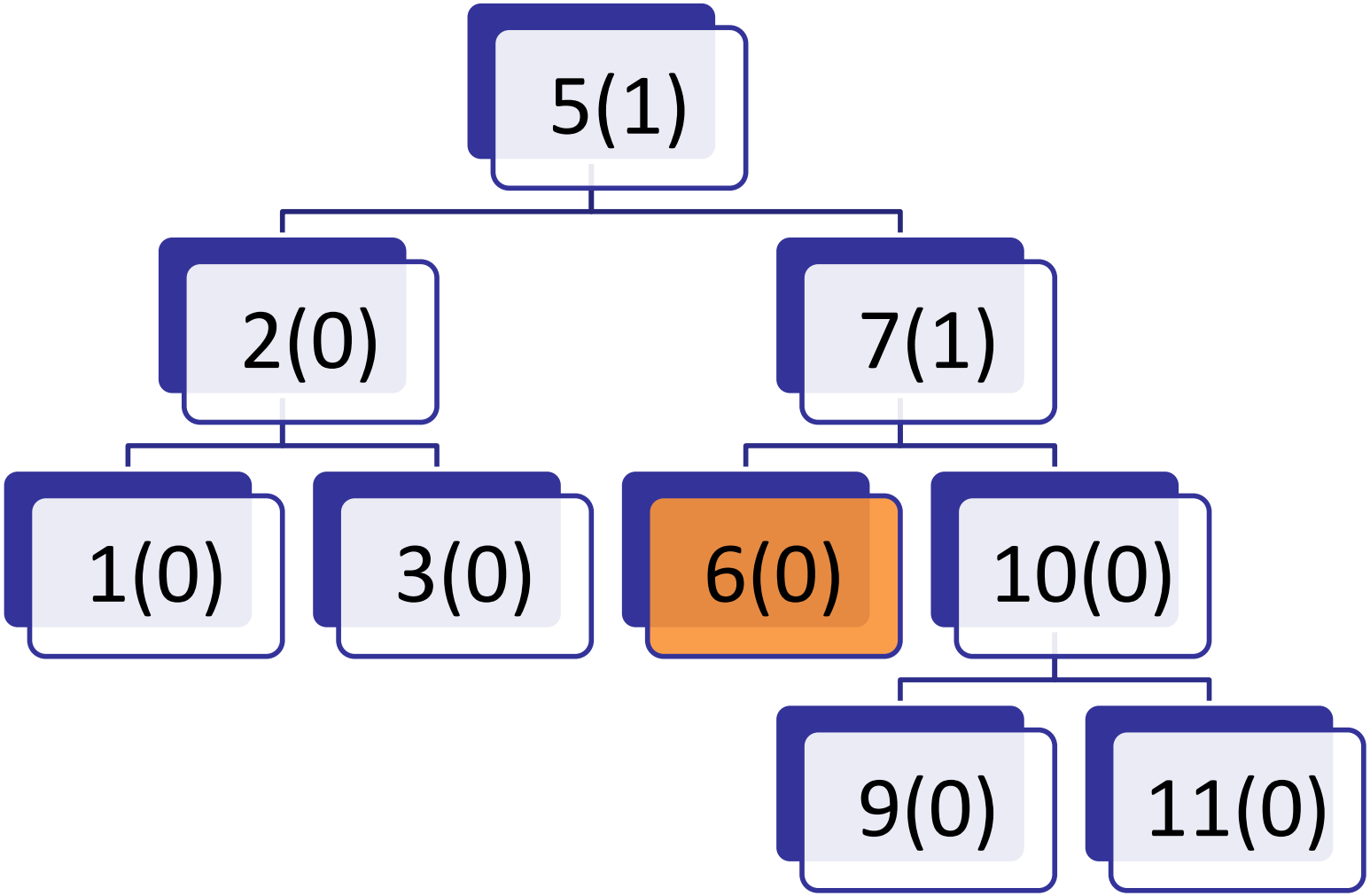
Удаление элемента из АВЛ дерева - 5



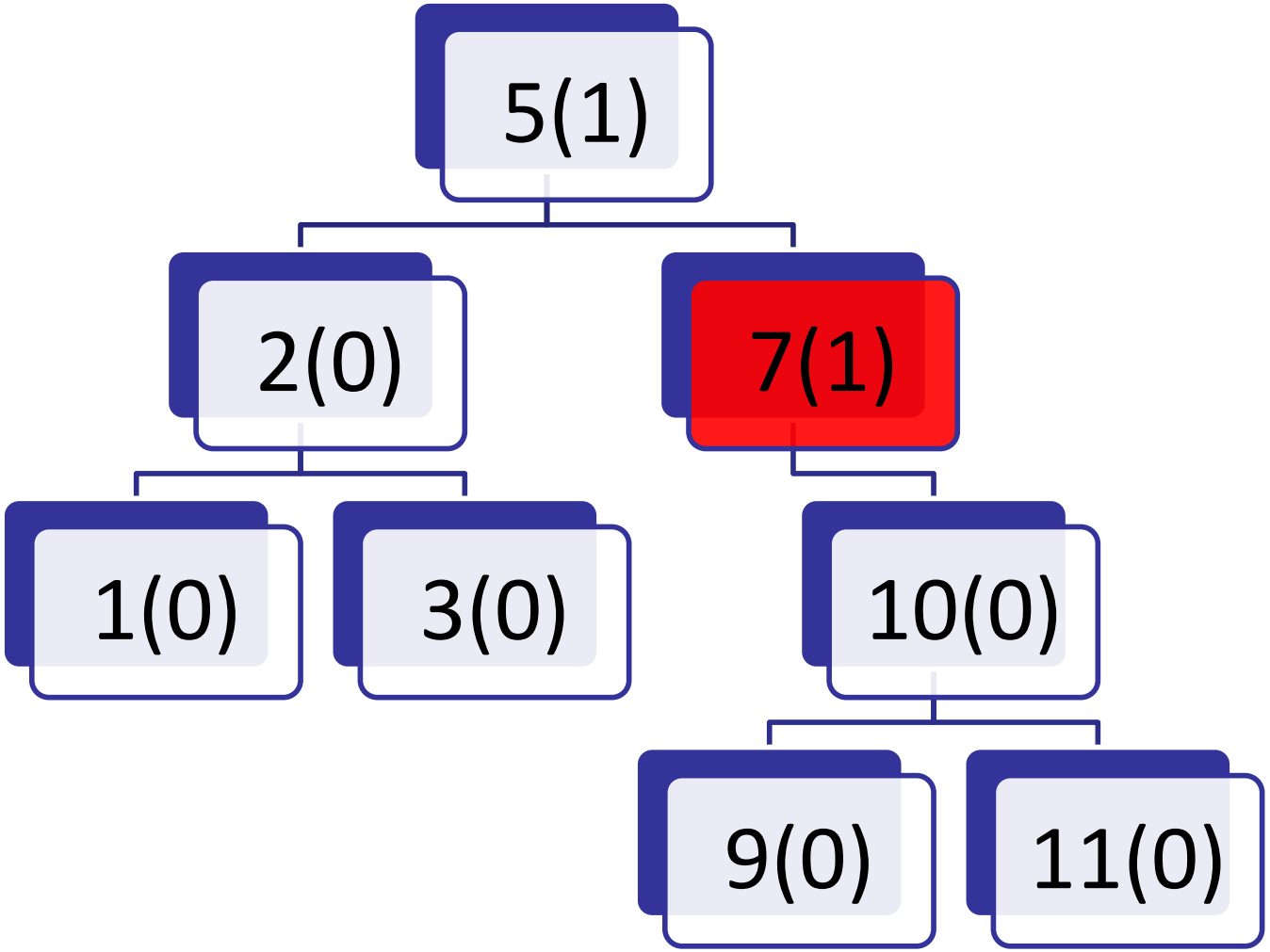
Удаление элемента из АВЛ дерева - 6



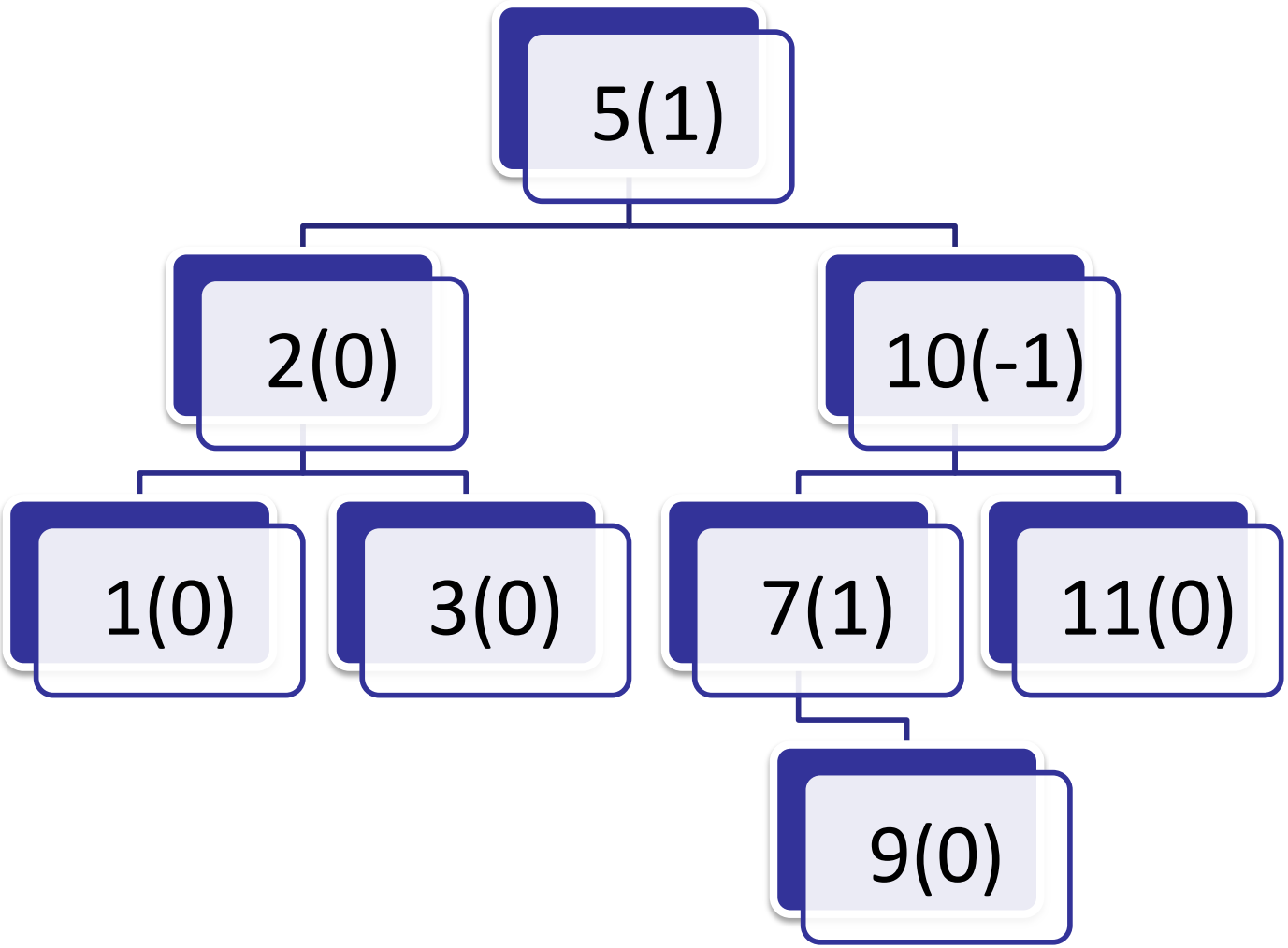
Удаление элемента из АВЛ дерева – 7



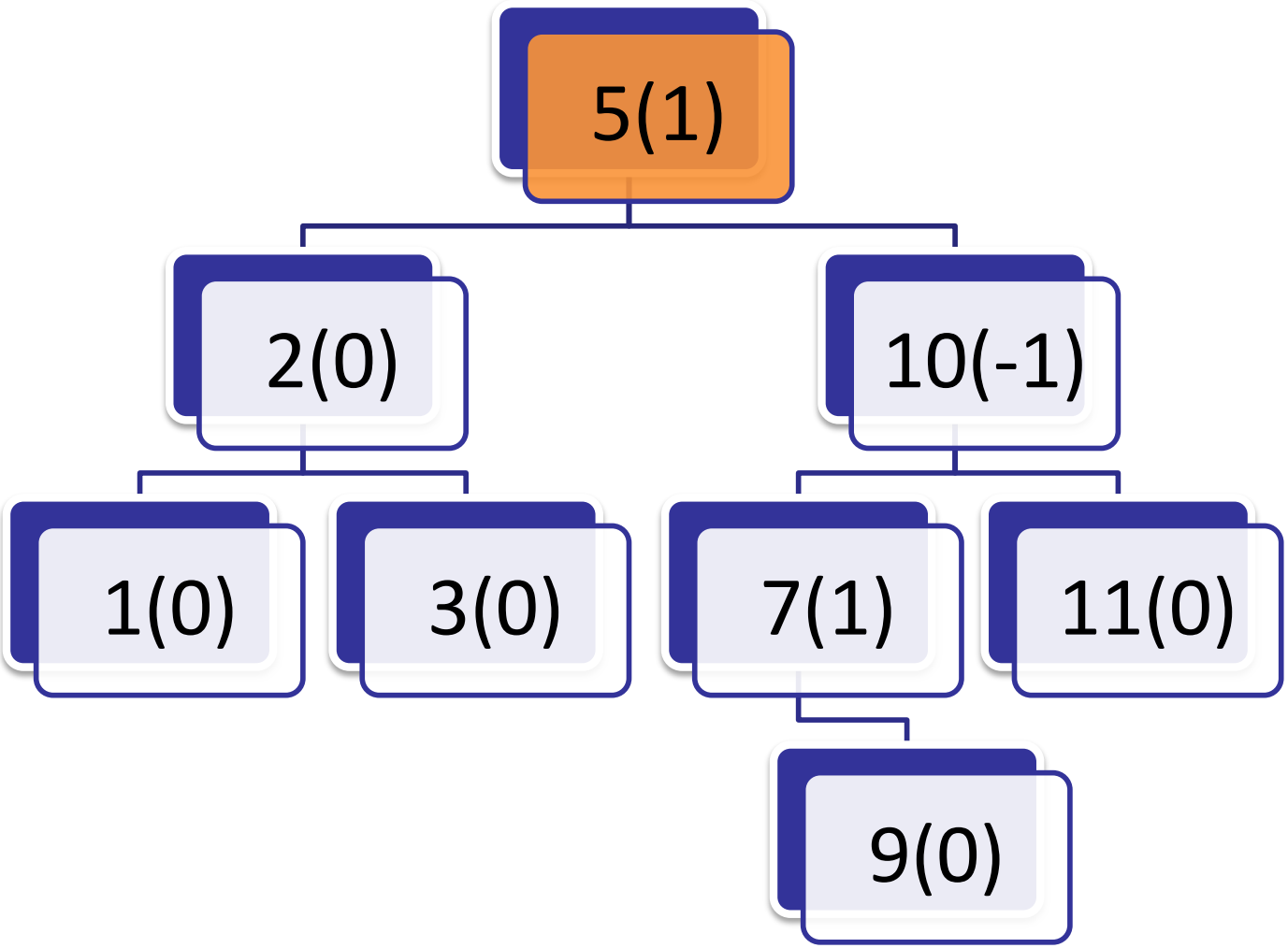
Удаление элемента из АВЛ дерева - 8



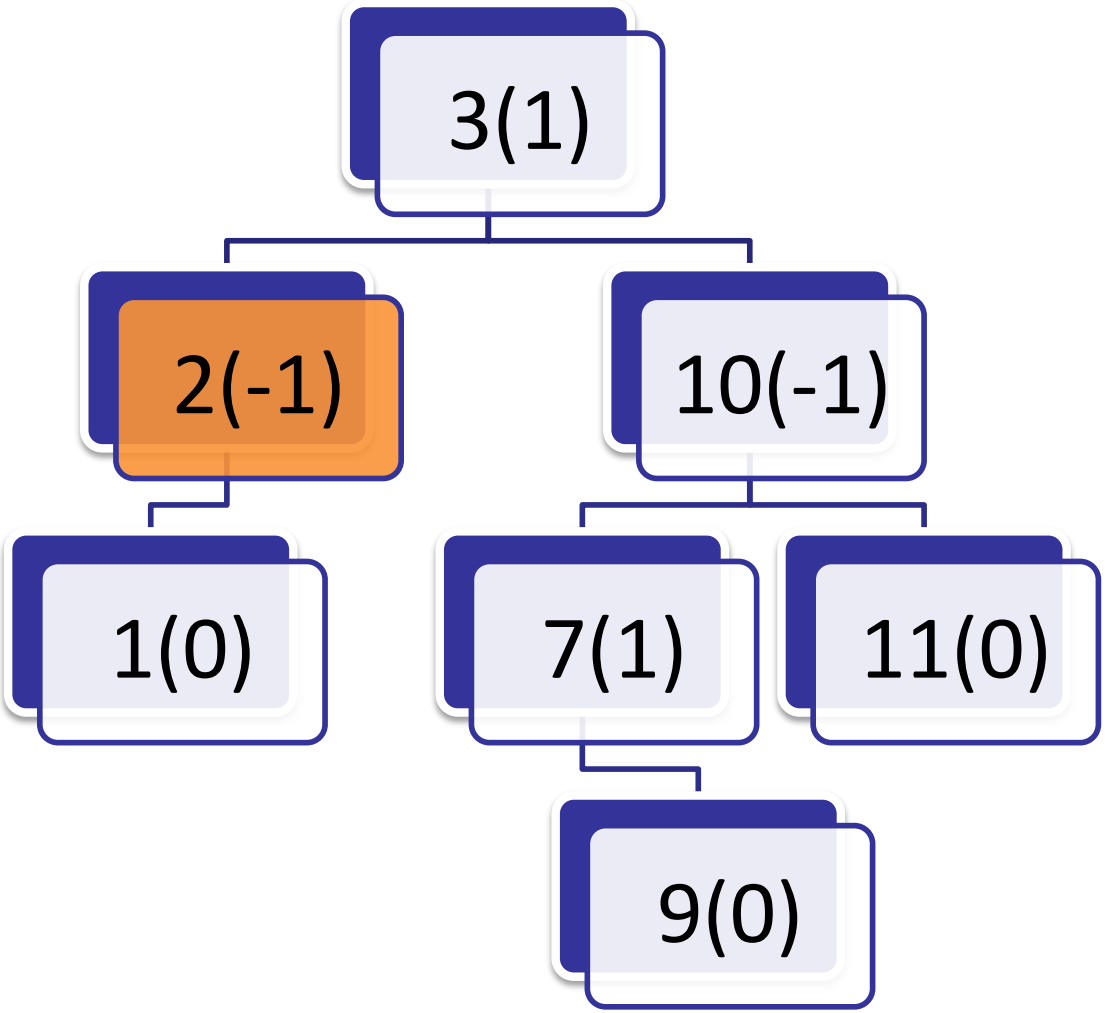
Удаление элемента из АВЛ дерева



