Introduction to
Algorithms and Data Structures

Lecture 13: Data Structure (4)
Data structures for graphs
and example in binary search tree

Professor Ryuhei Uehara,

School of Information Science, JAIST, Japan.

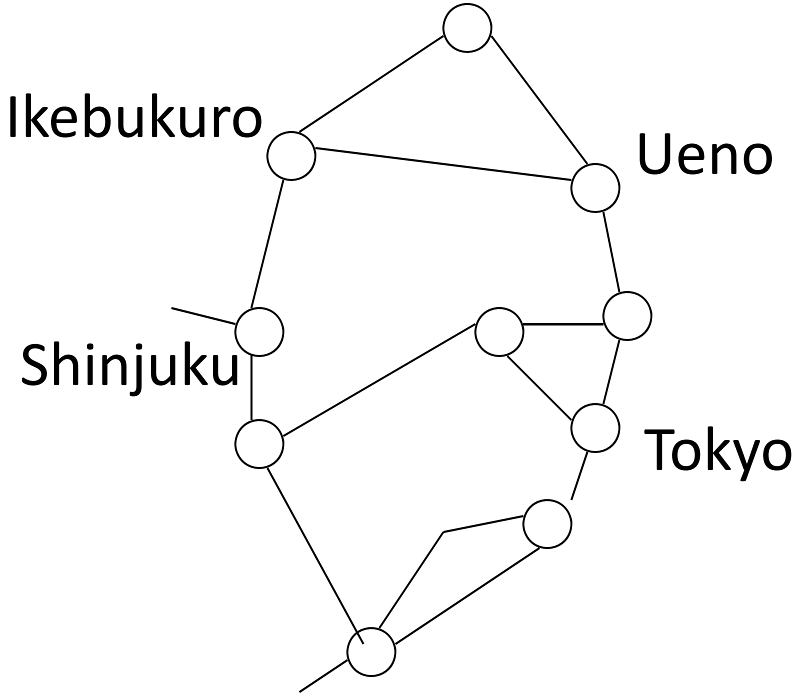
uehara@jaist.ac.jp

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

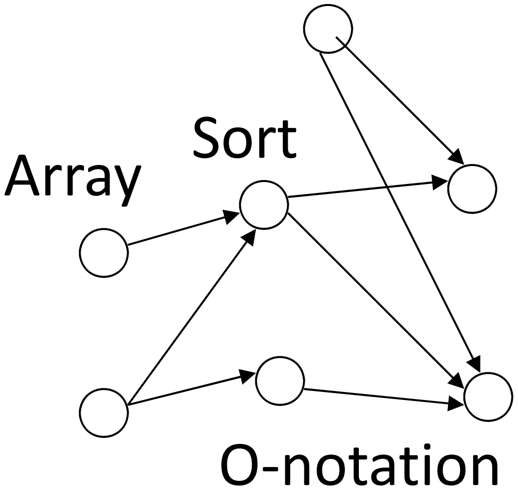
Graph

- "<u>Vertices</u>" (nodes) are joined by <u>edges</u> (arcs)
  - – Directed graph: each edge has direction
  - – Undirected graph: each edge has no direction

Example: railway in Tokyo



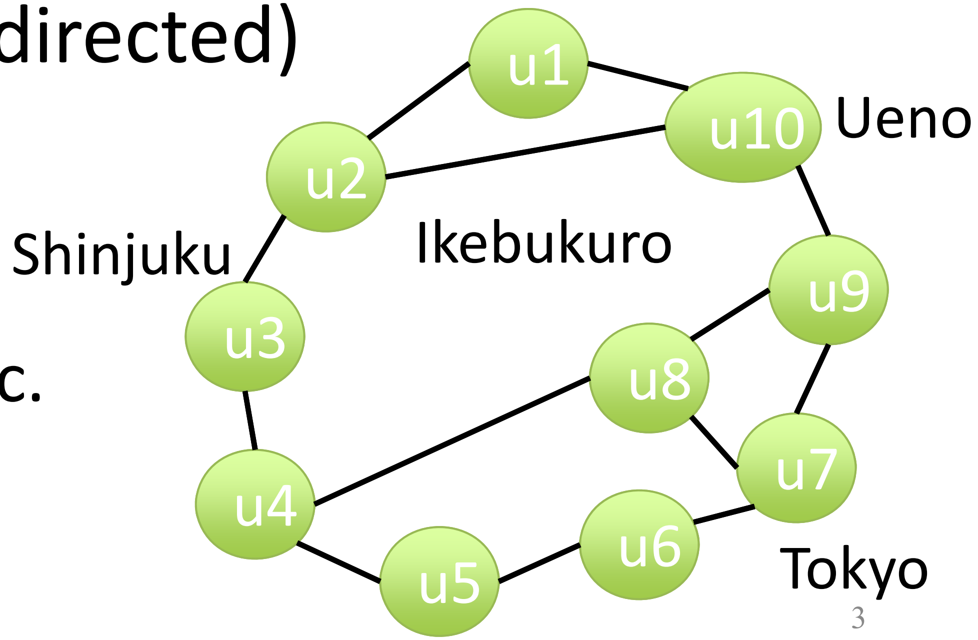
Example:
relationship between topics



Graph: Notation

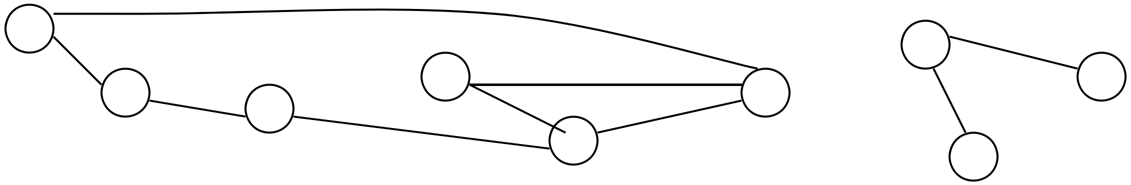
- Graph $G = (V, E)$
  - V: vertex set, E: edge set
- Vertices: $u, v, \ldots \in V$
- Edges: $e = \{u, v\} \in E$ (undirected)

  $a = (u, v) \in E$ (directed)
- Weighted variants;
  - $w(u)$, $w(e)$
  - Distance, cost, time, etc.

