Introduction to Algorithms and Data Structures

Lesson 8: Data Structure (2) Operations on linked lists, and Binary Search Tree

Professor Ryuhei Uehara,

School of Information Science, JAIST, Japan.

uehara@jaist.ac.jp

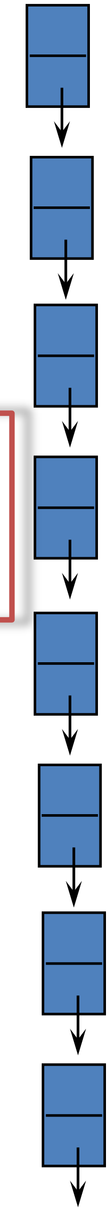
http://www.jaist.ac.jp/~uehara

Example of Data structures × Algorithms

- Usually, we can choose some data structure, e.g.,
  - array
  - linked list

  for the implementation of the same algorithm.
- Efficiency depends on “data structure” vs “basic operations” you will use on the data.
  - When we “add” and “remove” data, linked list is much better than array, and tree structure is much better than linked list (I’ll explain, say, at the last lesson?)
- We will show some simple examples

Sequential search by linked list



- Find x in the linked list from the top of linked list
  - It contains x ➔ address of the record
  - It doesn't contain x ➔ NULL

Satisfied?

YES

```
typedef struct{
  double data;
  struct list_t *next;
} list_t;
p = head;
while(p != NULL && p->data != x)
   p = p->next;
return p;
```

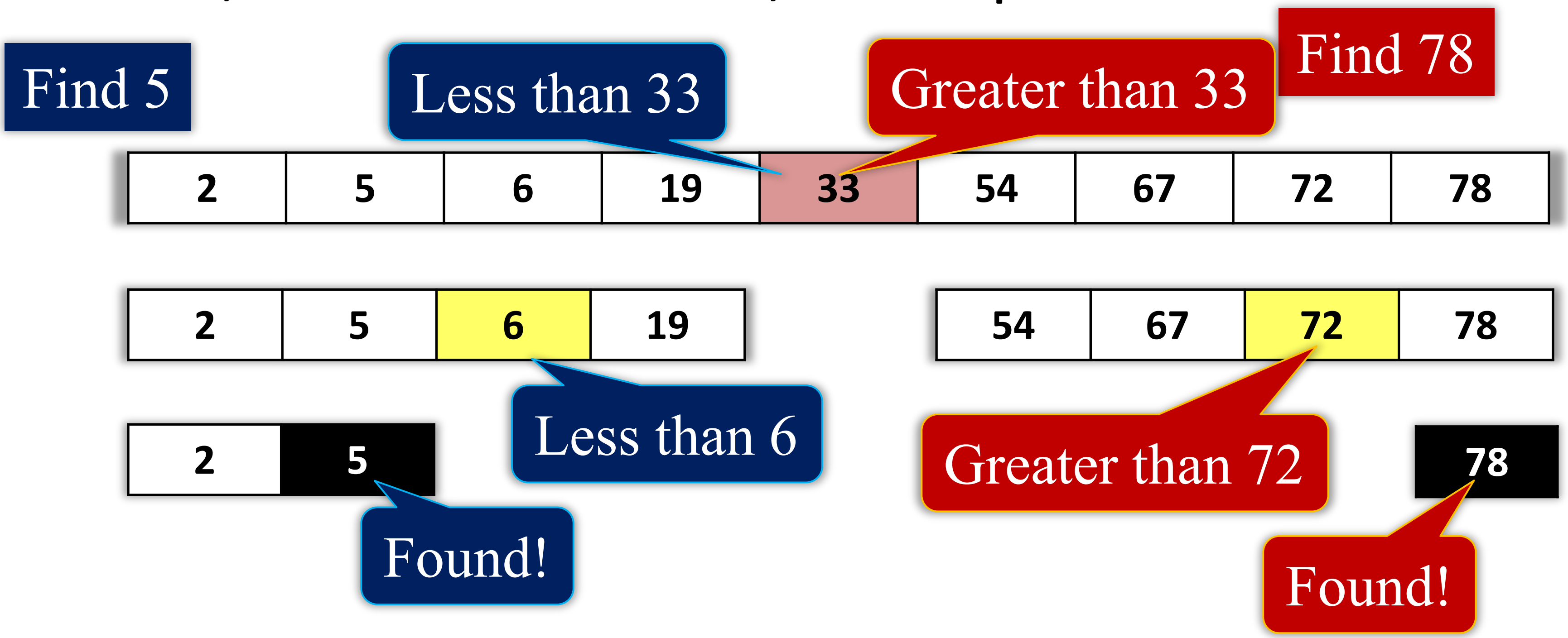
p==NULL or p->data == x it exits

Is this correct?

