Databases

Data Collection Stored in a Binary Tree *

* extra material: not required for the exam

* based on notions covered in high school and other courses in our faculty

Sabina S. CS

- binary search algorithm on ordered collections
  - very fast; reduced search time (+)
  - maintaining the records' sort order - costly (especially for dynamic collections with many INSERT, UPDATE, DELETE operations) (-)
- solution
  - store the collection using a *binary tree*
  - record - stored in a node
  - node with key value *v*
    - left subtree
      - records with key values < *v*
    - right subtree
      - records with key values > *v*

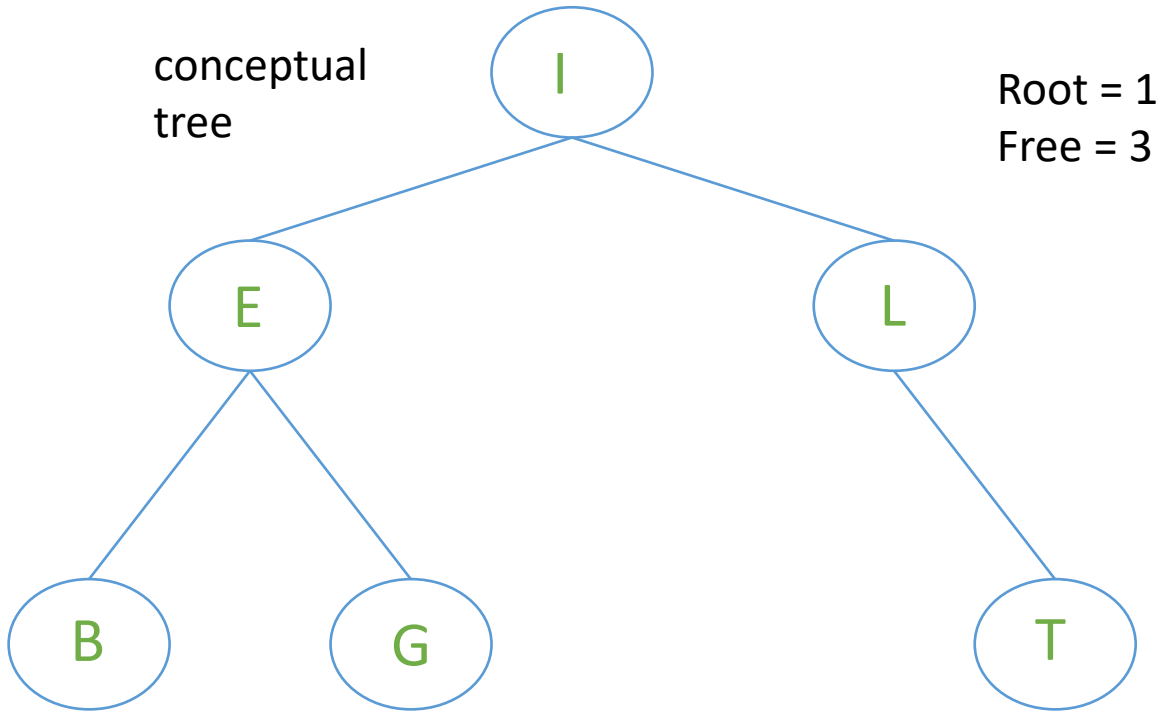
- memory structure for a binary tree node
  - record's values stored in K and INF

r

- memory structure for a binary tree
  - collection of nodes
  - pointer to the root
  - list of empty nodes
    - linked by PointerL

->

* Example
  - Root - pointer to the root
  - Free - pointer to the head of the empty nodes list



