ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Parallel in tree-related problems

References

- Michael J. Quinn. **Parallel Computing. Theory and Practice**. McGraw-Hill
- Albert Y. Zomaya. **Parallel and Distributed Computing Handbook**. McGraw-Hill
- Ian Foster. **Designing and Building Parallel Programs**. Addison-Wesley.
- Ananth Grama, Anshul Gupta, George Karypis, Vipin Kumar **. Introduction to Parallel Computing, Second Edition.** Addison Wesley.
- Joseph Jaja**. An Introduction to Parallel Algorithm.** Addison Wesley.
- Nguyễn Đức Nghĩa**. Tính toán song song.** Hà Nội 2003.

10.1 Prefix and Suffix

Prefix, Suffix Problems

- Concept: Let A[1..n] be a sequence of n integer elements
  - P[i] is called i_th prefix sum of array A, if : $P[i] = \sum A[j]$ with $j \in 1..i$
  - S[i] is called i_th suffix sum of array A, if : $S[i] = \sum A[j]$ with $j \in i..n$
- Problem: Build an algorithm on PRAM:
  - Input: A[1..n];
  - Output: P[1..n] (or S[1..n])
- These two problems (prefix or sufix) are the same.

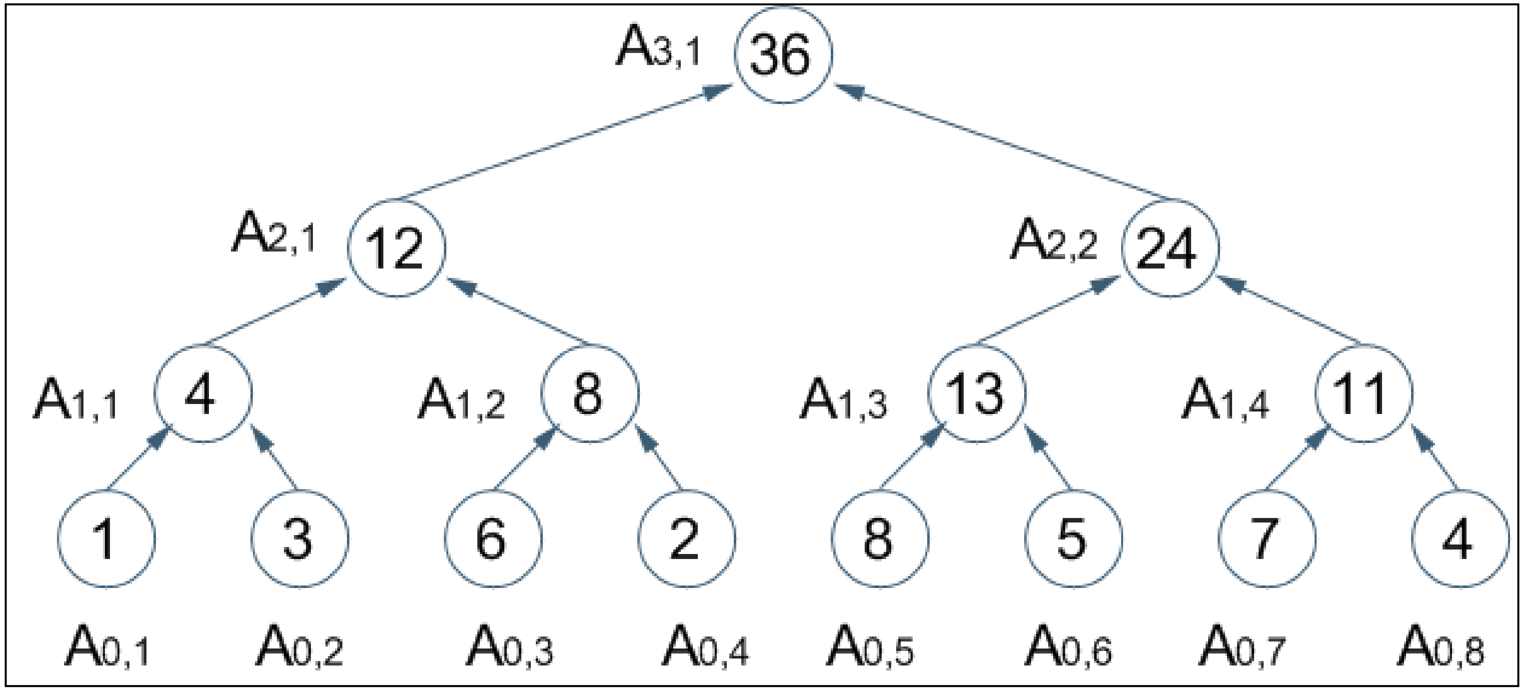
Prefix Sum

- Approaches:
  - Method 1: Using Balanced Tree + Growing by Doubling
  - Method 2: Recursive.
  - Method 3: Using Jumping Pointer

Using Balanced Tree + Growing by Doubling

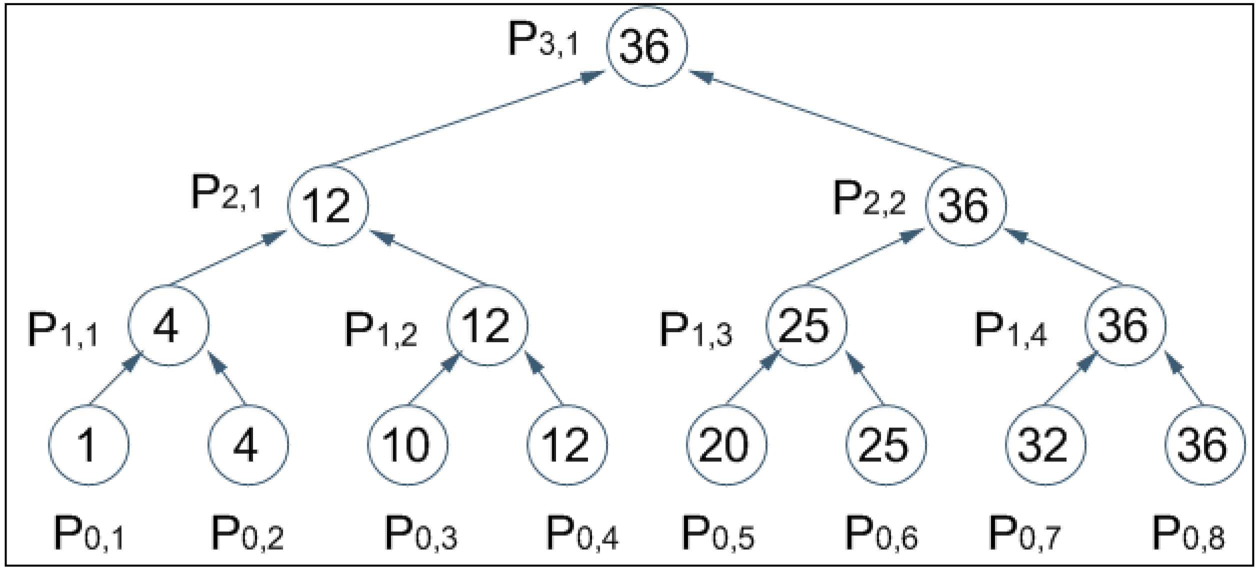
- Comment:
  - Balanced Tree Method returns a result at the top of the tree.
  - In order to obtain the sequence of prefix sum → using other nodes in the balanced tree created above.
- Idea:
  - Build a new tree P, each node is called: $P_{i,j}$ with i is the level index, j is the processor index.
  - Assuming that $P_{i,j}$ is root of a sub-tree, has leftmost leaf that is $P_{0,k}$, hence $P_{i,j} = \sum A[t]$ where $t \in 1..k$

Step1. Create a balanced tree



- Each node in the tree is represented by $A_{i,j}$ where:
  - i is the level index.
  - j is the processor index.
- And: $A_{i+1,j} = A_{i,2j-1} + A_{i,2j}$

Step 2. Build a new tree P



- Comments:
  - P tree is built from top to bottom with the top : $P_{k,1} = A_{k,1} = \sum A[i]$ where $i = 1..n$, $k = \log_2 n$.
  - $P_{i,1} = A_{i,1}$ where $i = k-1 .. 0$.
  - $P_{i,j} = P_{i+1,j/2}$ with j is even; $P_{i,j} = P_{i+1,[j/2]} + A_{i,j}$ with j is odd ($i = k-1..0$)

```
input       : A[1..n]; n = 2^k
output      : P[1..n] | P[i] = Prefix_sum[i]
begin
    for i = 1 to n do in parallel
        A[0,i]      =     A[i];
    end parallel
    for i = 1 to k do
        for j = 1 to n/2^i do in parallel
            A[i,j]=     A[i-1,2j-1] + A[i-1,2j];
        end parallel
    end for
    P[k,1]      =     A[k,1];
    for i = k -1 downto 0 do
        for j = 1 to 2^(k-i) do in parallel
            if j = 1 then P[i,1] = A[i,1];
            else if j chẵn then P[i,j] = P[i+1,j/2]
            else P[i,j] = P[i+1,[j/2]] + A[i,j]
        end parallel
    end for
    for i = 1 to n do in parallel
        P[i]  =     P[0,i];
    end parallel
end.
```

Complexity Evaluation

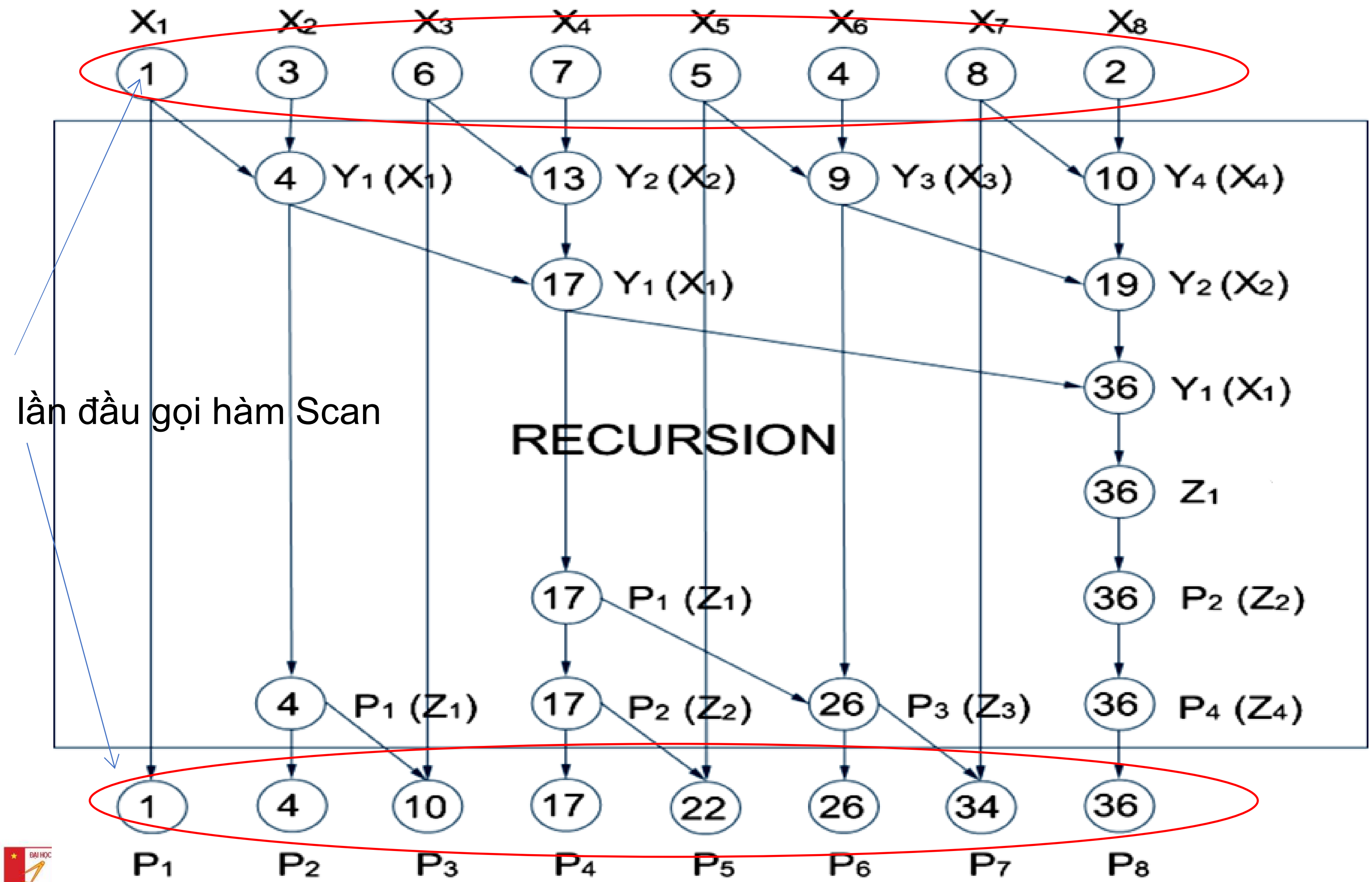
- The algorithm is divided into 2 parts:
  - Part 1: create the balanced tree : $O(\log_2 n)$ in PRAM EREW.
  - Part 2: build P tree with $O(\log_2 n)$ serial steps.
    - Node $P_{i,j}$ needs $P_{i+1,[j/2]}$: this value can be read in serial mode in PRAM EREW.
    - When j is even: $P_{i,j}$ needs $P_{i+1,j/2}$
    - When j is odd: $P_{i,j}$ needs $A_{i,j}$ first, then $P_{i+1,[j/2]}$

Recursive Method

- Idea:
  - Using recursive method to build A, P trees.
  - For example, we can build a balanced tree thanks to a recursive method as follows:

```
function S = Reduce(A[1..n])
begin
        if n = 1 then
                S = A[1];
                return S;
        end if
        for i = 1 to n/2 do in parallel
                A[i] = A[2i-1] + A[2i]
        end parallel
        S = Reduce(A[1..n/2];
end
```

Recursive Method's Illustration



Recursive Method

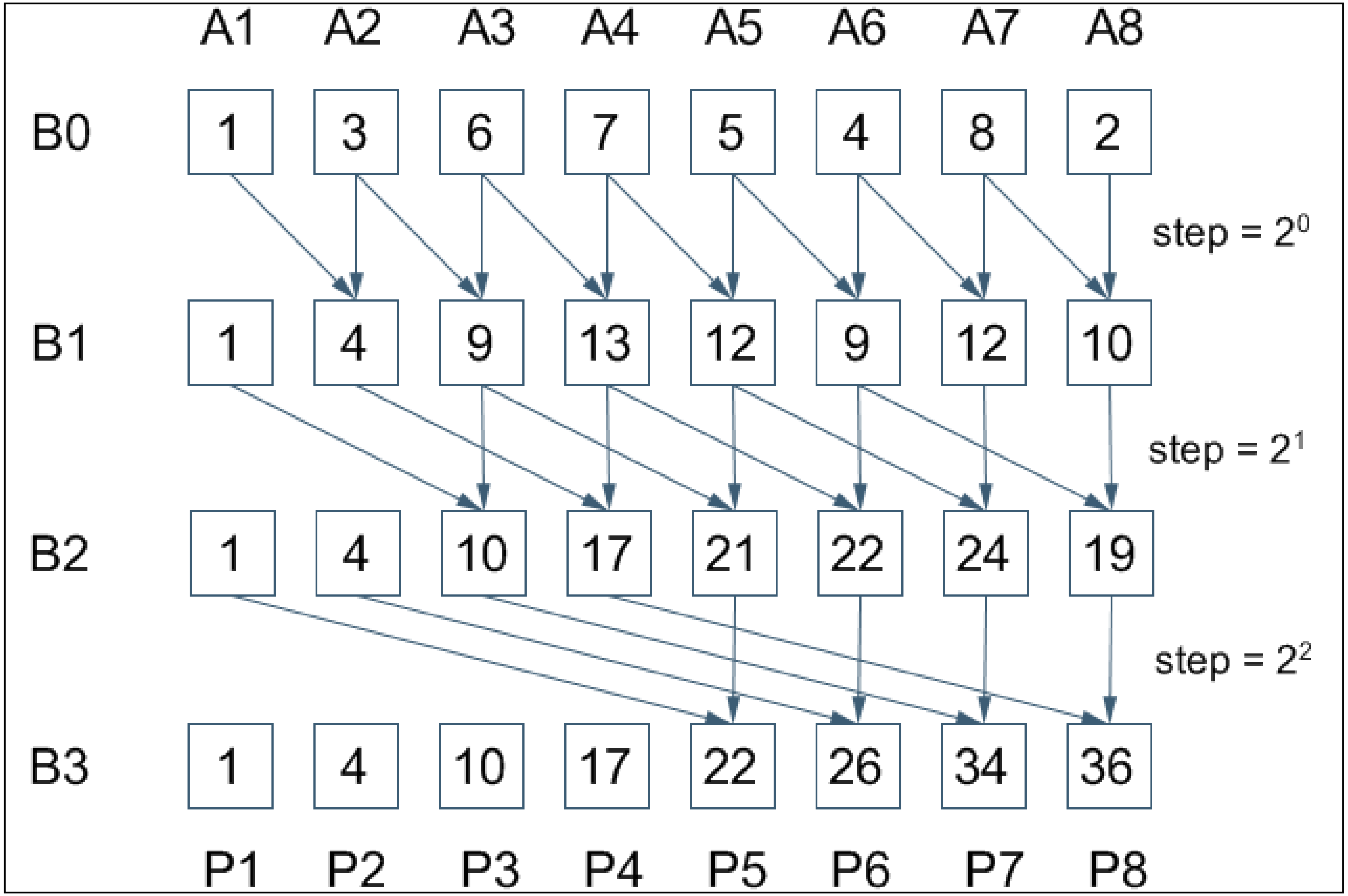
- The duration of the algorithm is equal to the number of recursive function calls.
- In a recursive call: O(1) time unit
- Total duration time: $O(\log_2 n)$.

```
function P[1..n] = Scan(X[1..n])
begin
        if n = 1 then
                P[1] = X[1];
                return P;
        end if
        for i = 1 to n/2 do in parallel
                Y[i] = X[2i-1] + X[2i]
        end parallel

        Z[1..n/2] = Scan(Y[1..n/2]);

        for i = 1 to n do in parallel
                if i chẵn then  P[i] = Z[i/2];
                elseif i = 1 then P[1] = X[1];
                else P[i] = Z[(i-1)/2] + X[i];

        end parallel
        return P;
end
```

Jumping Pointer's Algorithm

- Idea:
  - Initial step: $P_0[i] = A[i]$.
  - At step k:
    - step = 2^{k-1}.
    - $P_k[i] = P_{k-1}[i] + P_{k-1}[i\text{-step}]$ with $\forall$ i | step < i $\leq$ n.
  - We can see that:
    - At step k: $P_k[i] = \sum A[j]$ with $j \in 1..i$; $i \in 1..2^k$.
    - So to calculate P[1..n] requires $\log_2 n$ repeating steps.

