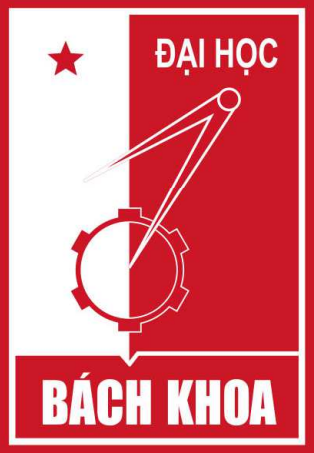


HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

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

Parallel in graph 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.

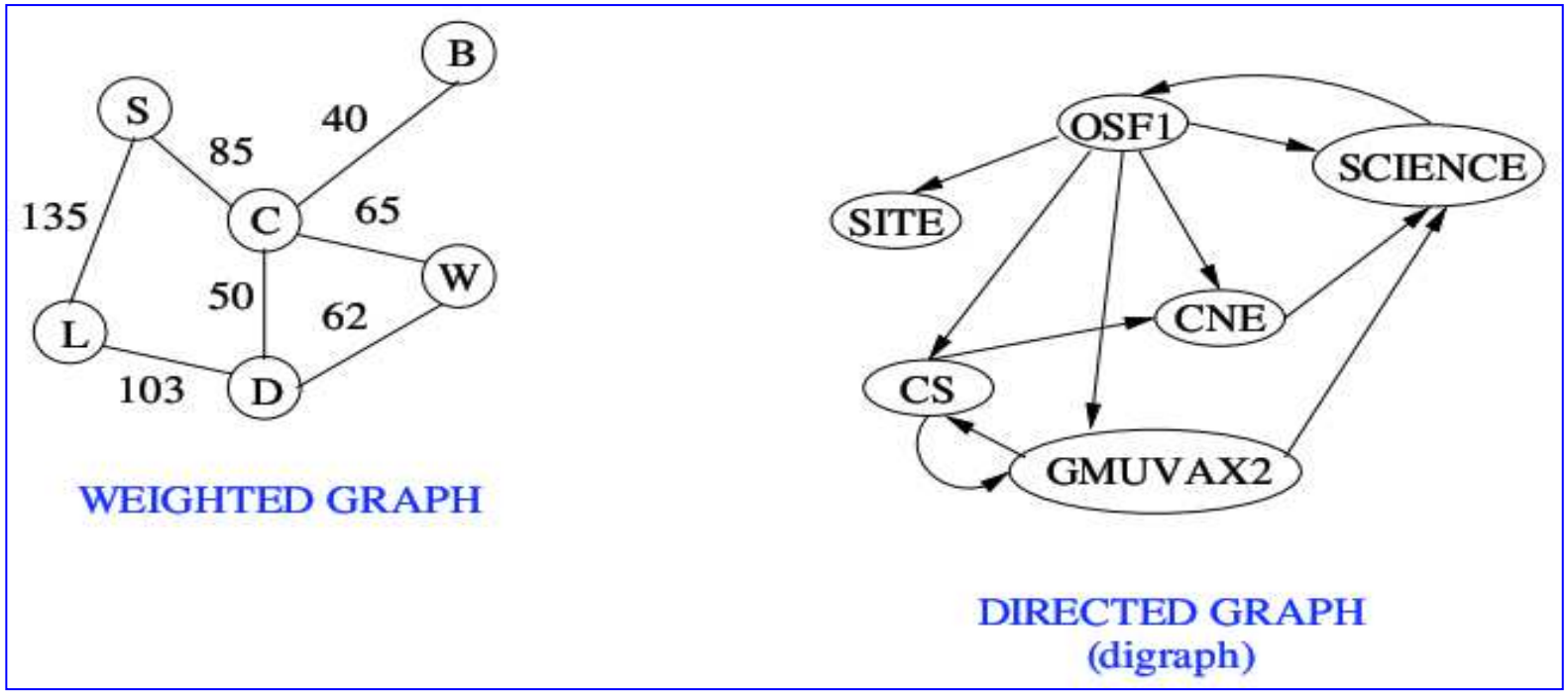
12.1 Parallel algorithm related to Euler tour

Basics

- A graph G = (V,E) is a set of vertexes and edges.
  - V is a limited set of vertexes.
  - E is a limited set of edges, e = (p,q) meaning node p conntects to node q, 2 nodes p,q $\in$ V.
- Sub-graph H = (V1, E1) of the G graph with:
  - $V_1 \in V$, $E_1 \in E$.
  - $e_1 \in E_1$: node p_1 connects to node q_1 $(p_1 , q_1) \in V_1$.

Types of graphs

- Considering some of the types of graphs used in this section:
  - Weighted graph.
  - Directed graph.

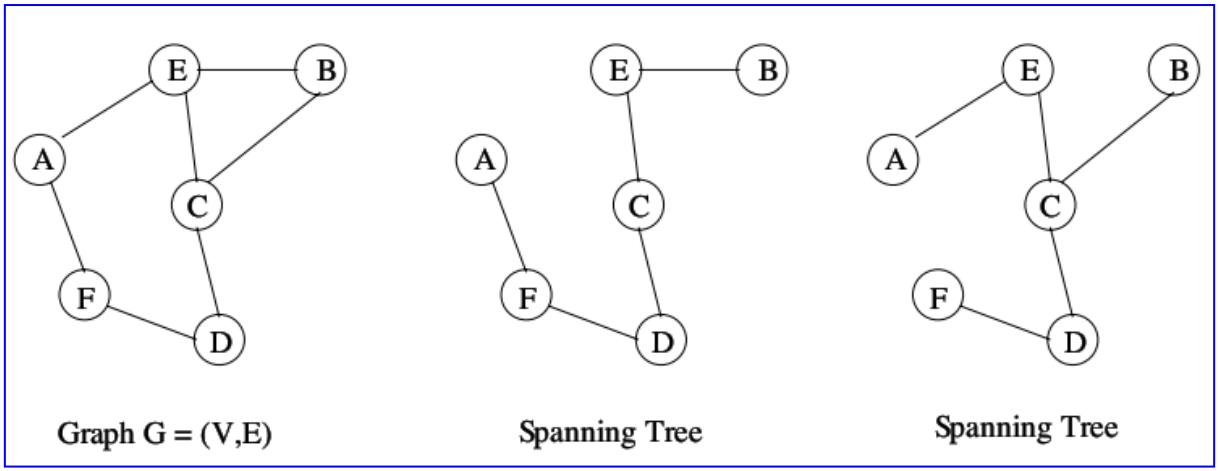


Basics

- Two vertexes p,q are two adjacent nodes if the edge e=(p,q) $\in$ E.
- The path between the two vertexes is a series of graph's vertexes where any two adjacent vertexes define one edge of the graph.
- Cycle is a path where the top and end nodes overlap.
- Graph called connected if there is 1 path between any two vertexes.

Basics

- T tree is a graph and it does not contain a cycle.
- G = (V,E) is a graph, T = (V', E') is a tree with V’ $\in$ V and E’ $\in$ E, hence it will be called the spanning tree of the graph.

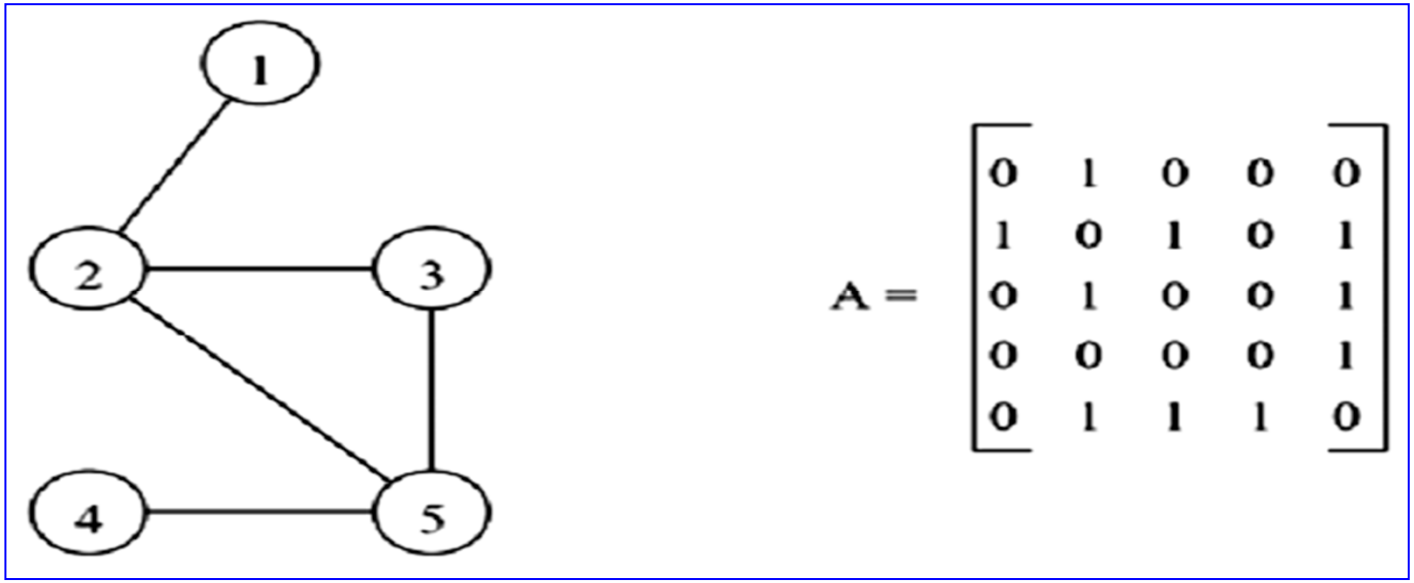


Graph demonstration

- How to represent a graph:
  - Adjacent matrix.
  - Adjacent list:
    - Adjacent-vertices list.
    - Adjacent list.
  - In parallel algorithms, instead of using adjacent lists we use arrays:
    - Adjacent-vertices array.
    - Adjacent array.