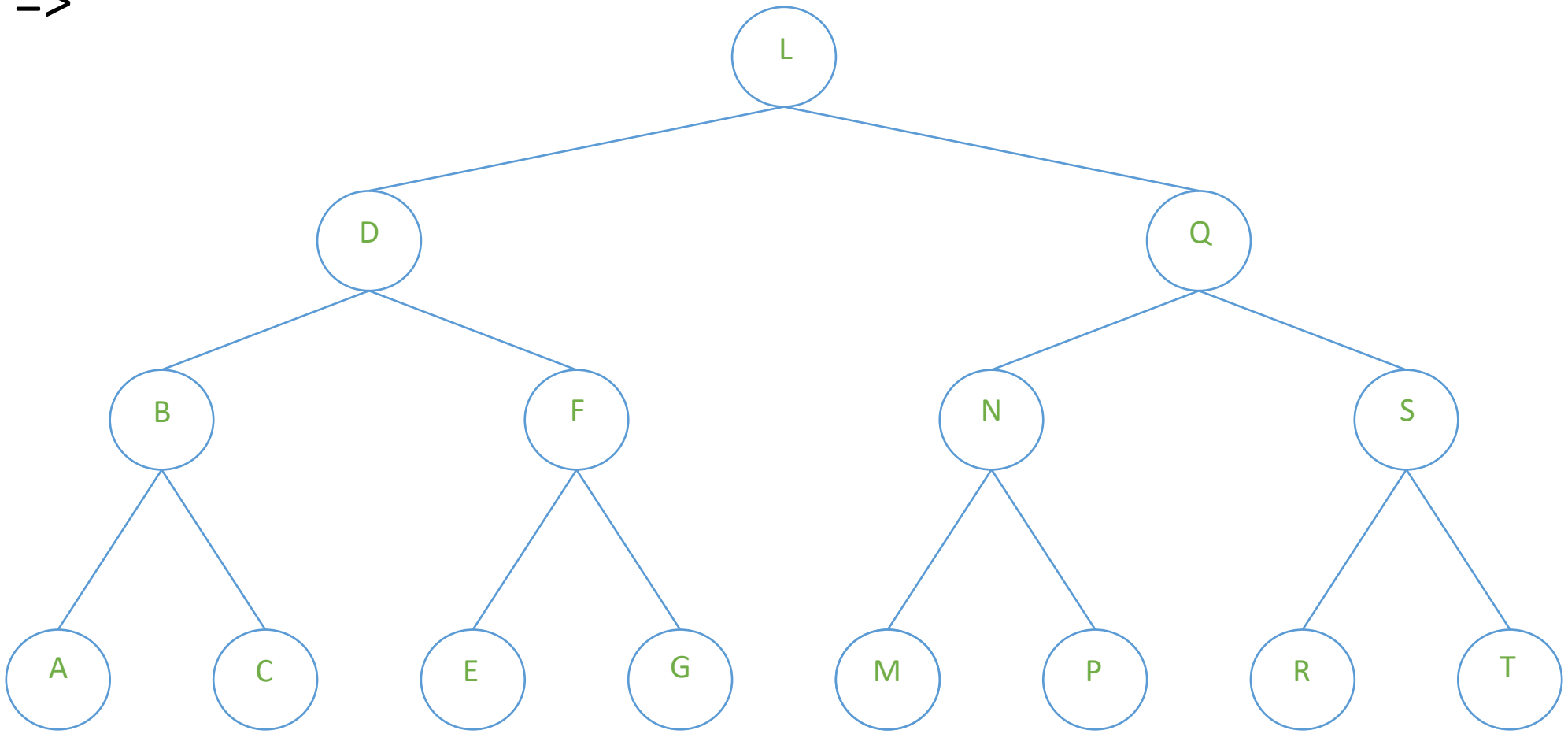
Root = 1
Free = 3

	PointerL	PointerR	K	INF
1	2	4	I	INF_I
2	8	7	E	INF_E
3	-6	NULL		
4	NULL	5	L	INF_L
5	NULL	NULL	T	INF_T
6	-9	NULL		
7	NULL	NULL	G	INF_G
8	NULL	NULL	B	INF_B
9	NULL	NULL		

- operations in a binary tree (BT)
  - searching for a record with key value K_0
  - inserting a record
  - removing a record
  - traversal - partial (between 2 values) / total

* Example - binary tree (only key values are shown)
  - key values {L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C} (added to the tree in the specified order)