Illustration



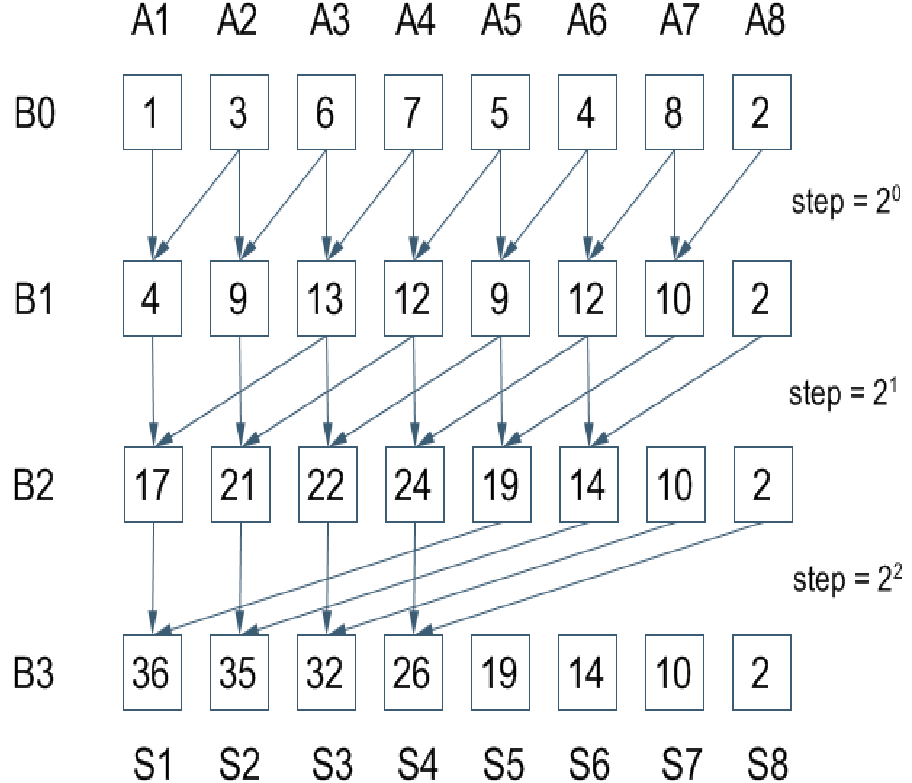
The algorithm's illustration

- Execution time: Serial iteration's number: O(k) = O($\log_2$n).
- Assignments can be performed by reading/writing separately → PRAM EREW.

```
input    : A[1..n]; n = 2^k
output   : P[1..n] | P[i] = Prefix_sum[i]
begin
    for i = 1 to n do in parallel
        P[i] = A[i];
    end parallel
    for i = 1 to k do
        for j = 1 to n do in parallel
            if j > 2^(i-1) then
                P[j] = P[j] + P[j-2^(i-1)];
            end if
        end parallel
    end for
end
```

Suffix sum

- Same as Prefix sum problem.
- For example, Jumping Pointer method for Suffix sum as follows:




```
input     : A[1..n]; n = 2^k
output    : S[1..n] | S[i] = Suffix_sum[i]
begin
    for i = 1 to n do in parallel
        S[i] = A[i];
    end parallel
    for i = 1 to k do
        for j = 1 to n do in parallel
            if j ≤ n-2^(i-1) then
                S[j] = S[j] + S[j+2^(i-1)];
            end if
        end parallel
    end for
end
```

10.2 Tree-related Problems

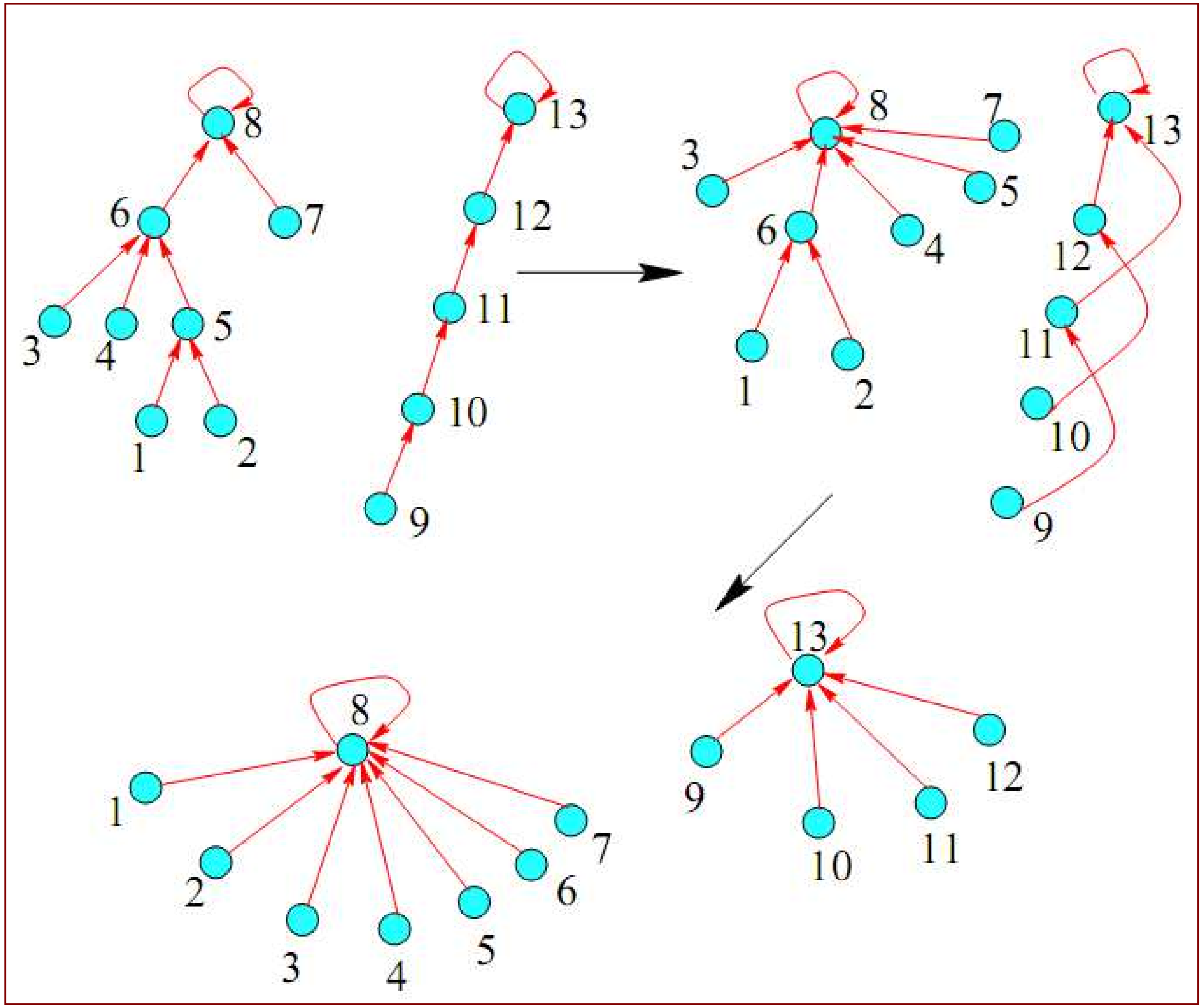
10.2.1 Rooted-directed Tree

- Definition: Rooted-directed Tree T is a directional graph with a special node r satisfying:
  - $\forall$ node v $\in$ V - {r} : have an out-of-degree outdegree(v) = 1 while the node r has outdegree(r) = 0.
  - $\forall$ node v $\in$ V - {r}: $\exists$ 1 path from v to r.
  - => The node r is called the root of tree.
- Tree T is presented by an array P[1..n],
  - P[i] = j if j is the father of i in the tree.
  - Root is the node that points to itself: P[r] = r.

Identifying tree root in the forest

- Problem speaking:
  - Let F be a forest of root-oriented trees.
  - F is presented by an array P[1..n].
  - For each node i in the forest, identifying the root of the trees that contains the node i, which is called S[i].
- Approach:
  - Use the jumping pointer technique.

Identifying tree root in the forest



Identifying tree root in the forest

```
input   : rừng F xác định bởi P[1..n]
output : S[1..n], S[i] -- gốc của cây con chứa nút i
begin
      for i = 1 to n do in parallel
            S[i] = P[i];
            while S[i] <> S[S[i]] do          pointer
                  S[i]  = S[S[i]];            jumping
            end while.
      end for.
end.
```

- Complexity: if h is the height of the highest tree in the forest.
  - Execution time: $O(\log_2 h)$.
  - Cost: $O(n.\log_2 h)$. →

Tree’s Suffix-sum problem

- Problem speaking:
  - Forest F is presented by an arrayP[1..n].
  - The nodes on the tree are weighted W[1..n].
  - Tree root’s weight equals 0.
  - Let's determine the total weight from any node (for example node v) to the root node r of a sub-tree containing node v.
- Approach:
  - Jumping pointer technique.

Tree’s Suffix-sum problem

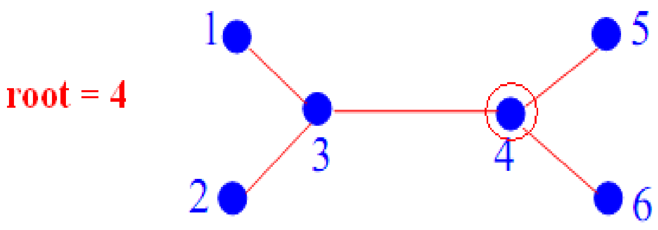
```
input   : rừng F xác định bởi P[1..n], W[1..n]
output : R[1..n], R[i] -- trọng số đi từ i tới S[i]
begin
      for i = 1 to n do in parallel
            S[i] = P[i];
            while S[i] <> S[S[i]] do
                  W[i] = W[i] + W[S[i]];
                  S[i]  = S[S[i]];
            end while.
      end for.
end.
```

- Complexity evaluation as before.
- PRAM model: CREW for 1 parent node can have multiple child nodes

Rooted Tree

- Let T = (V,E) be a tree defined by the list of adjacent vertexes and 1 node r $\in$V. Let's build the T tree with the root is r by defining p(v), with each node v $\neq$ r, as the parent node of v.
- Approach:
  - Set up a Euler cycle on T tree.
  - Let's assume u is the last node in the adjacent list of r. Set s(<u,r>) = 0.
  - Setting the weight for the <x,y> = 1 on T tree and performing suffix_sum in the tree.
  - For each <x,y>, defining x = p(y) if suffix_sum (<x,y>) is greater than suffix_sum(<y,x>).

Rooted Tree problem (root: 4)

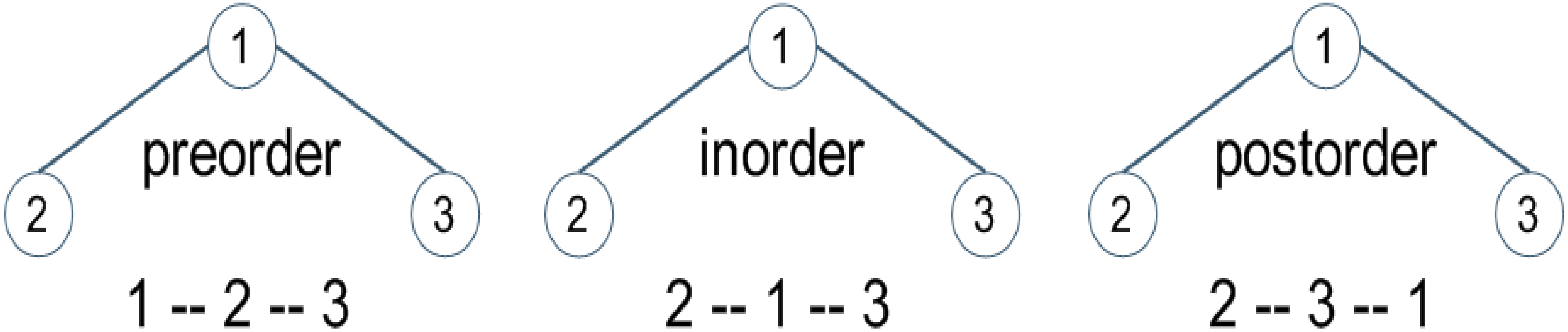


v	adj(v)
1	3
2	3
3	2, 1, 4
4	3, 5, 6
5	4
6	4

edge	successor
<3,1>	<1,3>
<3,2>	<2,3>
<2,3>	<3,1>
<1,3>	<3,4>
<4,3>	<3,2>
<3,4>	<4,5>
<5,4>	<4,6>
<6,4>	null
<4,5>	<5,4>
<4,6>	<6,4>

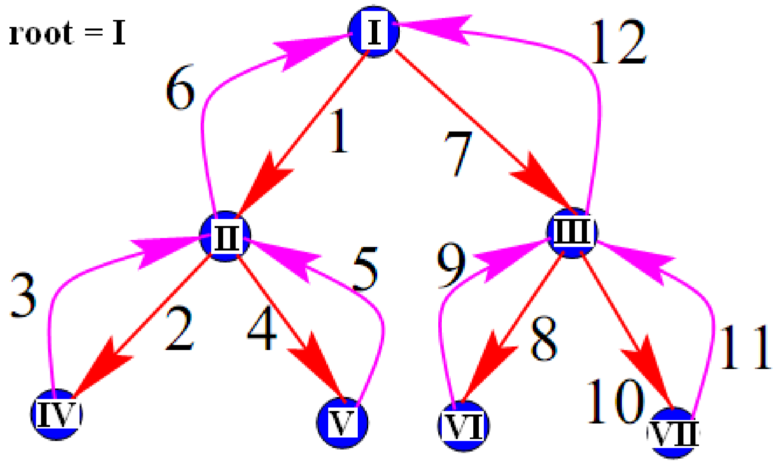
Euler Tour				Result	
Thứ tự	Cạnh	Giá trị	Suffix_sum	v	p(v)
1	<4,3>	1	10	1	3
2	<3,2>	1	9	2	3
3	<2,3>	1	8	3	4
4	<3,1>	1	7	4	4
5	<1,3>	1	6	5	4
6	<3,4>	1	5	6	4
7	<4,5>	1	4		
8	<5,4>	1	3		
9	<4,6>	1	2		
10	<6,4>	1	1		

10.2.2 Tree traversing problem



- Let T = (V,E) be a tree with root node r.
- 3 ways to traverse a binary tree:
  - Pre-Order.
  - In-Order.
  - Post-Order

Approach



Euler Tour		Tree	
Thứ tự	Cạnh	v	p(v)
1	<1,2>	1	1
2	<2,4>	2	1
3	<4,2>	3	1
4	<2,5>	4	2
5	<5,2>	5	2
6	<2,1>	6	3
7	<1,3>	7	3
8	<3,6>		
9	<6,3>		
10	<3,7>		
11	<7,3>		
12	<3,1>		

Post-Order

- Set up the Euler cycle on the T tree.
- With root r, identifying the rooted-directed tree (with $\forall$ v identify p(v) as the father of v).
- Set the weight to the edges:
- $w(<v,p(v)>) = 1$ & $w(<p(v),v>) = 0$.
- For each $<u,v>$, determine suffix_sum for $<u,v>$. It is called $S(<u,v>)$
- Traversing position of node v is: $|V| - S(<v,p(v)>)$.
- Finally traversing the root node r.

Post-Order

Euler Tour				Tree	
Thứ tự	Cạnh	Giá trị	Suffix_sum	v	p(v)
1	<1,2>	0	6	1	1
2	<2,4>	0	6	2	1
3	<4,2>	1	6	3	1
4	<2,5>	0	5	4	2
5	<5,2>	1	5	5	2
6	<2,1>	1	4	6	3
7	<1,3>	0	3	7	3
8	<3,6>	0	3		
9	<6,3>	1	3		
10	<3,7>	0	2		
11	<7,3>	1	2		
12	<3,1>	1	1		

Post-Order

- Position of vertexes :
  - S(<2,1>) = 4 → Position(2) = 7 - 4 = 3.
  - S(<3,1>) = 1 → Position(3) = 7 - 1 = 6.
  - S(<4,2>) = 6 → Position(4) = 7 - 6 = 1.
  - S(<5,2>) = 5 → Position(5) = 7 - 5 = 2.
  - S(<6,3>) = 3 → Position(6) = 7 - 3 = 4.
  - S(<7,3>) = 2 → Position(7) = 7 - 2 = 5.
- The traversing order is:
  - [4 → 5 → 2 → 6 → 7 → 3 → 1]

Pre-Order

- Set up the Euler cycle on the T tree.
- With root r, identifying the rooted-directed tree (with ∀ v, identifying p(v) as father of v).
- Set the weight to the edges:
- w(<v,p(v)>) = 0 & w(<p(v),v>) = 1.
- For each <u,v>, determining suffix_sum for <u,v>. It is called S(<u,v>)
- Traversing position v is: |V| - S(<p(v),v>).
- We traverse root node first.

Pre-Order

Euler Tour				Tree	
Thứ tự	Cạnh	Giá trị	Suffix_sum	v	p(v)
1	<1,2>	1	6	1	1
2	<2,4>	1	5	2	1
3	<4,2>	0	4	3	1
4	<2,5>	1	4	4	2
5	<5,2>	0	3	5	2
6	<2,1>	0	3	6	3
7	<1,3>	1	3	7	3
8	<3,6>	1	2		
9	<6,3>	0	1		
10	<3,7>	1	1		
11	<7,3>	0	0		
12	<3,1>	0	0		

Pre-Order

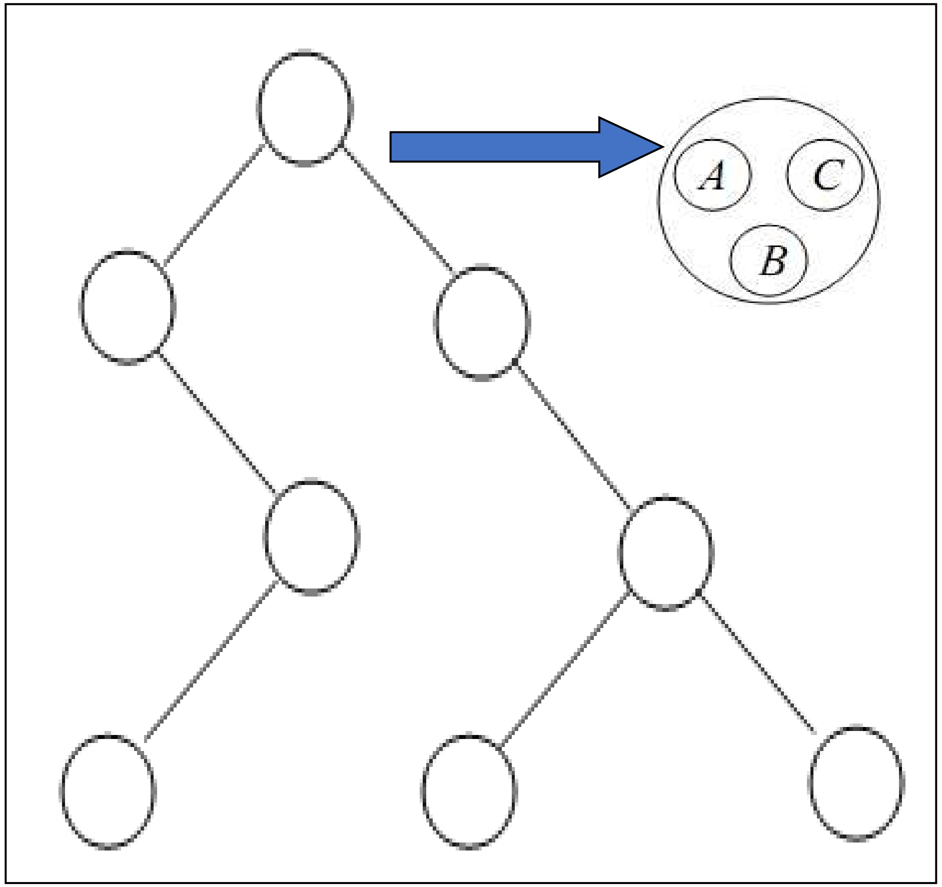
- Position of vertexes as follows:
  - S(<1,2>) = 6 → Position(2) = 7 - 6 = 1.
  - S(<1,3>) = 3 → Position(3) = 7 - 3 = 4.
  - S(<2,4>) = 5 → Position(4) = 7 - 5 = 2.
  - S(<2,5>) = 4 → Position(5) = 7 - 4 = 3.
  - S(<3,6>) = 2 → Position(6) = 7 - 2 = 5.
  - S(<3,7>) = 1 → Position(7) = 7 - 1 = 6.
- The traversing order is:
  - [1 → 2 → 4 → 5 → 3 → 6 → 7]