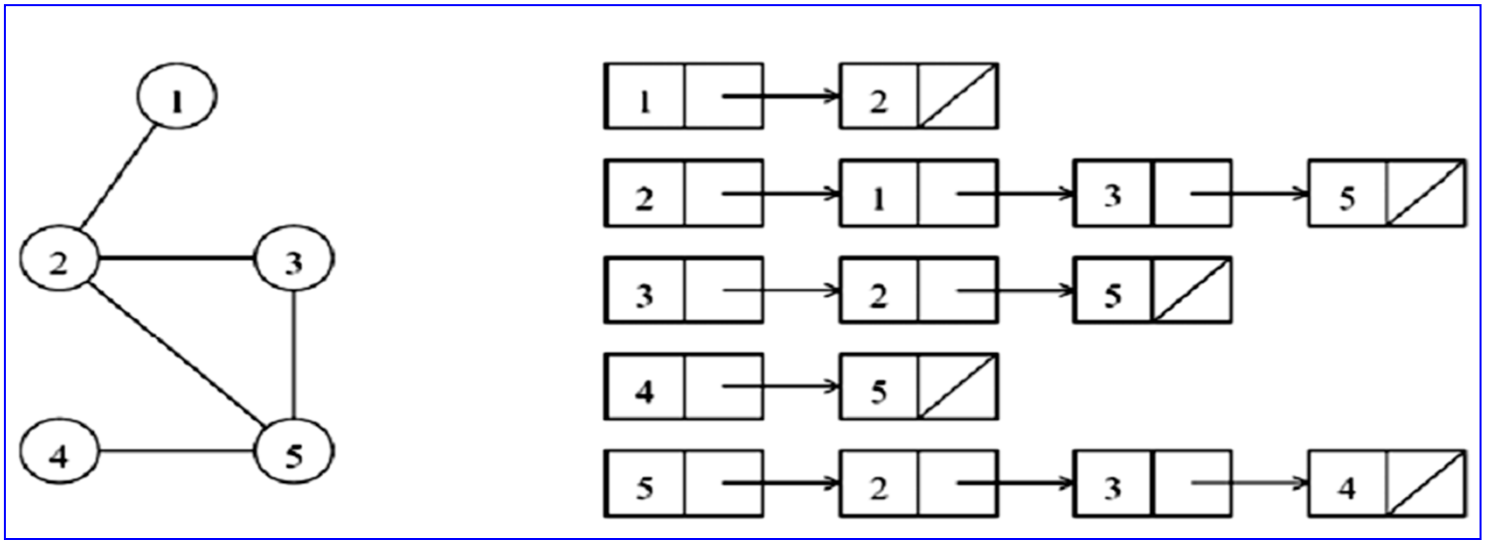
Adjacent matrix

- If | V| = n, the graph is expressed by matrix A[n x n] with rules:
  - A[i][j] = 1 if (i,j) is a graph's edge
  - A[i][j] = 0 in the remaining cases.

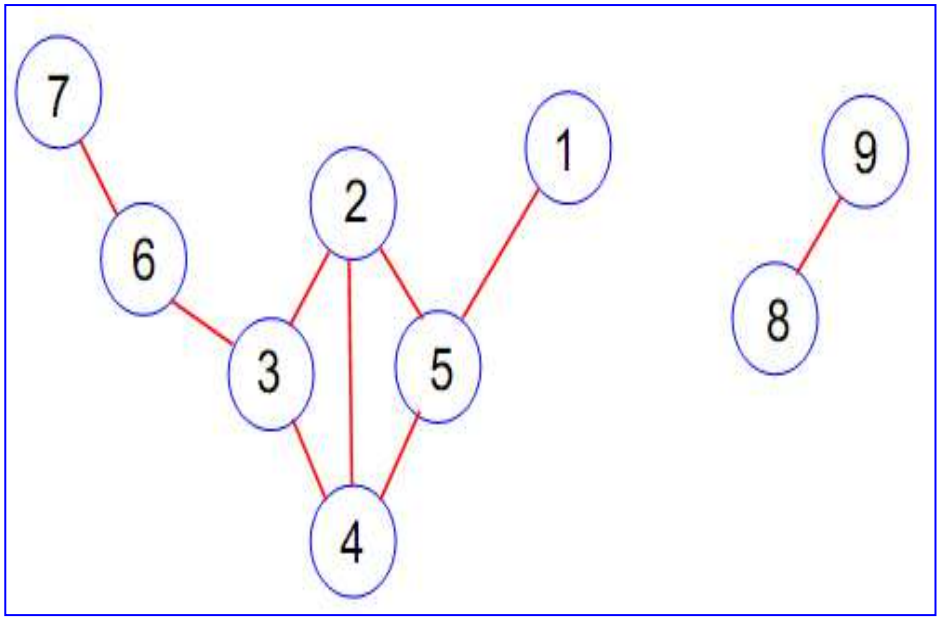


Adjacent list

- An array of adjacent lists, each corresponding to a set of its adjacent vertices.
- Set of vertices can be presented by an array for easier access.



Determine the rank of vertexes



	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0	0	0
2	0	0	1	1	1	0	0	0	0
3	0	1	0	1	0	1	0	0	0
4	0	1	1	0	1	0	0	0	0
5	1	1	0	1	0	0	0	0	0
6	0	0	1	0	0	0	1	0	0
7	0	0	0	0	0	1	0	0	0
8	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	1	0

- Approach:
  - Using adjacent matrix.
  - For each vertex we calculate the total on each row.

Algorithm with O(n) and $O(n^2)$ processors

```
input    : G = (V,E) xác định bởi A[1..n, 1..n].
output   : d[1..n] với d[v] là bậc của đỉnh v
begin
      for i = 1 to n do in parallel
            d[i] = PSum(A[i][1..n], n)
      end parallel
end.

PSum(X[1..n],n) tính tổng song song theo
mô hình cây cân bằng.
```

- PRAM $O(n^2)$ processor.
- $O(\log_2 n)$ time units

- PRAM O(n) processor.
- O(n) time units

```
input    : G = (V,E) xác định bởi A[1..n, 1..n].
output   : d[1..n] với d[v] là bậc của đỉnh v
begin
      for i = 1 to n do in parallel
            d[i] = 0;
            for j = 1 to n do
                  d[i]  = d[i] + A[i][j];
            end for
      end parallel
end.
```

Euler’s graph

- Euler's theothothical statement: If G is a graph, directionless and the number of vertex are even, then G is a Euler’s graph.
- Approach:
  - Calculate the ranks on the graph.
  - Check the parity of the vertex

Illustration of the algorithm

- Representing the parity of the vertices through the flag flag[1..n] : boolean.
- Depending on the number of processors we have the corresponding boolean value assembly.

```
input   : G = (V,E) liên thông
        : xác định bởi A[1..n, 1..n].
output  : xác định G là đồ thị Euler hay không
begin
    B1. Tính bậc các đỉnh: d[1..n]
    B2. for v = 1 to n do in parallel
          if d[v] lẻ then flag[v] = false;
          else flag[v] = true;
    end parallel
    B3. Result = AND{flag[v]}, v = 1..n
end.
```

Clique

- Set of C vertexes $\in$ V called a graph's clique if a child graph from C vertexes is a complete graph, i.e. between any two vertexes there is a connecting edge $\in$ E.
- Approach:
  - Calculate the degree of vertexes in C.
  - If degree of the vertex = k-1 with | C| = k then C will be a clique of G.

Algorithm with EW and CW

```
input   : G = (V,E) : A[1..n,1..n],
        : tập con C của V, |C| = k.
output : kiểm tra C có là clique không?
begin
      for all v in C do in parallel
            dc[v] = Sum{A[v][u]}, với u thuộc C.
      end parallel

      d = Sum{dc[v]}, với v thuộc C.

      if d = k(k-1) then clique = true
      else clique = false;
end.
```

- PRAM EREW.
- T(n) depends on the number of processors

- PRAM ERCW.
- T(n) = O(1) with k^2 processor

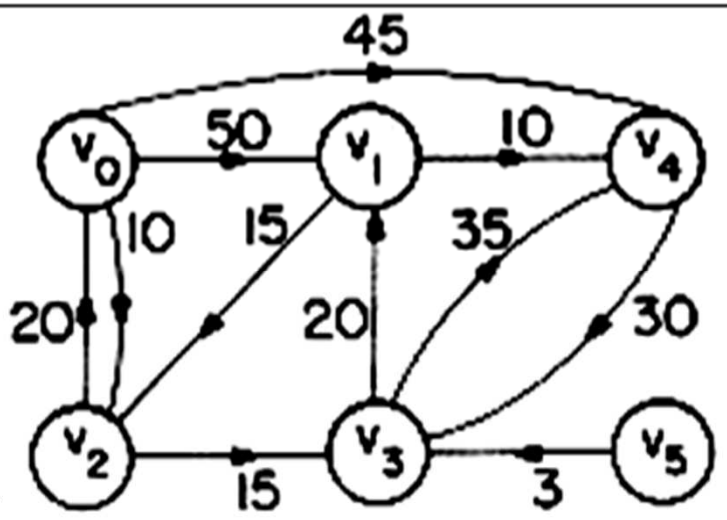
```
input   : G = (V,E) : A[1..n,1..n],
        : tập con C của V, |C| = k.
output : kiểm tra C có là clique không?
begin
      clique = true;
      for all u,v in C do in parallel
            if (u <> v) & A[u][v] = 0 then
                  clique = false;
            end if
      end parallel
end.
```

All Shortest Paths

- The shortest path's problem from a vertex to one or many vertexes does not have an optimal parallel algorithm.
- Building a parallel algorithm that finds the shortest path between all vertex pairs (i,j) using an algorithm-graph: Floyd-Warshall.

Floyd - Warshall algorithm

- Matrix of weighted graph :
  - If (i,j) is a weighted graph's edge, considered as w[i][j] then assigning A[i][j] = w[i][j];
  - If (i,j) is not an edge then: A[i][j] = ∞.
  - Assigning A[i][i] = 0 with ∀ i, j ∈ 1..n



Ma trận kề

0	50	10	∞	45	∞
∞	0	15	∞	10	∞
20	∞	0	15	∞	∞
∞	20	∞	0	35	∞
∞	∞	∞	30	0	∞
∞	∞	∞	3	∞	0

Floyd - Warshall algorithm

- Algorithm ideas:
  - Choose 1 vertex as intermediate, assuming vertex k.
  - Optimizing the path from vertex u to vertex v via k. That is, we compare A[u][v] with the total of A[u][k] and A[k][v]. If we go from u to v longer than going from u to k and then v, then we will optimize the value A[u][v].
  - We see that there are n methods to choose vertex k, with each k having n methods to choose the starting vertex u and with each u we have n methods to choose the ending vertex v. So we have three nested loops.