YES

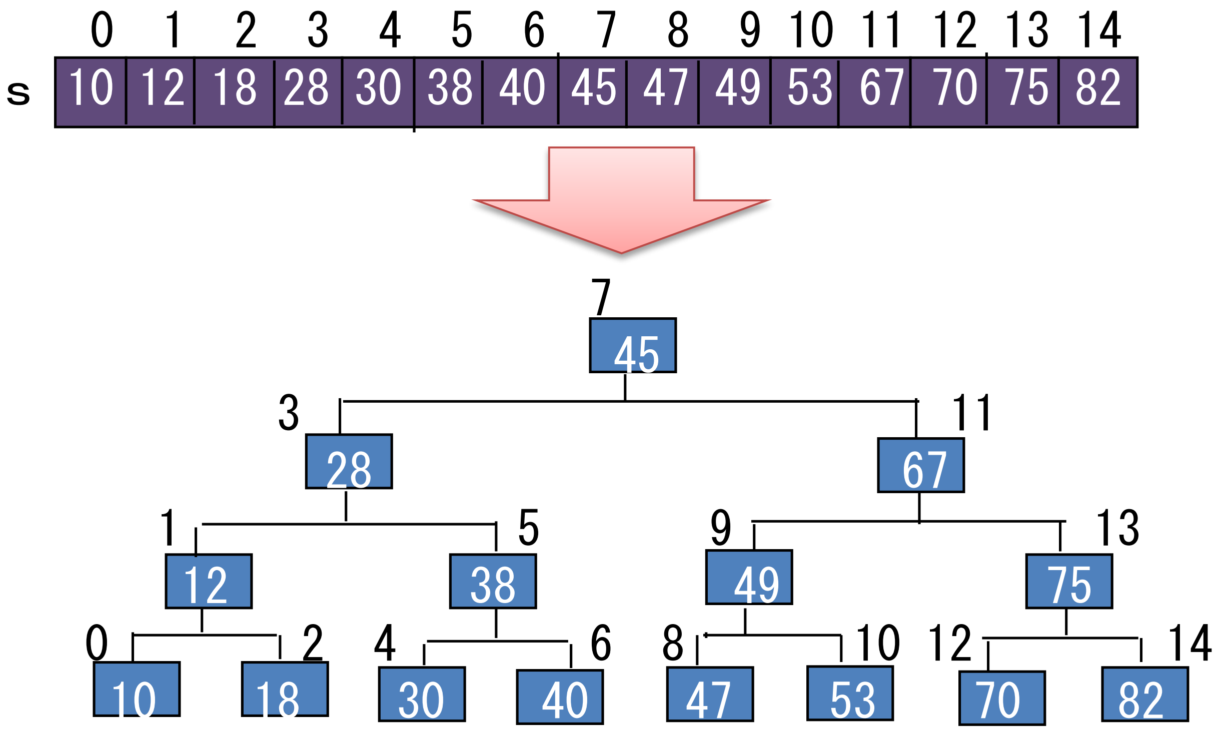
Binary search method

- Search, divide into halves, and repeat to find



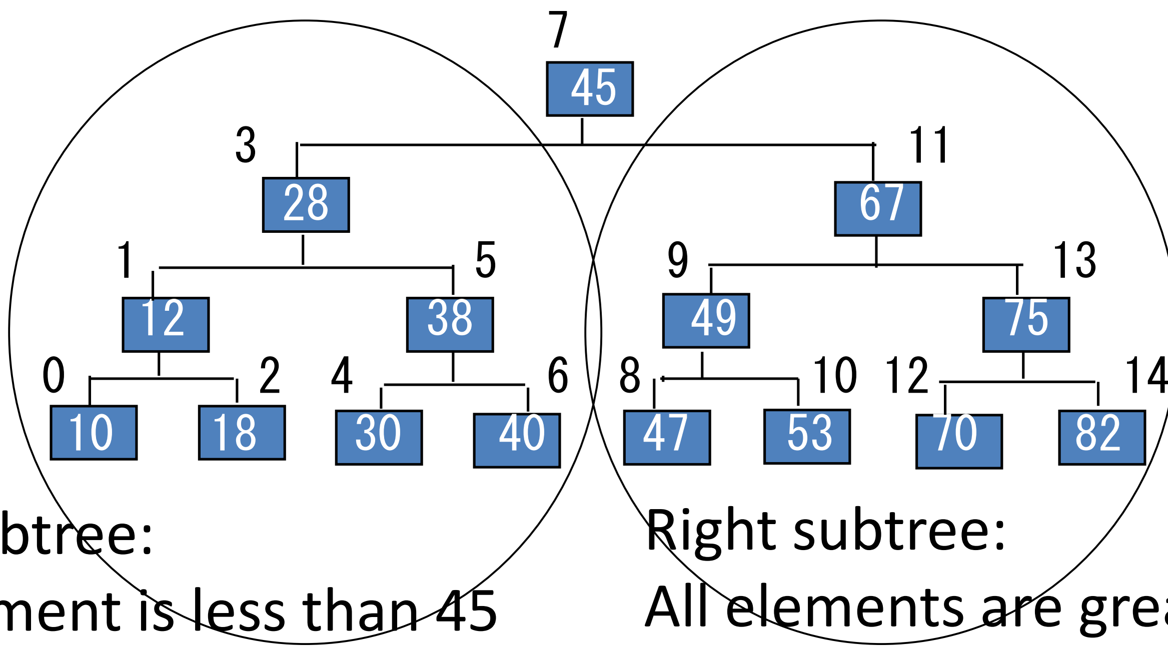
  - Key issue: Divide at the center point.

Binary search tree: data structure of binary search



- When data size is fixed, we can compute the central positions beforehand

Property of binary search tree



Left subtree:
All element is less than 45

Right subtree:
All elements are greater than 45

- In general, for a node n,
  - All elements in right subtree are greater than (or equal to) n
  - All elements in left subtree are less than (or equal to) n

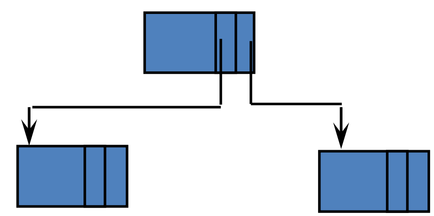
Search in binary search tree

```
BSTnode *root, *v;
x=/*some value*/;
v = root;
while( v ){
  if(v->data == x)
    break;
  if(v->data > x)
    v = v->lson;
  else
    v = v->rson;
}
return v;
```

Left if small

Right if large

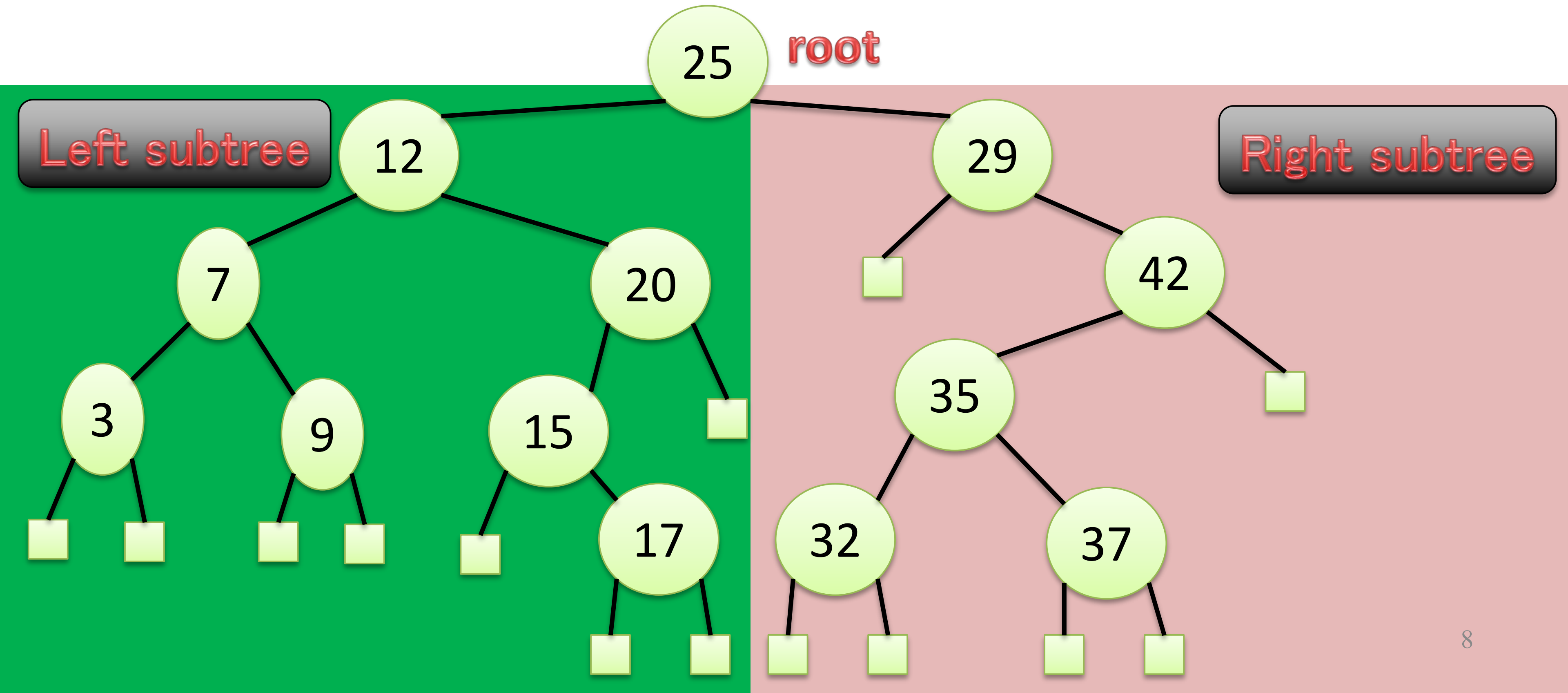
```
typedef struct{
  int data;
  struct BSTnode
    *lson, *rson;
}BSTnode;
```