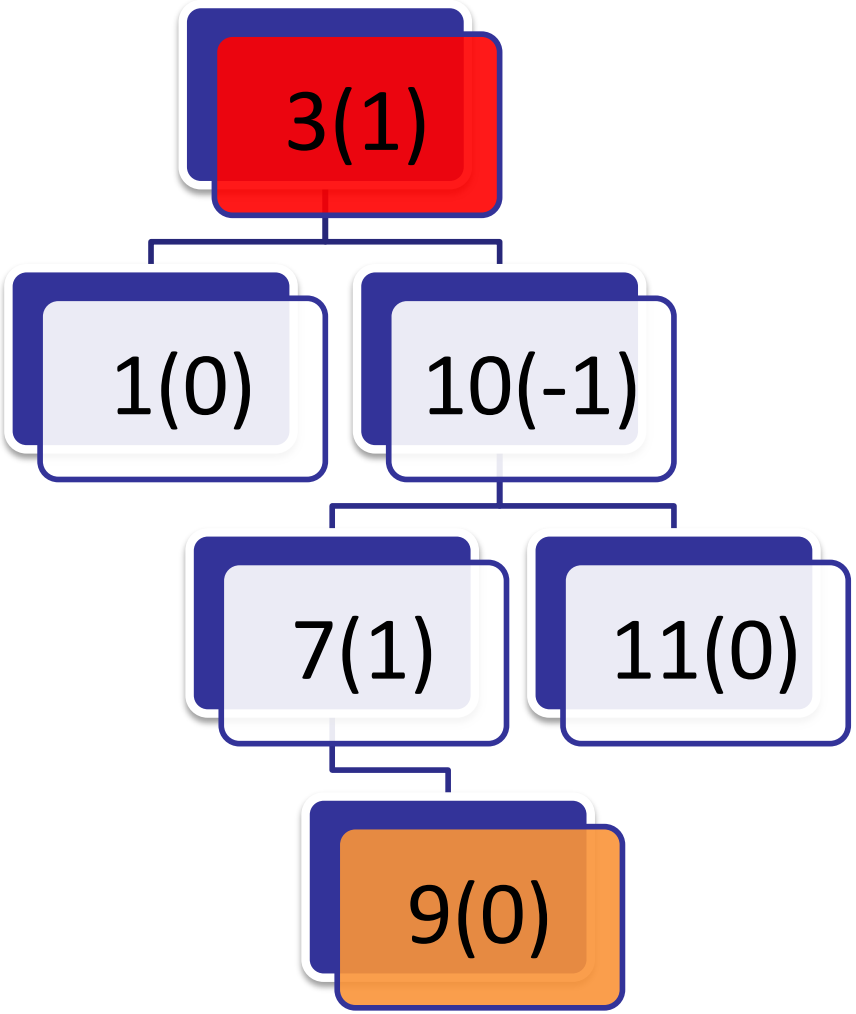
Удаление элемента из АВЛ дерева - 9



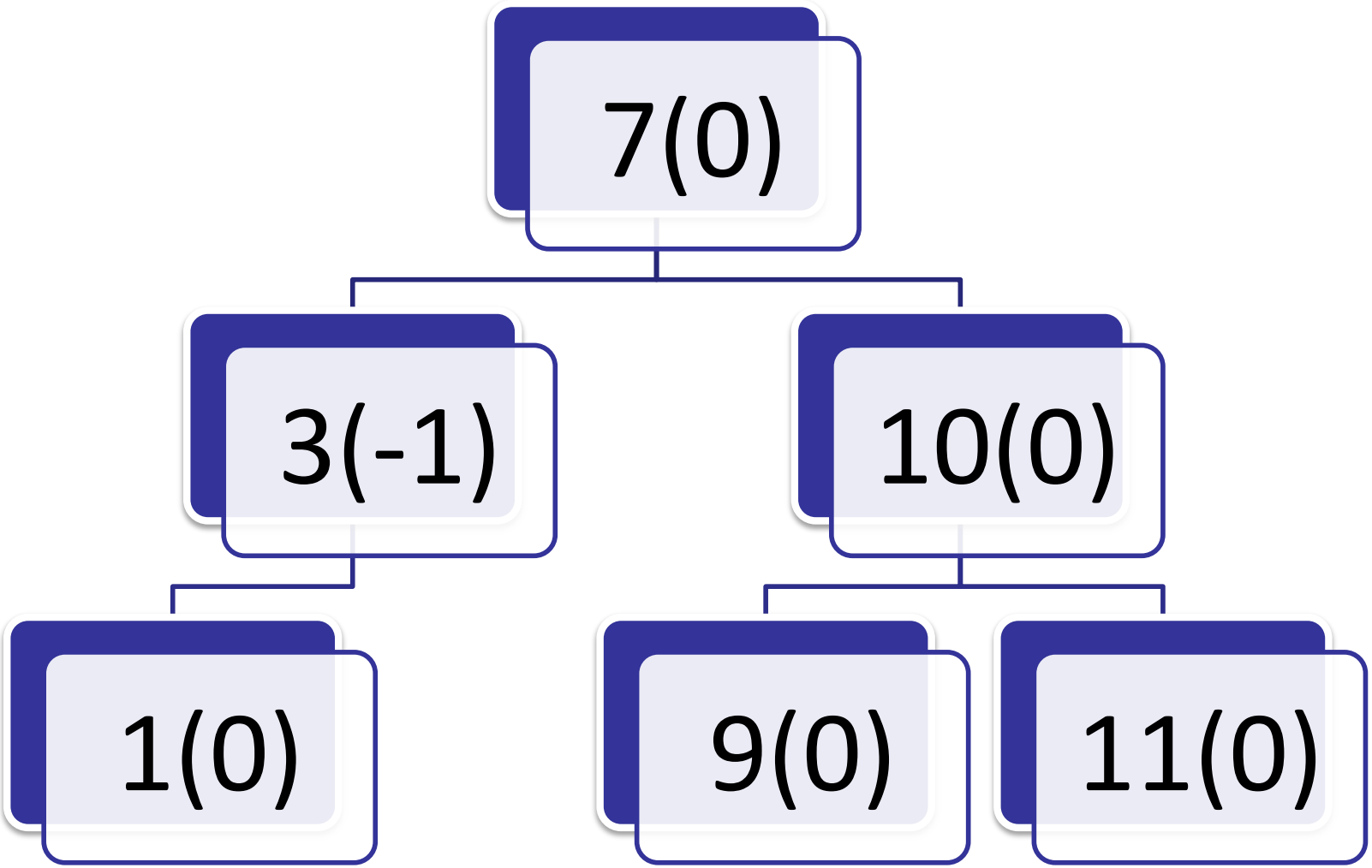
Удаление элемента из АВЛ дерева - 10



Удаление элемента из АВЛ дерева - 11



Удаление элемента из АВЛ дерева - 12



$\sim O(1)$

ХЕШ-ТАБЛИЦЫ:
ПОИСК НА ОСНОВЕ МАССИВА

H: К→А

NB: поиск по ключам

f.e.: Поиск по фамилии, состоящей не более чем из 10 букв.

- Множество возможных ключей ~30^{10}