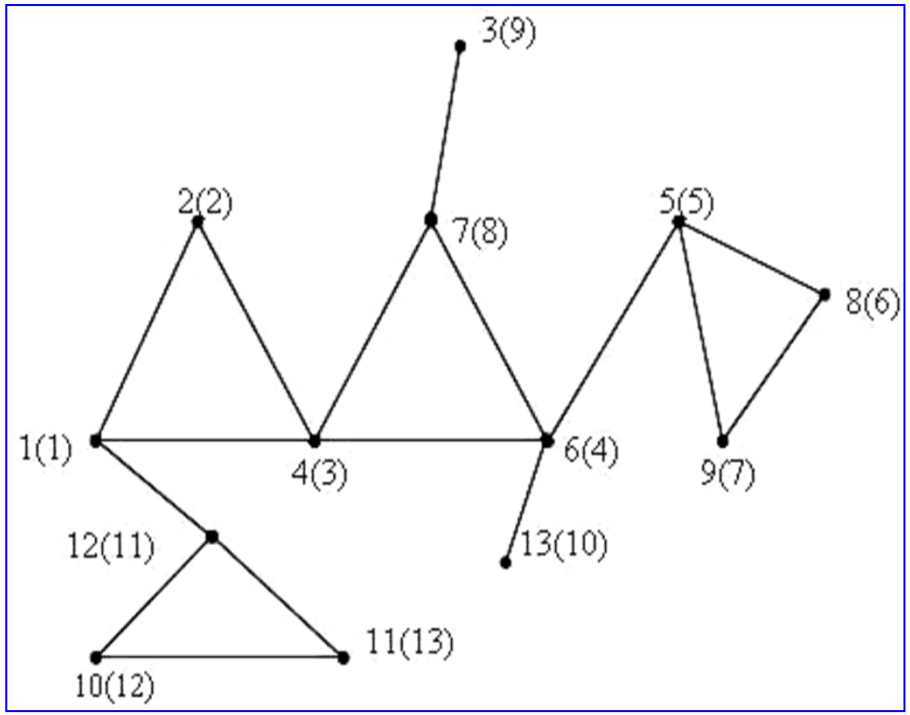
Floyd - Warshall algorithm

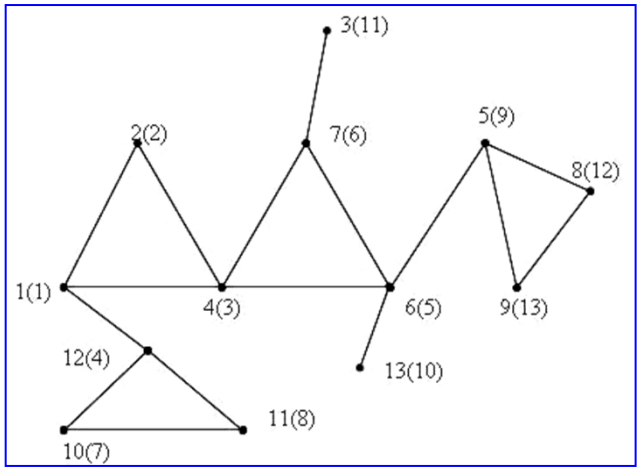
```
input   : G =(V,E,W) : w[1..n, 1..n] : ma trận trọng số.
output : d[1..n,1..n] với d[i][j] là đường đi ngắn nhất (i,j)
         : ma trận p[1..n,1..n] ghi nhận đường đi từ i đến j.
begin
      for 1 ≤ i, j ≤ n do in parallel
            d[i][j]  = w[i][j];
            p[i][j]  = i;
      end parallel

      for k = 1 to n do
            for i = 1 to n do in parallel
                  for j = 1 to n do in parallel
                        if d[i][j] > d[i][k] + d[k][j] then
                              d[i][j] = d[i][k] + d[k][j];
                              p[i][j] = p[k][j];
                        end if
                  end for
            end for
      end for
end
```

12.2 Parallel algorithms for graph traversing

Algorithms for graph's traversing





- There are two common ways to traverse a graph:
  - Depth First Search.
  - Breadth First Search.

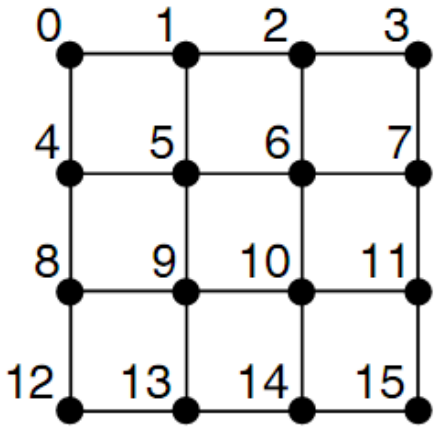
Algorithms for graph's traversing

- Parallelization:
  - The essence of the DFS algorithm is recursive method so there is no corresponding parallel algorithm.
  - The BFS algorithm will traverse adjacent vertexes, so it can be parallelized.

Parallelization of BFS

- Approach:
  - Starting from any vertex: called s.
  - Build BFS trees with root being s:
    - The leaves of the BFS tree at k_th iteration's step are determined in parallel by the adjacent vertexes of leaves at (k-1)_th iteration's step.
    - Iteration process until no more leaves are considered.
  - Problem: handle repeated case (a new leaf node is the child of two or more previous nodes)

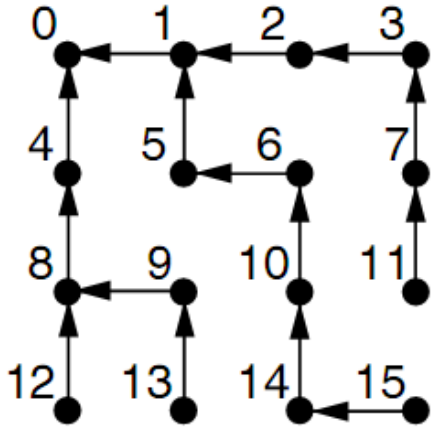
Parallelization of BFS



(a)

Step	Frontier
0	[0]
1	[1,4]
2	[2,5,8]
3	[3,6,9,12]
5	[7,10,13]
6	[11,14]
7	[15]

(b)



(c)

Parallelization of BFS

- Data structure's usage:
  - Graph shown by an adjacent array (each vertex has an array of vertices adjacent to it).
  - BFS trees are characterized by T[1.n]. T[u] = v meaning that v is the father of u in the tree.
  - Edge vertex's layer F[1..n] : boolean in each iteration step. If v is a edge vertex → F[v] = 1, else F[v] = 0.
  - Handling repeated vertex using the CW-ACR mechanism.

```
input    : G = (V,E), s là 1 đỉnh bất kỳ.
output   : BFS tree với root = s
begin
    for all v in V do in parallel
        F[v] = 0;  T[v] = -1;
        if  v = s then
            F[s] = 1; T[s] = s;
        end if
    end parallel

    while (OR {F[v]}, v in V)
    begin
        for all v in V do in parallel
            Fnew[v] = 0;
        end parallel
        for all e = (u,v) in E do in parallel
            // nếu 1 trong 2 đỉnh là biên, đỉnh kia chưa xét
            if F[v] = 1 && T[u] = -1 then
                T[u]  = v;   // CW - ACR
                Fnew[u] = 1;   // cập nhật lại biên.
            end if
        end parallel
        for all v in V do in parallel
            F[v]  = Fnew[v];
        end parallel
    end
end.
```

Connected components

- In the sequential mode, the connected components are determined by the algorithm that traverses the graph.
- In parallel, if BFS is used to determine the number of connected components, it is not effective.
- Some other approaches:
  - Algorithm: HSC – Hirschberg, Chandra, Sarawate.
  - Algorithm: SV – Shiloach, Vishkin

