Древовидные структуры

Бинарные деревья:

Идеально сбалансированное и сбалансированное АВЛ дерево

Древовидные структуры

- Дерево ::= Ø | Вершина
  | Вершина ¤ Дерево ¤ … ¤ Дерево
- Терминология:

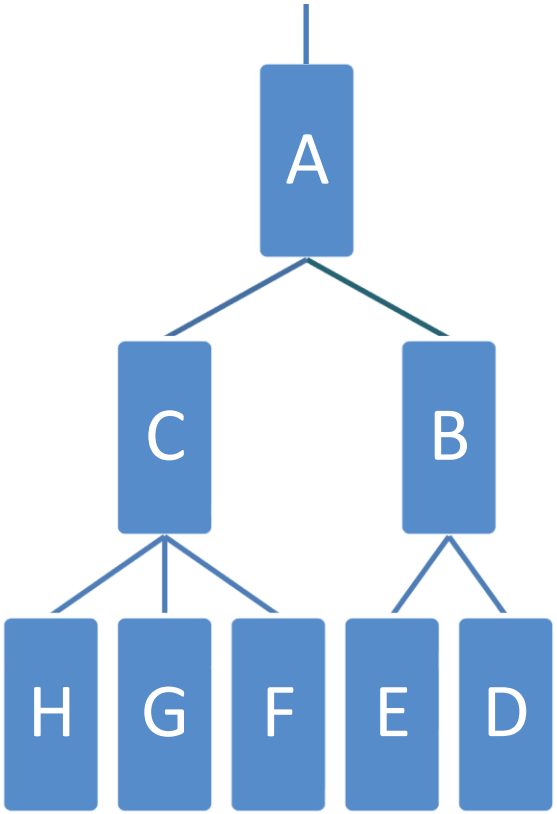
✓ A – корень дерева;

✓ C предок G или G потомок C;

✓ Корень на уровне 1;

✓ Максимальный уровень – это высота или глубина дерева

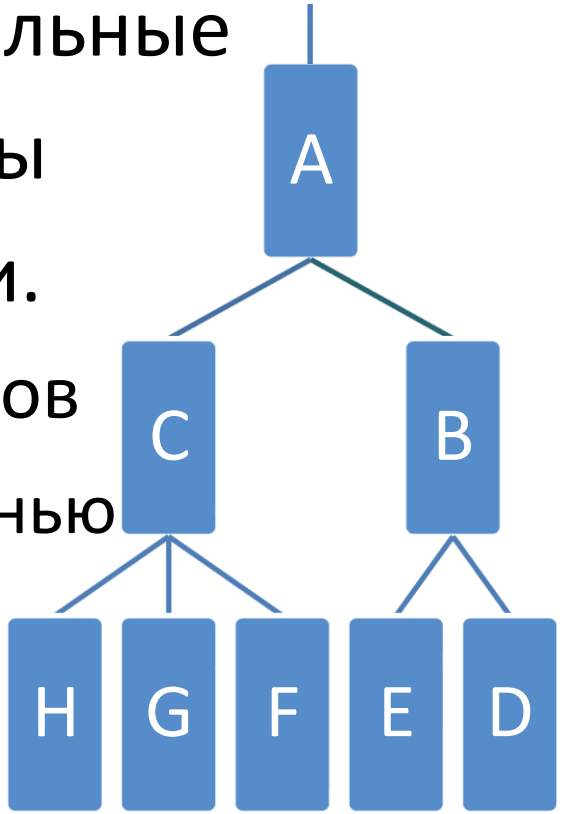
✓ Число ребер, которые нужно пройти от корня до узла *x*, называют длиной пути к *x*.



Древовидные структуры

Корень имеет длину пути 1, его непосредственные потомки – 2, узлы на уровне i имеют длину пути i.

- ✓ Элементы D,E,F,G,H – терминальные узлы. Не терминальные элементы называются внутренними узлами.
- ✓ Число непосредственных потомков внутреннего узла называют его степенью
- ✓ Степень дерева – максимальная степень среди всех его узлов.



Бинарные деревья

- Описание вершины

```
template <typename T>
        struct Item {T n; Item *left, *right;};
```

- Построение дерева с n узлами минимальной высоты:

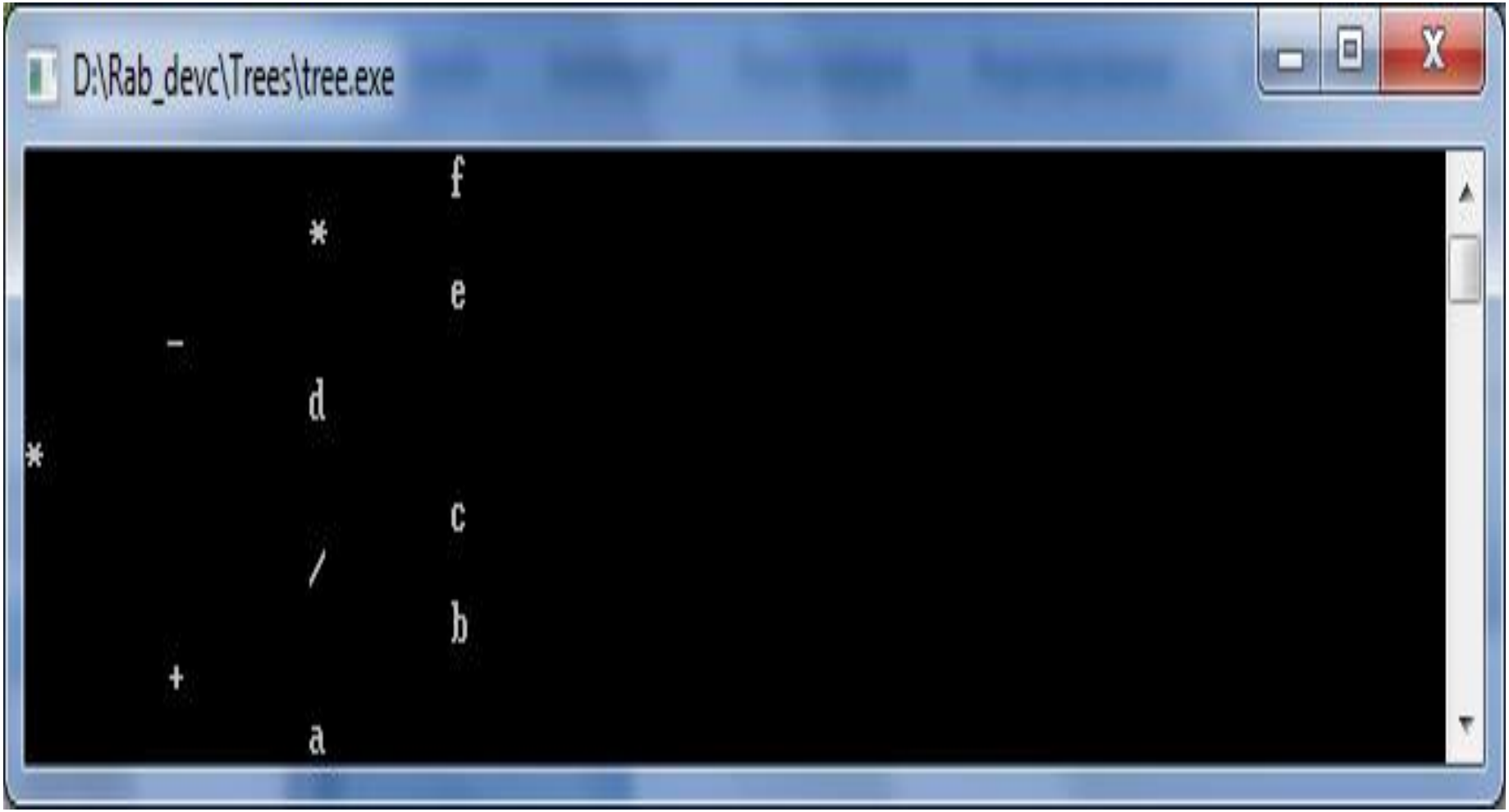
1) Взять один узел в качестве корня дерева;
2) Построить левое поддерево с ***nl=n/2*** узлами;
3) Построить правое поддерево с ***nr=n-nl-1*** узлами.

```
#include <iostream>
using namespace std;
template <typename T> struct Item {  T n; Item *left, *right;
   Item(T const& a, Item<T> *l=NULL, Item<T> *r=NULL)
                                                { n=a; left=l; right=r; }
   void operator+(T const& a) { right= new Item(a); }
   friend void operator+(T const& a, Item& b) { b.left =  new Item(a); }
   Item<T>& operator[](int i) { if(i==1) return *left; else return *right; }
   friend ostream& operator<<(ostream& a, Item<T> const& b) {
   static int k=0;
         k++; if(b.right!=NULL) a << *b.right; k--;
         for(int i=0;i<k;i++) a << '\t'; a << b.n << '\n';
         k++; if(b.left!=NULL) a << *b.left; k--;
         return a; }
};
```

```
typedef Item<int> *Tree;

Tree tree(int n) { Tree ptr; int nl=n/2, nr=n-nl-1;
    if(!n) return NULL;
    return ptr = new Item<int>(n,tree(nl),tree(nr)); }

int main() {
   Item<char> a('*');
   '+'+a; a+'-'; 'a'+a[1]; a[1]+'/'; 'd'+a[2];a[2]+'*';
   'b'+a[1][2]; a[1][2]+'c'; 'e'+a[2][2]; a[2][2]+'f';
   cout << a;
Tree T=tree(32);
    cout << *T;
   return 0;  }
```



(a+b/c)*(d-e*f)

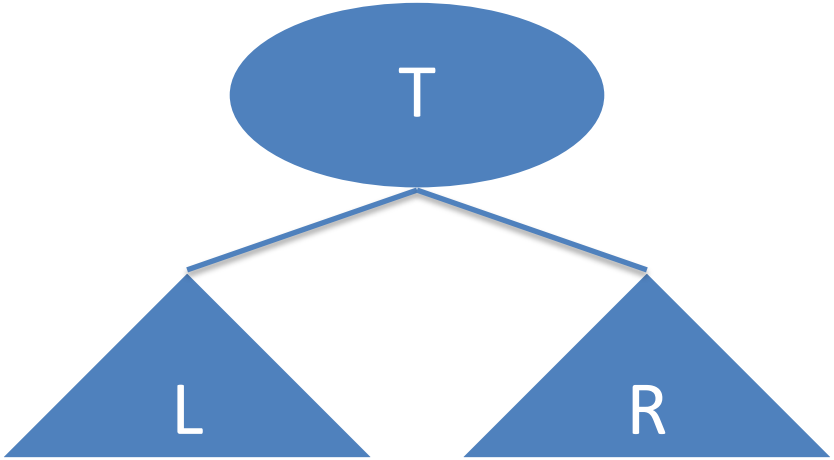
D:\Rab_devc\Trees\tree.exe

```
                    1
          3
                    1
    7
                    1
          3
                    1
15
                    1
          3
                    1
    7
                    1
          3
                    1
32
                    1
          3
                    1
    7
                    1
          3
                    1
16
                    1
          3
                    1
    8
                    1
          4
                    2
                        1
```