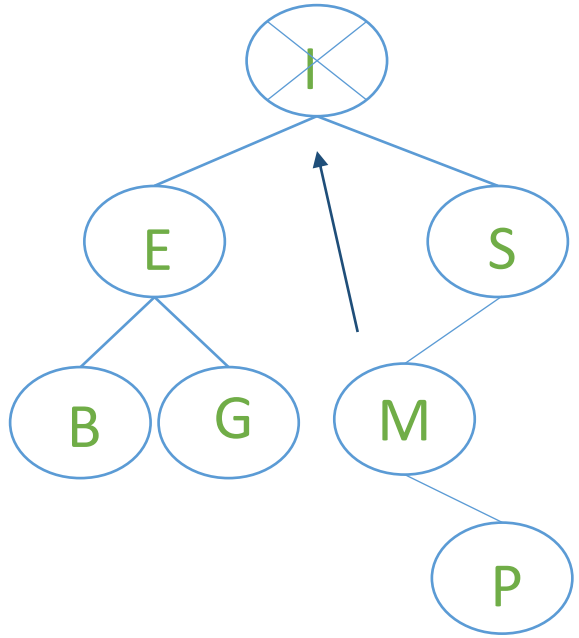
    =>



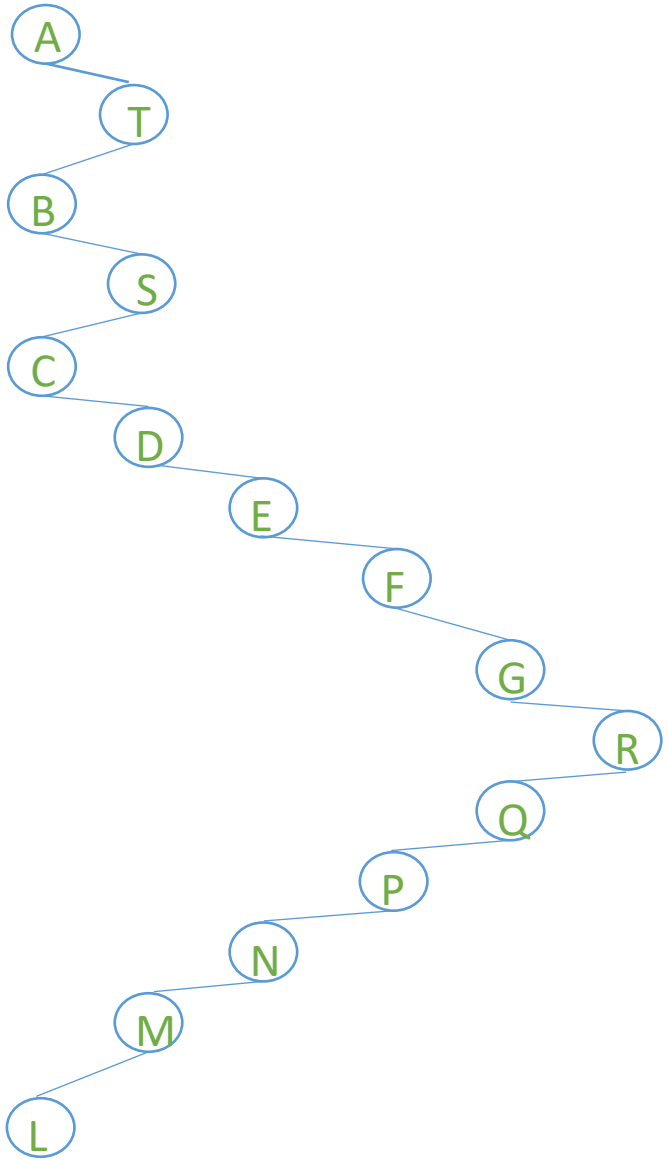
- searching for a record
  - search starts at the root, follows PointerL / PointerR - based on the result of a comparison, a subtree is not considered anymore
  - how many comparisons to find record with key value M in the previous tree?
  - analysis, e.g.:
    - collection of English texts; tuples: (L, 50k), (D, 100k), etc
    - analyzing the most common letters in English; how many times does a letter appear in the collection?

- inserting a record
  - detect the position of the record
  - store the record in a free node
  - link the node to its parent

- removing a record
  - search for the record
  - 3 cases
    - it doesn't have children
      - the parent's pointer := NULL
    - it has 1 child
      - the child is attached to the parent
    - it has 2 children
      - it's replaced with the closest value (e.g., in-order successor)
  - node - added to the empty nodes list