A different approach

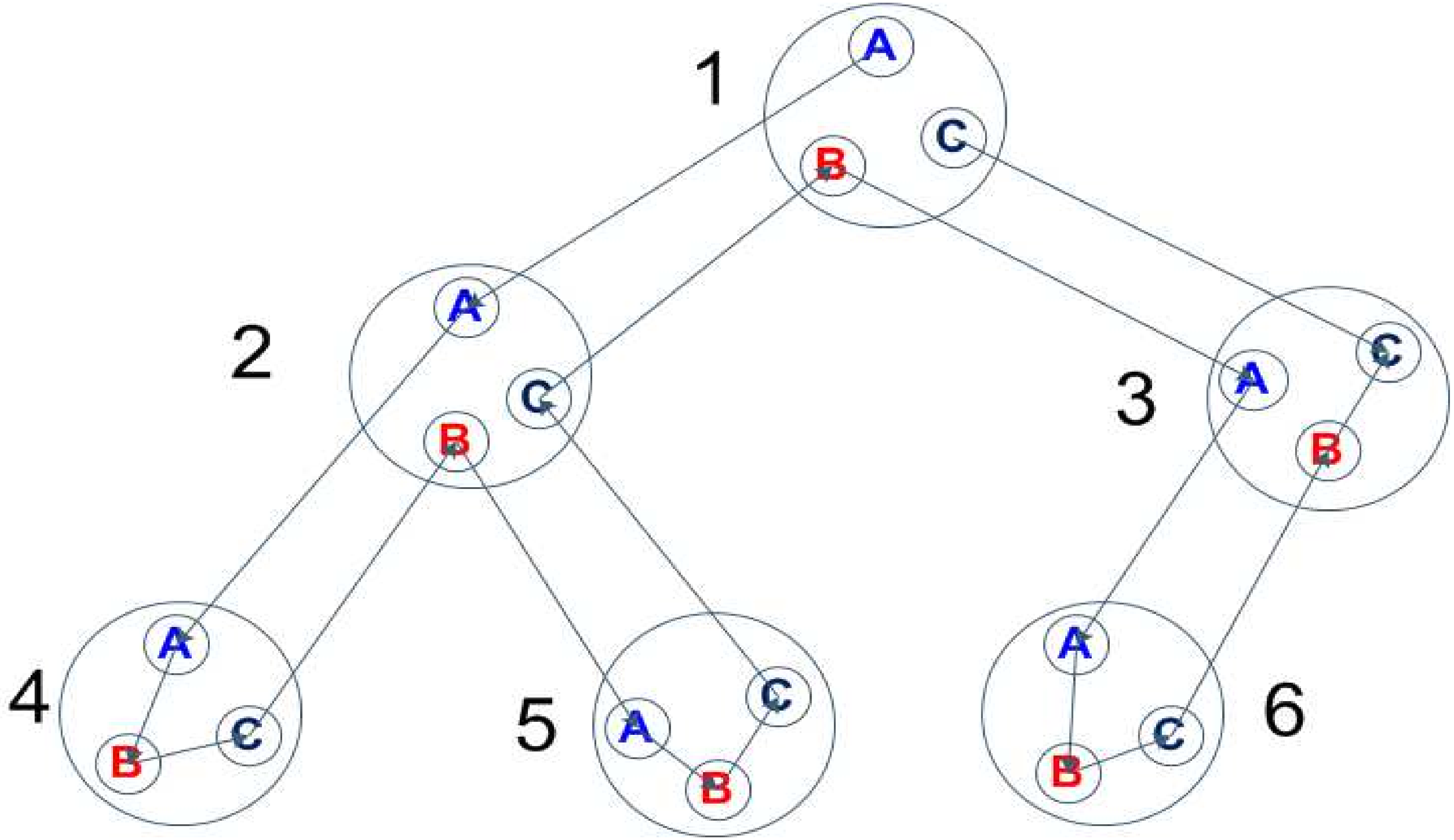
- For the binary tree:
  - Each v node is considered to be 3 child nodes: v[a], v[b], v[c].
  - Rules of the node [a]:
    - If v has a left child that is u, then: v[a] → u[a].
    - If v does not have left child: v[a] → v[b].
  - Rules of the node [b]:
    - If v have a right child that is u, then: v[b] → u[a].
    - If v does not have right child: v[b] → v[c].
  - Rules of the node [c]:
    - If v is u's left child, then: v[c] → u[b].
    - If v is u's right child,: v[c] → u[c].
    - If v is the root node: v[c] → NULL.

Illustration

- Look at the tree as pictured on the side. Each node is presented by a set of 3 child nodes A, B, C
- Assigning the appropriate values A,B,C to problems:
  - Traversing trees
  - Calculate height, number of child nodes, …

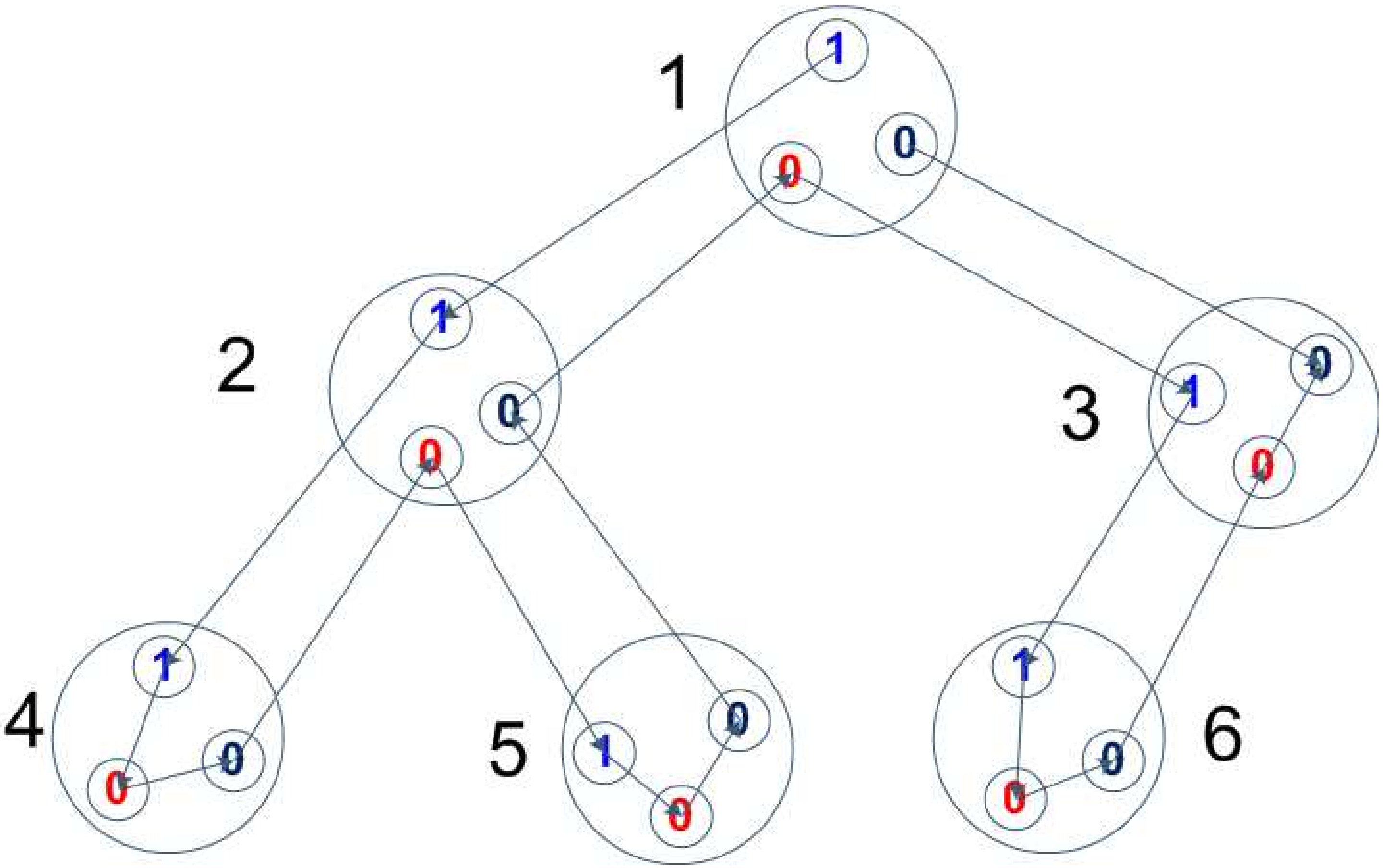


Euler cycle and Linked List

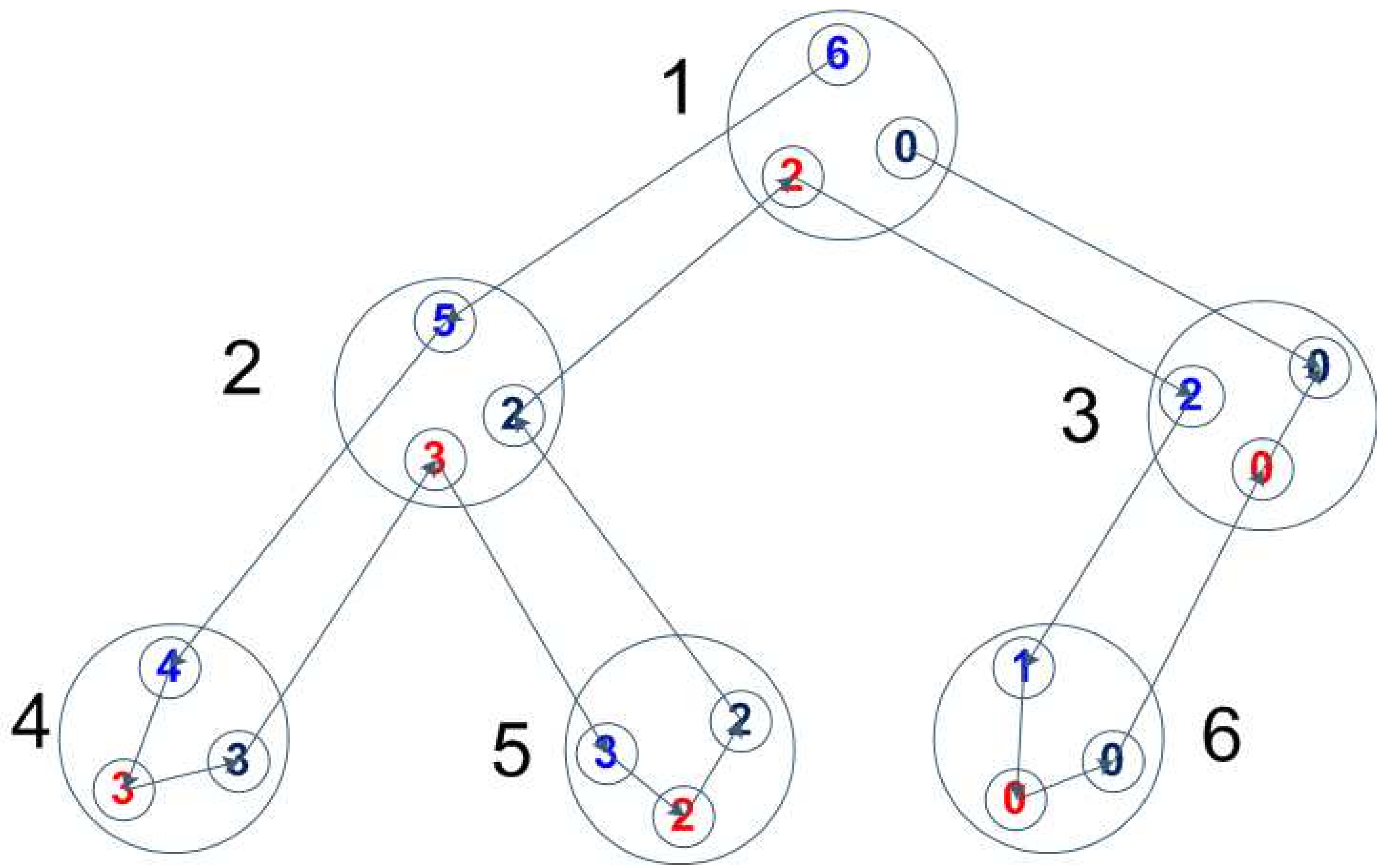


1[A] → 2[A] → 4[A] → 4[B] → 4[C] → 2[B] → 5[A] → 5[B] →5[C] → 2[C] → 1[B] → 3[A] → 6[A] → 6[B] → 6[C] → 3[B] → 3[C] → 1[C] → ●(NULL)

PreOrder: A = 1, B = 0, C = 0.



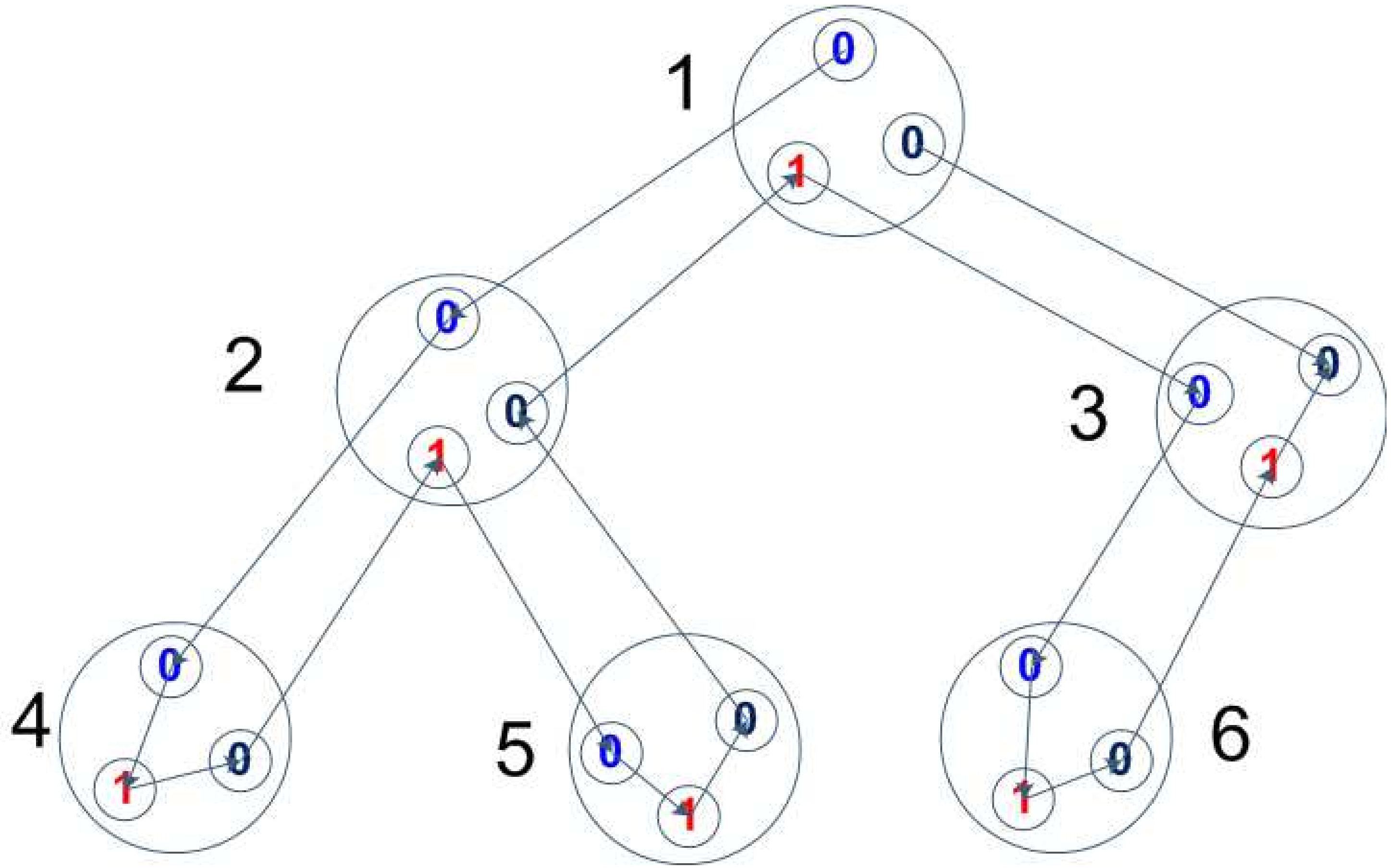
Calculate the Suffix-Sum of the List.



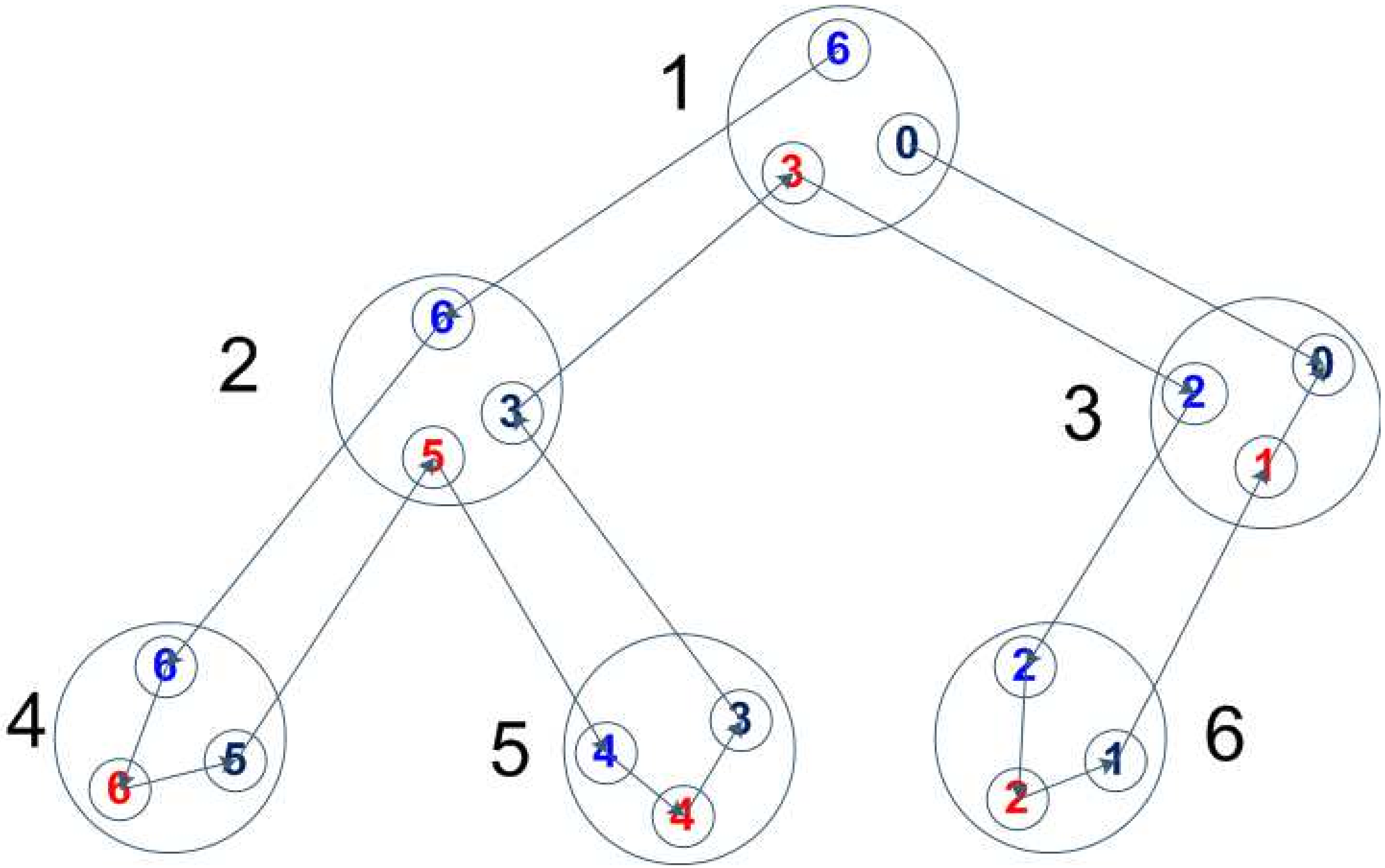
PreOrder's order

- Node's traversing order v: |V| - v[A] + 1.
  - 1[A] = 6 → Position(1) = 6 – 6 + 1 = 1.
  - 2[A] = 5 → Position(2) = 6 – 5 + 1 = 2.
  - 3[A] = 2 → Position(3) = 6 – 2 + 1 = 5.
  - 4[A] = 4 → Position(4) = 6 – 4 + 1 = 3.
  - 5[A] = 3 → Position(5) = 6 – 3 + 1 = 4.
  - 6[A] = 1 → Position(6) = 6 – 1 + 1 = 6.
- The traversing order is: [1 → 2 → 4 → 5 → 3 → 6]

InOrder: A = 0, B = 1, C = 0.