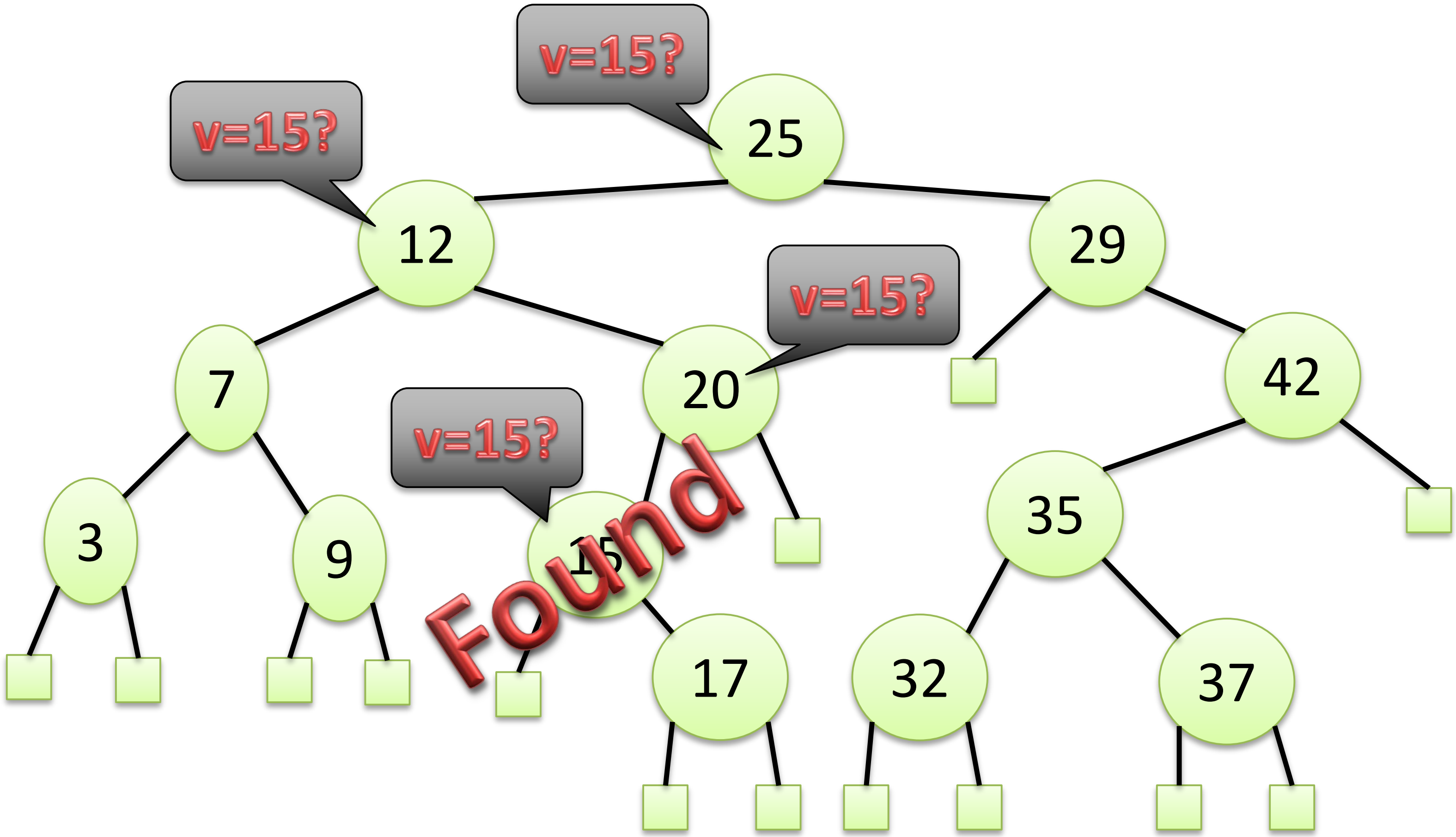
Each record has two pointers to left child and right child.