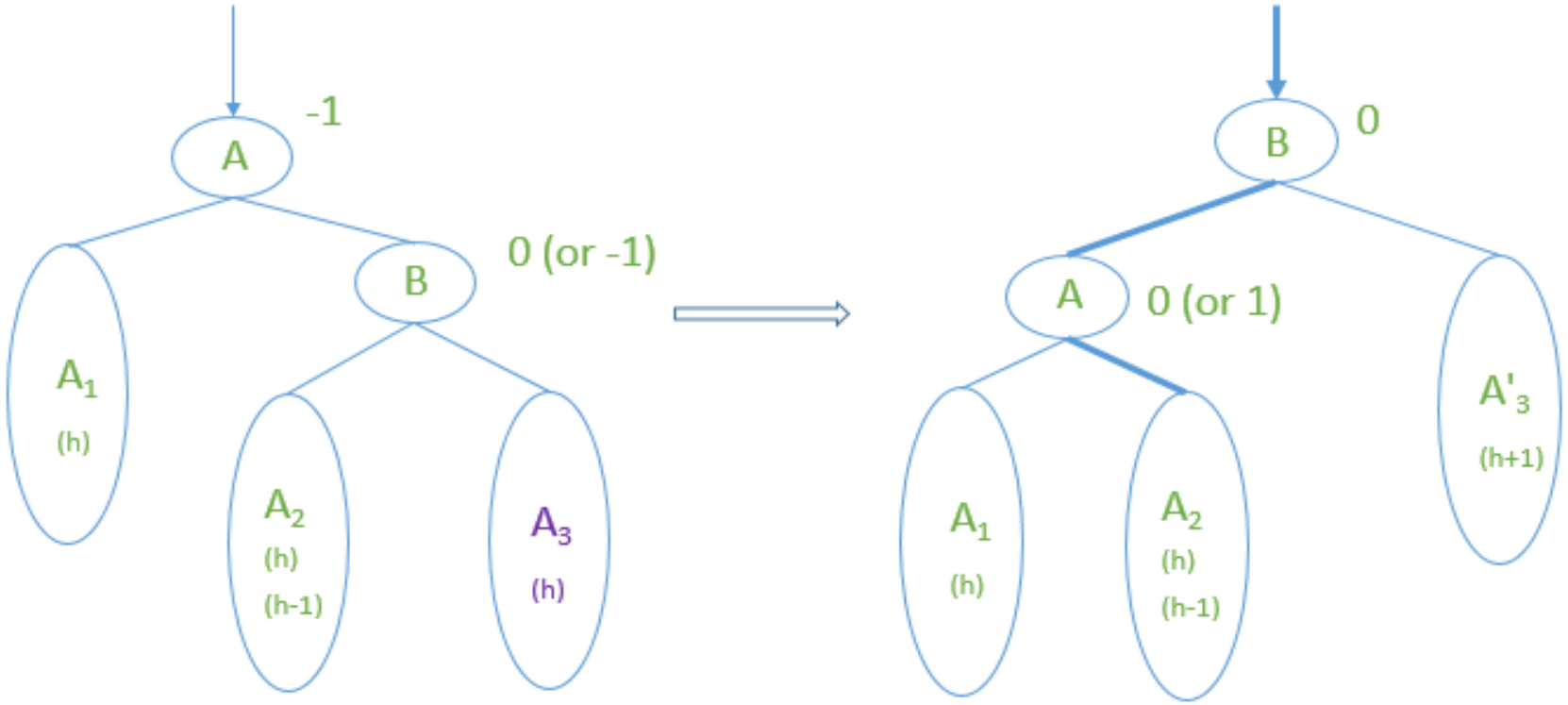


* Example
  - key values {L, D, B, Q, N, F, S, R, T, M, E, G, P, A, C} are now provided in a different order: {A, T, B, S, C, D, E, F, G, R, Q, P, N, M, L}
  - the tree on the right is obtained
  - how many comparisons to find record with key value M now?
- *degenerate tree*, sequential search
- the shape of the tree, and hence the search time, depends on the order in which data is added to the collection