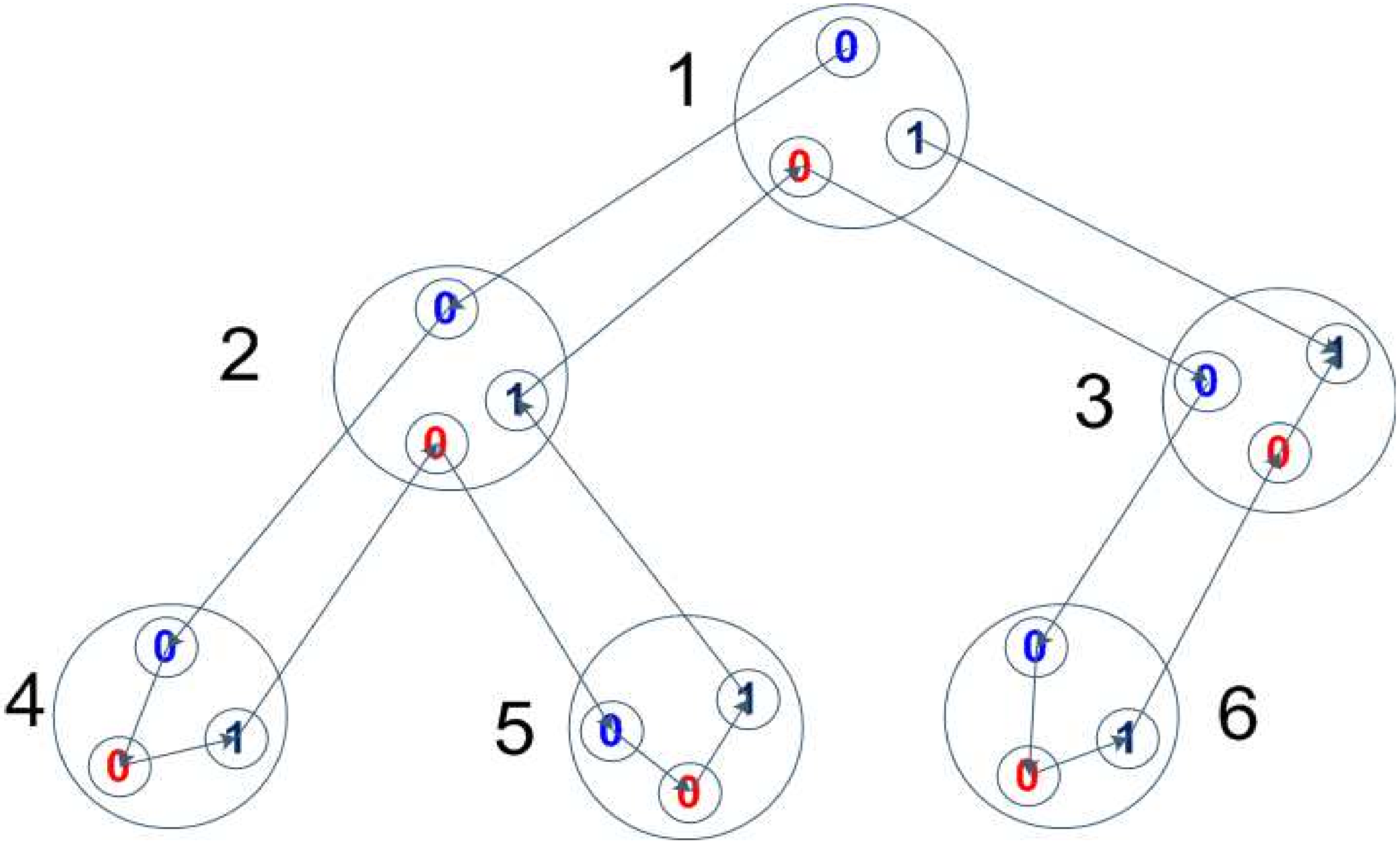
Calculate the Suffix-Sum of the List



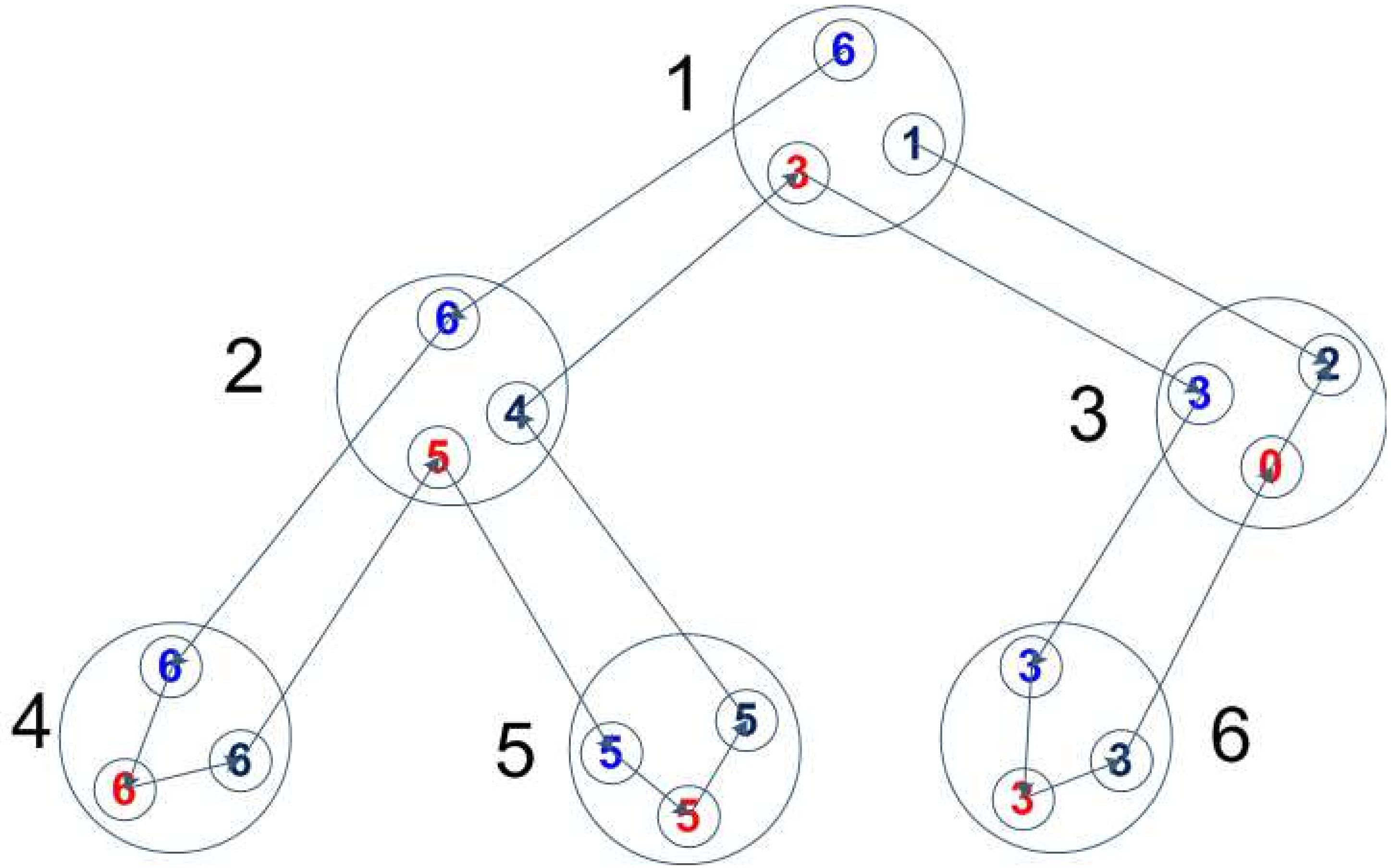
InOrder's order

- Node's traversing order v: |V| - v[B] + 1.
  - 1[B] = 3 → Position(1) = 6 – 3 + 1 = 4.
  - 2[B] = 5 → Position(2) = 6 – 5 + 1 = 2.
  - 3[B] = 1 → Position(3) = 6 – 1 + 1 = 6.
  - 4[B] = 6 → Position(4) = 6 – 6 + 1 = 1.
  - 5[B] = 4 → Position(5) = 6 – 4 + 1 = 3.
  - 6[B] = 2 → Position(6) = 6 – 2 + 1 = 5.
- The traversing order is: [4 → 2 → 5 → 1 → 6 → 3]

PostOrder: A = 0, B = 0, C = 1.



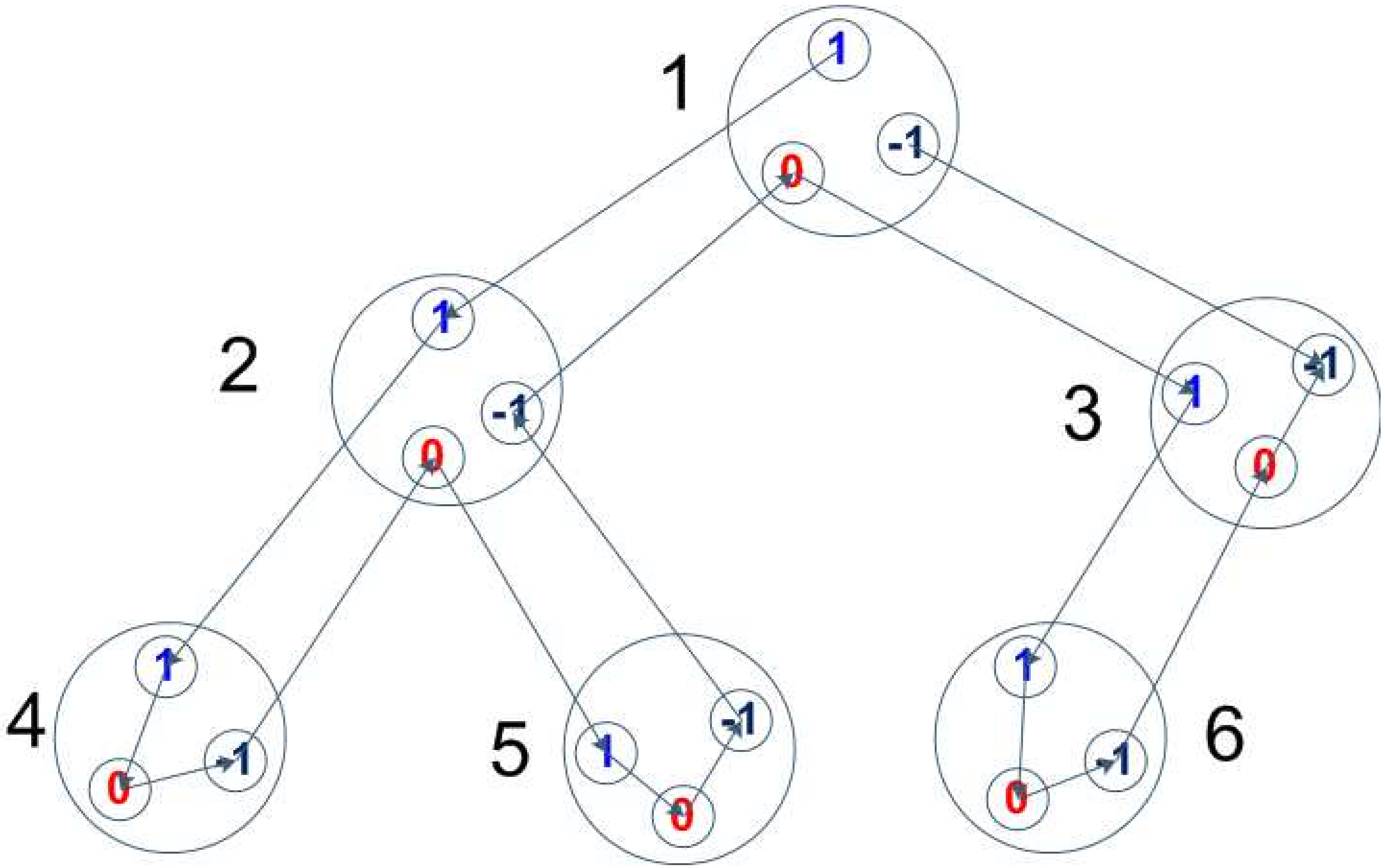
Calculate the Suffix-Sum of the List



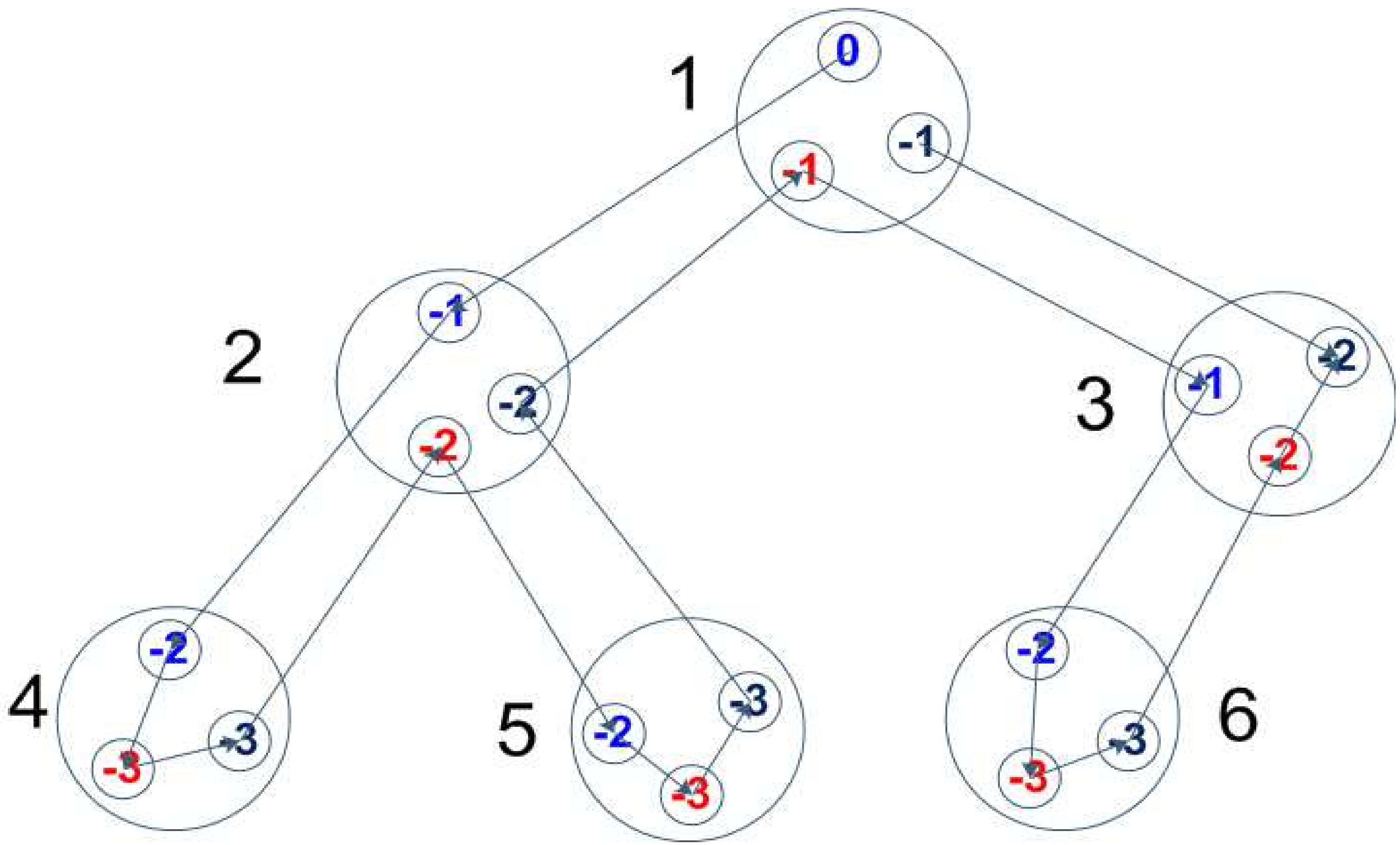
PostOrder's order

- Node's traversing order v: |V| - v[C] + 1.
  - 1[C] = 1 → Position(1) = 6 – 1 + 1 = 6.
  - 2[C] = 4 → Position(2) = 6 – 4 + 1 = 3.
  - 3[C] = 2 → Position(3) = 6 – 2 + 1 = 5.
  - 4[C] = 6 → Position(4) = 6 – 6 + 1 = 1.
  - 5[C] = 5 → Position(5) = 6 – 5 + 1 = 2.
  - 6[C] = 3 → Position(6) = 6 – 3 + 1 = 4.
- The traversing order is: [4 → 5 → 2 → 6 → 3 → 1]

Depth(v) : A = 1, B = 0, C = -1



Calculate the Suffix-Sum of the List

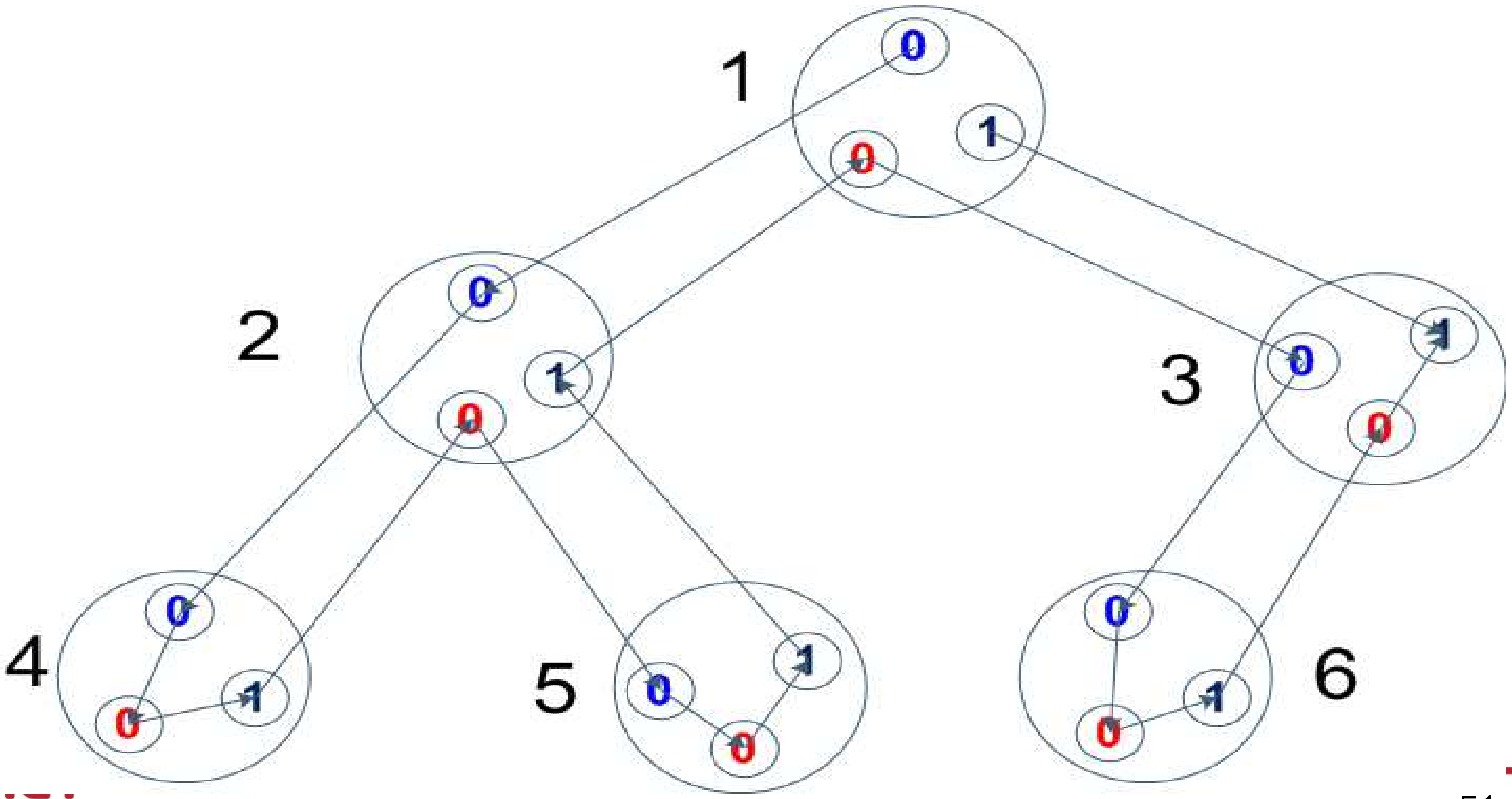


Specify the depth of nodes

- Depth of node v: abs(v[A])
  - 1[A] = 0 → Depth(1) = 0.
  - 2[A] = -1 → Depth (2) = 1.
  - 3[A] = -1 → Depth (3) = 1.
  - 4[A] = -2 → Depth (4) = 2.
  - 5[A] = -2 → Depth (5) = 2.
  - 6[A] = -2 → Depth (6) = 2.
- Node's Height: Height(v) = H - Depth(v) where H = max{ Depth(v)}.