Consider: binary search tree

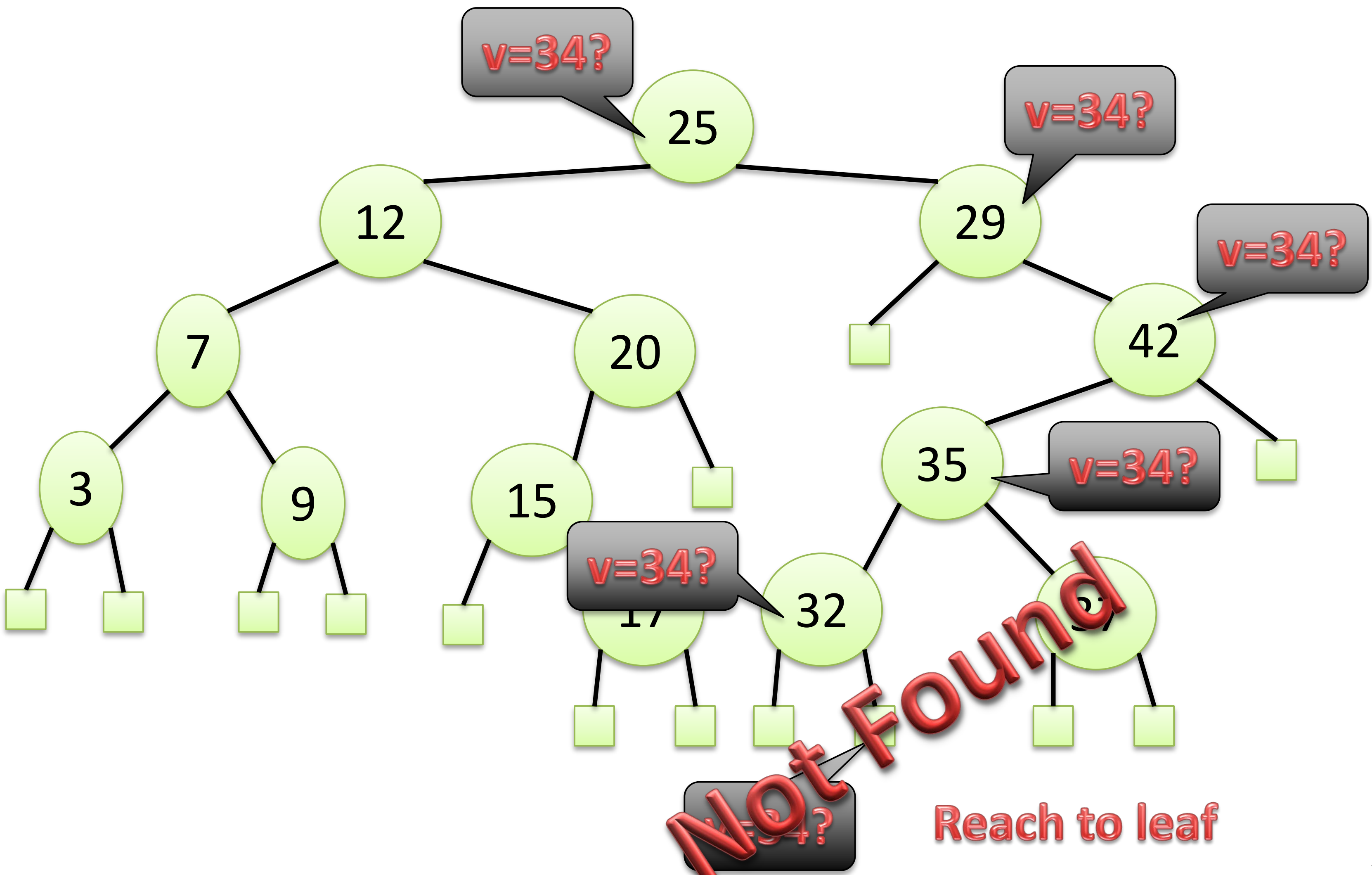
- On a binary search tree, it holds for each vertex v;
  - data in v > each data in left subtree of v
  - data in v < each data in right subtree of v