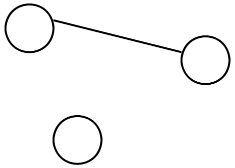


Graph: basic notions/notations (1/2)

- Path: sequence of vertices joined by edges
  - Simple path: it never visit the same vertex again

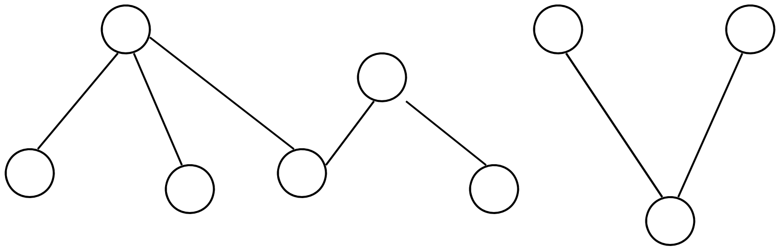


- Cycle, closed path: path from *v* to *v*
- Connected graph: Every pair of vertices is joined by path

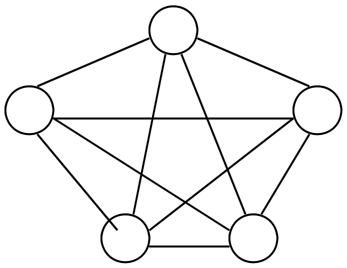


Graph: basic notions/notations (2/2)

- Forest: Graph with no cycle
- Tree: Connected, and no cycle



- Complete graph: Every pair of vertices is connected by an edge
  - Example: K_5



Computational complexity of graph problems

- The number n of vertices, the number m of edges;
  - Undirected graph: $m \leqq n(n-1)/2$
  - Directed graph: $m \leqq n(n-1)$
    - $m \in O(n^2)$

- Every tree has $m=n-1$ edges, so $m \in O(n)$.

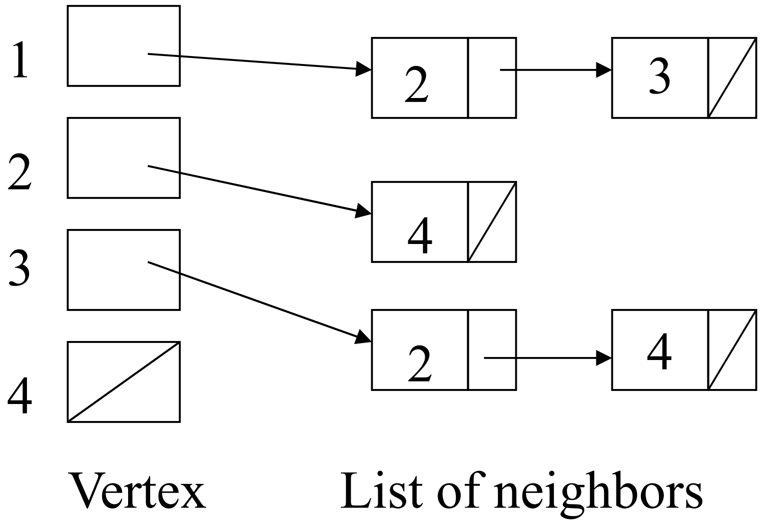
- Computational complexity of graph algorithm is described by equations of n and m.

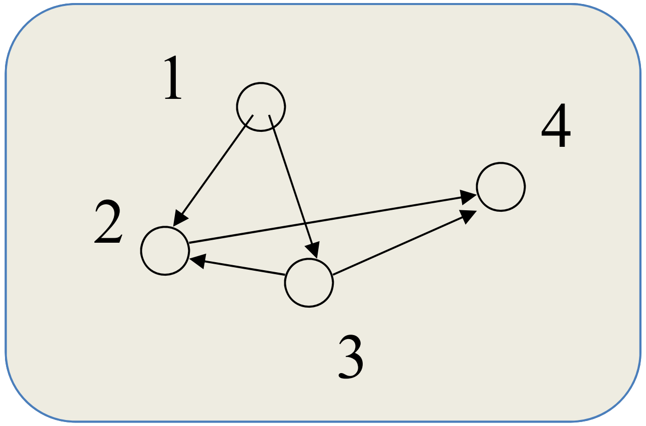
Representations of a graph in computer

- Adjacency matrix

$$M = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

- Adjacency list



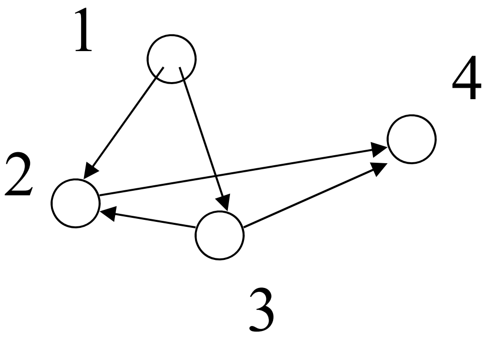


Representation of a graph:

matrix representation (adjacency matrix)

- $(u, v) \in E \Rightarrow M[u, v] = 1$
- $(u, v) \notin E \Rightarrow M[u, v] = 0$

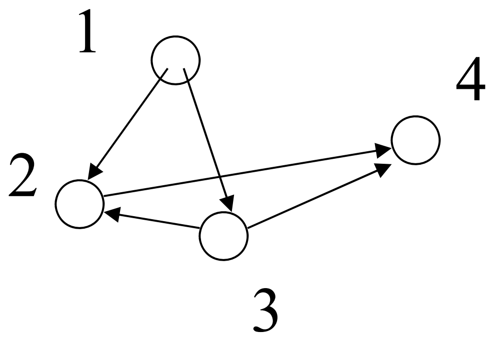
It is easy to extend edge-weighted graph.



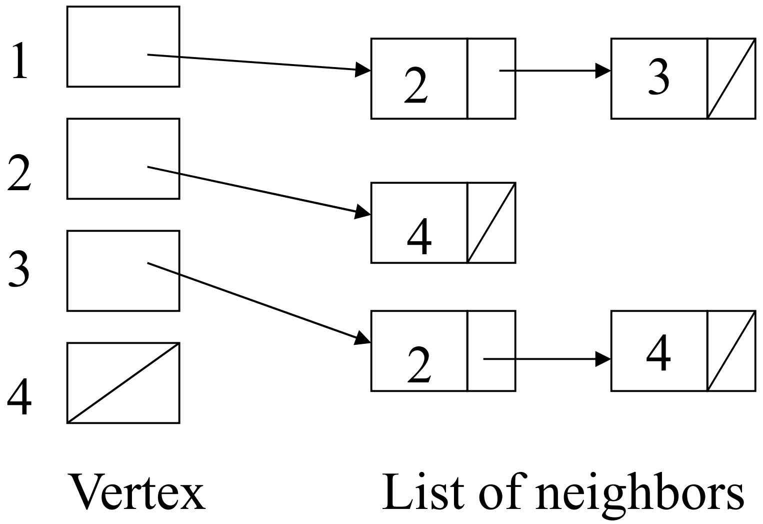
$$M = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Representation of a graph: list representation (adjacency list)

- $(u, v) \in E \Leftrightarrow v \in L(u)$
  - $L(u)$ is the list of neighbors of u



It is easy to extend vertex-weighted graph.