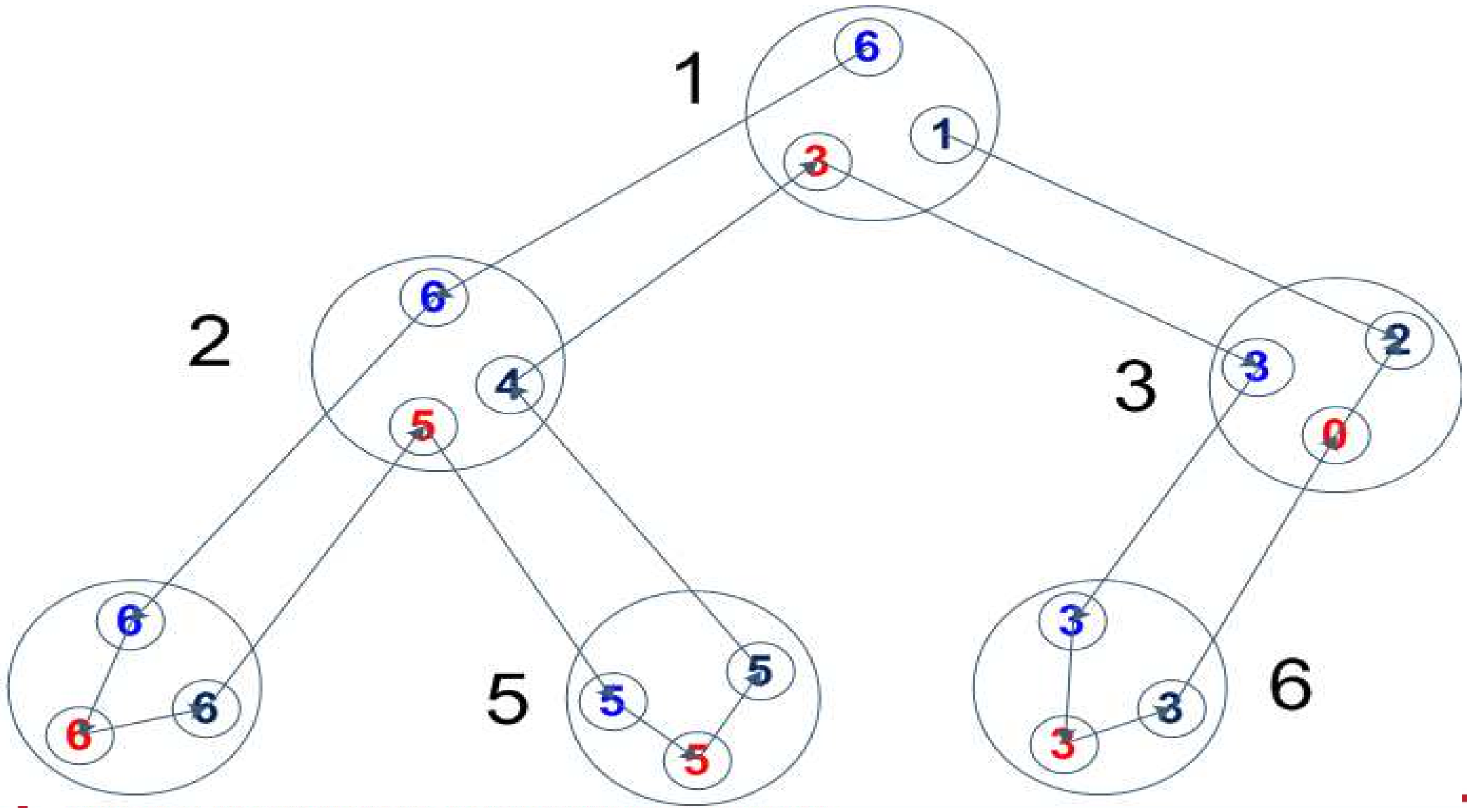
Determining the size of the tree with the root v

- For all v, specifying the number of nodes in the sub-tree that consider v as the root. Set A = 0, B = 0, C = 1.



Determining the size of the tree with the root v

- Calculate the Suffix-sum of the list based on the Euler cycle



Determine the size of the tree with the root v

- Size(v) = v[A] – V[C] + 1.
  - Size(1) = 6 – 1 + 1 = 6.
  - Size(2) = 6 – 4 + 1 = 3.
  - Size(3) = 3 – 2 + 1 = 2.
  - Size(4) = 6 – 6 + 1 = 1.
  - Size(5) = 5 – 5 + 1 = 1.
  - Size(6) = 3 – 3 + 1 = 1.

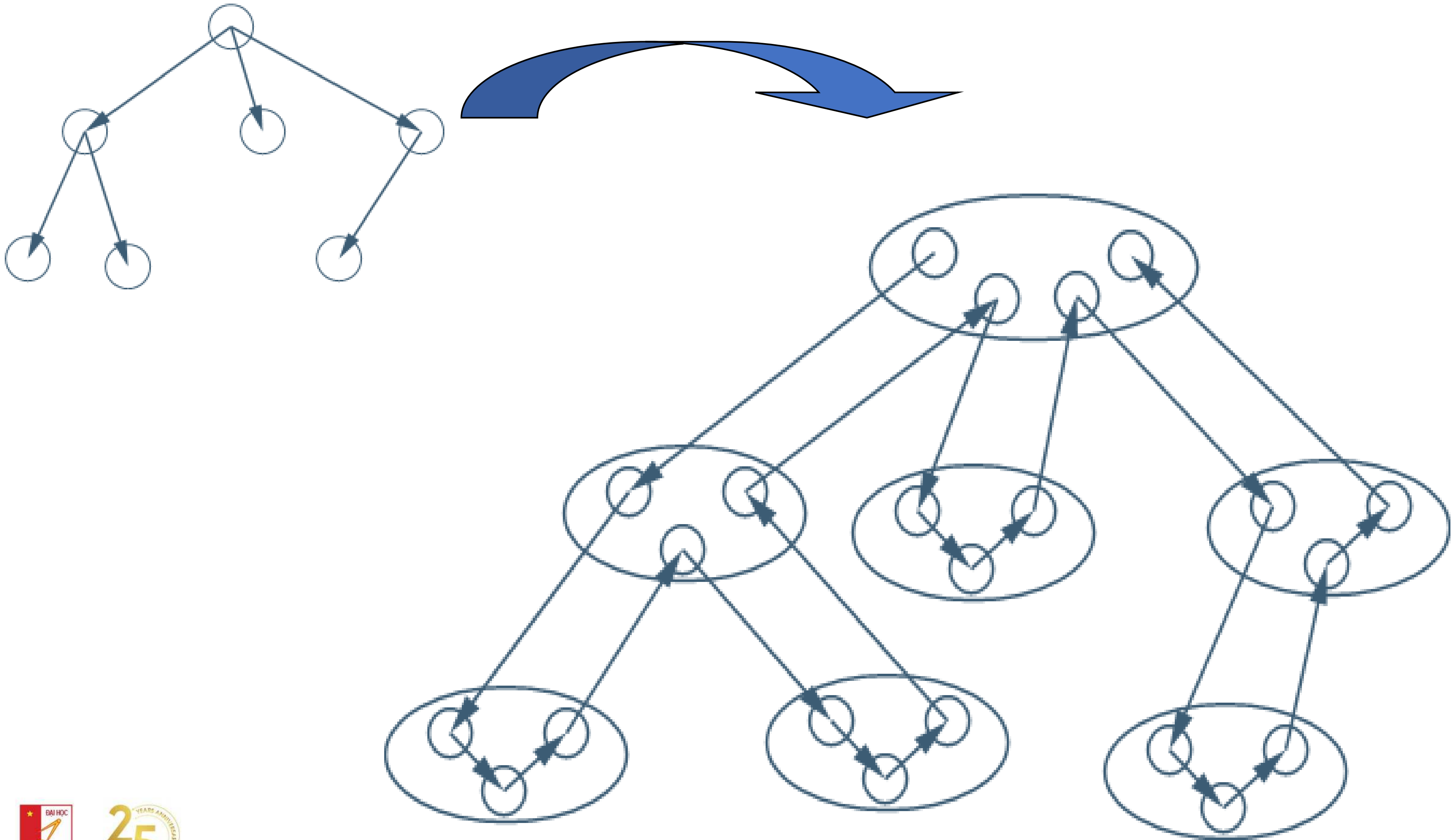
Euler cycle for general tree

- Consider node v. Supposing {v1, v2, .., vm} are the children of v from left to right.
- Node v is presented by m+1 child nodes:
  - v[A] : the entrance point of v in the Euler cycle.
  - v[C] : point out of v in euler cycle.
  - v[B_k]: connect to the child nodes of v_{k+1}. (k = 1..m-1)
- If v is a leaf node or has only 1 child, v is still presented by v[A], v[B], v[C].

Node's connecting rules

- Rules for A:
  - If v has an outer-left child v_1 then v[A] connected to v_1[A].
  - If v does not have children, then v[A] connects to v[B].
- Rules for B:
  - If v is the leaf node, then v[B] connects to v[C].
  - If v has child nodes $\{v_1, v_2, .., v_m\}$ then v[B_k] connected to v_{k+1}[A] with k = 1..m-1.
- Rules for C:
  - If v is the outer-right child of u then v[C] connects to u[C].
  - If v is the k-child of u then v[C] connects to u [B_k].
  - If v is the root, then v[C] connects to NULL.

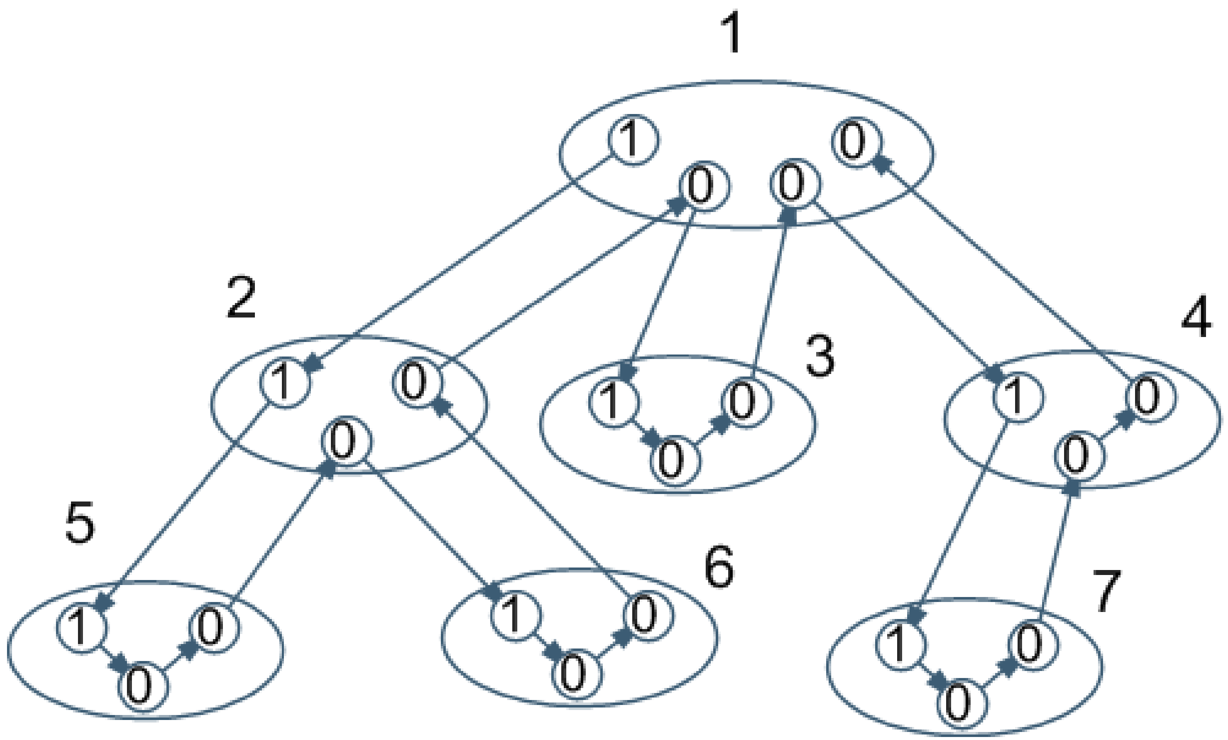
Illustration



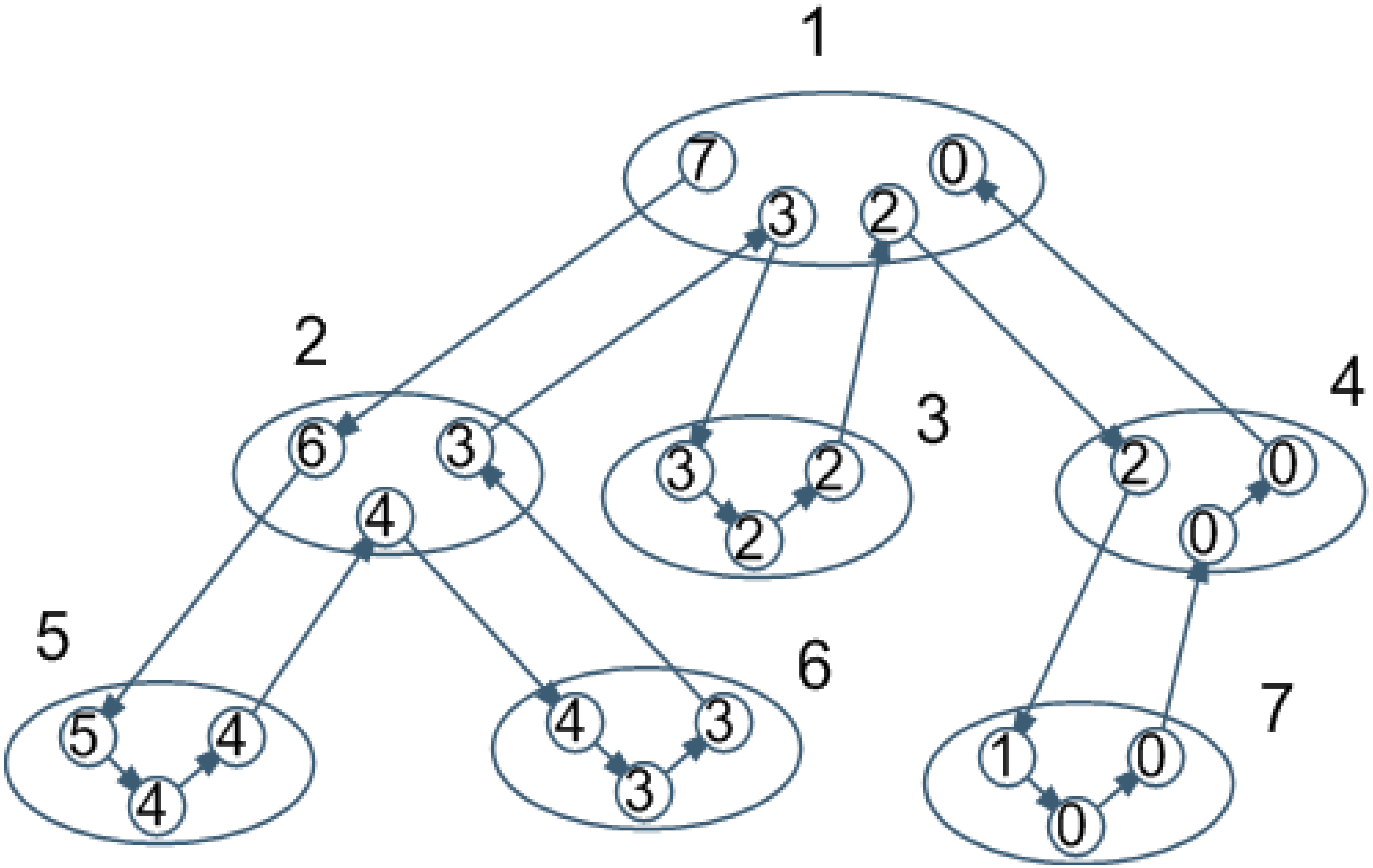
Problems with the general tree

- Traversing problem:
  - No Inorder concept.
  - PreOrder and PostOrder's traversing are based on the A input node A and output node C. Values $B_k = 0$.
    - Preorder : A = 1; C = 0.
    - Postorder: A = 0; C = 1.
- Depth problem: A = 1; C = -1.
- Sub-tree size problem: A = 0; C = 1.