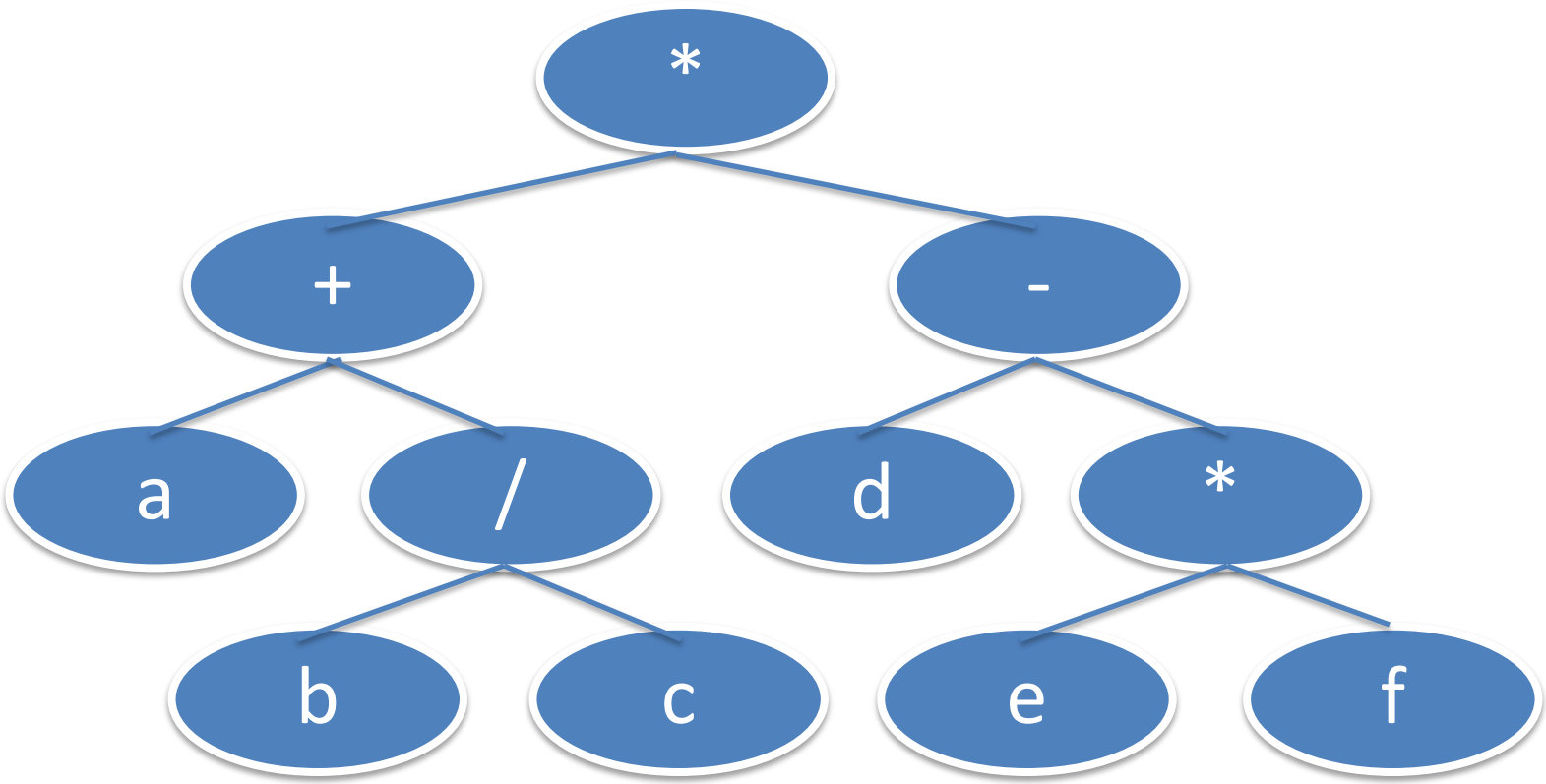
Бинарное дерево

- Обход дерева:

1. Сверху вниз: T,L,R
2. Слева направо: L,T,R
3. Снизу вверх: L,R,T



(a+b/c)*(d-e*f)