Search in binary search tree: case v=15



Search in binary search tree: case v=34

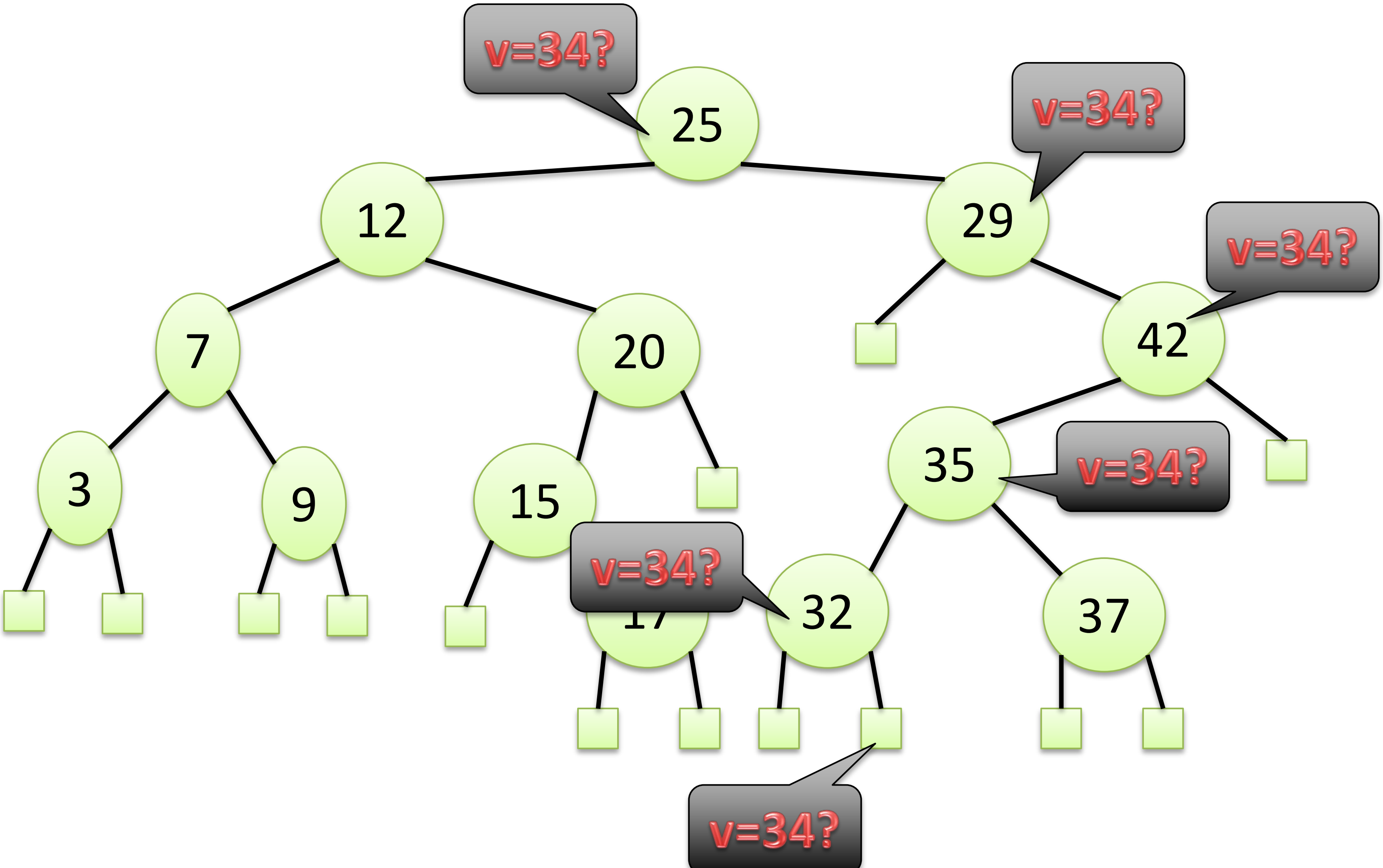


Add a data to binary search tree

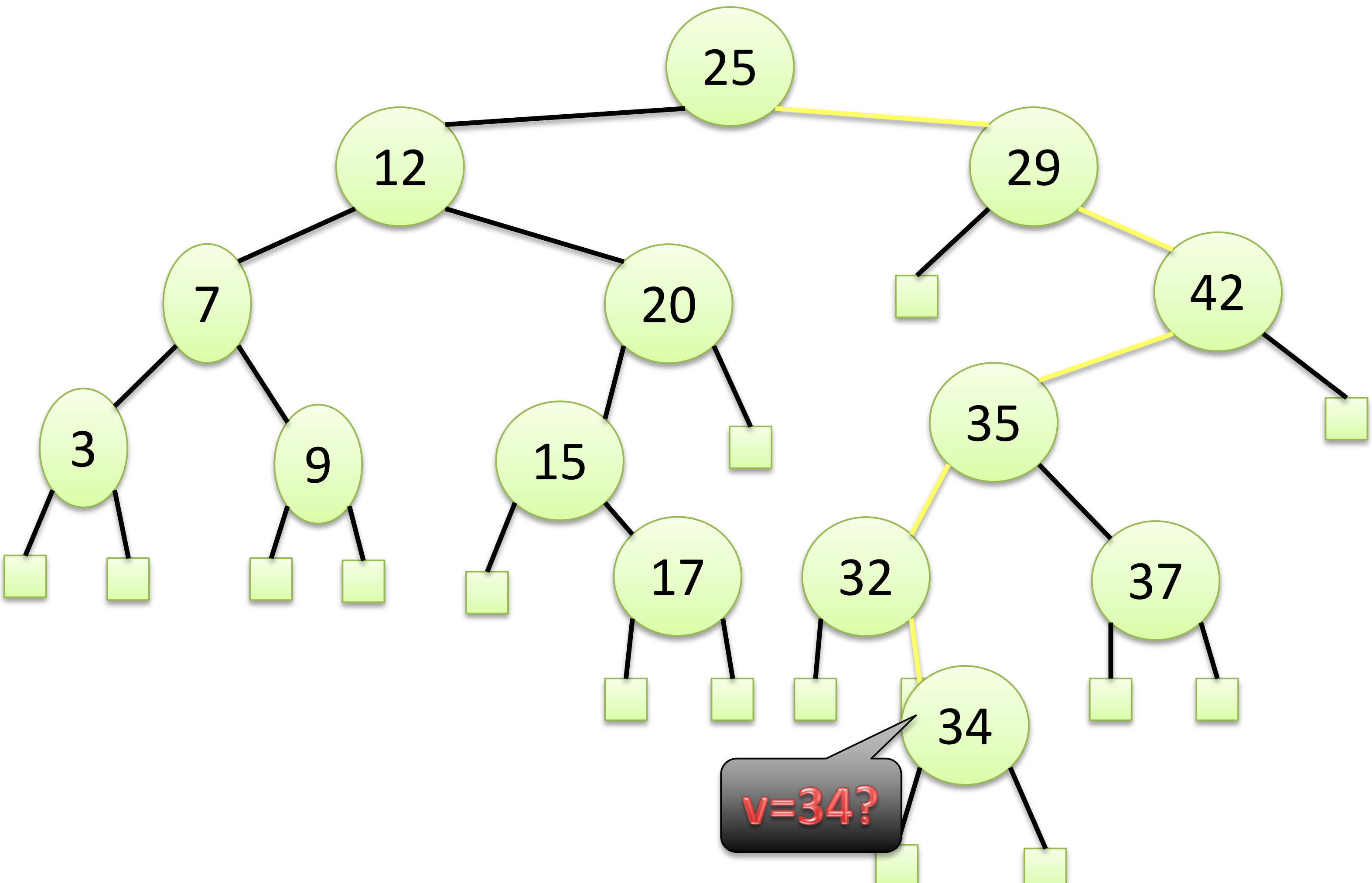
- Perform binary search from the root
- If it reach to the leaf, store data on it

```
insert(x,tree){
  v ← root(tree);
  while(v is not a leaf){
     if( x <= data(v) ) then
       v ← left child of v;
     else
       v ← right child of v;
  }
  make a node v at the leaf;
  data(v) ← x;
}
```

Add a data to binary search tree
Example: add x=34



Add a data to binary search tree
Example: add x=34



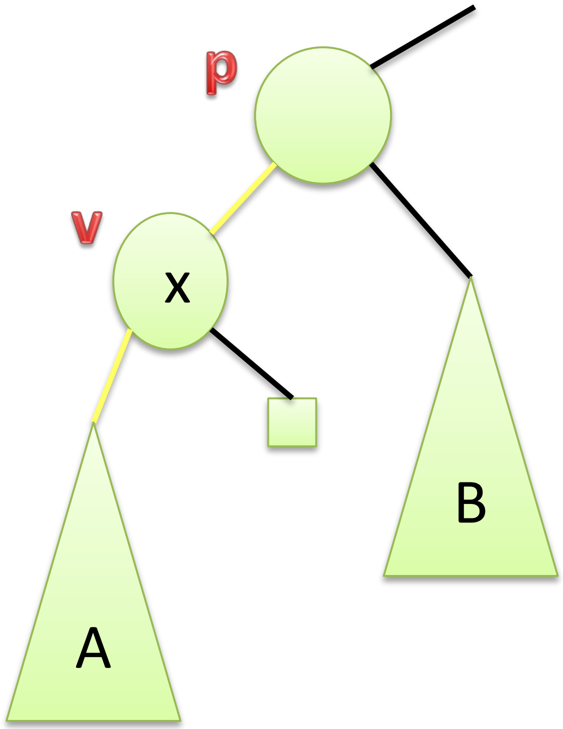
Add a data to binary search tree (cnt'd)

```
void insert(tree *p, int x){
  if(p == NULL){
    p = (tree*) malloc( sizeof(tree) );
    p->key = x;
    p->lchild =NULL; p->rchild = NULL;
  }else
    if( p->key < x )
      insert( p->rchild, x);
    else
      insert( p->lchild, x);
}
```