PreOrder: $A = 1, B_k = 0, C = 0$



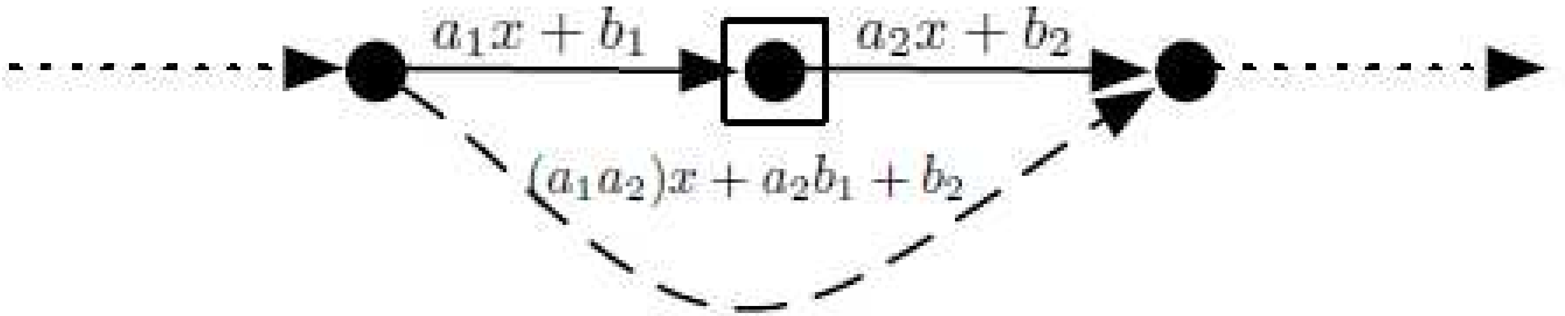
PreOrder: $A = 1, B_k = 0, C = 0$



[1 → 2 → 5 → 6 → 3 → 4 → 7]

10.2.3 Tree Contraction

- For a binary tree. Let's reduce a main tree to a smaller tree consisting of 1 root and 2 child nodes.
- For example, the example shortens 3 nodes to 2 nodes:

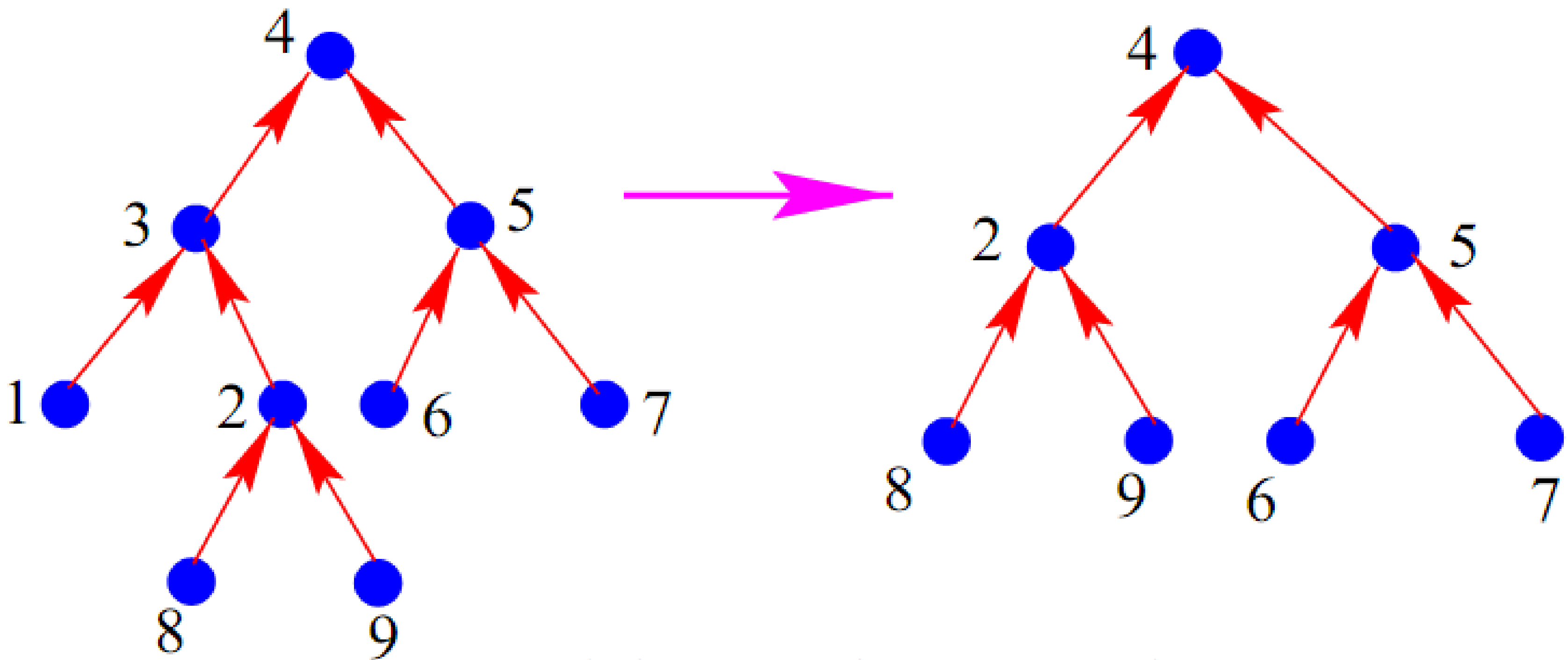


Approach

- Tree T = (V,E) is a binary tree with root r:
  - p(v) is the parent node of v on the T tree.
  - sib(v) is the brother node of v: being the child of the same parent node. (sib = sibling).
- RAKE operation for leaf node v: p(v) ≠ r
  - Delete nodes v, p(v) on the T tree.
  - Connect sib(v) to p(p(v)) on the T tree.

Approach

- RAKE operation - reduce the leaf nodes:



Applying Rake to node 1

Approach

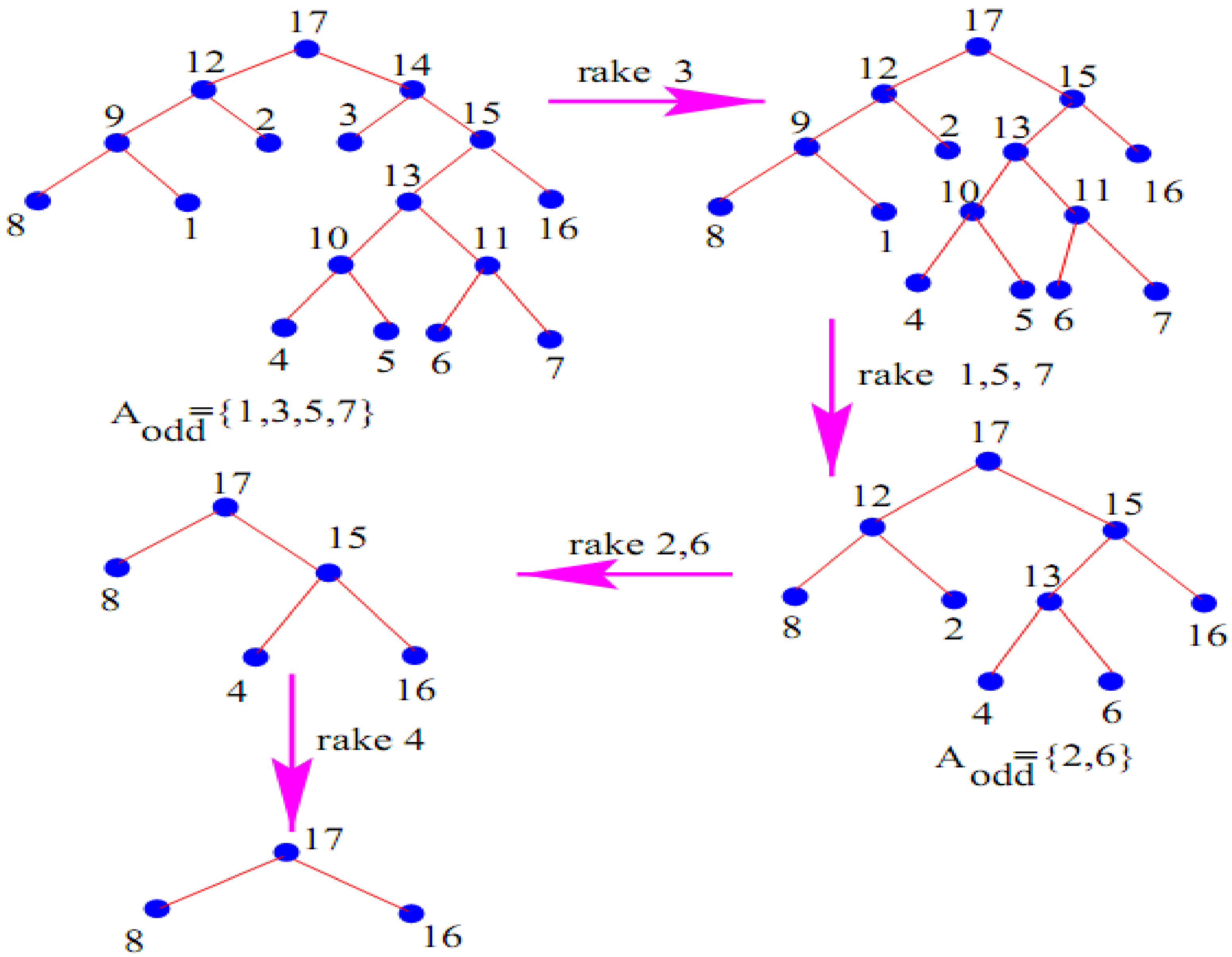
- Problems arises:
  - The RAKE can't be performed with the leaf node connected to the root.
  - All leaves cannot be excluded by a RAKE operation in parallel?
  - → it works only on the leaves that their fathers do not adjacent to each other.
  - For example, nodes of 1, 8, 9 cannot be RAKE together.

Approach

- Solution:
  - Each parent node must store information about its left child and its right child nodes.
  - Highlight leaf nodes in the order from 1..n
  - Consider nodes with odd index:
    - The nodes are the left child, and their fathers won't be together. It is called as Group 1.
    - The same with the right-child nodes. It is called as Group 2.
  - → implemented in parallel on each group in turn will ensure that RAKE operation’s condition is not violated.

Algorithm steps

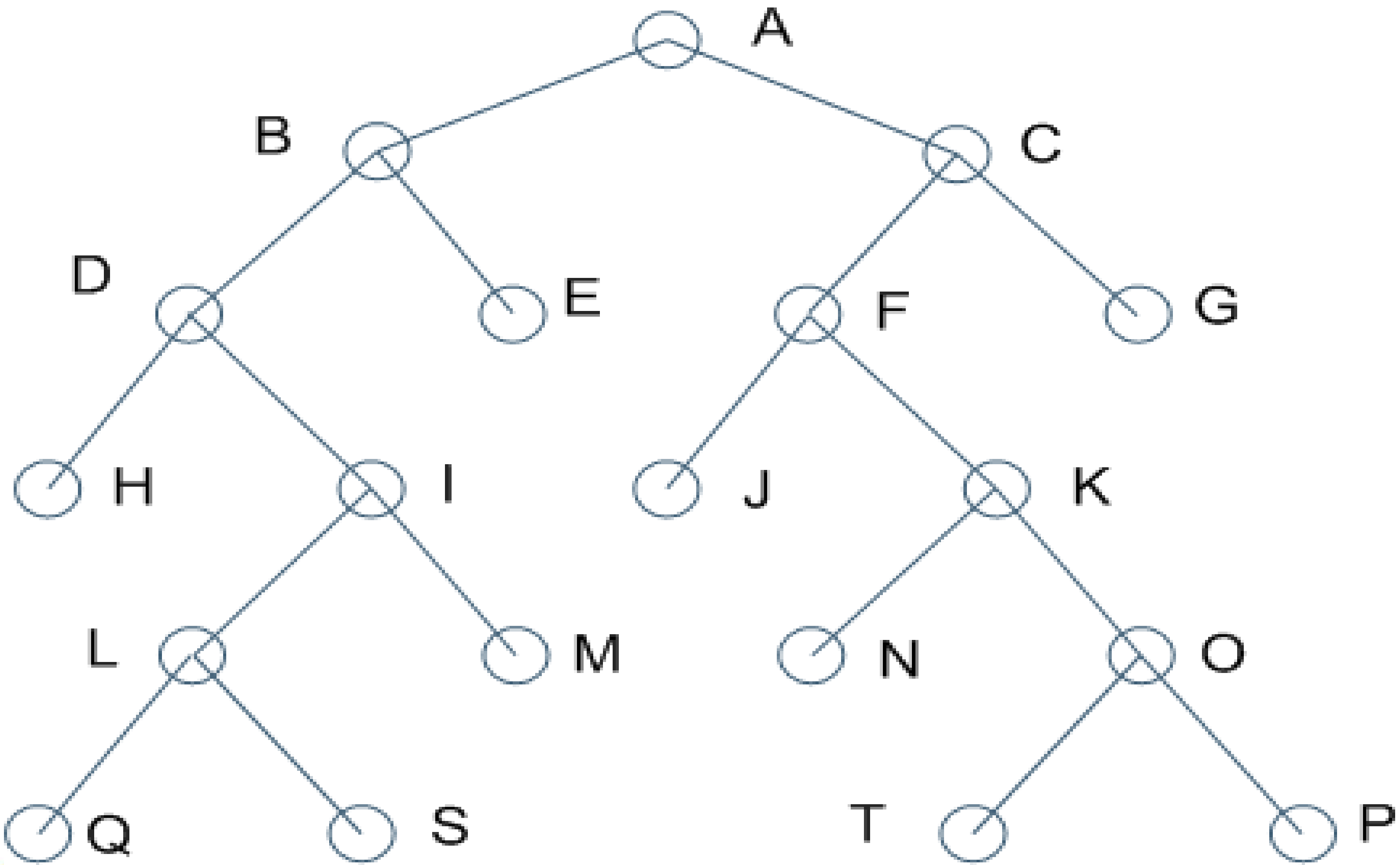
- S1. Marking the leaf nodes in order from 1..n to save to array Z, except for 2 leaf nodes located on the left, on the right end.
- Repeat:
  - S2. Performing RAKE with Z[k] nodes if k is odd and the node must be left child.
  - S3. Performing RAKE with Z[k] nodes with remaining odd-values k.
  - S4. Assign Z = set of Z[k] if k is even.
- Until there are 3 nodes left, then the algorithm stops.

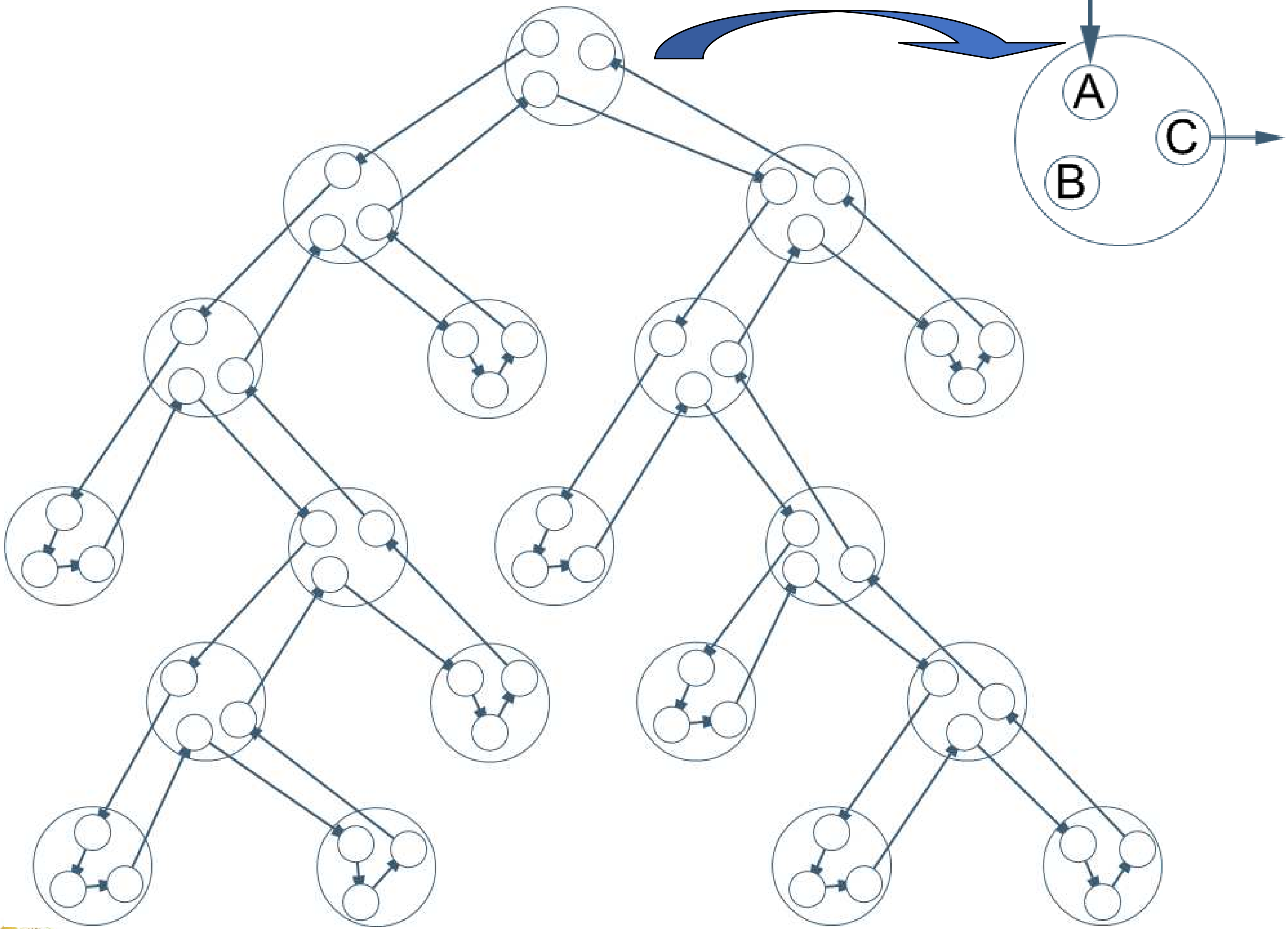

17
12
14
9
2
3
15
13
8
1
10
11
16
4
5
6
7
A odd ={1,3,5,7}
rake 3
17
12
15
9
2
13
8
1
10
11
16
4
5
6
7
rake 1,5, 7
17
12
15
8
2
13
16
4
6
A odd ={2,6}
rake 2,6
17
15
8
4
16
rake 4
17
8
16

Detailed steps

- Solving step 1 of the algorithm:
  - Given tree T = (V,E).
  - Numbering the leaves from left to right (except for left-end/right-end nodes) in order from 1...n.
- Solution:
  - Using Euler cycle.
- Illustration with binary tree (each node has exactly 2 sub-nodes).