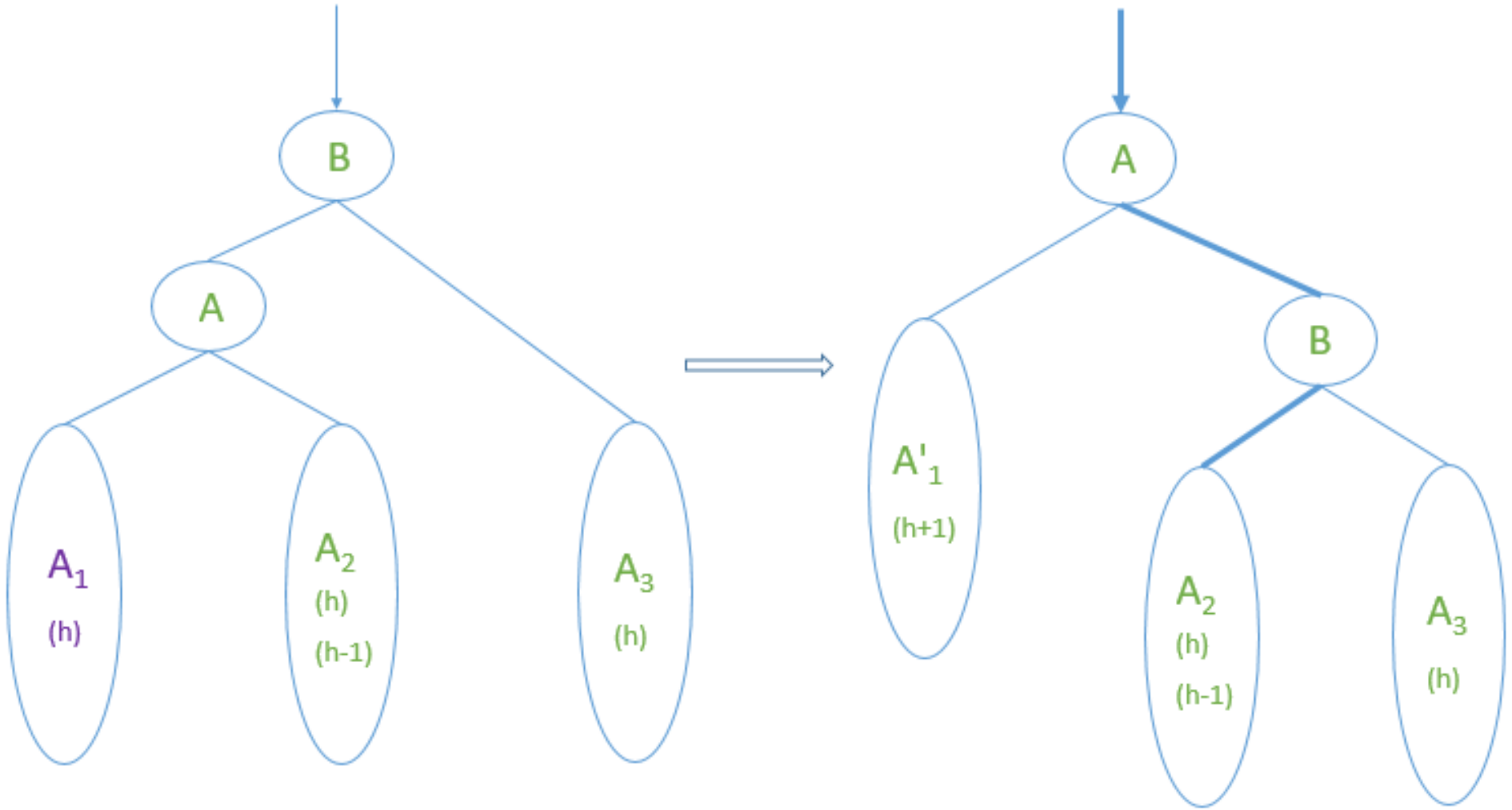
- definitions
  - *height* of the tree
    - length of longest path from root to terminals
  - a binary tree is *balanced* if, for every node N, the difference between the heights of N's subtrees is 0, 1 or -1
- obs. operations on a balanced tree can unbalance it; the tree can be rebalanced through a small number of changes
- a value is added to the A_3 tree on the next page (v1), causing its height to change (it increases by 1)
  - the subtrees' heights are shown between parentheses; for nodes A and B, the difference between the heights of their subtrees is also shown
  - after the insertion, the tree becomes unbalanced
  - the right-hand side of the figure shows a transformation that rebalances the tree with the root in A