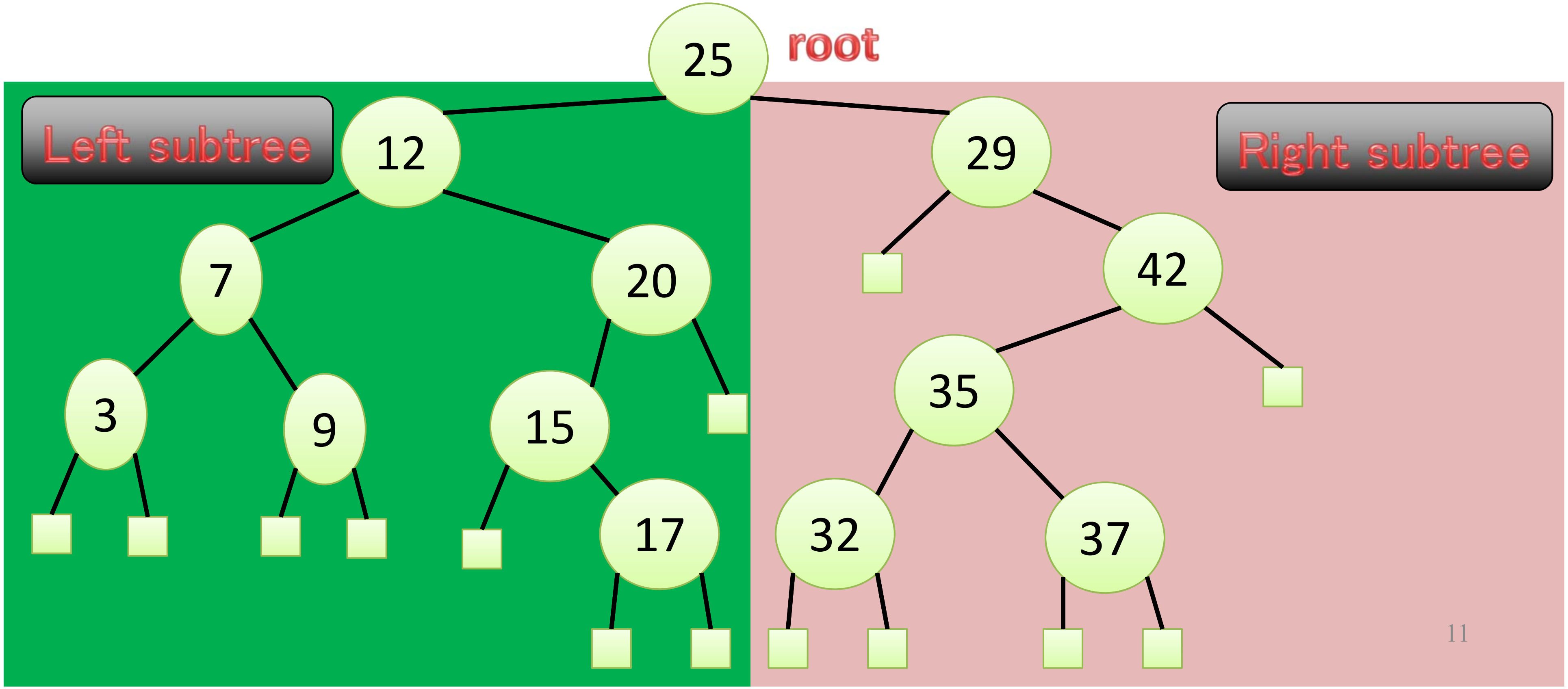
Adj. matrix v.s. Adj. list

- Space complexity
  - Adjacency matrix: $\Theta(n^2)$
  - Adjacency list: $\Theta(m \log n)$
- Time complexity of checking if $(u, v) \in E$?
  - Adjacency matrix: $\Theta(1)$
  - Adjacency list : $\Theta(n)$

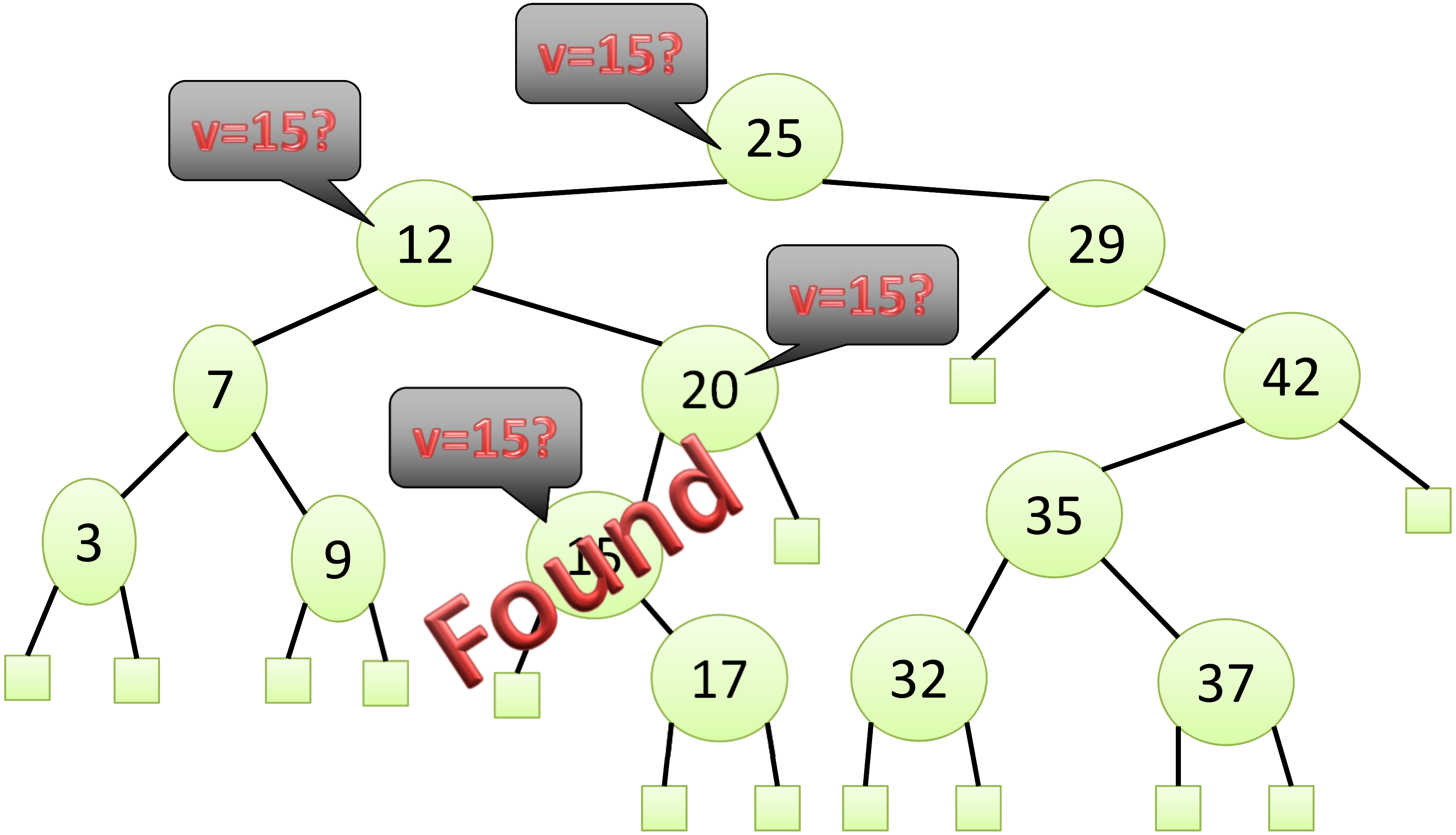
Q. How about update graph?
(e.g., add/remove vertex/edge)