- Множество индексов массива ~10^3

Выбор функции преобразования

- $H(k) = ord(k) \bmod N$ – функция расстановки
  - $ord(k) = \sum \alpha_i \cdot 30^i$
  - N – простое число
  - $H(k) \in \{0,\ldots,N-1\}$
- $a = H(k') = H(k'')$ – конфликт
- Получение альтернативного индекса – задача разрешения конфликтов

Разрешение конфликтов

1. Привлечение дополнительной памяти для размещения элементов с одинаковым первичным индексом – <u>область переполнения</u>

2. Размещение элементов с одинаковыми первичными индексами в незанятых ячейках того же массива – <u>открытая адресация</u>

1. Область переполнения

```
#include <iostream>
using namespace std;
#include "My_str.h"
#include "SQL.h"
char Rand() { return 'A'+(char)26.*rand()/RAND_MAX; }
void Rand(char a[], int n=6) { for(int i=0;i<n;i++) a[i]=Rand(); }
long h(const char a[], int n=6) { long s=0; for(int i=0;i<n;i++) s=s*26 + a[i]-'A'; return s; }
unsigned long long start, end;
unsigned long long access_counter() { asm("rdtsc"); }
int main() {
List<My_string> A[23], B;
char s[7]={0}; const char *k="MFTMLY";
My_string K(k);
     for(int i=0;i<20;i++) {
            Rand(s); int j = h(s) % 23; A[j].push_back(My_string(s));
            B.push_back(My_string(s)); cout << s << endl; }
     for(int i=0;i<23;i++) { cout << i << ' '; cout << A[i]; cout << endl; }
    start = access_counter();
          B.remove(K);
    end = access_counter();
    cout << "Elapsed time: " << end - start << " \n";
    start = access_counter();
          A[h(k)%23].remove(K);
    end = access_counter();
    cout << "Elapsed time: " << end - start << " \n";
    return 0;
}
```