v1

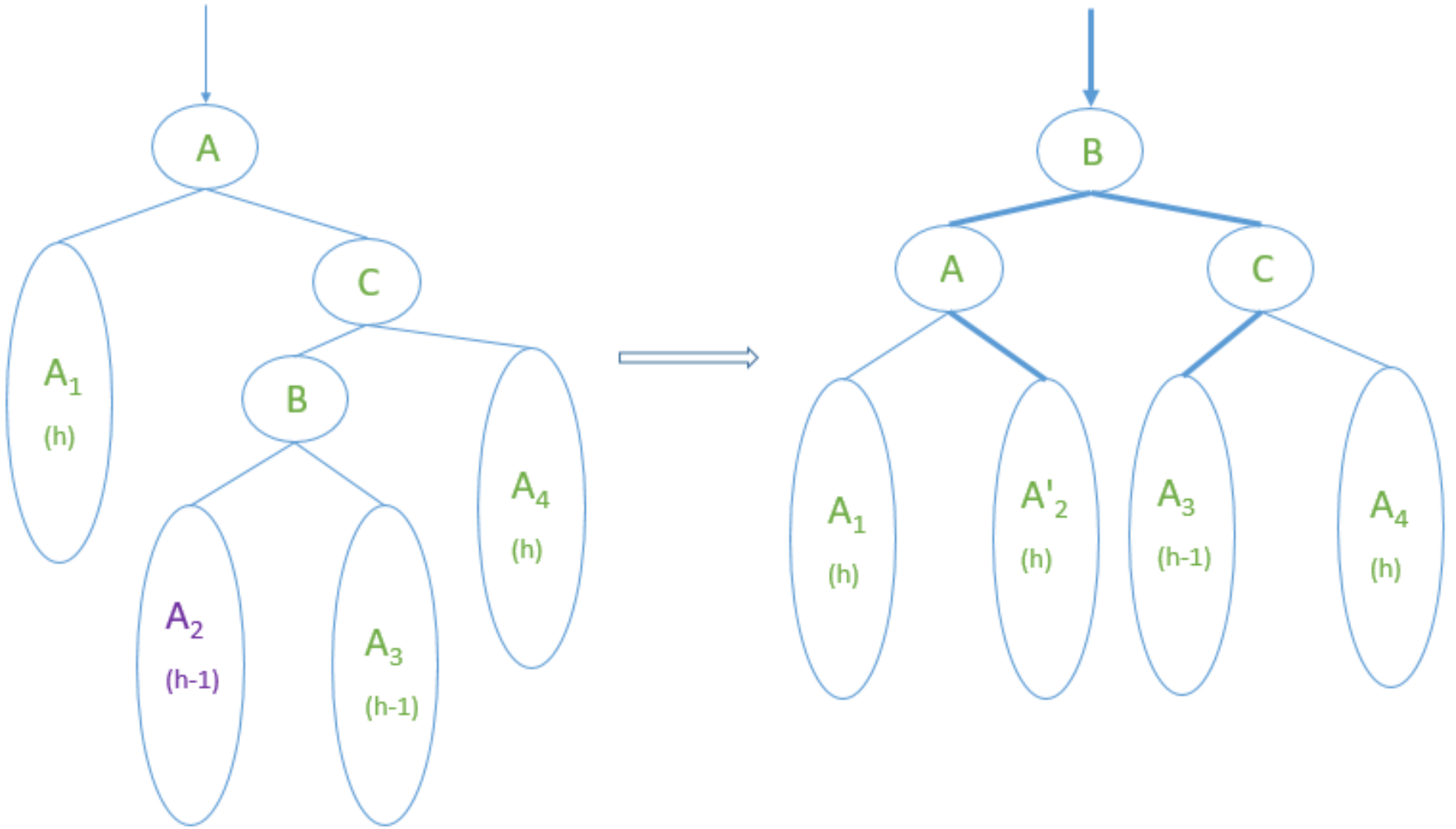


- some assignments are necessary for this rebalancing in the memory area that stores the tree (PointerR for A, PointerL for B, pointer referring to A will refer to B)
- [Kn76] enumerates all 6 rebalancing transformations