1. TLR: *+a/bc-d*ef – префиксная запись
2. LTR: a+b/c*d-e*f – инфиксная запись
3. LRT: abc/+def*-* – постфиксная запись

Бинарное дерево

- **Дерево поиска**: $N_L < N_T < N_R$

Здесь N_L – все ключи левого поддерева,

N_R – все ключи правого поддерева

- **Поиск по ключу**

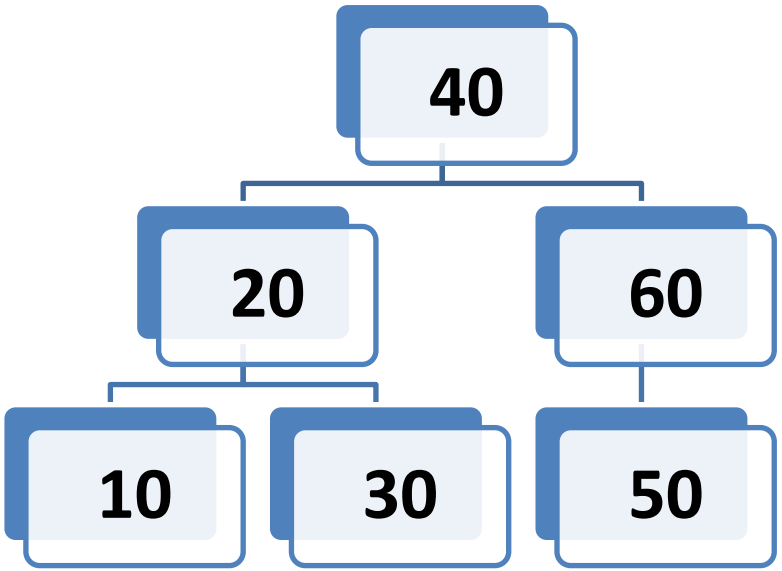
```
Item* loc(Item* ptr, int key) {
while( ptr ) if(ptr->n==key) return ptr; else
   if(ptr->n>key) ptr=ptr->left; else ptr=ptr->right;
return NULL; }
```

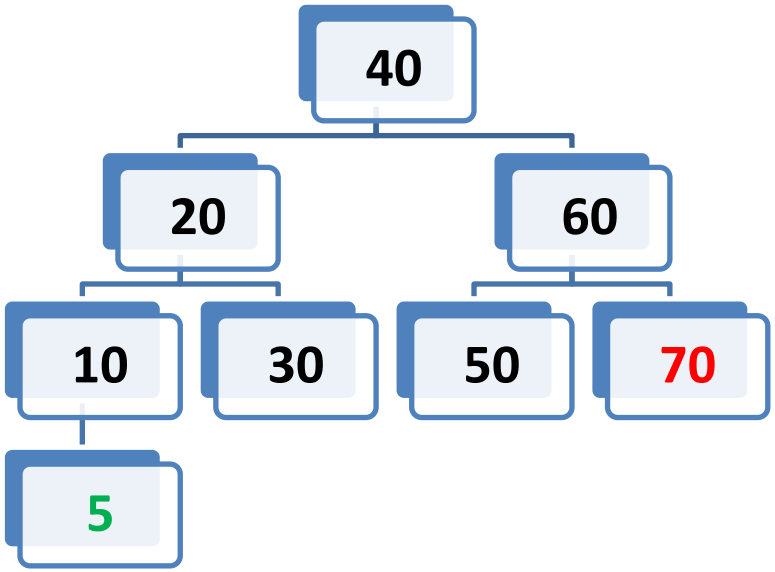
- **NB**: Чем меньше высота дерева – тем быстрее поиск.

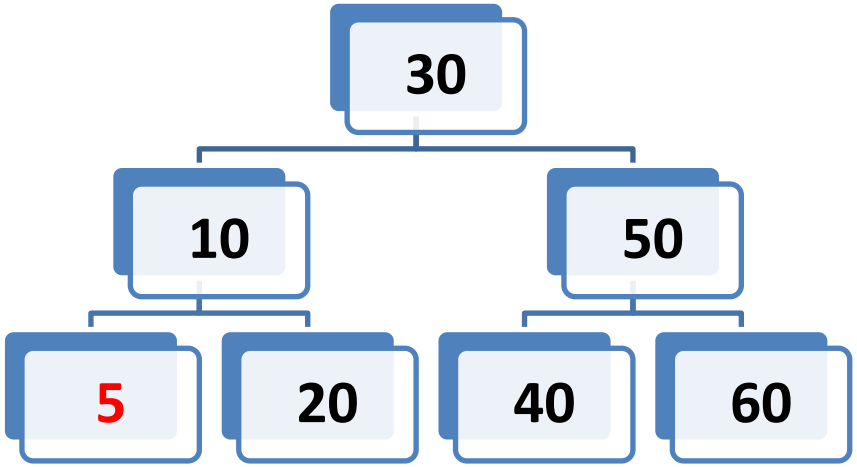
Бинарное дерево

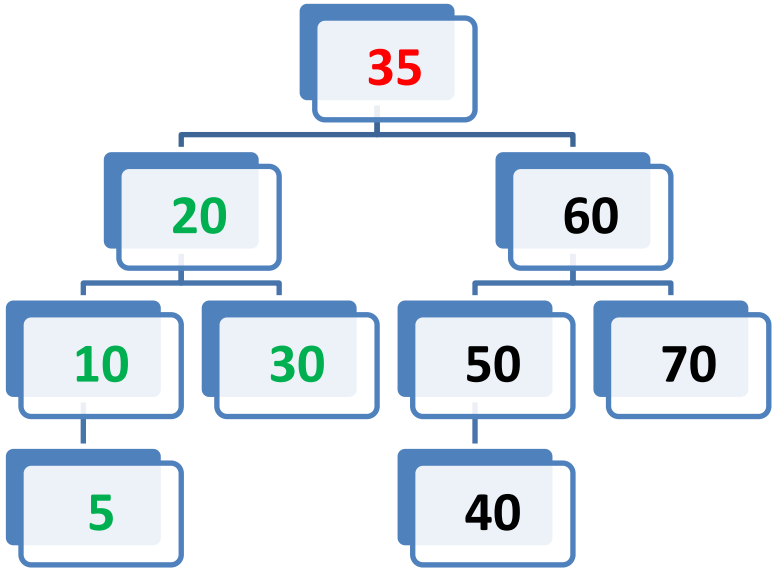
- Идеально сбалансированное дерево – это дерево, у которого для каждого его узла количества узлов в левом и правом поддеревьях различаются не более чем на 1.