Order leaves from left to right

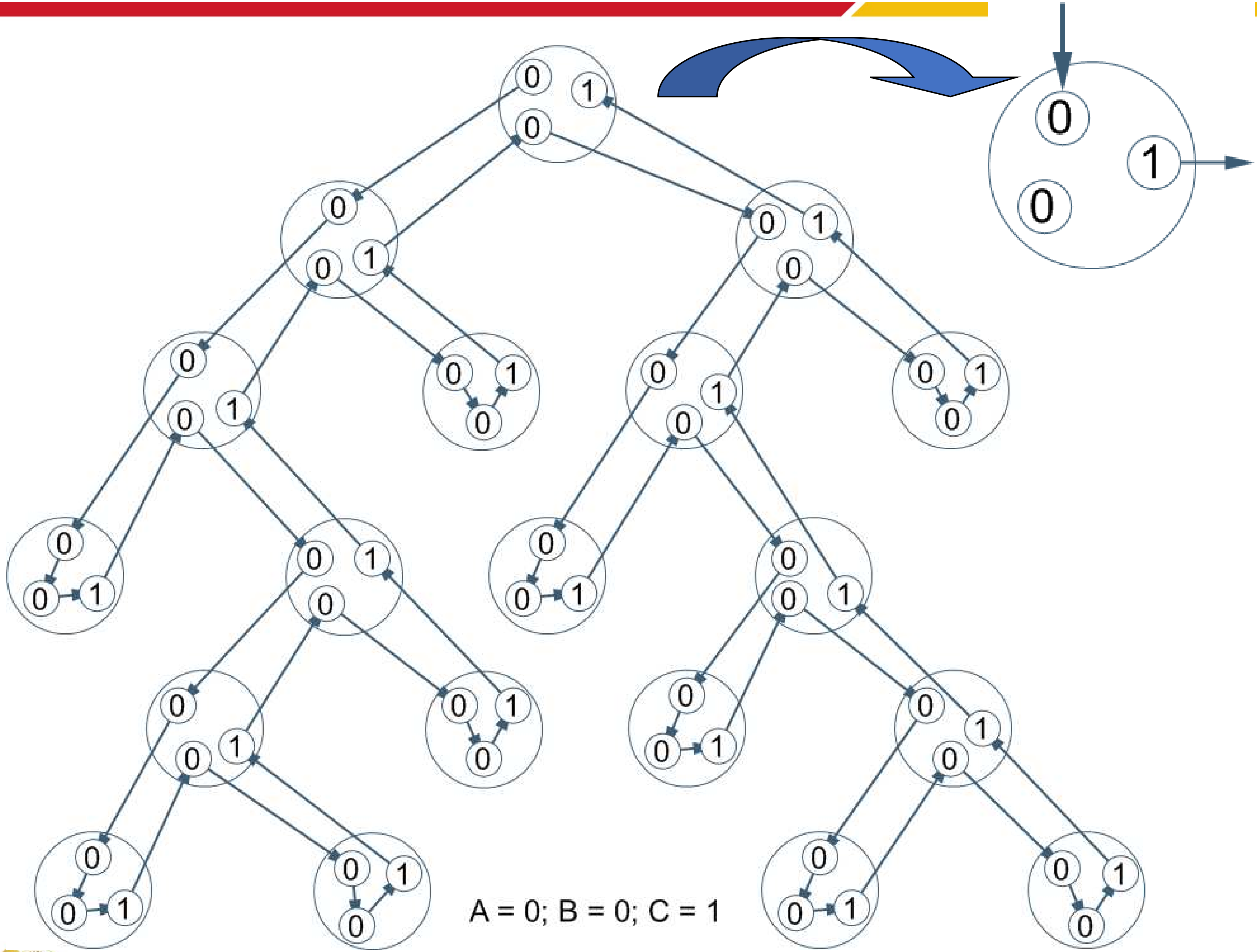


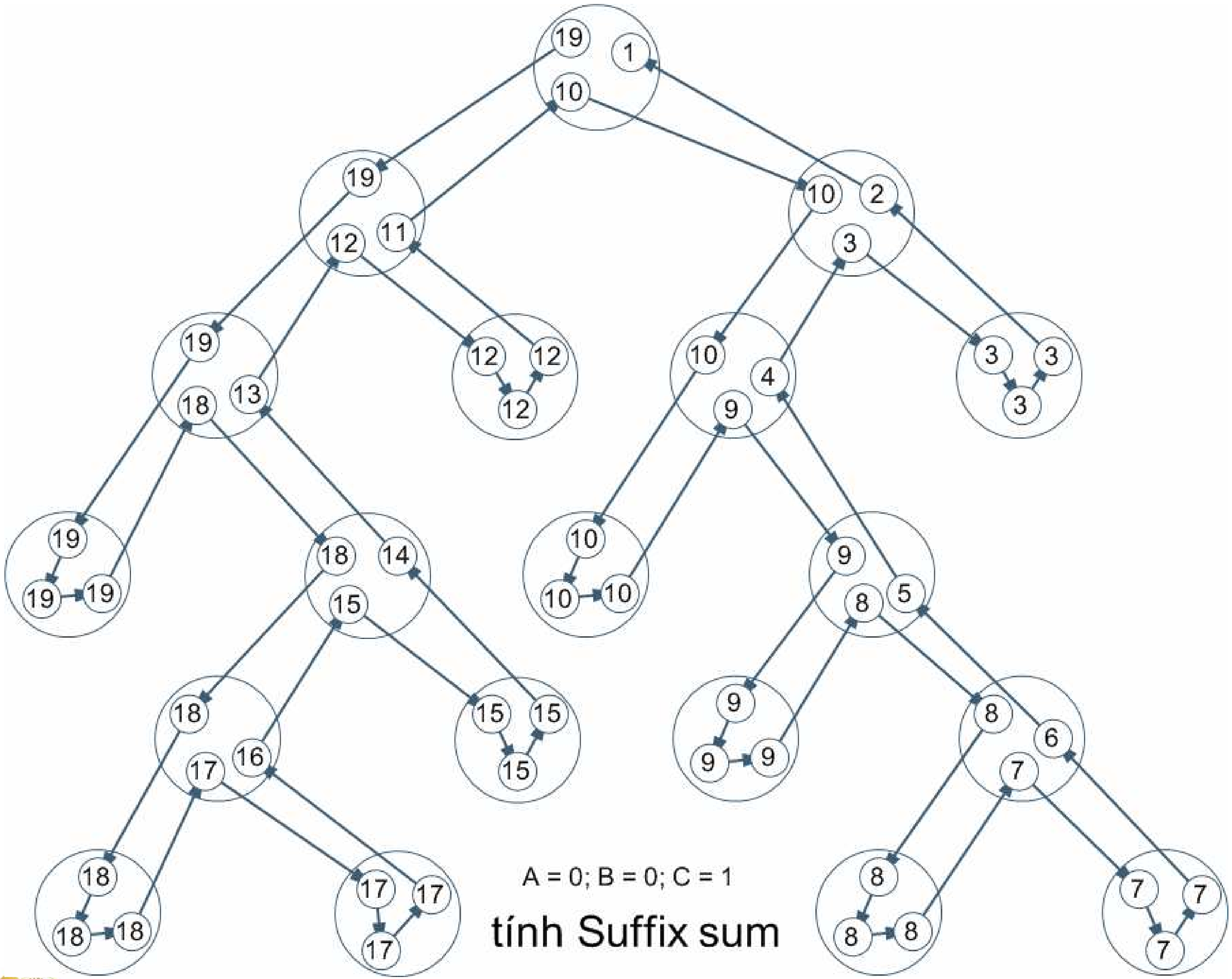

A
B
C

Defining leaf nodes

- Building the Euler cycle on tree.
- At each node v : v[A] = 0; v[B] = 0; v[C] = 1.
- Calculate the Suffix-Sum for nodes on the list generated from the Euler cycle.
- Leaf node has following characteristics: suffix-sum values at its child nodes are equal: v[A] = v[B] = v[C].
- From the picture, we have the leaves as follows:

  [H Q S M E J N T P G]


A = 0; B = 0; C = 1

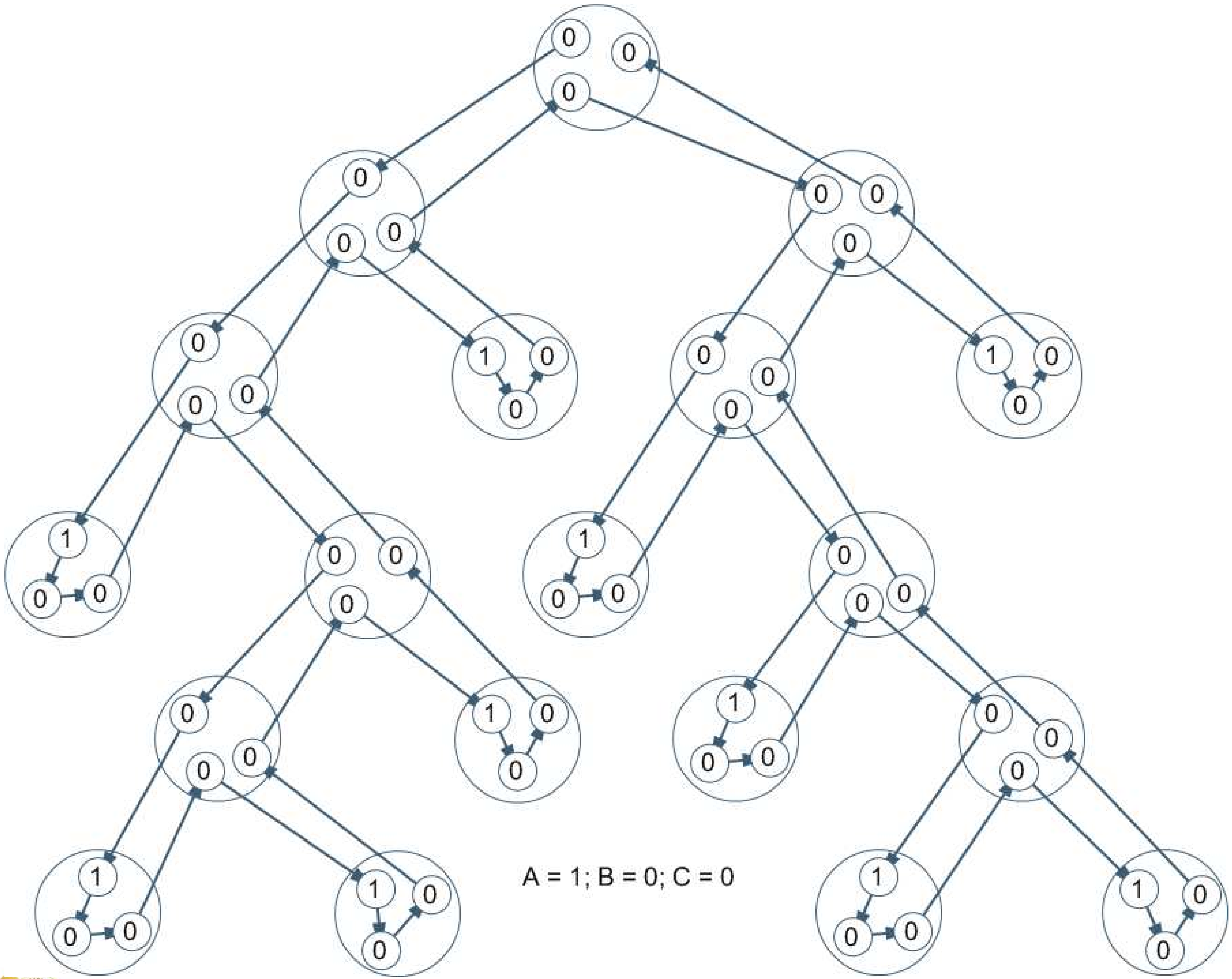


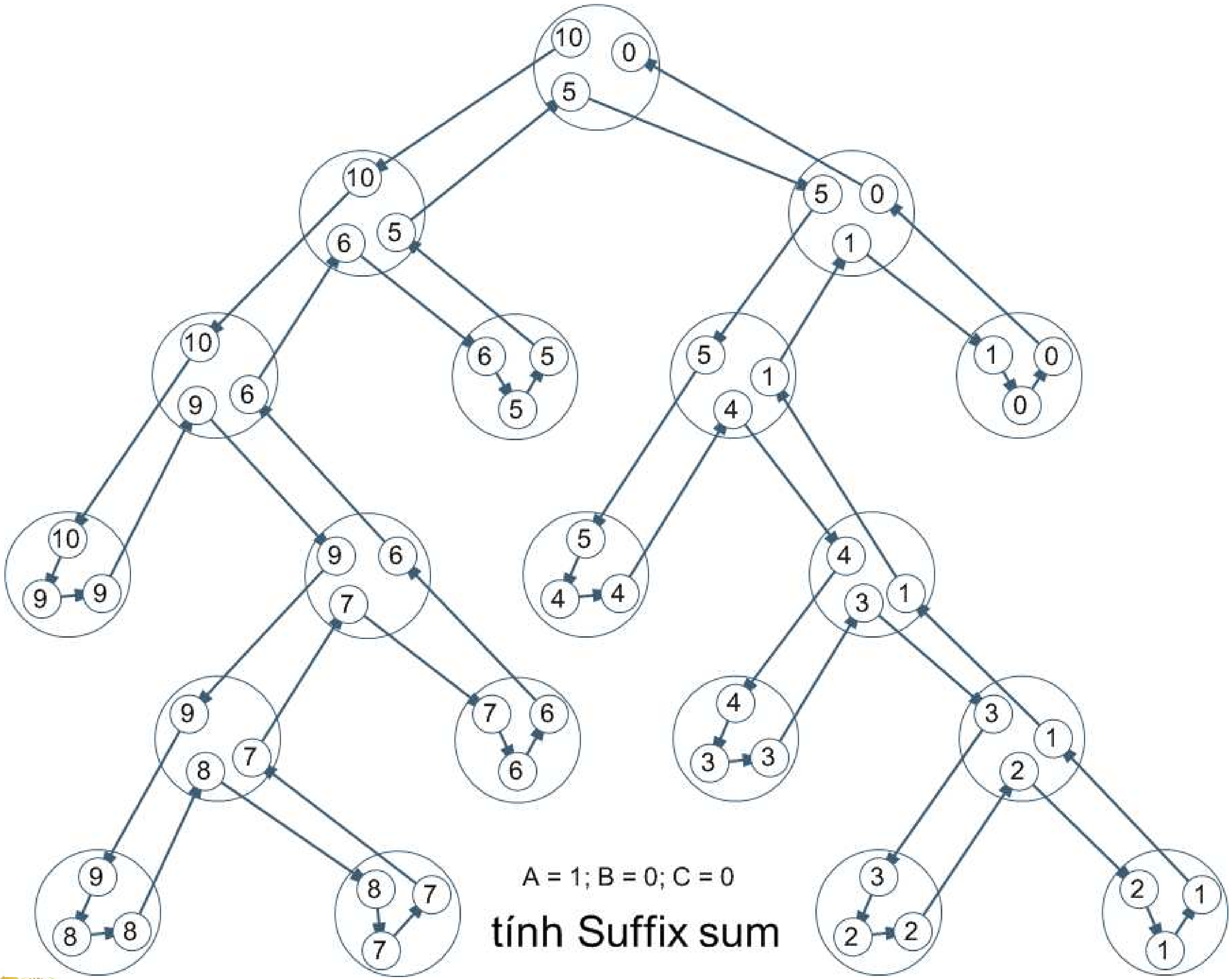
A = 0; B = 0; C = 1

tính Suffix sum

Set the order for the leaves

- Set A = 1, B = 0, C = 0.
- Calculate the Suffix-sum for nodes on the list generated from the Euler cycle.
- Order of the leaves sorted from right to left through value v[A]
- Node can be numbered left-to-right using formula: |number of leaves| - v[A] + 1.
- Store the leaves except for the leaf on the left end and for the leaf on the right end in array Z[1..n].