Example: binary search tree

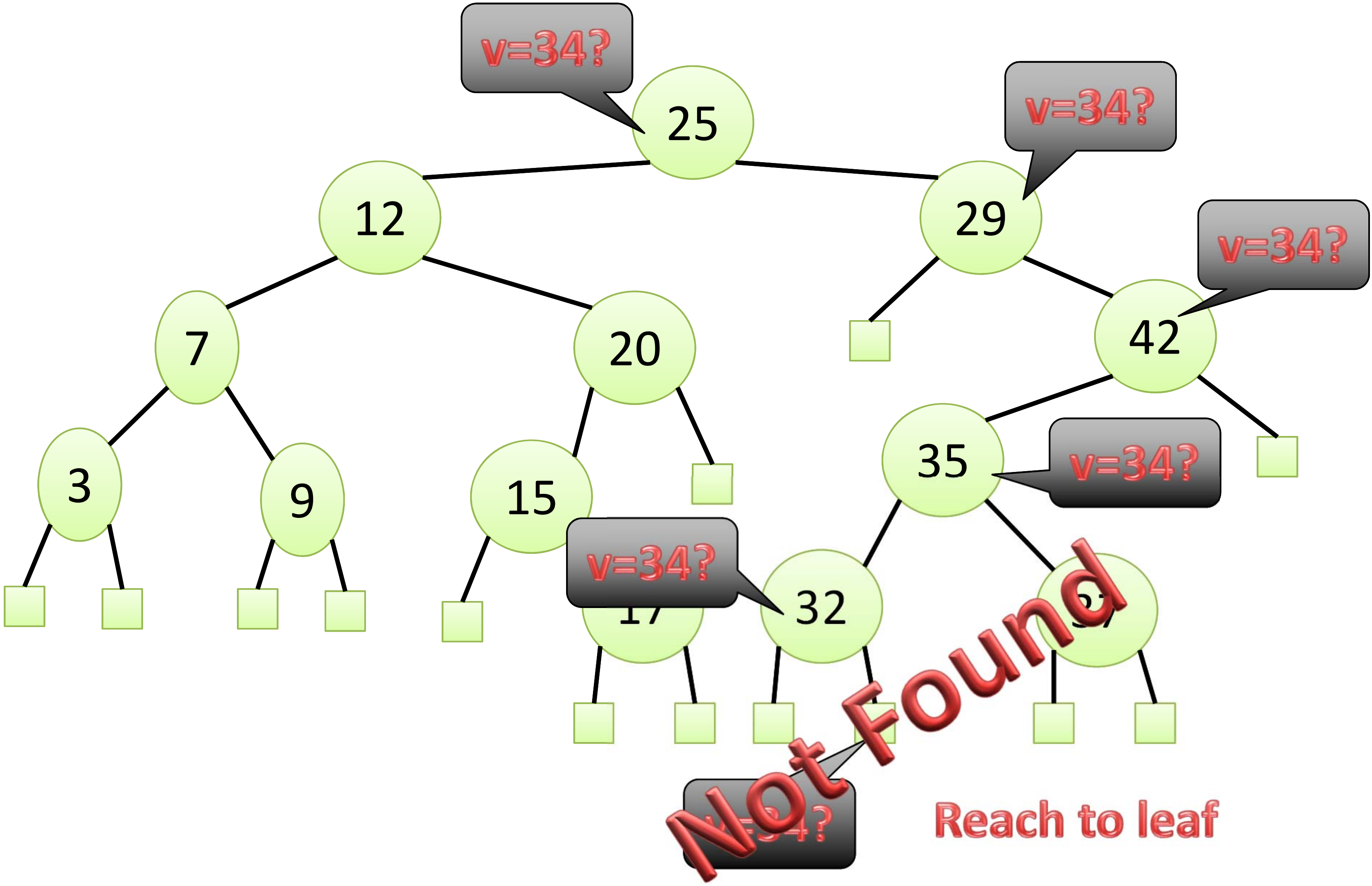
- On a binary search tree, it holds for each vertex v;
  - data in v > each data in left subtree of v
  - data in v < each data in right subtree of v