Идеально сбалансированное дерево









Бинарное дерево

- Идеально сбалансированное дерево – это дерево, у которого для каждого его узла количества узлов в левом и правом поддеревьях различаются не более чем на 1.

Бинарное дерево

- Идеально сбалансированное дерево – это дерево, у которого для каждого его узла количества узлов в левом и правом поддеревьях различаются не более чем на 1.
- Сбалансированное дерево – это дерево, у которого для каждого его узла высоты левого и правого поддеревьев различаются не более чем на 1.
  Другое их название - АВЛ-деревья (по фамилиям их изобретателей Адельсон-Вельский и Ландис).

$$\mathbf{h_{ИСД} \leq h \leq h_{ДФ}}$$

Адельсон-Вельский,
Георгий Максимович

Ландис,
Евгений Михайлович

Теорема Адельсон-Вельского и Ландиса

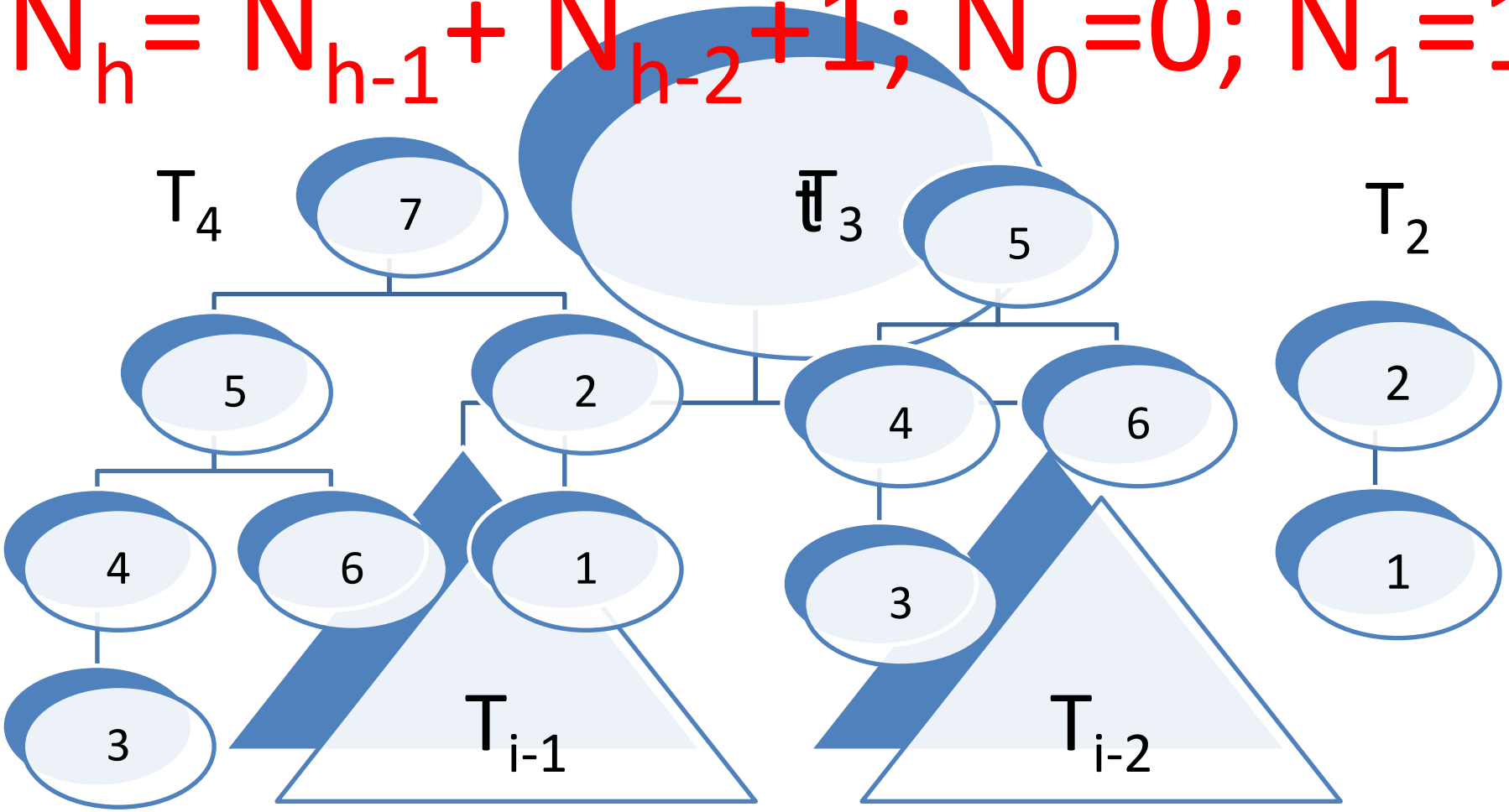
$$\log(n+1) \le h(n) < 1{,}4404 * \log(n+2) - 0{,}328$$