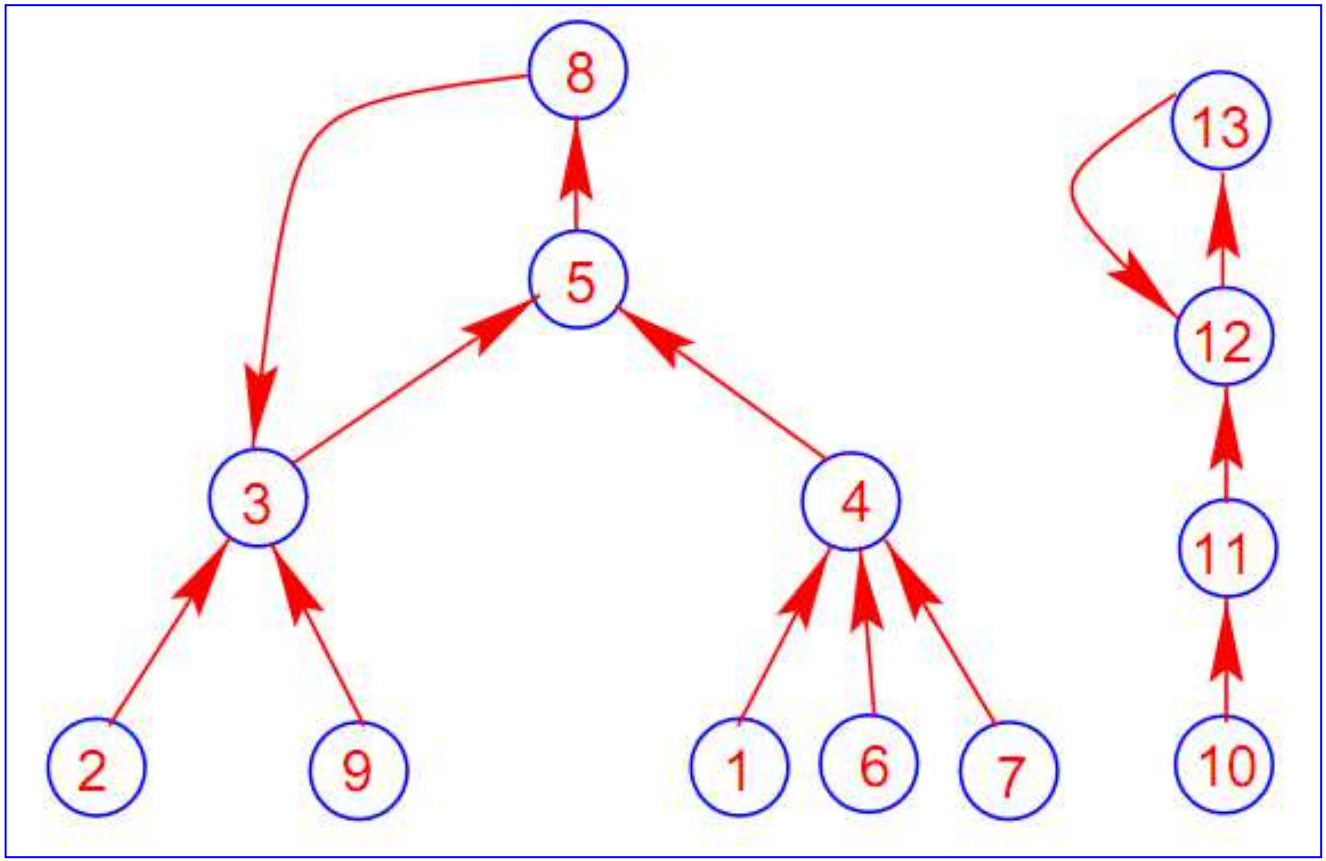
Parallel solved

- HSC – Hirschberg, Chandra, Sarawate:
  - Use of data structure: adjacent matrix.
  - Efficiency: $O(\log^2 n)$ time units at a cost $O(n^2)$.
- SV – Shiloach, Vishkin:
  - Use of data structure: adjacent list.
  - Efficiency: $O(\log_2 n)$ time units at a cost $O(m+n)$.

Some concepts

- Rooted tree:
  - All nodes point to the parent, the out-degree of each node (except the root) are equal to 1.
- Star tree:
  - It's the rooted tree, but there's only one root and the leaves.
  - Root points to itself.
- Pseudo-forest:
  - This is a directed graph where each vertex has an outdegree equal to 1.

Pseudo-Forest



HCS algorithm

- Algorithm's idea:
  - Each connected component is presented as a tree in the forest (pseudoforest).
  - Each connected component is assigned a label, because the role of vertexes is the same, so in order to break the symmetry we choose the vertex with the smallest index which is characteristic for the set of vertexes belonging to the intersex component.

HCS algorithm

- Algorithm's idea:
  - Need to build an algorithm that converts G to pseudoforest.
  - The graph G = (V,E) is presented by an adjacent matrix A[1.n][1..n]
  - Building a function C on V as follows:
    - C(v) = min {u | A[u][v] = 1 }, i.e.:
      - C(v) = v if v is the individual vertex.
      - C(v) is the smallest vertex's index with v.

HCS algorithm

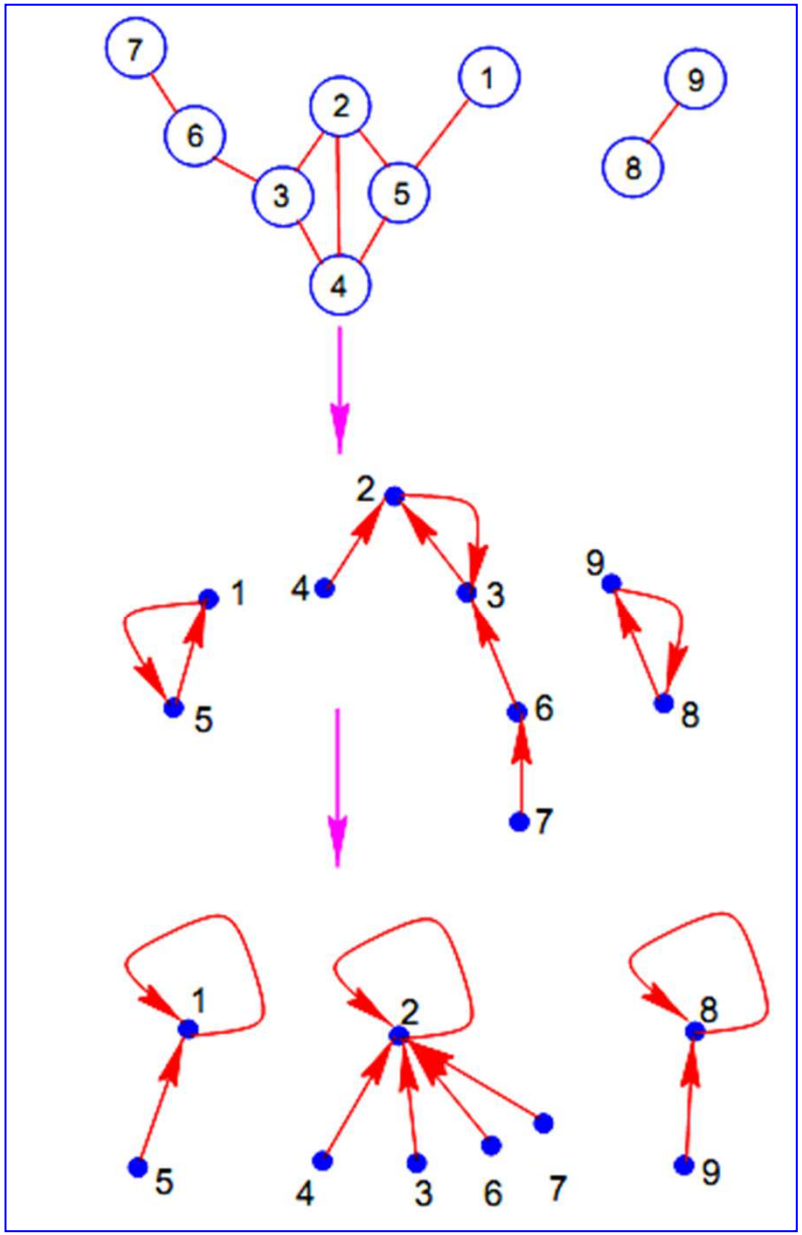
- Complement: function C creates a pseudo-forest F and divides V into sub-set $V_1, V_2, ..V_s$ so that each V_i is a sub-set of the vertexes of a root tree T_i .
- Features:
  - The vertexes in V_i belong to connected component of G.
  - Each cycle in F is either a vertex point to itself or having exactly 2 edges.
  - Each cycle in T_i contains the smallest vertex of T_i

HCS algorithm

- Algorithm's idea:
  - Determining the C(v) value is relatively simple when using the adjacent matrix. C(v) is the smallest index on row v which has the value equal to 1.
  - Compressing the sub-set V_i into a star-shaped tree, we call its root as a super-vertex. The super-vertex has the smallest value in V_i.
  - Setting the matrix for the super-vertexes.

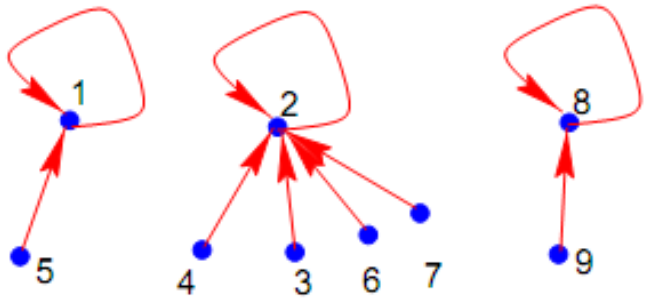
HCS algorithm

- Algorithm's idea:
  - The compression of a sub-set V_i to super vertex is done as follows:
    - Breaking the cycle in a tree T_i .
    - Using pointer jumping technique to build the star tree.
  - The re-establishment of the super-vertex matrix is as follows:
    - If u,v belong to the super-vertex r_u and r_v and A[u][v] = 1
    - Then $A[r_u][r_v] = 1$.