AOFVPM
JXVTEW
SNHACJ
DEZLDA
AJNOPP
ERLJBP
UUNHWS
YYODMG
WFUVZZ
PKGHVA
JCRBAX
HHPRVS
MFTMLY
TCPKTP
OJDFLU
NZTIER
MBSNDY
DXSHLB
ZRDWVP
EEVMEN

```
0
1 AJNOPP
DXSHLB

2 WFUVZZ
NZTIER

3
4
5 PKGHVA

6
7
8 ERLJBP

9 UUNHWS
HHPRVS
OJDFLU

10 TCPKTP

11 JCRBAX

12 YYODMG
ZRDWVP

13
14
15 JXVTEW

16
17
18 EEVMEN

19 SNHACJ
DEZLDA

20 AOFVPM
MFTMLY

21
22 MBSNDY

Elapsed time: 6440
Elapsed time: 3094
```

2. Открытая адресация

- H(k,i)
  - i – номер исследования
- Линейное исследование

$$h(k,i) = (h'(k) + i) \bmod m$$

- Квадратичное исследование

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

- Двойное хеширование

$$h(k,i) = (h_1(k) + i h_2(k)) \bmod m$$

```
20 AOFVPM
15 JXVTEW
19 SNHACJ
19 20 21 DEZLDA
1 AJNOPP
8 ERLJBP
9 UUNHWS
12 YYODMG
2 WFUVZZ
5 PKGHVA
11 JCRBAX
9 10 HHPRVS
20 21 22 MFTMLY
10 11 12 13 TCPKTP
9 10 11 12 13 14 OJDFLU
2 3 NZTIER
22 0 MBSNDY
1 2 3 4 DXSHLB
12 13 14 15 16 ZRDWVP
18 EEVMEN
0        MBSNDY
1        AJNOPP
2        WFUVZZ
3        NZTIER
4        DXSHLB
5        PKGHVA
6
7
8        ERLJBP
9        UUNHWS
10       HHPRVS
11       JCRBAX
12       YYODMG
13       TCPKTP
14       OJDFLU
15       JXVTEW
16       ZRDWVP
17
18       EEVMEN
19       SNHACJ
20       AOFVPM
21       DEZLDA
```

```
20 AOFVPM
15 JXVTEW
19 SNHACJ
19 20 0 DEZLDA
1 AJNOPP
8 ERLJBP
9 UUNHWS
12 YYODMG
2 WFUVZZ
5 PKGHVA
11 JCRBAX
9 10 HHPRVS
20 21 MFTMLY
10 11 14 TCPKTP
9 10 13 OJDFLU
2 3 NZTIER
22 MBSNDY
1 2 5 10 17 DXSHLB
12 13 16 ZRDWVP
18 EEVMEN
0        DEZLDA
1        AJNOPP
2        WFUVZZ
3        NZTIER
4
5        PKGHVA
6
7
8        ERLJBP
9        UUNHWS
10       HHPRVS
11       JCRBAX
12       YYODMG
13       OJDFLU
14       TCPKTP
15       JXVTEW
16       ZRDWVP
17       DXSHLB
18       EEVMEN
19       SNHACJ
20       AOFVPM
21       MFTMLY
22       MBSNDY
```

```
20 AOFVPM
15 JXVTEW
19 SNHACJ
19 7 DEZLDA
1 AJNOPP
8 ERLJBP
9 UUNHWS
12 YYODMG
2 WFUVZZ
5 PKGHVA
11 JCRBAX
9 7 5 3 HHPRVS
20 16 MFTMLY
10 TCPKTP
9 1 16 8 0 OJDFLU
2 16 7 21 NZTIER
22 MBSNDY
1 19 14 DXSHLB
12 5 21 14 7 0 16 9 2 18 ZRDWVP
18 15 12 9 6 EEVMEN
0        OJDFLU
1        AJNOPP
2        WFUVZZ
3        HHPRVS
4
5        PKGHVA
6        EEVMEN
7        DEZLDA
8        ERLJBP
9        UUNHWS
10       TCPKTP
11       JCRBAX
12       YYODMG
13
14       DXSHLB
15       JXVTEW
16       MFTMLY
17
18       ZRDWVP
19       SNHACJ
20       AOFVPM
21       NZTIER
22       MBSNDY
```

```
#include <iostream>
using namespace std;
#include "My_str.h"
#include "SQL.h"

const int N=23;

char Rand() { return 'A'+(char)26.*rand()/RAND_MAX; }
void Rand(char a[], int n=6) { for(int i=0;i<n;i++) a[i]=Rand(); }
long H(char a[], int n=6) { long s=0; for(int i=0;i<n;i++) s=s*26 + a[i]-'A'; return s; }

unsigned long long start, end;
unsigned long long access_counter() { asm("rdtsc"); }

inline int h1(int k) { return k%N; }
inline int h2(int k) { return 1+k%(N-1); }
inline int h(int k,int i) { return (h1(k)+i*h2(k))%N; }
inline int h1(int k,int i) { return (h1(k) + i)%N; }
inline int h2(int k,int i) { return (h1(k) + i*i)%N; }

int main() {
List<My_string> A;
List<My_string> B[N];
My_string C1[N],C2[N],Cd[N] ;
char s[7], *k="MFTMLY"; s[6]=0;
int x,y;
My_string K(k),S;

    for(int i=0; i<N; i++) { C1[i]=""; C2[i]=""; Cd[i]=""; }
```

```
 for(int i=0;i<20;i++) { Rand(s); S=s;
     A.push_back(S);
 B[h1(H(s))].push_back(S);
 y=0; while(C1[x=h1(H(s),y++)].size()>0); C1[x]=S;
 y=0; while(C2[x=h2(H(s),y++)].size()>0); C2[x]=S;
 y=0; while(Cd[x=h(H(s),y++)].size()>0); Cd[x]=S; }

 cout << A << endl;
 for(int i=0;i<N;i++) cout << i << ": " << B[i]; cout << endl;
 for(int i=0;i<N;i++) cout << i << ": " << C1[i] << '\t' << C2[i] << '\t' << Cd[i] << endl;

start = access_counter();
      A.remove(K);
end = access_counter();
cout << "Elapsed time: " << end - start << " \n";

start = access_counter();
      B[h1(H(k))].remove(K);
end = access_counter();
cout << "Elapsed time: " << end - start << " \n";
```

```
    start = access_counter();
            y=0; for(;;)if(C1[x=h1(H(k),y)]==K) break; else y++;
            C1[x]="      ";
    end = access_counter();
    cout << "Elapsed time: " << end - start << " \n";

    start = access_counter();
            y=0; for(;;)if(C2[x=h2(H(k),y)]==K) break; else y++;
            C2[x]="      ";
    end = access_counter();
    cout << "Elapsed time: " << end - start << " \n";

    start = access_counter();
            y=0; for(;;)if(Cd[x=h(H(k),y)]==K) break; else y++;
            Cd[x]="      ";
    end = access_counter();
    cout << "Elapsed time: " << end - start << " \n";

     cout << A << endl;
     for(int i=0;i<N;i++) cout << i << ": " << B[i]; cout << endl;
     for(int i=0;i<N;i++) cout << i << ": " << C1[i] << '\t' << C2[i] << '\t' << Cd[i] << endl;

    return 0;
}
```