Построение АВЛ дерева высоты h с минимальным количеством вершин

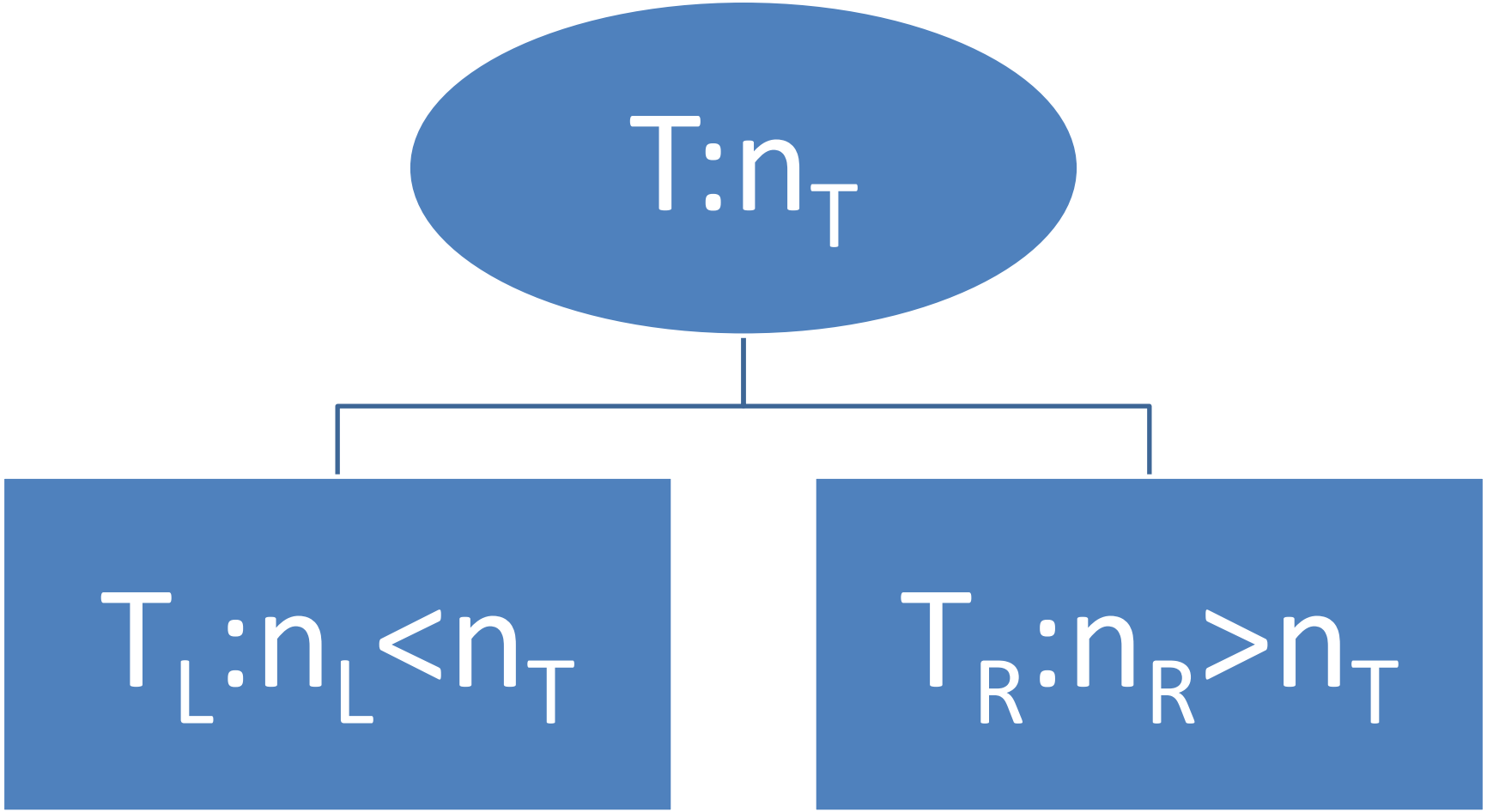
T_0 – пустое дерево
T_1 – дерево, состоящее из одной вершины: t ¤ T_0
T_2 – второго уровня (из 2 вершин): t ¤ T_1 ¤ T_0