7
6
2
1
3
5
4
9
8
2
4
1
3
9
5
6
8
7
1
2
8
5
4
3
6
7
9

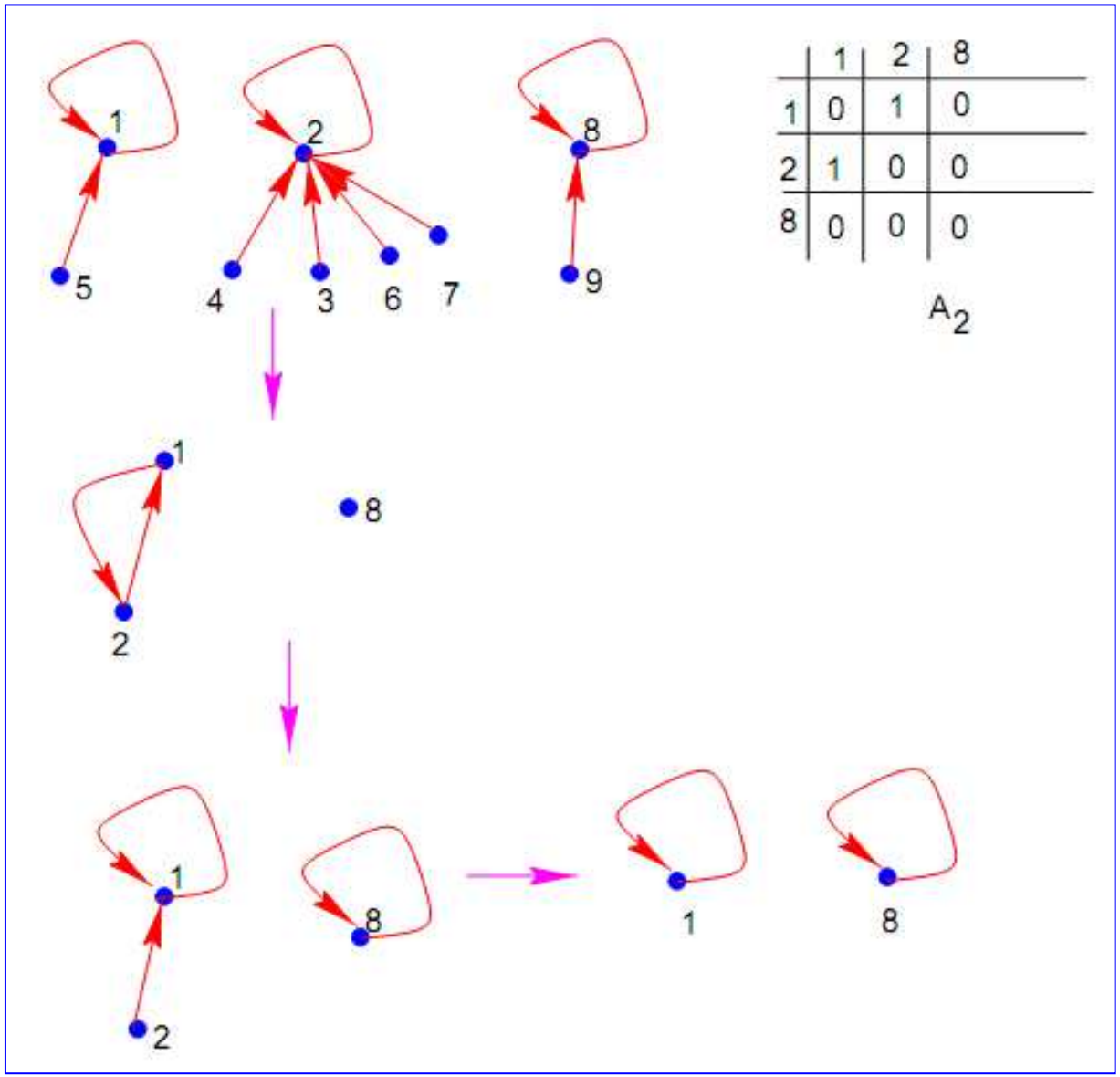
	1	2	3	4	5	6	7	8	9
1	0	0	0	0	1	0	0	0	0
2	0	0	1	1	1	0	0	0	0
3	0	1	0	1	0	1	0	0	0
4	0	1	1	0	1	0	0	0	0
5	1	1	0	1	0	0	0	0	0
6	0	0	1	0	0	0	1	0	0
7	0	0	0	0	0	1	0	0	0
8	0	0	0	0	0	0	0	0	1
9	0	0	0	0	0	0	0	1	0

A_1



	1	2	8
1	0	1	0
2	1	0	0
8	0	0	0

A_2



	1	2	8
1	0	1	0
2	1	0	0
8	0	0	0

A_2

```
input   : G = (V,E) : A[1..n][1..n]
output  : các thành phần liên thông của G
begin
    for i = 1 to n do in parallel
        P[i] = i;  active[i] = 1;
    end parallel
    for k = 1 to log2n do
        for i = 1 to n do in parallel
            if active[i] = 1 then P[i] = C[i];
        end parallel
        for i = 1 to n do in parallel
            if P[P[i]] = i && P[i] > i then P[i] = i;
        end parallel
        for t = 1 to log2n do
            for i = 1 to n do in parallel
                P[i]  = P[P[i]];
            end parallel
        end for
        for 1 ≤ i,j ≤ n do in parallel
            if A[i][j] = 1 && active[i] && active[j] then A[P[i]][P[j]] = 1;
        end parallel
        for i = 1 to n do in parallel
            A[i][i] = 0;
            if P[i] <> i then active[i] = 0;
        end parallel
end
```

P[u] =v nếu v là cha u

xác định supervertex

phá vỡ chu trình

Chuyển từ rooted tree về star tree

cập nhật ma trận supervertex

SV algorithm

- HCS algorithm's analysis:
  - In each iteration step that compresses the tree into a star-shaped tree, this work requires $O(\log_2 n)$ steps.
  - Calculating C(v) value should also be done with $O(\log_2 n)$ steps.
- SV algorithm will solve these 2 works with O(1) time units.

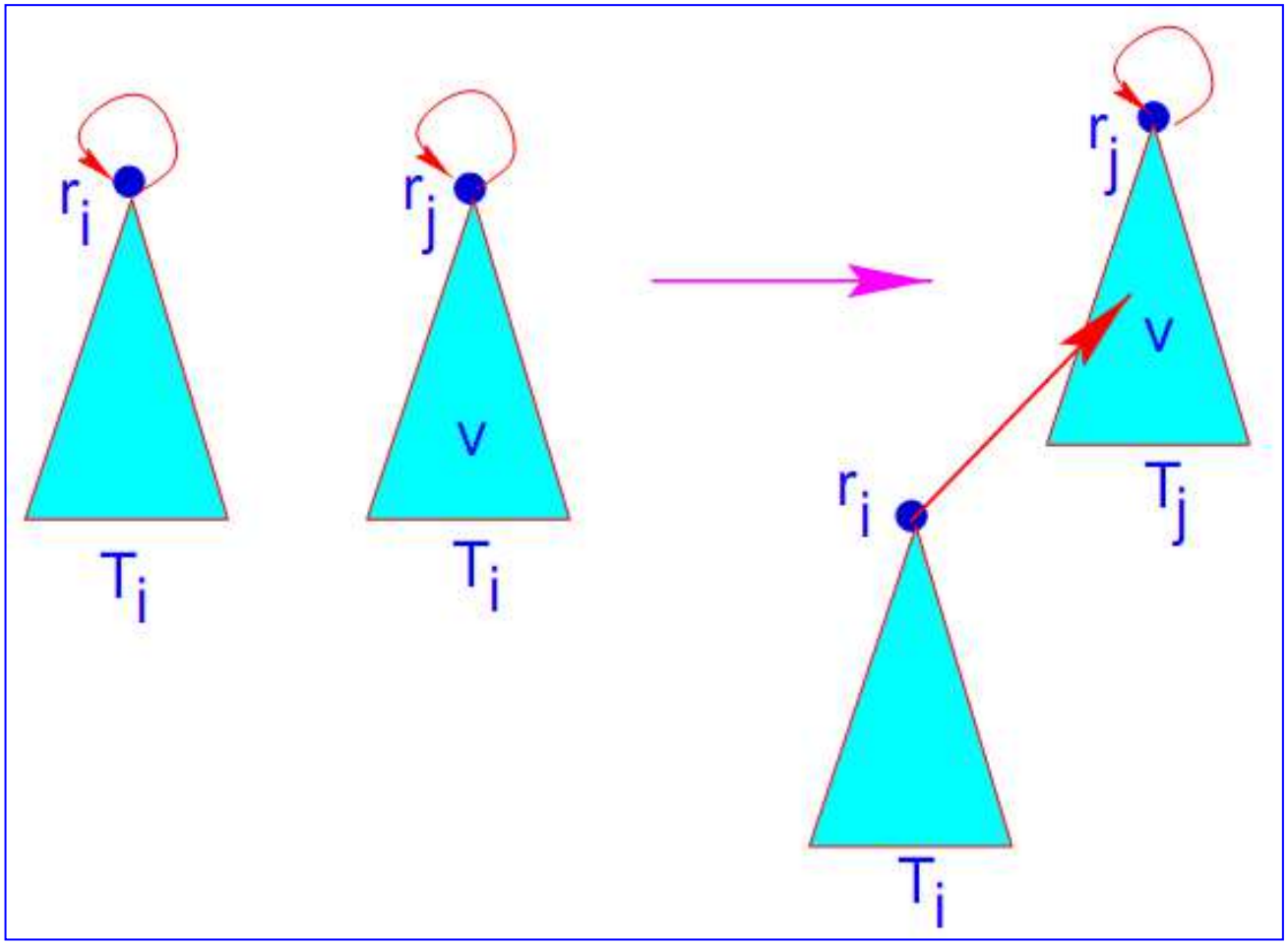
SV algorithm

- Algorithm's idea:
  - Replace the rooted tree in the HCS algorithm with a star-shaped tree in each iteration step.
  - Jump the pointer step by step, do not do it thoroughly as in HCS.
  - Build a satisfying D-function on V:
    - D(u) = v means that u points to v.
    - Initial: D(v) = v with $\forall$ v
    - Function D creates a fake forest (pseudo-forest).