How to call: `insert(root,x)` ← Pointer to root

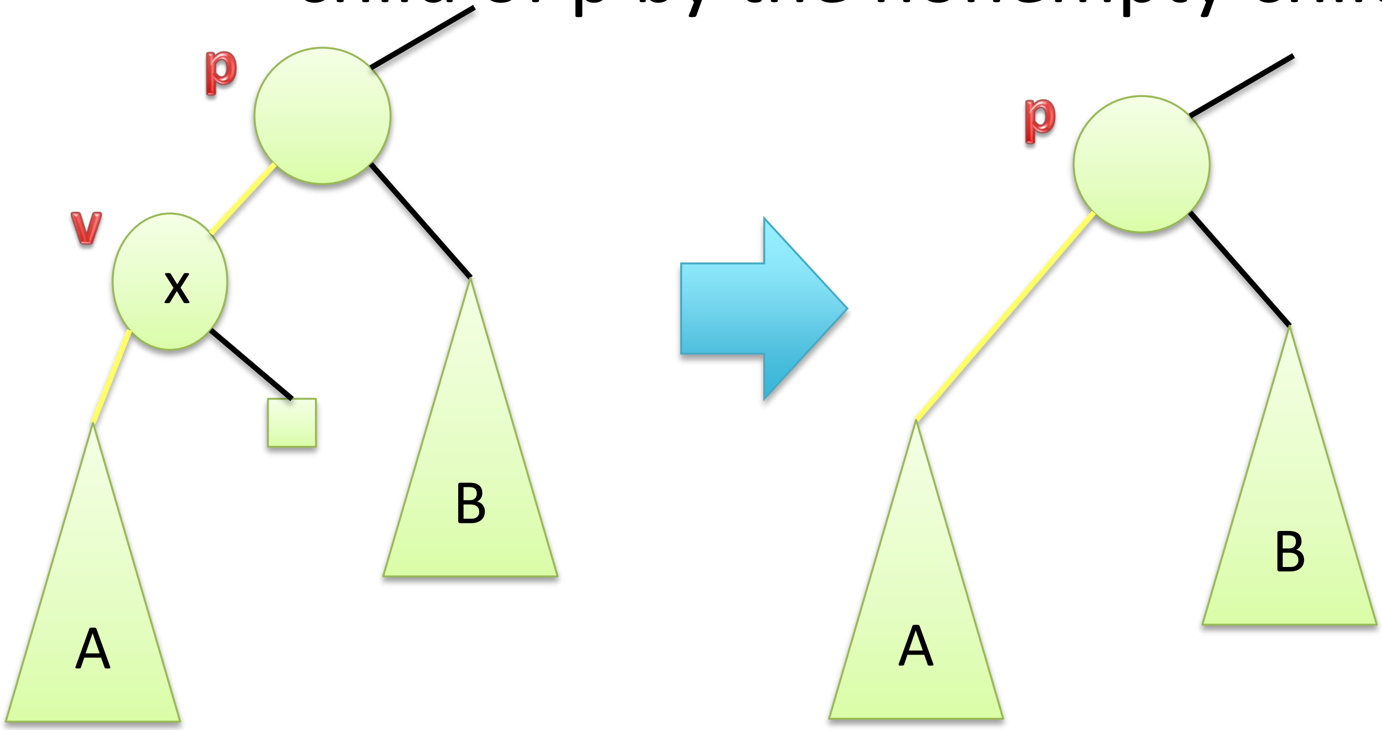
Remove a data to binary search tree : find a vertex of data x, and remove it!

- Case analysis based on the vertex v that has data x
  - Case 0. v has two leaves;
    - This is easy; just remove v!
  - Case 1. v has one leaf
  - Case 2. v has no leaves



Remove a data to binary search tree: Case 1. v has one leaf

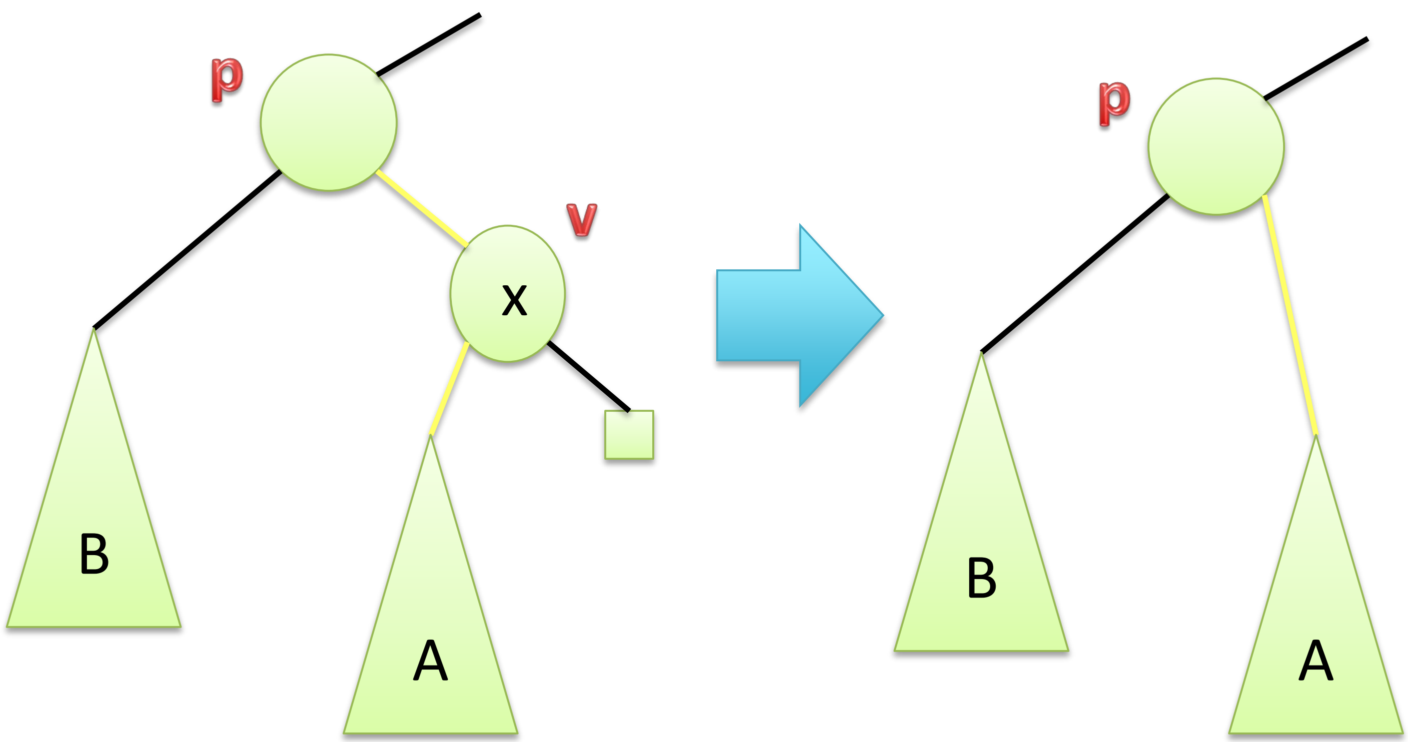
(1a) v is left child of parent p: update the left child of p by the nonempty child of v



Report 3. Is property of binary search tree OK?