Деревья Фибоначчи

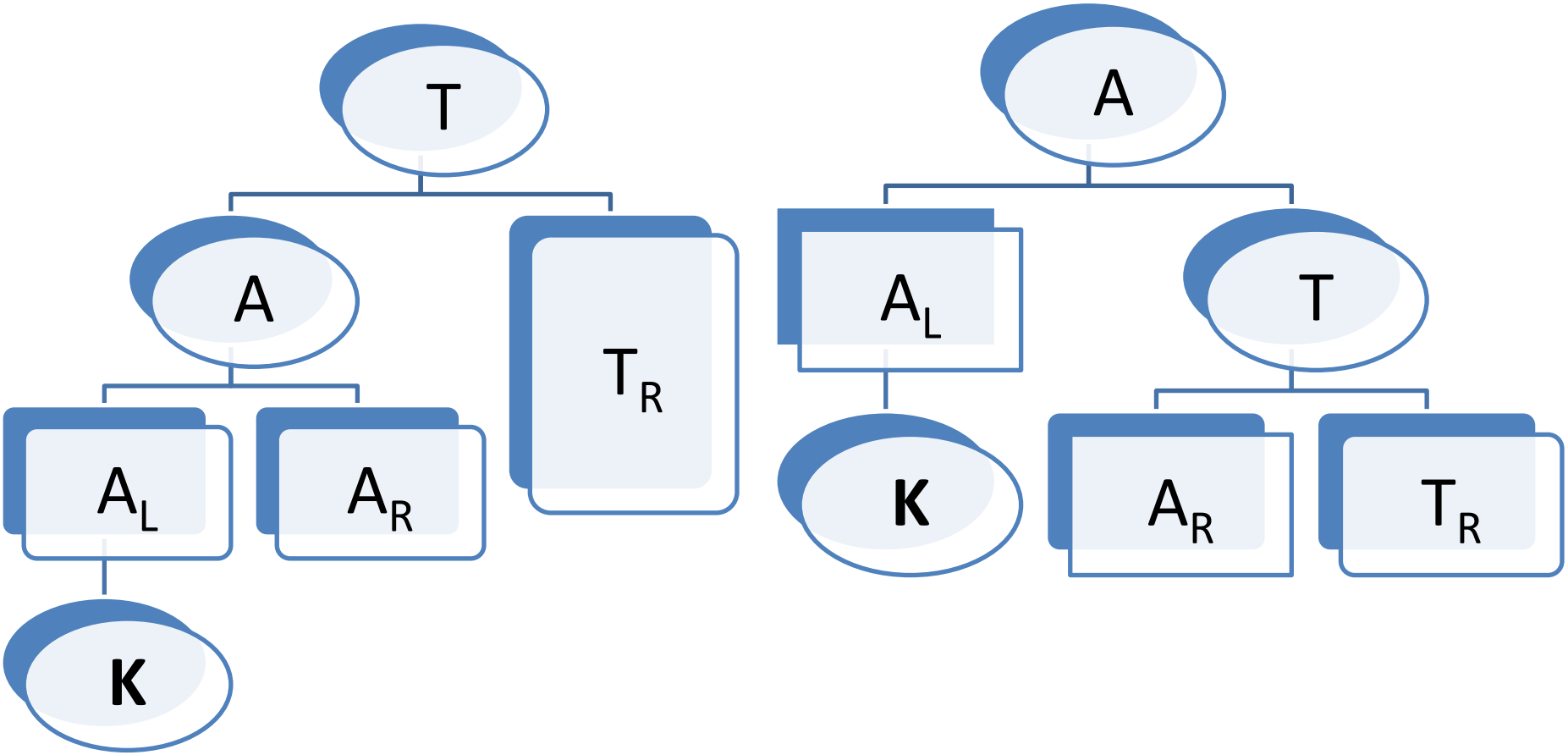
$$N_h = N_{h-1} + N_{h-2} + 1;\ N_0 = 0;\ N_1 = 1$$



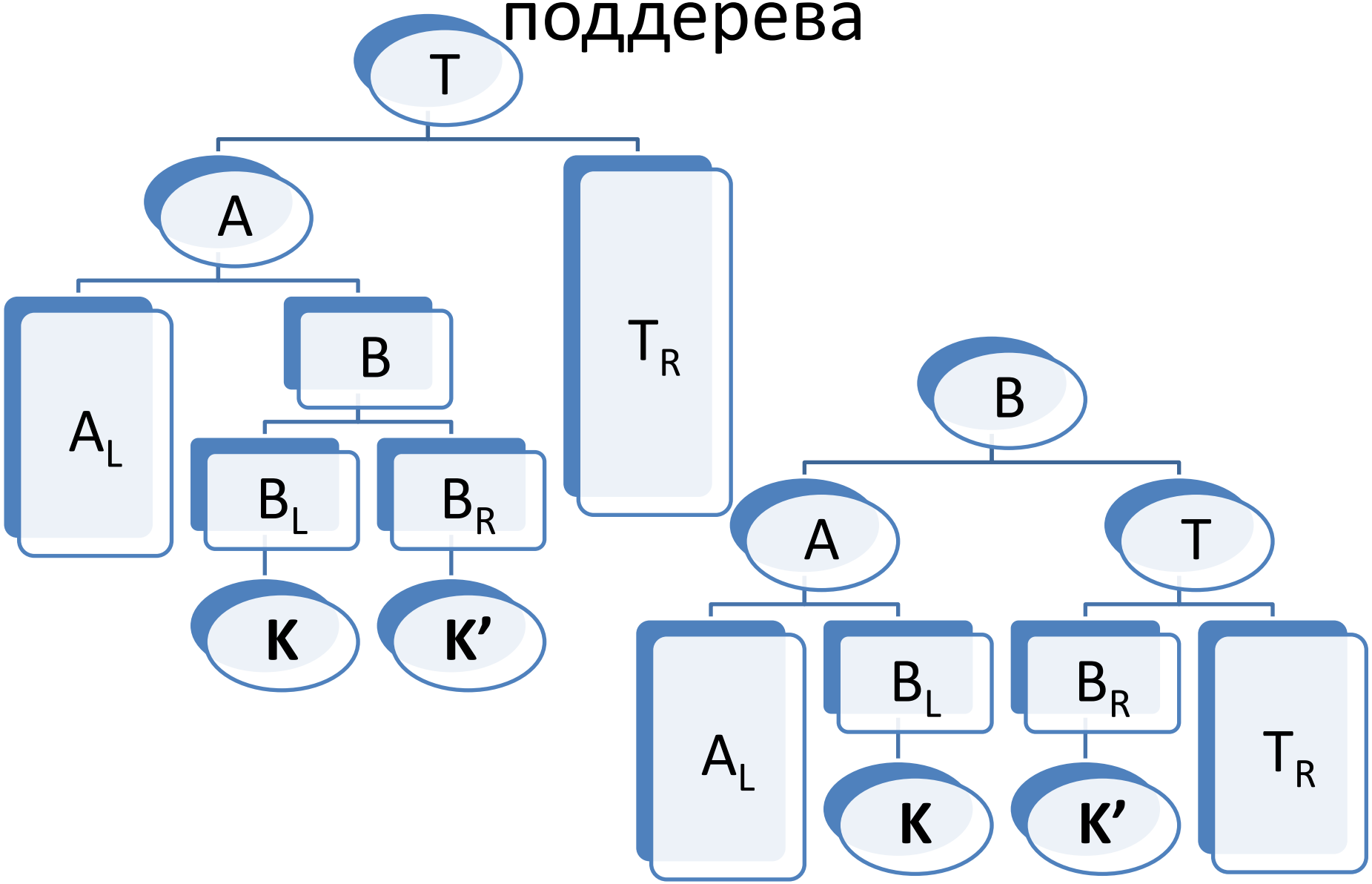
Включение в АВЛ дерево ключа $K<N_T$



Включение левого узла левого поддерева



Включение правого узла левого поддерева



ХЕШ-ТАБЛИЦЫ

Цепочки переполнения.