Search in binary search tree: case v=15



Search in binary search tree: case v=34



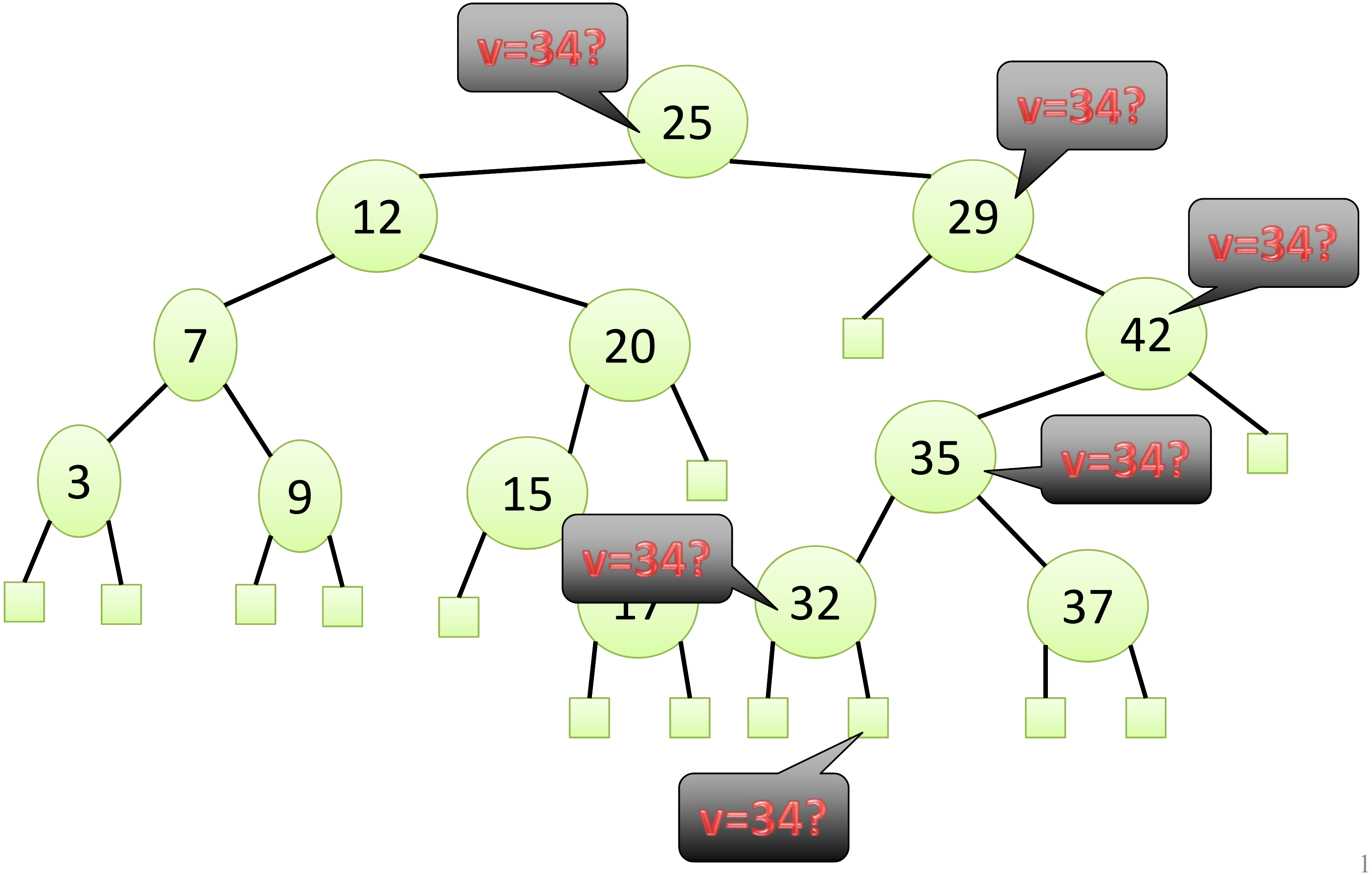
Reach to leaf

Add a data to binary search tree

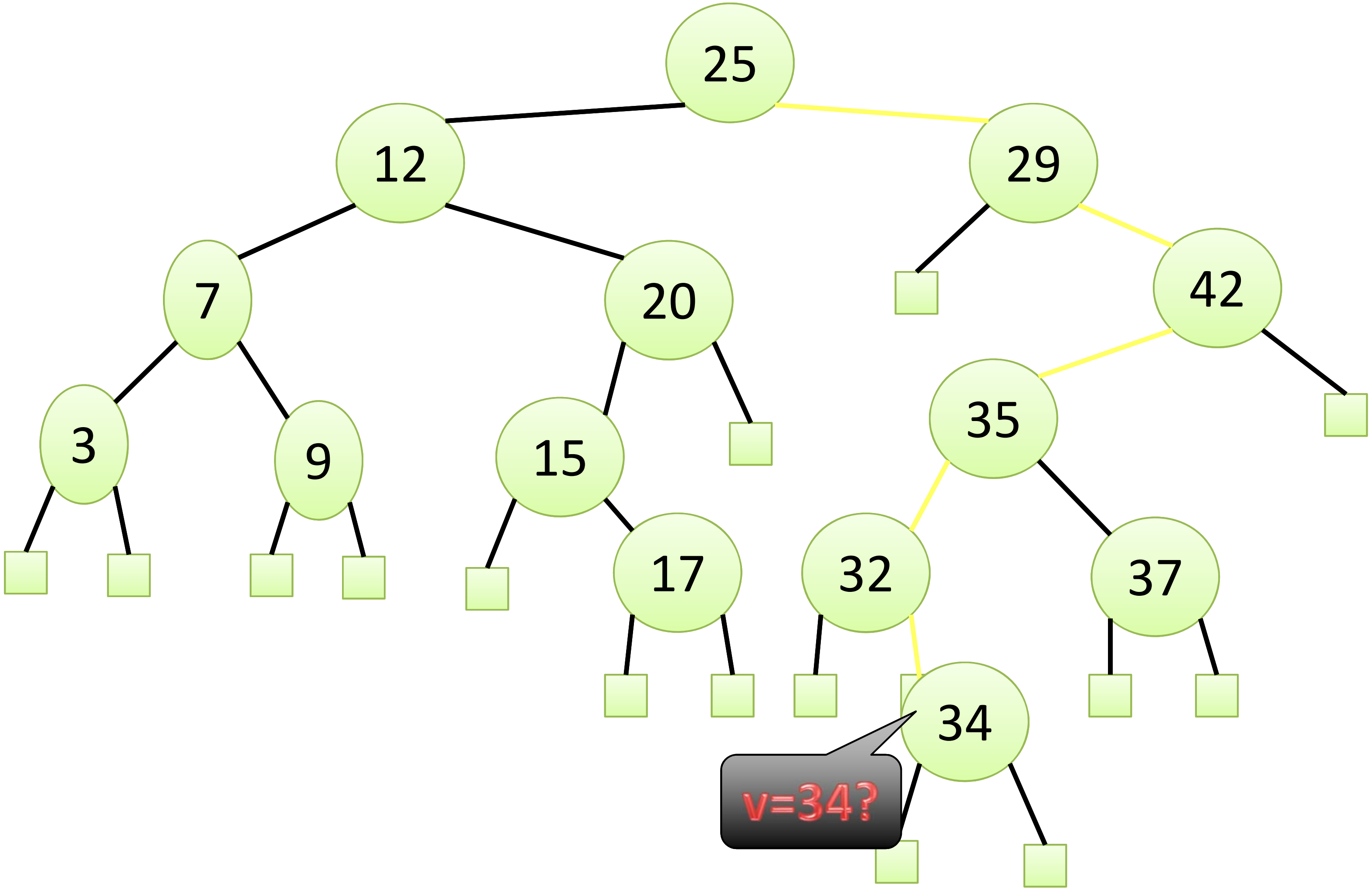
- Perform binary search from the root
- If it reach to the leaf, store data on it

```
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)

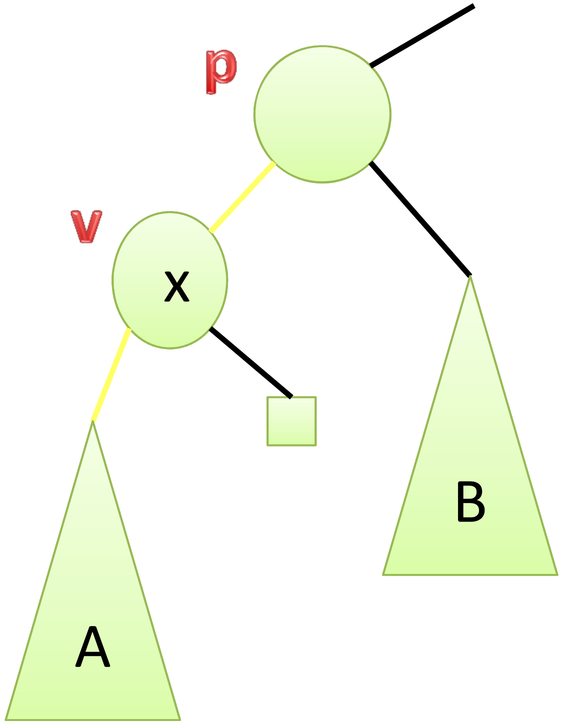
```
void insert(tree *p, int x){
  if(p == NULL){