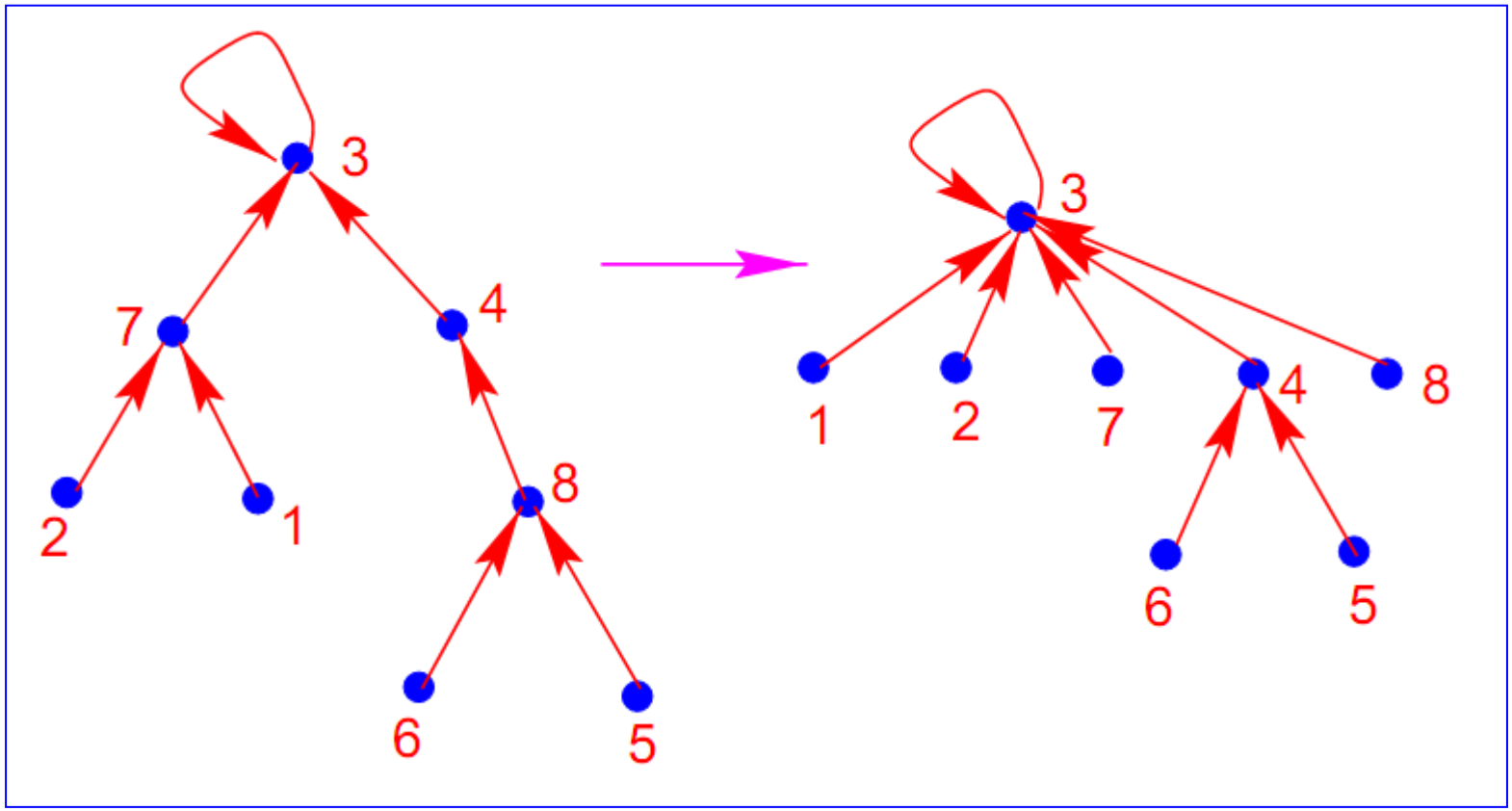
SV algorithm

- Two operations performed on pseudo-forest:
  - Grafting operation: Considering 2 trees T_i and T_j in pseudo-forest defined by D. Call r_i is the root of T_i and v is a vertex of T_j, the grafting operation of T_i on T_j is defined: $D(r_i) = v$.
  - Pointer jumping: considering v is a node on the T tree, the jumping operation for v is defined as follows: D(v) = D(D(v)).

Grafting (hooking)

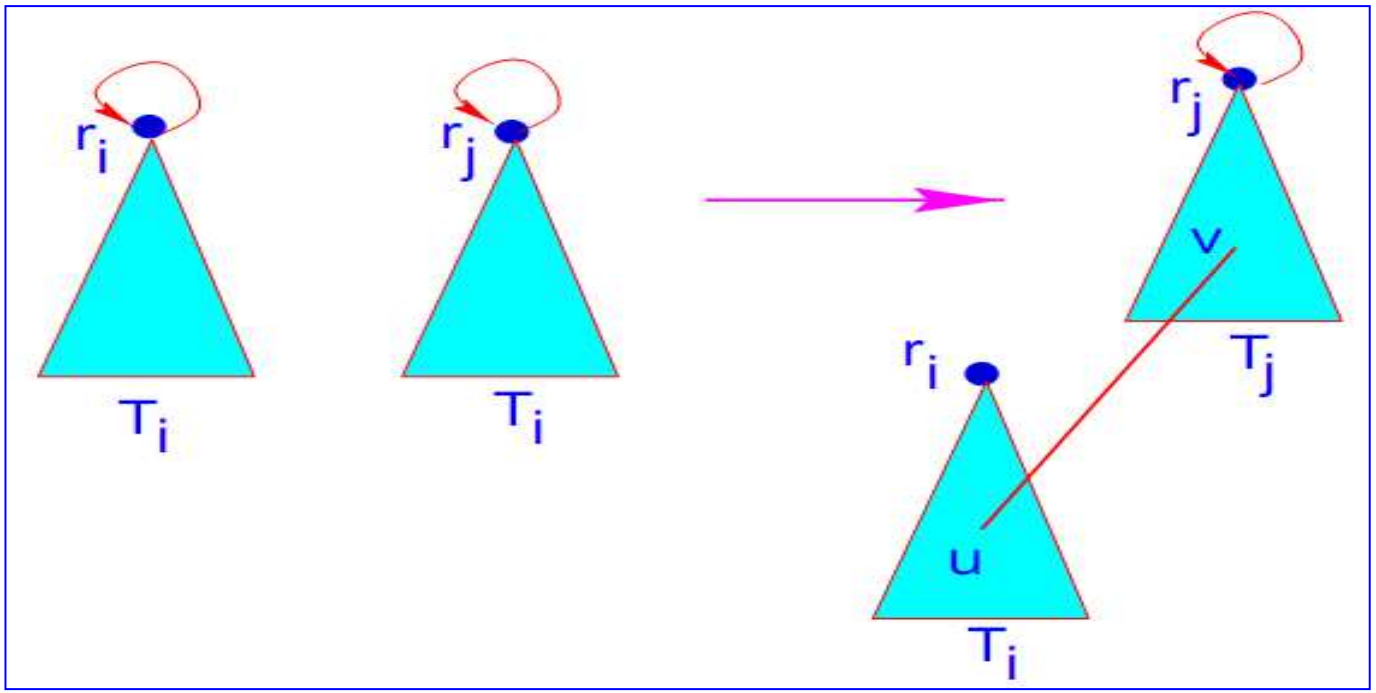


Pointer jumping



SV algorithm

- Problems with Grafting:
  - If $\exists$ u $\in$ T_i & v $\in$ T_j and (u,v) $\in$ E then all the nodes of T_i and T_j belong to the same connected component.

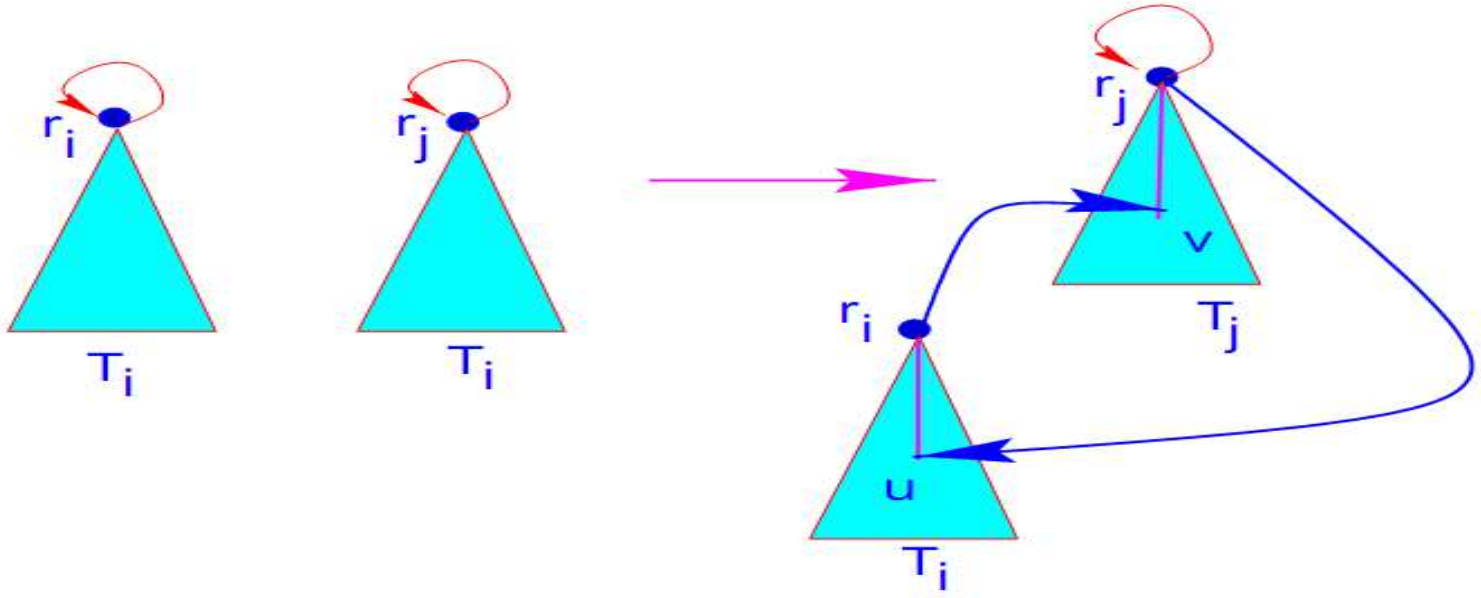


SV algorithm

- Problems with Grafting:
  - Grafting operation: grafting only a root into another tree's trunk, so it is necessary to determine which root to graft.
  - Since each tree is represented by the root, it is not known where the u, v's root is (unless we perform a operation for root determination).
  - If 1 of the 2 nodes u,v is the root or the root's child, the grafting operation is also easy.
    - Root's child or root node: D(v) = D(D(v)).