Открытая адресация.

Сравнение

	Прямая адресация	Хеш-таблицы
Множество возможных значений ключа: U		
Множество реальных значений ключа: $K \subset U$		
Память:	**$O(\|U\|)$**	**$\Theta(\|K\|)$**
Коэффициент заполнения α:		
	$\|K\|/\|U\|$	**$\|K\|/\Theta(\|K\|) = O(1)$**
Время доступа:	**$O(1)$***	**$O(1)$****
Индекс эл-та:	**k**	**h(k)**

- хеш-значение ключа k

*** В худшем случае**

Цель хеш-функции – уменьшить рабочий диапазон значений индексов массива

**** В среднем**

Коллизии

- Хеширование двух (и более) ключей в одну ячейку – называется ***коллизией***.
- Т.к. $|U| > |K|$ возникновение коллизий неизбежно.
- Цель: выбор хеш-функции, приводящей к минимизации количества коллизий и создание метода разрешения возникающих коллизий.

Выбор хеш-функции

- Метод деления

  h(k) = k mod m, где m = |K|

  Плохо: $m = 2^n$

  Удачно: m - простое число

- Метод умножения

  0 < *a* < 1, h(k) = [m*(ka mod 1)]

  Кнут: $a = (\sqrt{5} - 1)/2 \approx 0.6180339887\ldots$

```
void main() { const   K=9,m=11,U=20; List<int> T[m];
for(int i=0,k=9;i<K;i+=2,k+=2) {
                                T[h(i)].put(i); T[h(k)].put(k); }
for(i=0;i<m;i++) { cout << i << ":\t"  << T[i]; } }
```

```
int h(k) {return (k%m);}
```

```
0:  11    0
1:
2:  2     13
3:
4:  4     15
5:
6:  6     17
7:
8:  8
9:  9
10:
```

```
int h(int k) { double a=0.61800339887;
return (int)m*(a*k-int(a*k)); }
```

```
0:  13    0
1:
2:  2     15
3:
4:
5:  4     17
6:  9
7:  6
8:  11
9:
10: 8
```

Открытая адресация

Второй параметр i – номер исследования:

$\{h(k,0), h(k,1), \ldots, h(k,m-1)\} \rightarrow \{0, 1, \ldots, m-1\}$

- Линейное исследование

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

- Квадратичное исследование

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

- Двойное хеширование

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

Пример

```
const  m=13;
int h1(int k) { return k%m; }
int h2(int k) { return 1+k%(m-1); }
int h(int k,int i) { return (h1(k)+i*h2(k))%m; }
void main() { int T[m]; for(int i=0;i<m;i++) T[i]=0;
for(int k=118;k>0;k-=m)
   for(int j=0;j<m;j++) {
        if(!T[i=h(k,j)]) { T[i]=k; break; }
for(i=0;i<m;i++) cout << i << '\t' << T[i] << endl; }
```

Результат:

0	0
1	118
2	0
3	1
4	14
5	27
6	40
7	53
8	66
9	79
10	92
11	105
12	0

Задача number-game-Хэш-таблица

- Вася и Петя играют в игру. Вася произносит числа, а Петя старается их запомнить. Когда Вася произносит последнее число X, Петя должен вспомнить, какое число Y Вася произносил после X в прошлый раз; или сказать что такого числа произнесено не было (-1). Петя знает, что Вася будет называть более-менее случайные числа. Можно считать, что Петя попросил Васю подбирать числа так, чтобы остаток от деления на 100003 совпадал не более чем у 10 различных чисел.
- Вход

Числа от 1 до $2*10^8$, не более 10^7 штук. Из них различных не более 100000.

- Выход

Число, находящееся справа от предпоследнего вхождения последнего входного числа; или -1, если такого вхождения нет.

```
template <typename T, int  N, int M> class Hash { T S[N][2];
    int h1(T k) { return k%N; }
    int h2(T k) { return 1+k%M; }
    int h(int k,int i) { return (h1(k)+i*h2(k))%N; }
public:
    Hash() { for(int i=0;i<N;i++) S[i][1]=S[i][2]=0; }
    bool ins(T k, T z) { for(int j,i=0;i<M;i++)
                    if(S[j=h(k,i)][1]<1) { S[j][1]=k; S[j][2]=z; return true; }
                    return false; }
    T find(T k) { for(int j,i=0;i<M;i++) if(S[j=h(k,i)][1]==k) return S[j][2];
                                                else if(!S[j][1]) return -1;
                    return -1; }
   bool del(T k) { for(int j,i=0;i<M;i++)
                     if(S[j=h(k,i)][1]==k) { S[j][1]=-1; return true; }
                     else if(!S[j][1]) return false; } };
```

```
int main() {
Hash<int,200000,100003> T;
int k,i;
    cin >> k;
    while(cin >> i) { T.del(k); T.ins(k,i); k=i; }
    k=T.find(i);
    cout << k << endl;
    return 0; }
```