    p = (tree*) malloc( sizeof(tree) );
    p->key = x;
    p->lchild =NULL; p->rchild = NULL;
  }else
    if( p->key < x )
      insert( p->rchild, x);
    else
      insert( p->lchild, x);
}
```

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

Pointer to root

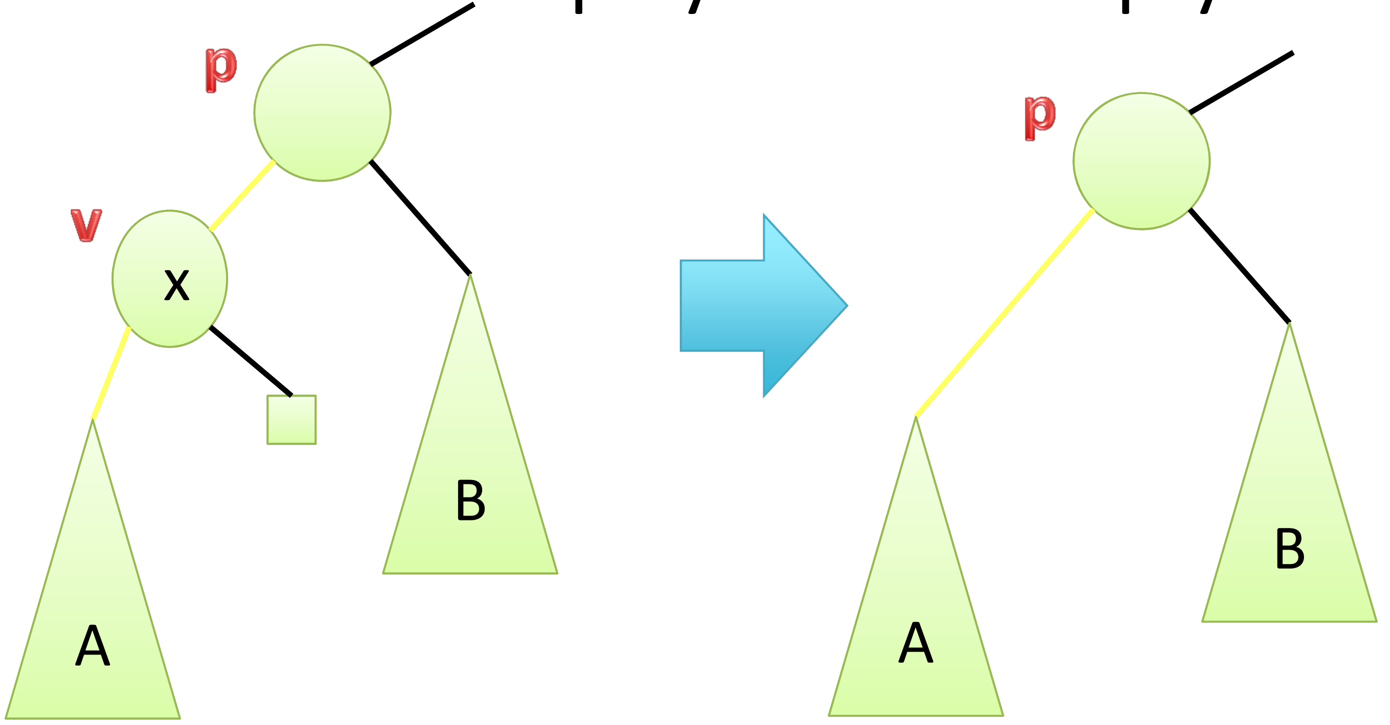
Remove a data to binary search tree : find a vertex of data x, and remove it!

- 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