Исходные данные

	Список	Массив списков		Линейное	Квадратичное	Двойное
0	AOFVPM			MBSNDY	DEZLDA	OJDFLU
1	JXVTEW	AJNOPP	DXSHLB	AJNOPP	AJNOPP	AJNOPP
2	SNHACJ	WFUVZZ	NZTIER	WFUVZZ	WFUVZZ	WFUVZZ
3	DEZLDA			NZTIER	NZTIER	HHPRVS
4	AJNOPP					
5	ERLJBP	PKGHVA			A	PKGHVA
6	UUNHWS					EEVMEN
7	YYODMG					DEZLDA
8	WFUVZZ	ERLJBP				ERLJBP
9	PKGHVA	UUNHWS	HH		WS	UUNHWS
10	JCRBAX	TCPKTP			S	TCPKTP
11	HHPRVS	JCRBAX			X	JCRBAX
12	MFTMLY	YYODMG	ZR		MG	YYODMG
13	TCPKTP			TCPKTP	OJDFLU	
14	OJDFLU			OJDFLU	TCPKTP	DXSHLB
15	NZTIER	JXVTEW		JXVTEW	JXVTEW	JXVTEW
16	MBSNDY			ZRDWVP	ZRDWVP	MFTMLY (20)
17	DXSHLB				DXSHLB	
18	ZRDWVP	EEVMEN		EEVMEN	EEVMEN	ZRDWVP
19	EEVMEN	SNHACJ	DEZLDA	SNHACJ	SNHACJ	SNHACJ
20		AOFVPM	MFTMLY	AOFVPM	AOFVPM	AOFVPM
21				DEZLDA	MFTMLY (20)	NZTIER
22		MBSNDY		MFTMLY (20,21)	MBSNDY	MBSNDY

Elapsed time: 4018
Elapsed time: 1456
Elapsed time: 3682
Elapsed time: 2212
Elapsed time: 1946

Среднее количество проб*)

Таблица цепочек

	$\alpha=0.5$	$\alpha=0.9$	
■ Успешный поиск			Неудачный поиск
■ $\approx 1 + \alpha/2$	1.25	1.45	$\approx 1 + \alpha^2/2$

Открытая адресация

■ Линейное исследование

$\frac{1}{2}\left(1 + \frac{1}{1-\alpha}\right)$	1.5	5.5	$\frac{1}{2}\left(1 + \left(\frac{1}{1-\alpha}\right)^2\right)$

■ С двойным хешированием

$ln\left(\frac{1}{1-\alpha}\right)\Big/_{\alpha}$	1.4	2.6	$\approx (1-\alpha)^{-1}$

*) Д.Кнут Искусство программирования. Том 3.