SV algorithm

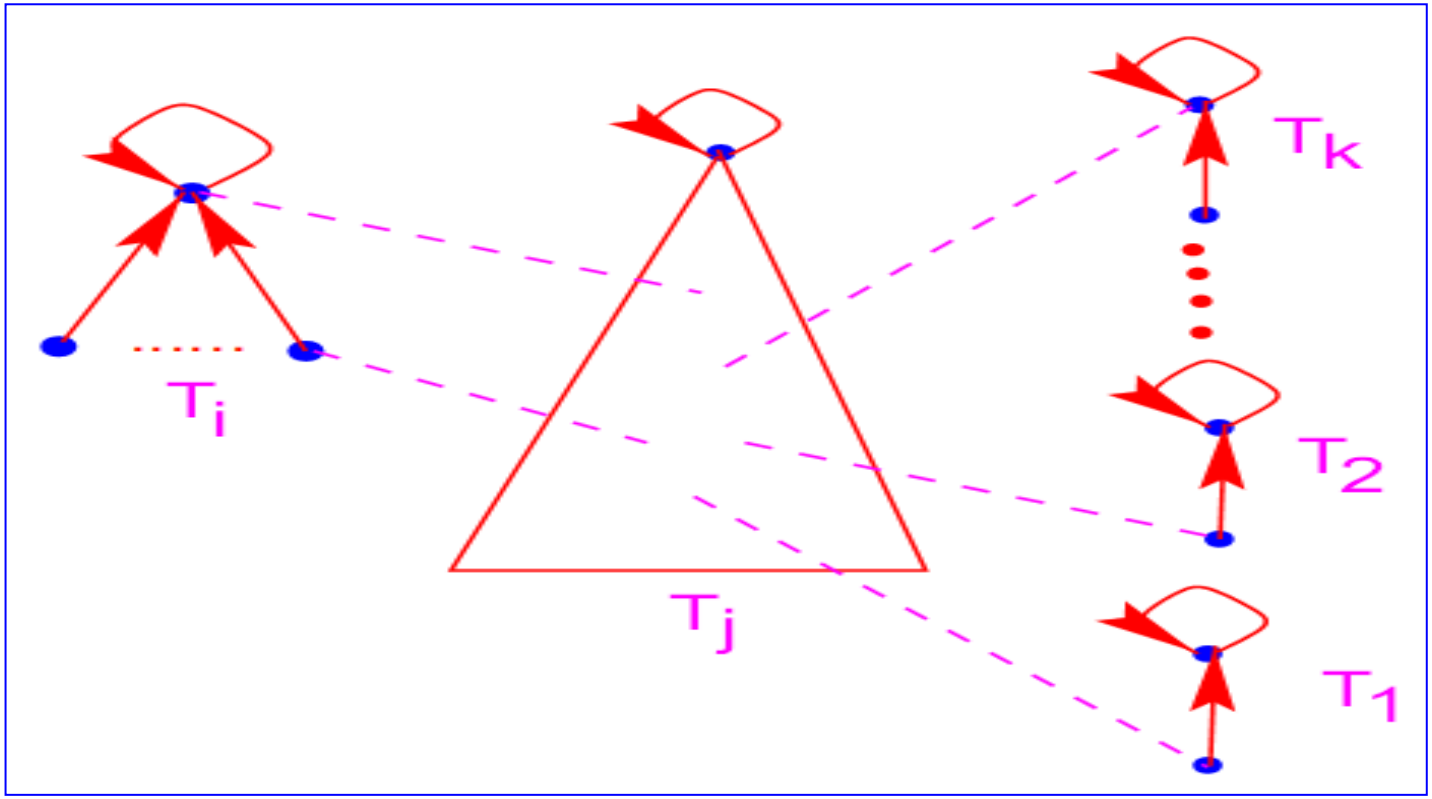
- Problems with Grafting:
  - If both nodes u,v are the same root or the same root's child, then execution in parallel could leading to the cycle (in this algorithm we do not want to have a cycle)



SV algorithm

- Problems with Grafting:
  - To avoid the cycle we use symmetry breaking technique:
  - With each e = (u,v) $\in$ E ,u $\in$ T_i & v $\in$ T_j and if D(u) = D(D(u)) we'll assign it to T_j if satisfied D(v) < D(u). Then: D(D(u)) = D(v).
  - We call this operation: conditioning operator.
  - However, with this only operation to transplant all the trees can take O(n) steps → not yet optimal $O(\log_2 n)$.

- Problems with Grafting: considering the example:
  - Tree T_i containing nodes which are smaller than that in tree T_j
  - Tree T_j has smaller adjacent vertexes which are located on the trees T_1, T_2, .. The trees T_1 , T_2 , ... have no connecting edge.

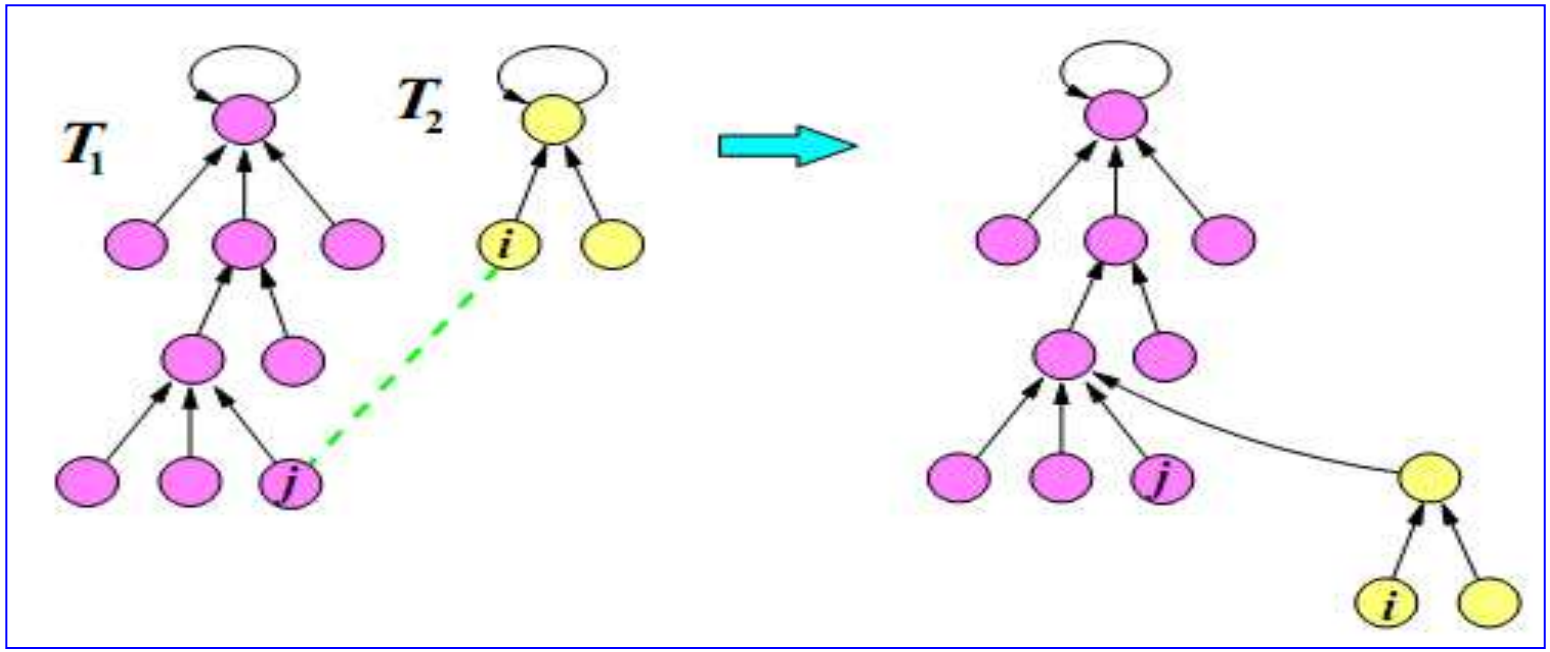


SV algorithm

- In this example, the following situation may occur:
  - First of all T_j connected to T_1, creating a new tree T_j.
  - Then T_j is reconnected to T_2 ,.. Repeated until T_s
  - Finally it is grafted into the tree T_i
- Thus the execution time can be O(n).
- New grafting operation needs to be added: Unconditioning grafting.

SV algorithm

- Algorithm's idea:
  - In each iteration step, after performing Conditioning grafting, we perform unconditioning grafting.
  - Unconditioning grafting: Grafting star-shaped trees into other trees. (don't check the condition D(v) < D(u))