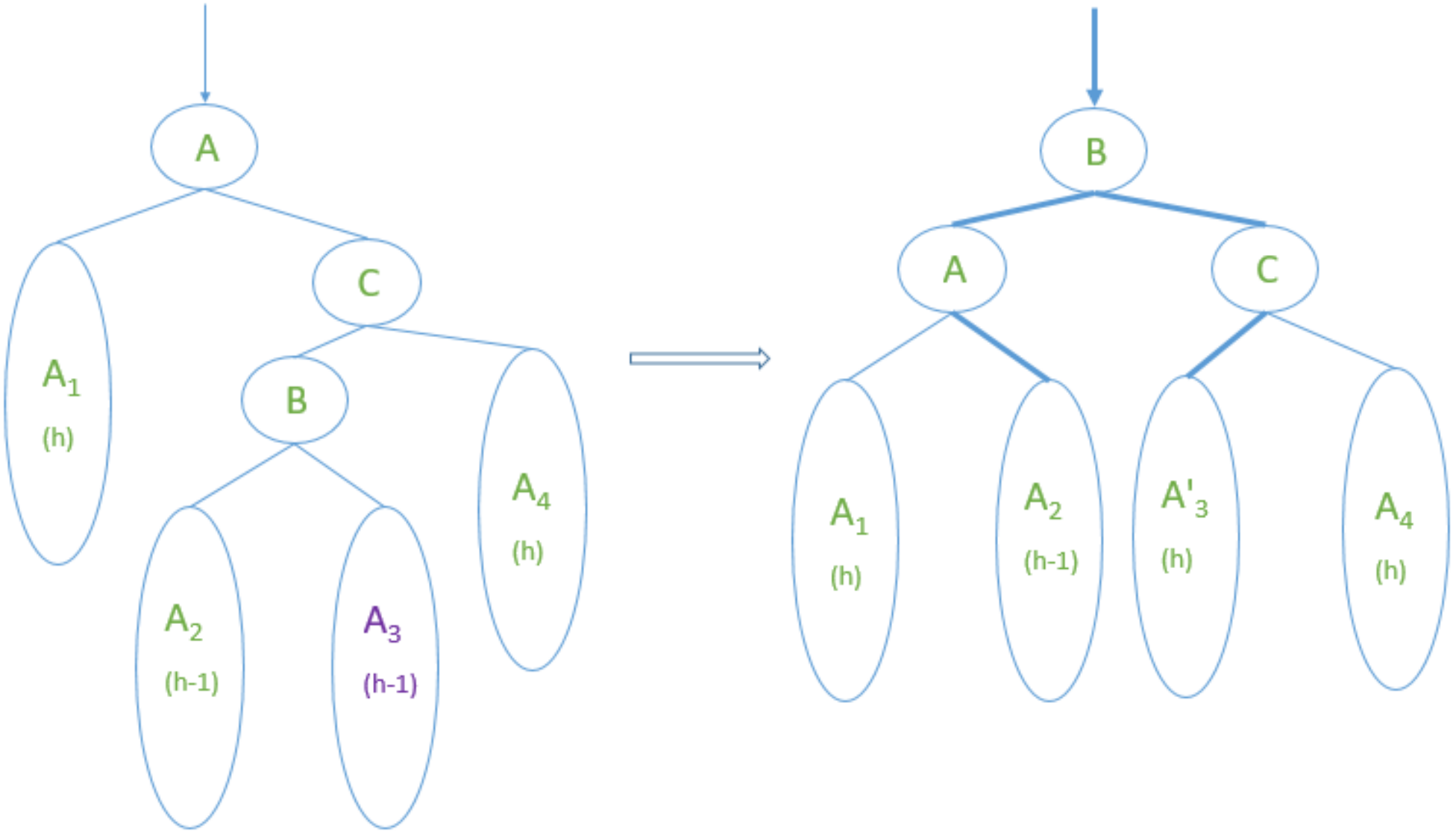
v2



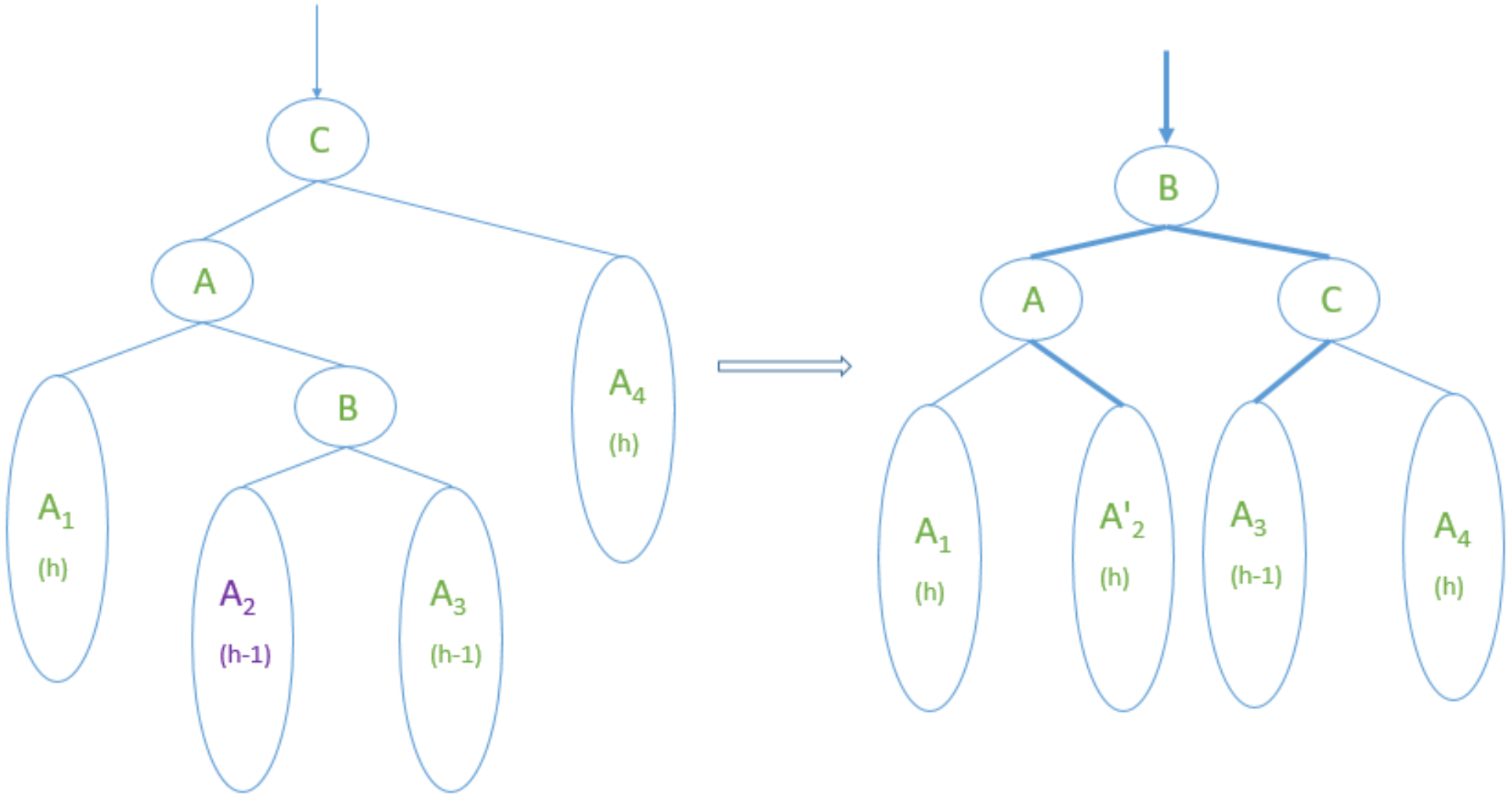
v3



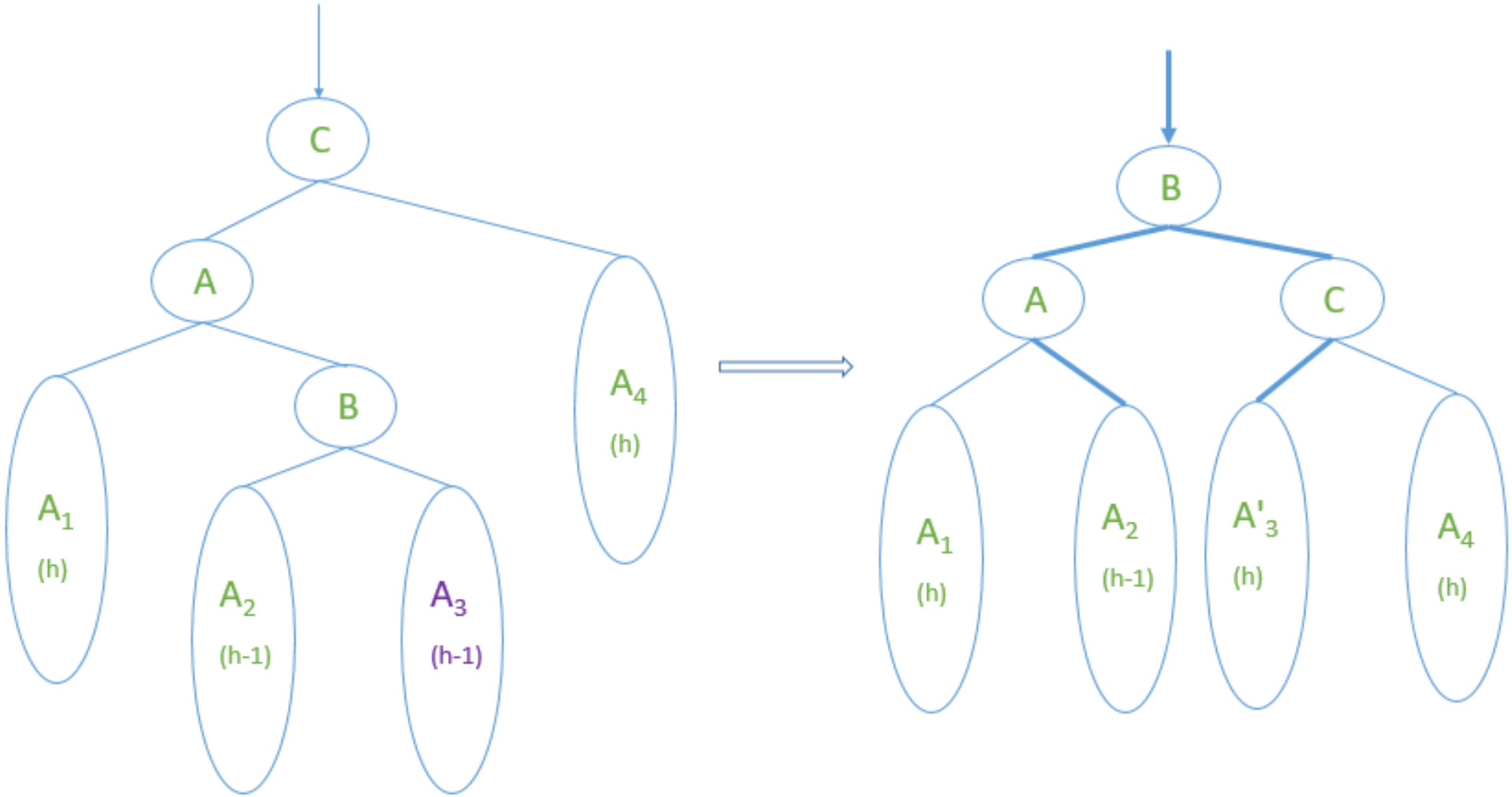
v4



v5



v6



References

- [Ta13] ȚÂMBULEA, L., Curs Baze de date, Facultatea de Matematică și Informatică, UBB, 2013-2014
- [Kn76] KNUTH, D.E., Tratat de programare a calculatoarelor. Sortare și căutare. Ed. Tehnică, București, 1976