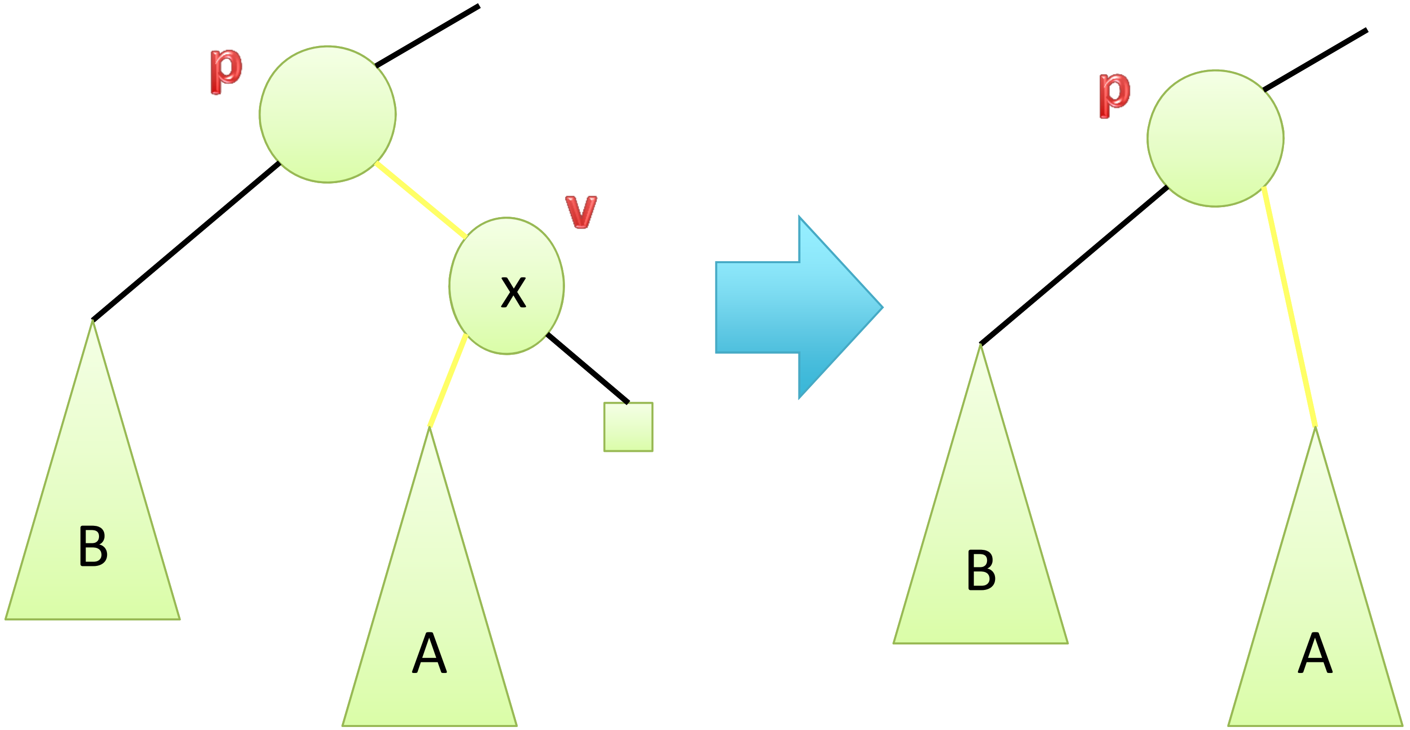
(1a) v is left child of parent p: update the left child of p by the nonempty child of v



Q. Is property of binary search tree OK?

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

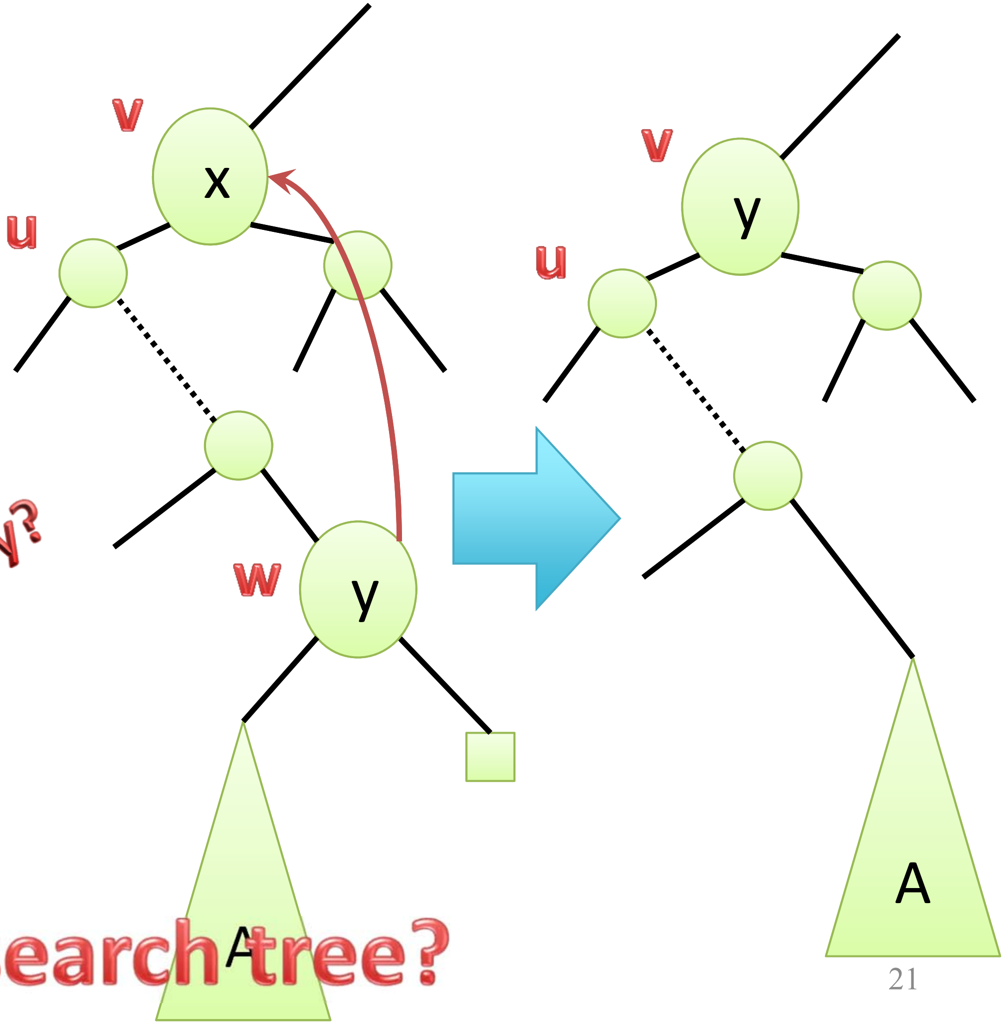
(1b) v is right child of parent p: update the right child of p by the nonempty child of v



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