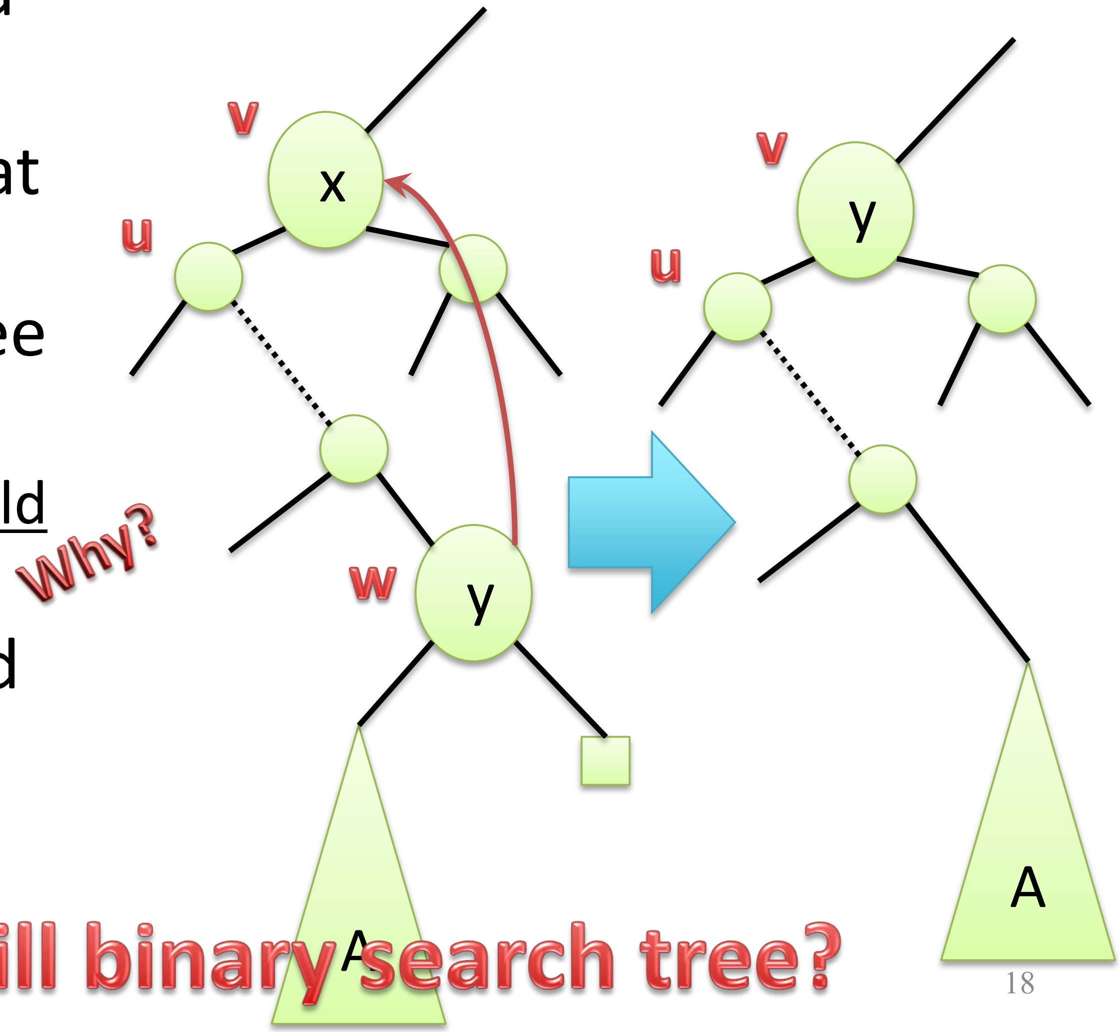
Remove a data to binary search tree: Case 1. v has one leaf

(1b) v is right child of parent p: update the right child of p by the nonempty child of v



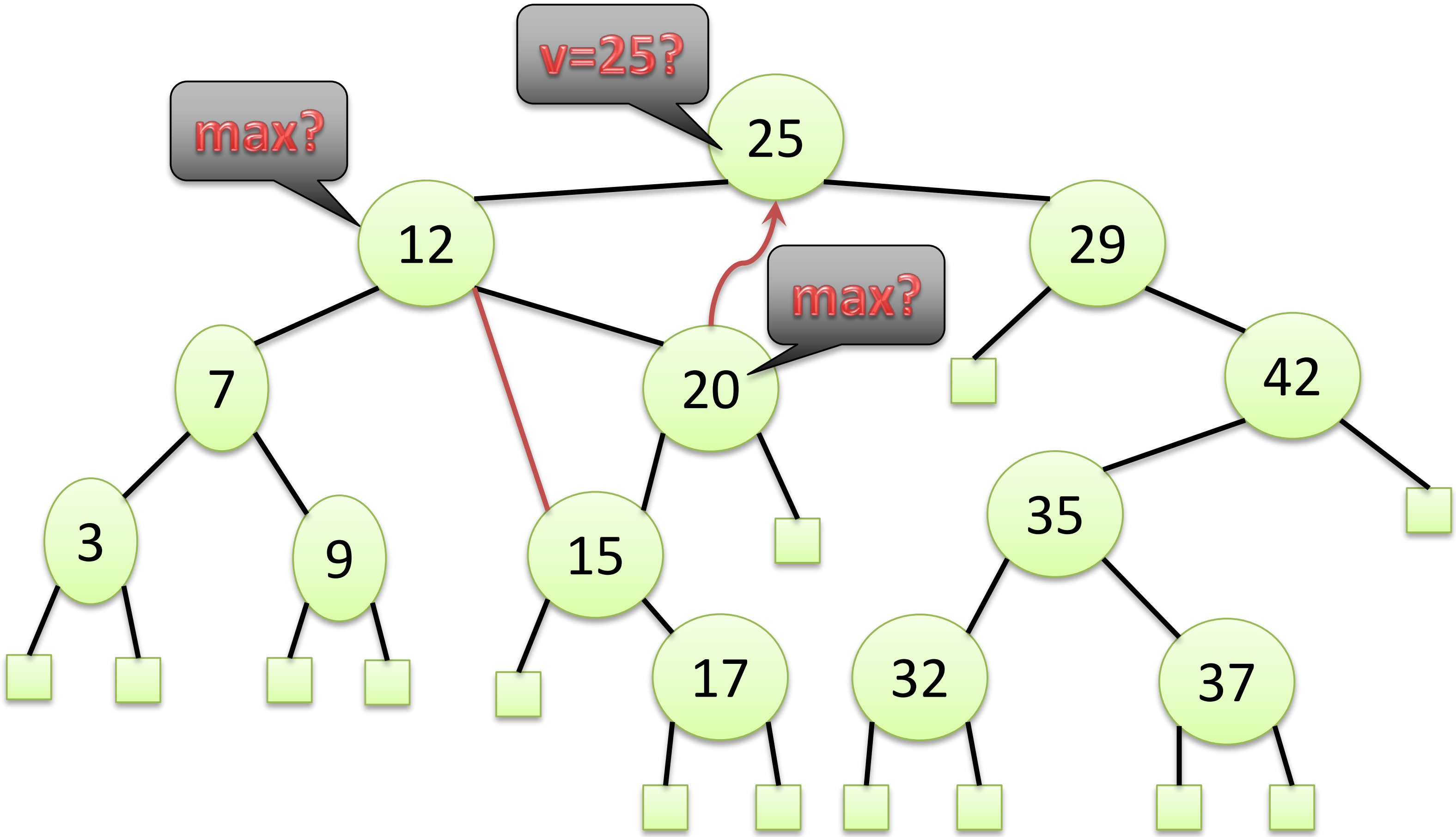
Remove a data to binary search tree : Case 2. v has no leaves