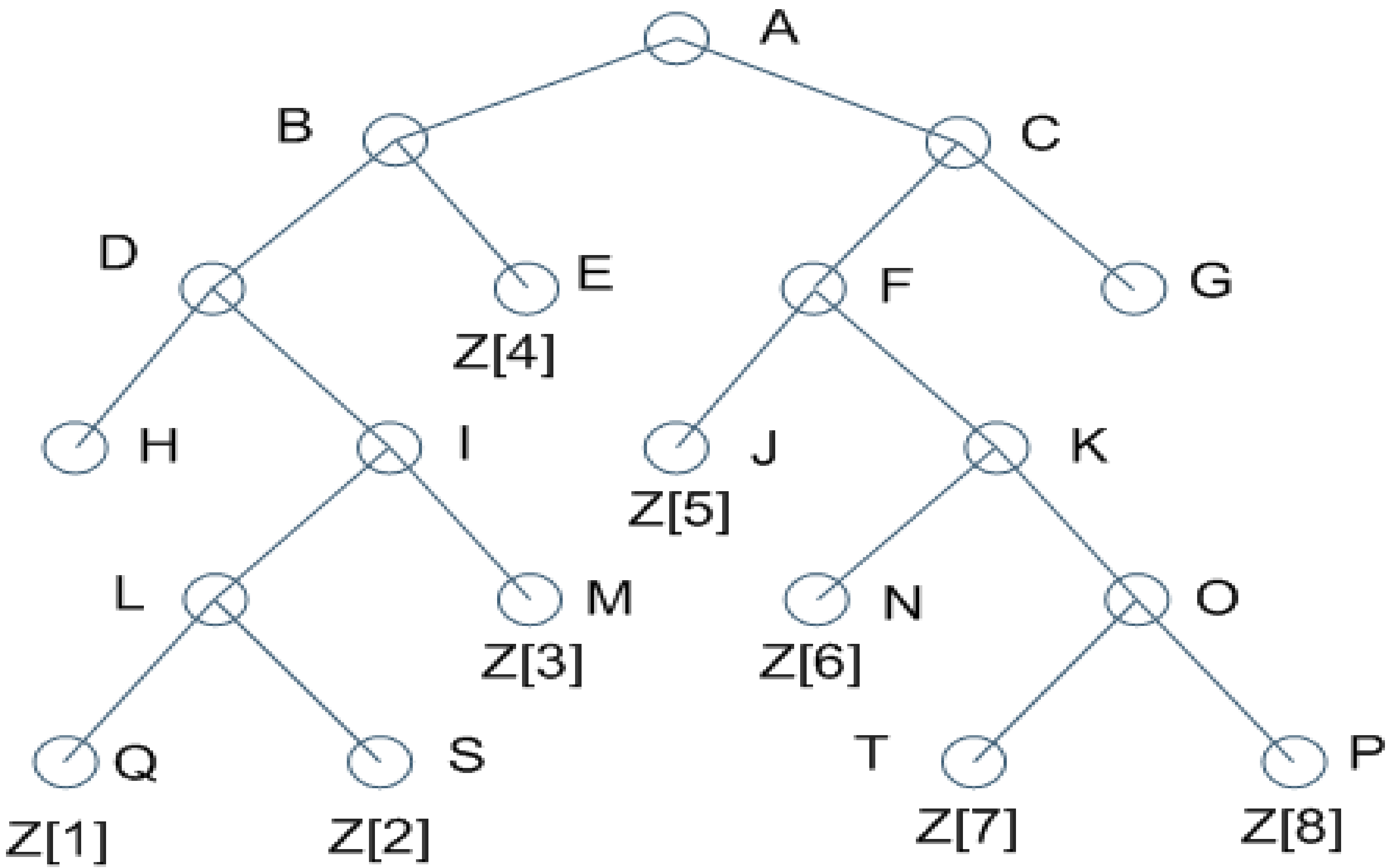

A = 1; B = 0; C = 0



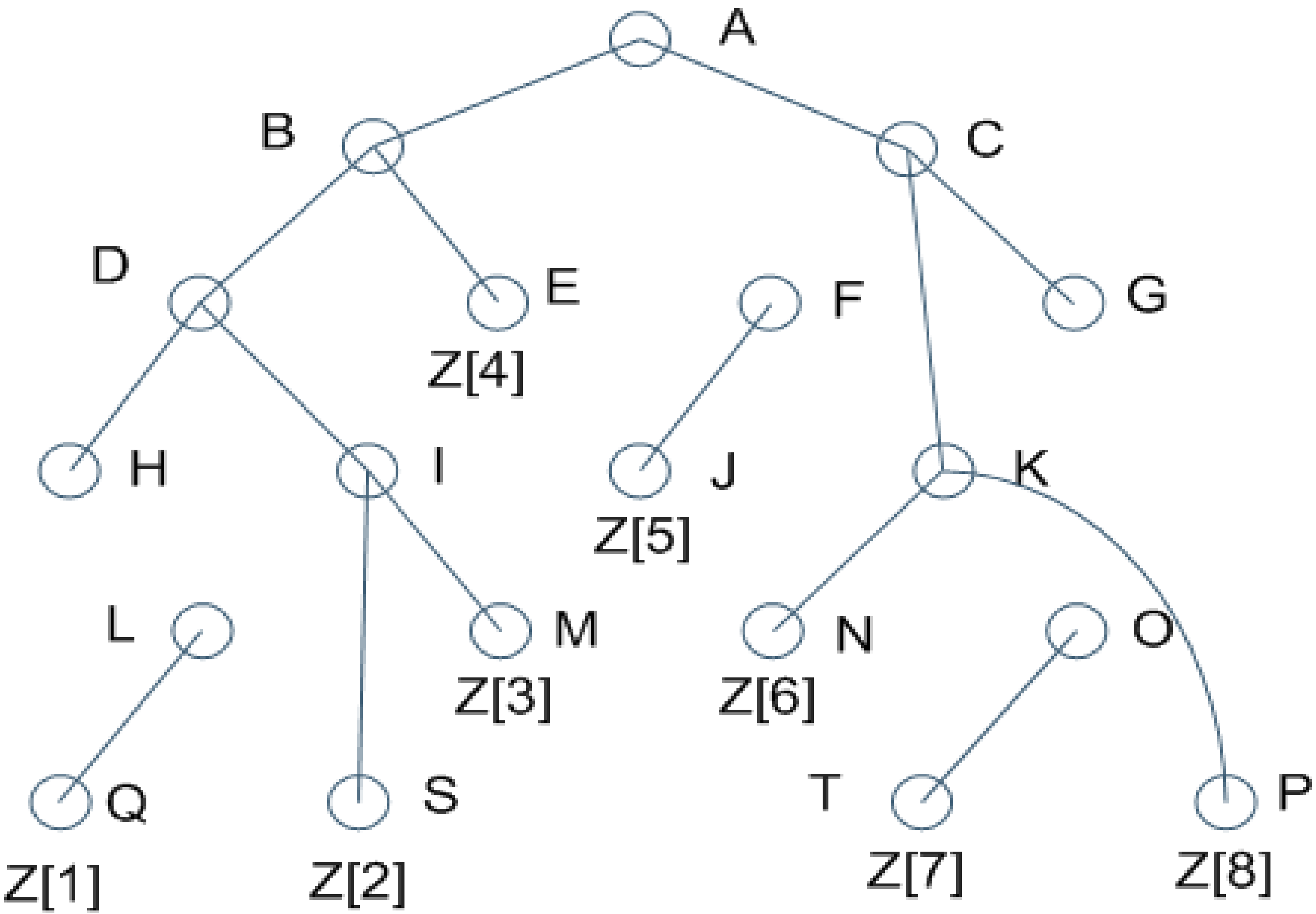
A = 1; B = 0; C = 0

tính Suffix sum

Assigning the order of leaves from left to right

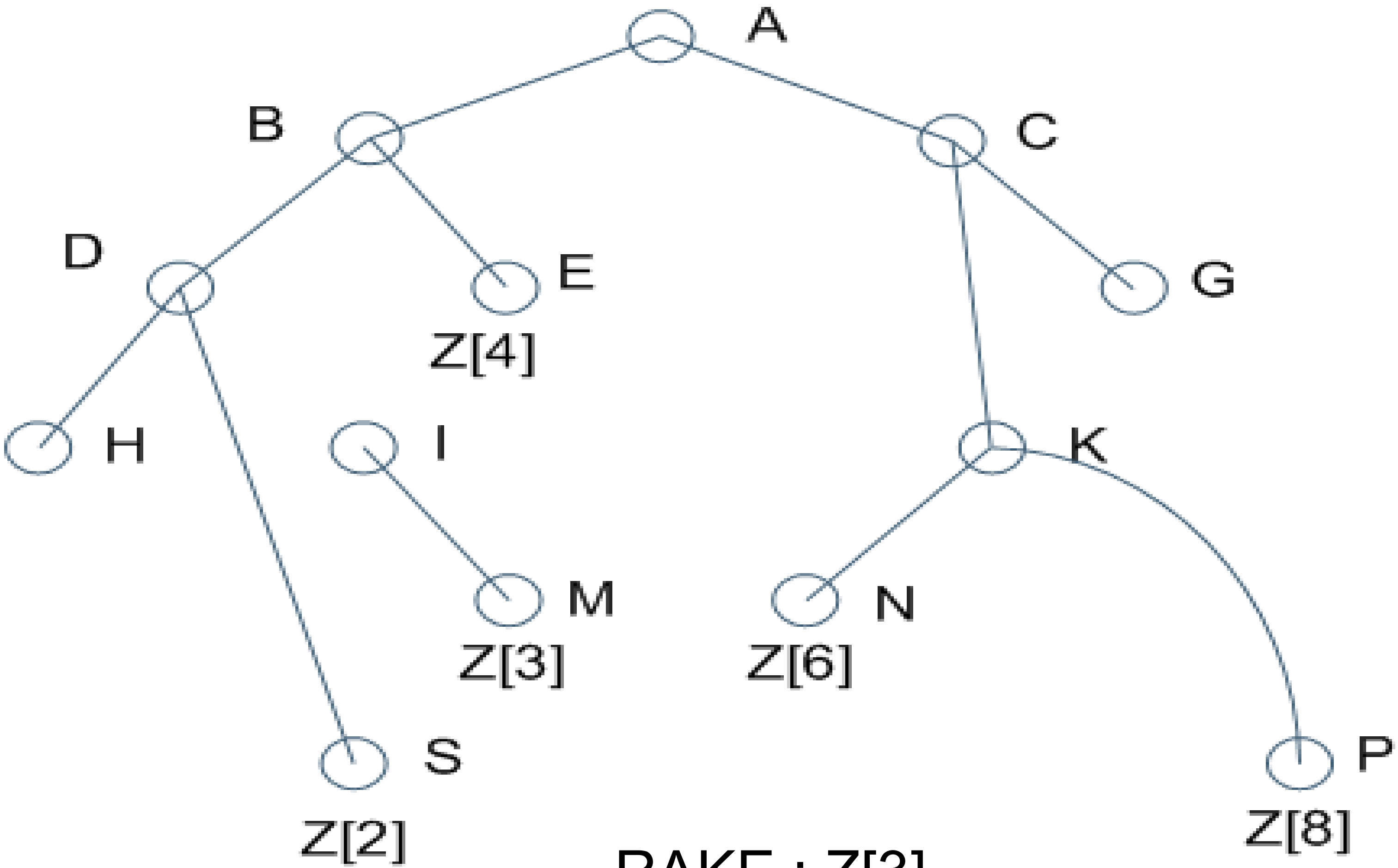


Tree Contraction (Step 1.1)



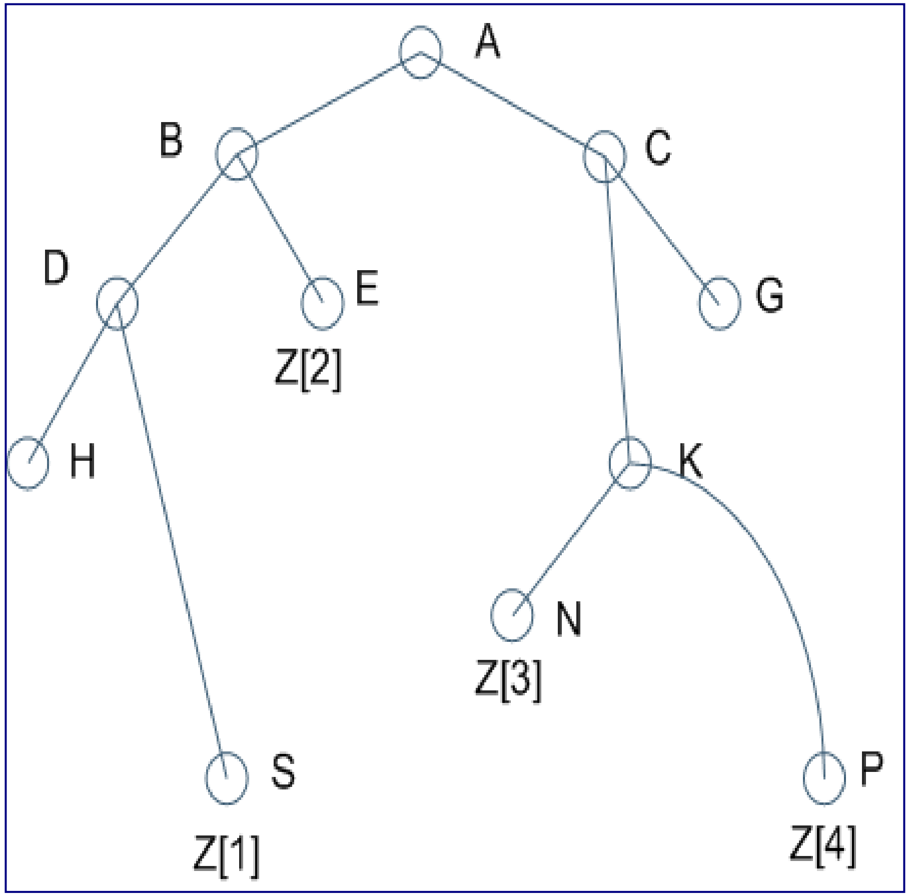
RAKE : Z[1], Z[5], Z[7]

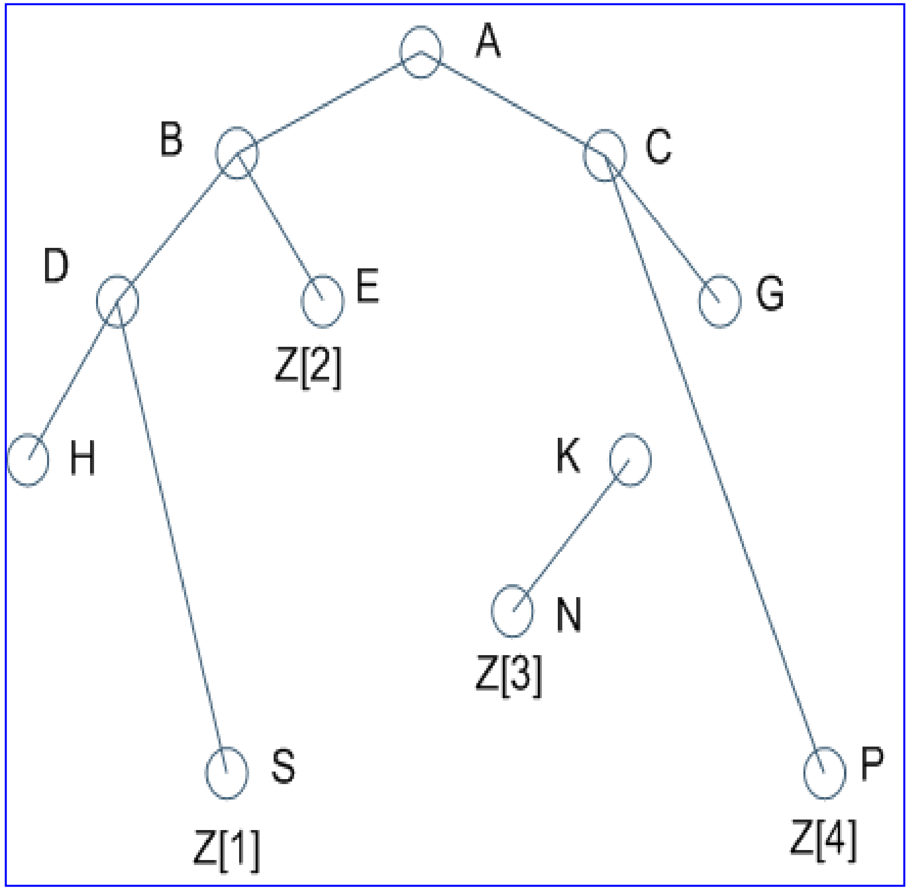
Tree Contraction (Step 1.2)



RAKE : Z[3]

Tree Contraction (Steps 1.3-2.1)

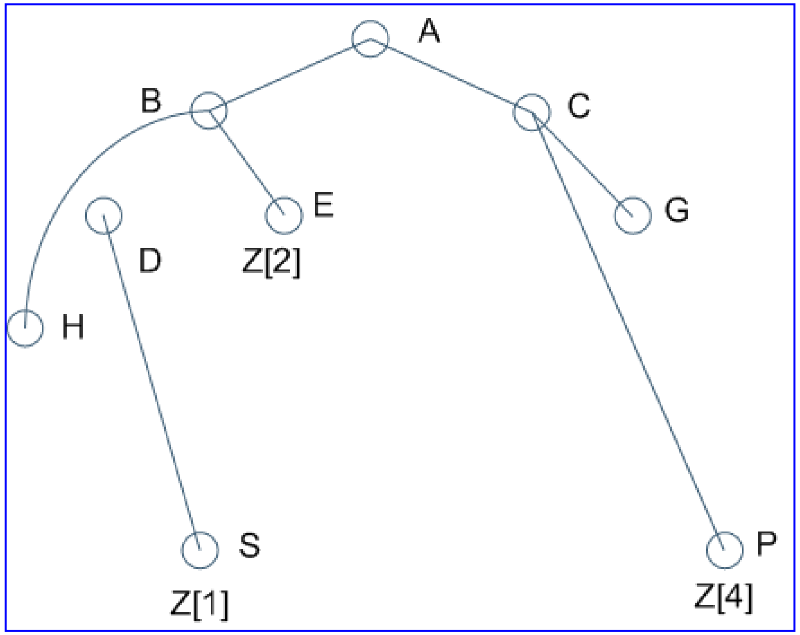




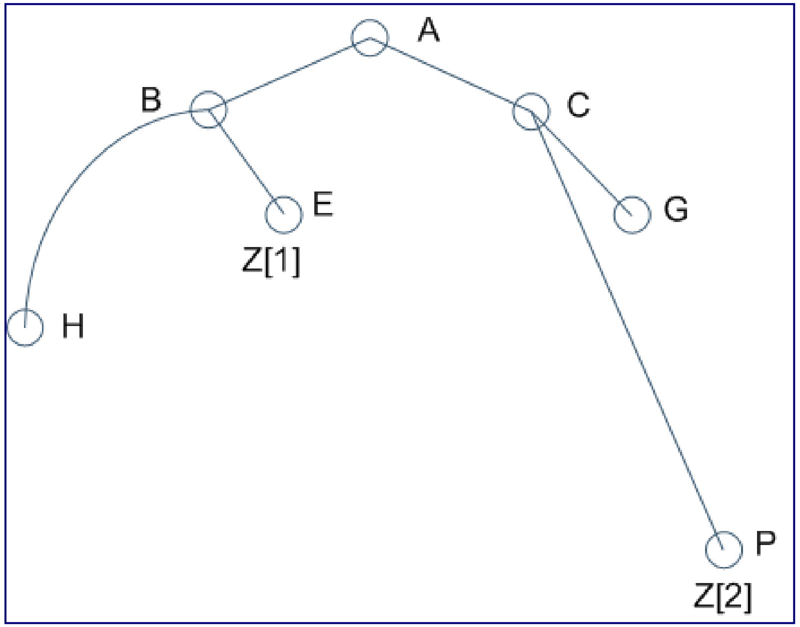
Z = Z[k] with k is even

RAKE: Z[3]

Tree Contraction (Steps 2.2-2.3)

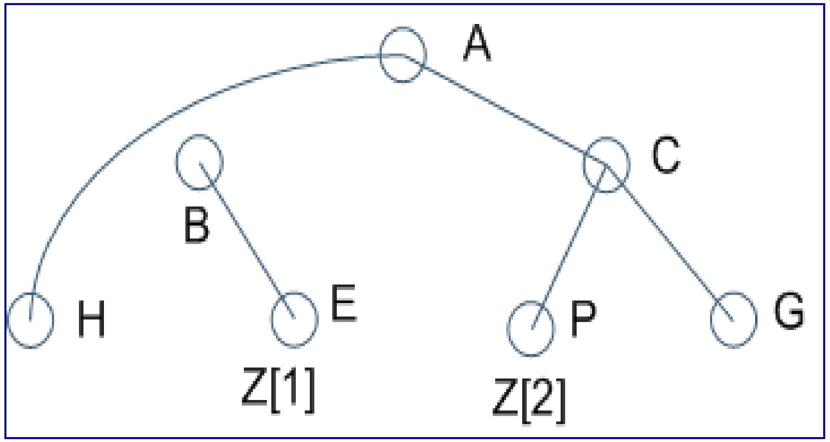


RAKE : Z[1]

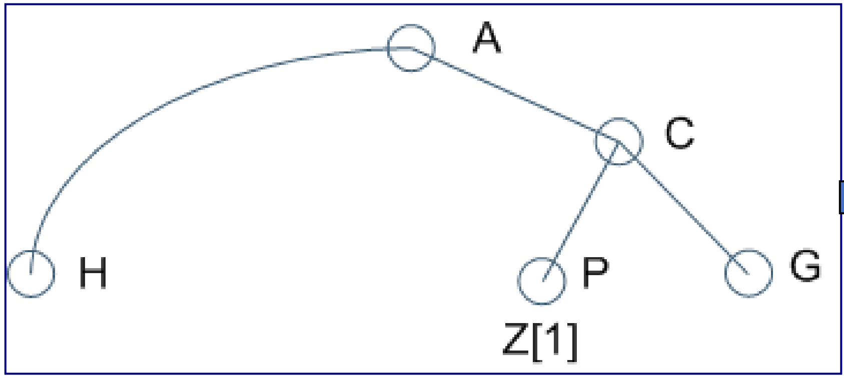


Z = Z[k] with k is even

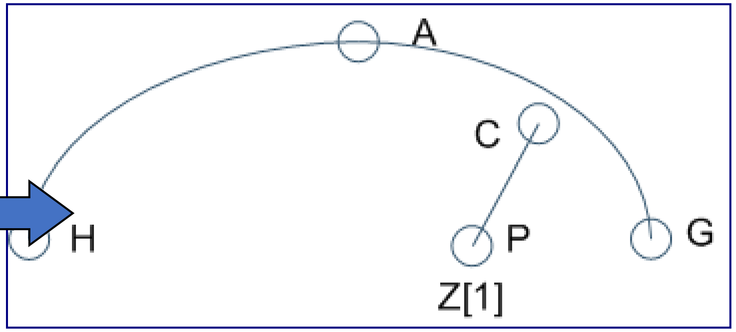
Tree Contraction (steps 3.1-3.2)



RAKE: Z[1]

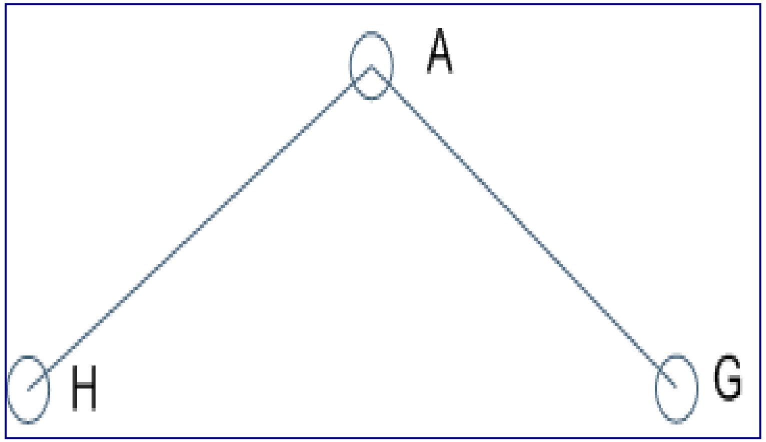


Assign: Z = Z[k] with even



RAKE : Z[1]

End



10.3 Write parallel tree-based programs

Write parallel tree-based programs

- Choose a tree-based algorithm
- Write a program implemented the chosen algorithm
- Run the program in a cluster consisting at least 2 connected linux-based computers.
- Evaluating the performance of the algorithm

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Thank you for your attentions!

soict.hust.edu.vn/ fb.com/groups/soict