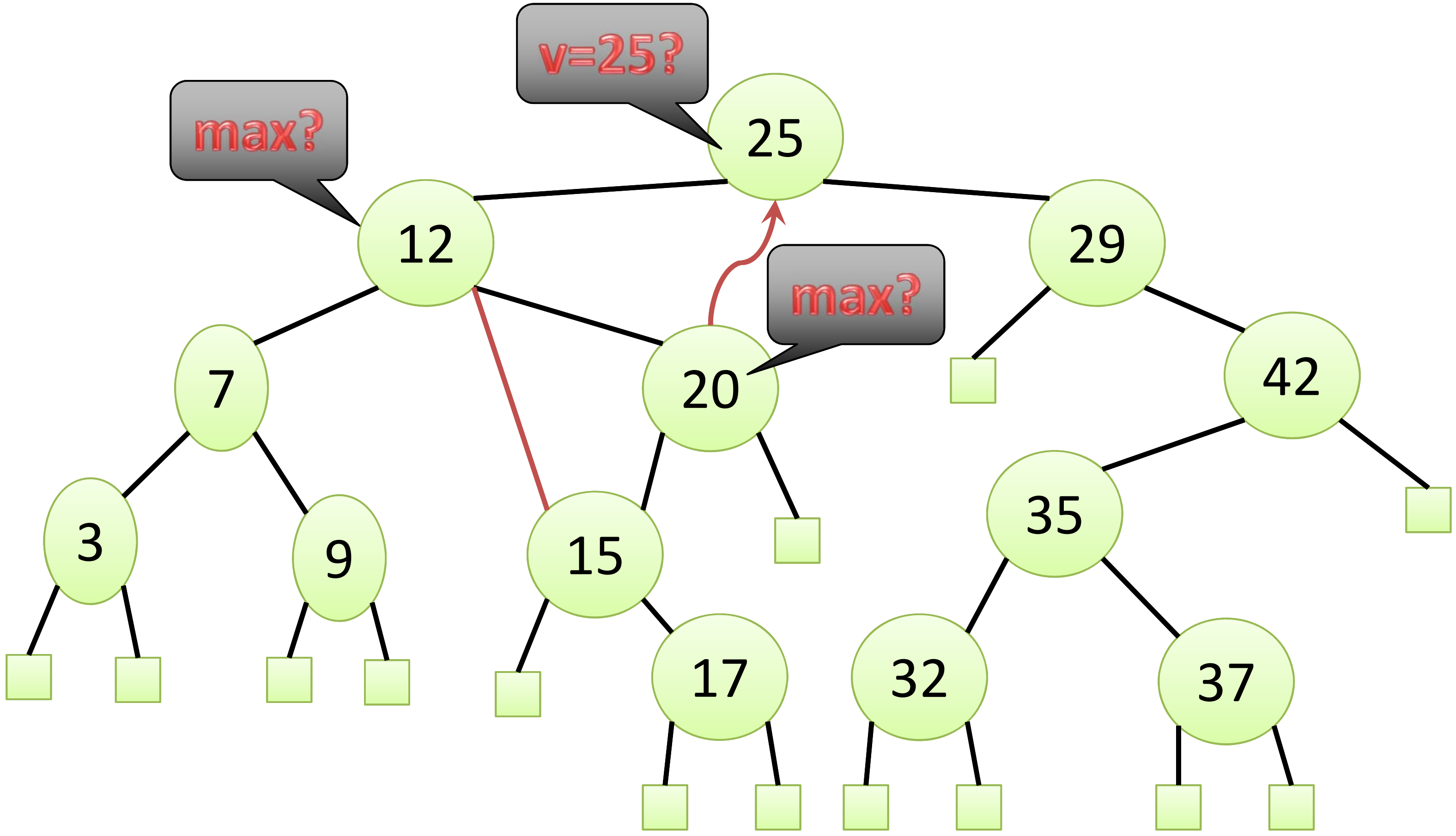
- Let u be the left child of v.
- Find the vertex w that has the maximum value w in the subtree rooted at u.
  - Right child of w should be a leaf
- Value y in w is copied to v, and remove w.
  - As same as case 1

Why?

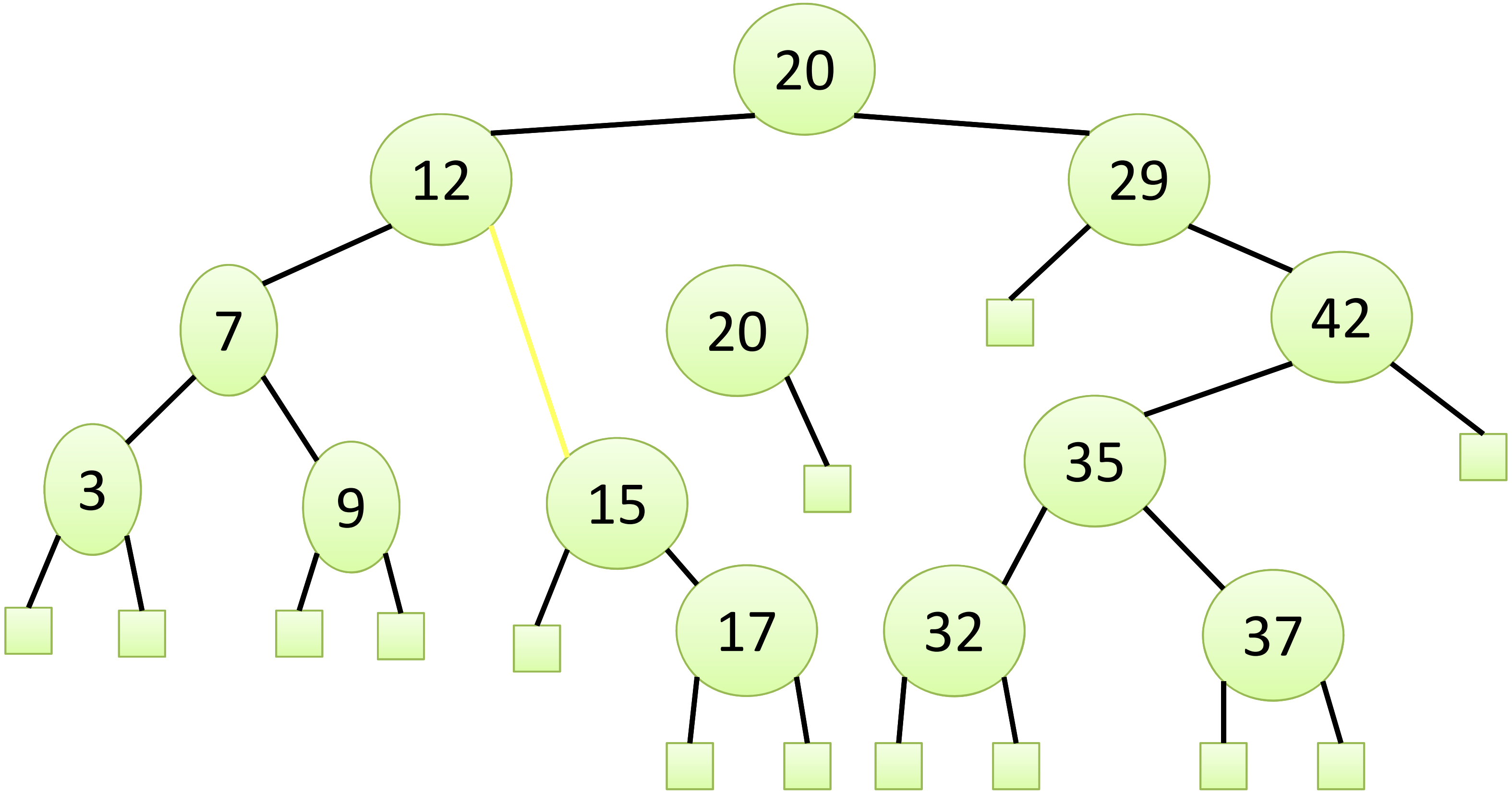


Q. Is this still binary search tree?

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 Θ(n). (In this case, it is almost the same as a linked list.)