- Let u be the left child of v.
- Find the vertex w that has the maximum value w in the subtree rooted at u.
  - Right child of w <u>should</u> be a leaf
- Value y in w is copied to v, and remove w.
  - As same as case 1

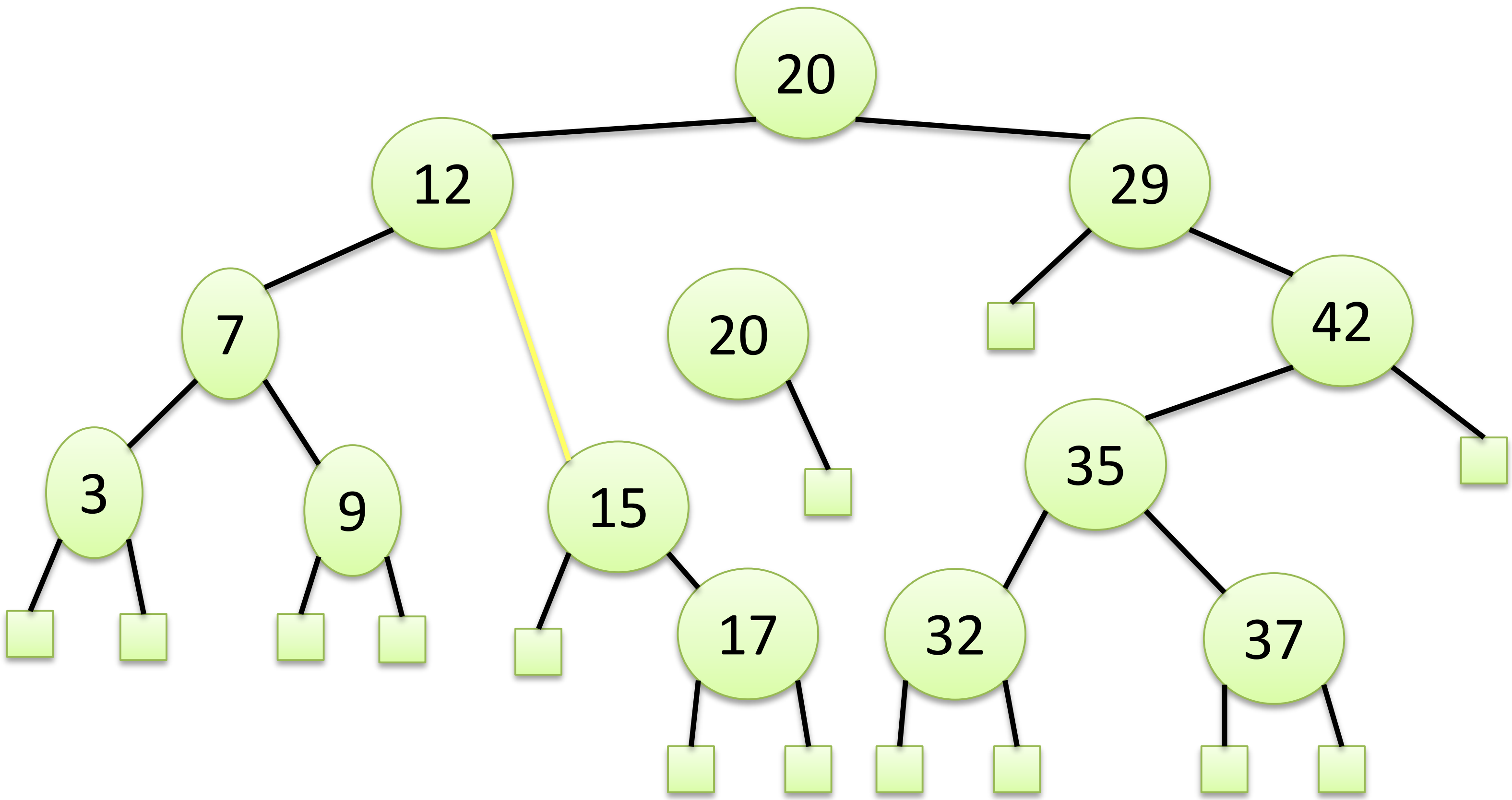


Report 3. Is this still binary search tree?

Remove a data to binary search tree : Remove x=25



Remove a data to binary search tree : Remove x=25



Some comments

- The shape of binary search tree depends on
  - Initial sequence of data
  - Ordering of adding/removing data
- So, it may be a quite unbalanced tree if these ordering is not good…
  - If you can hope that it is “random”, the expected level of tree is $O(\log n)$.
  - If you may have quite unbalanced data, the level can be $\Theta(n)$. (In this case, it is almost the same as a linked list.)