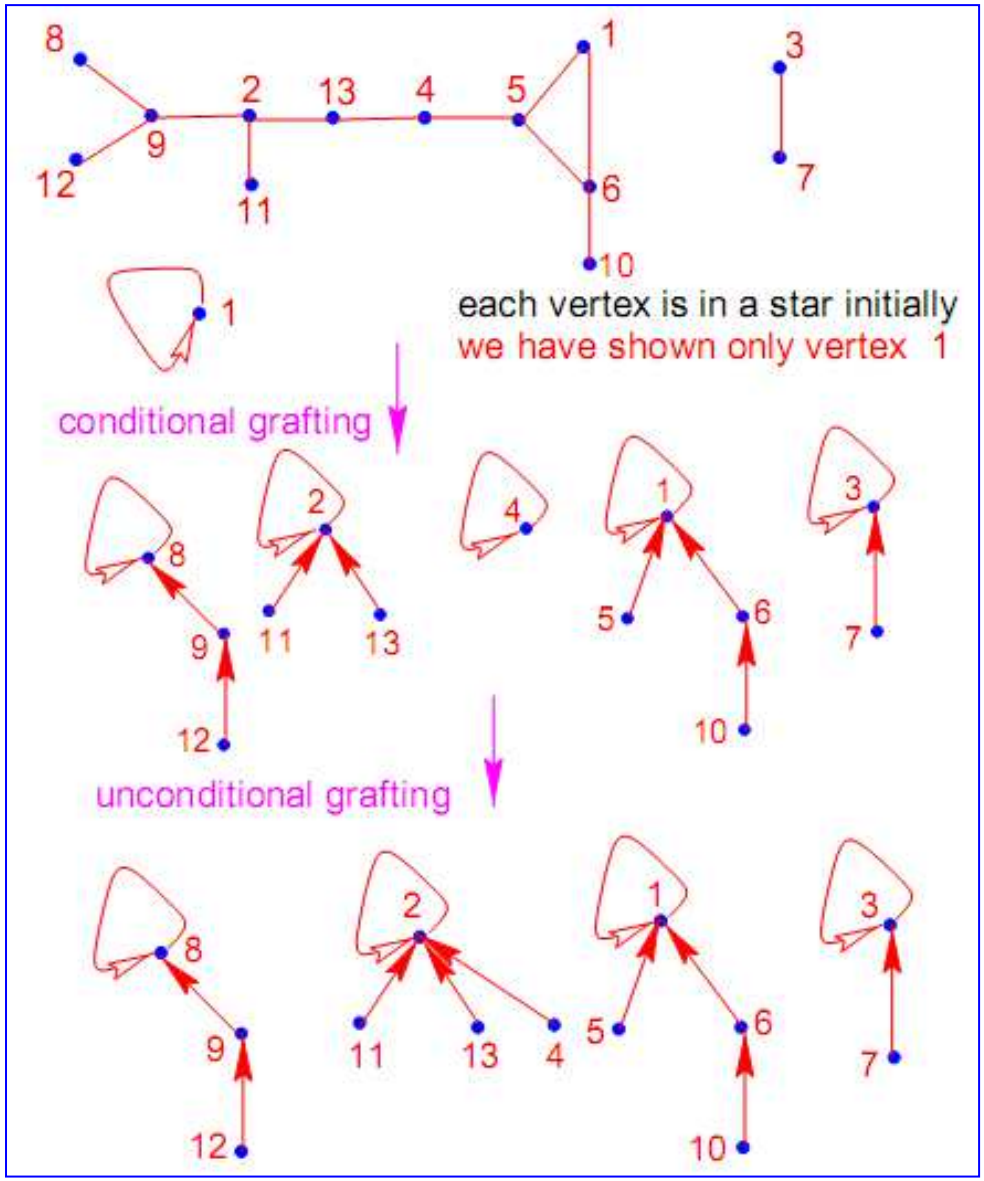


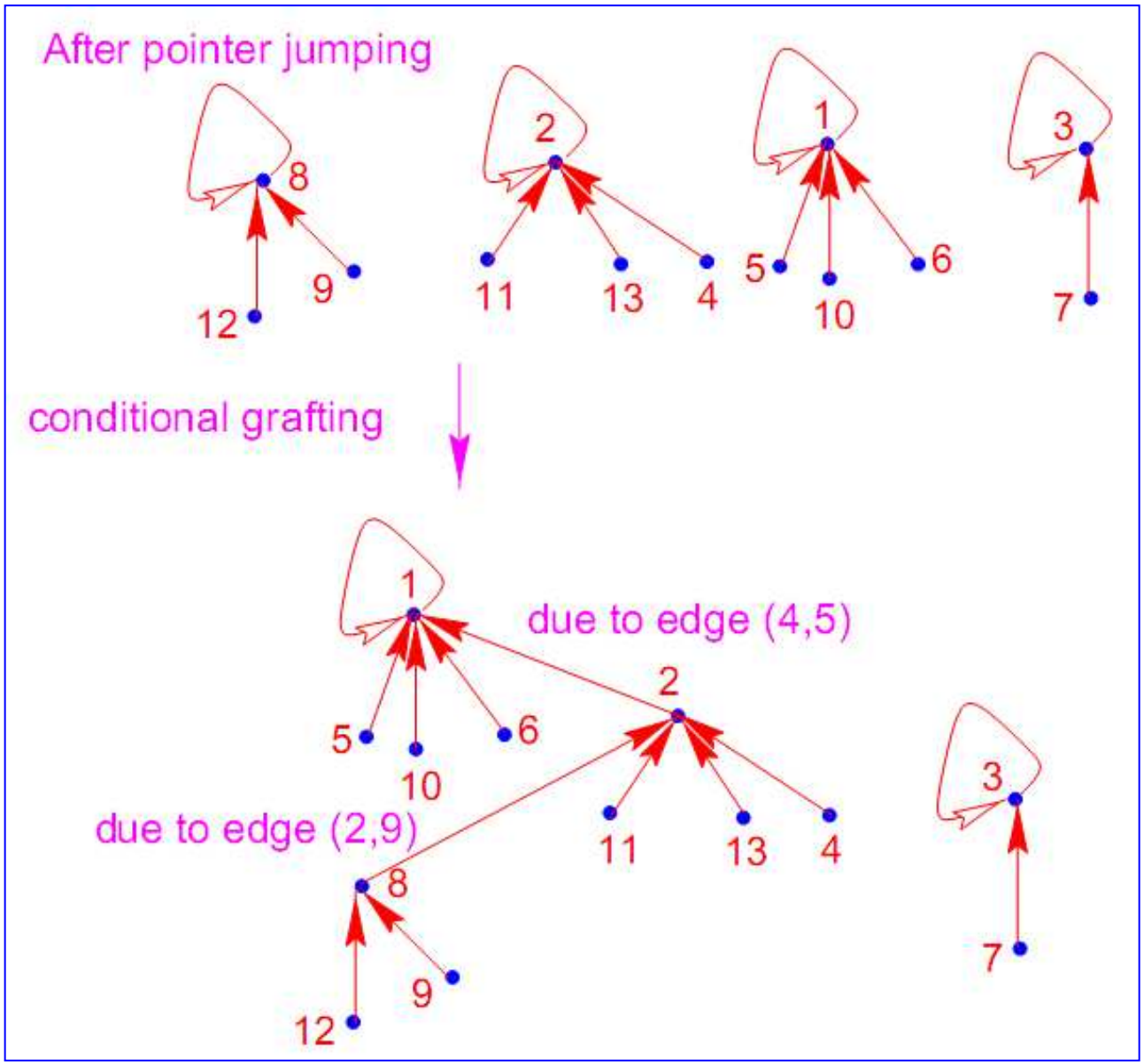
SV algorithm

- Unconditioning grafting: this ensures that other star-shaped trees, that have not been grafted at conditioning grafting, will be grafted.

```
Unconditioning_Grafting
begin
      for all e = (i,j) do in parallel
            if star(i) && D(i) <> D(j) then D(D(i)) = D(j)
      end parallel
end.
```

```
function star(v) : boolean
begin
      star(v)  = true;
      if (D(v) <> D(D(v)) then
            star(v) = star(D(v)) = star(D(D(v))) = false;
      end if.
      star(v) = star(D(v));
end.
```

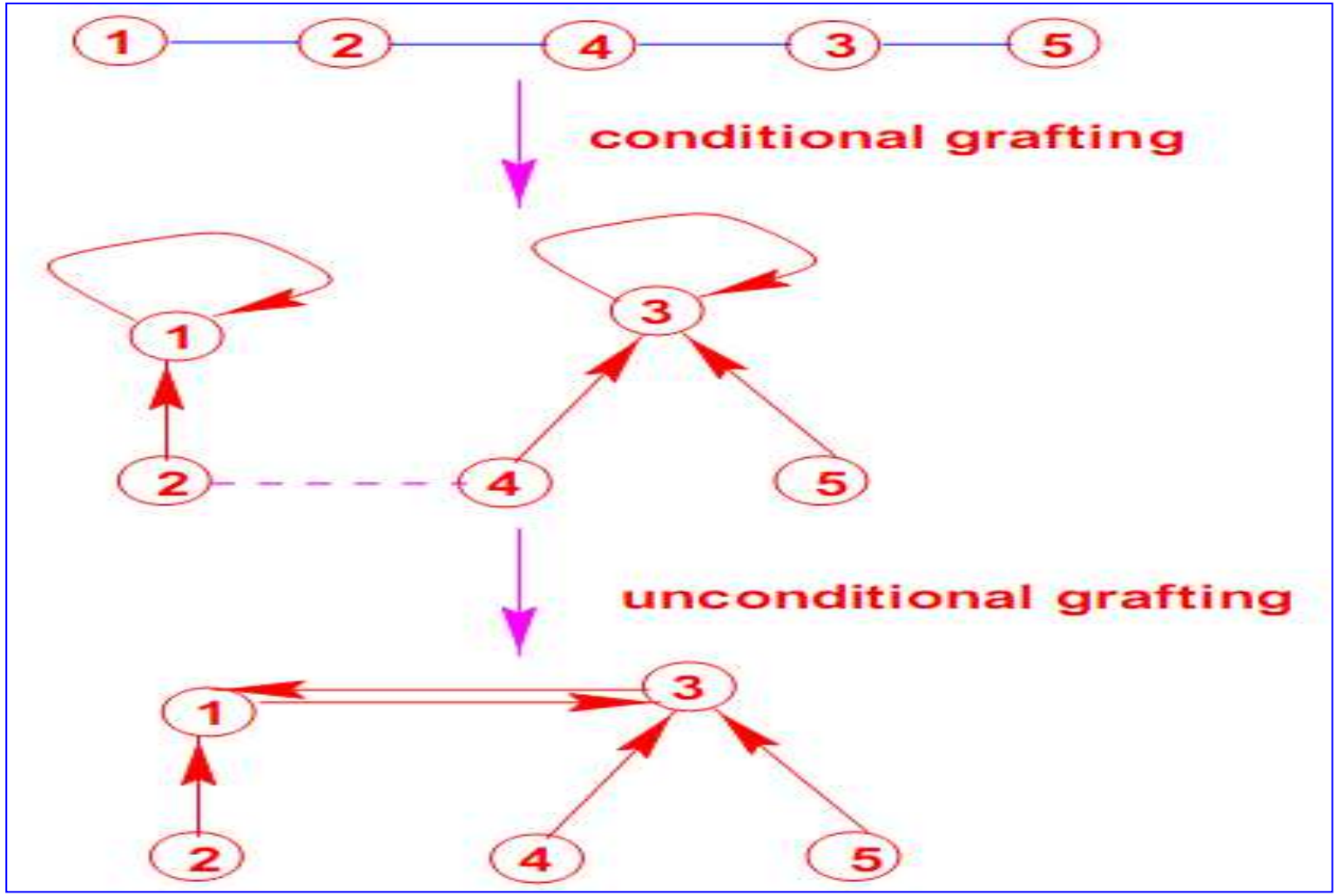

8
2
13
4
5
1
3
9
12
11
6
7
10
each vertex is in a star initially
we have shown only vertex 1
1
conditional grafting
2
4
1
3
8
9
11
13
5
6
7
12
10
unconditional grafting
2
1
3
8
9
11
13
4
5
6
7
12
10


After pointer jumping
8
9
12
2
11
13
4
1
5
6
10
3
7
conditional grafting
1
due to edge (4,5)
5
6
10
2
11
13
4
due to edge (2,9)
8
9
12
3
7

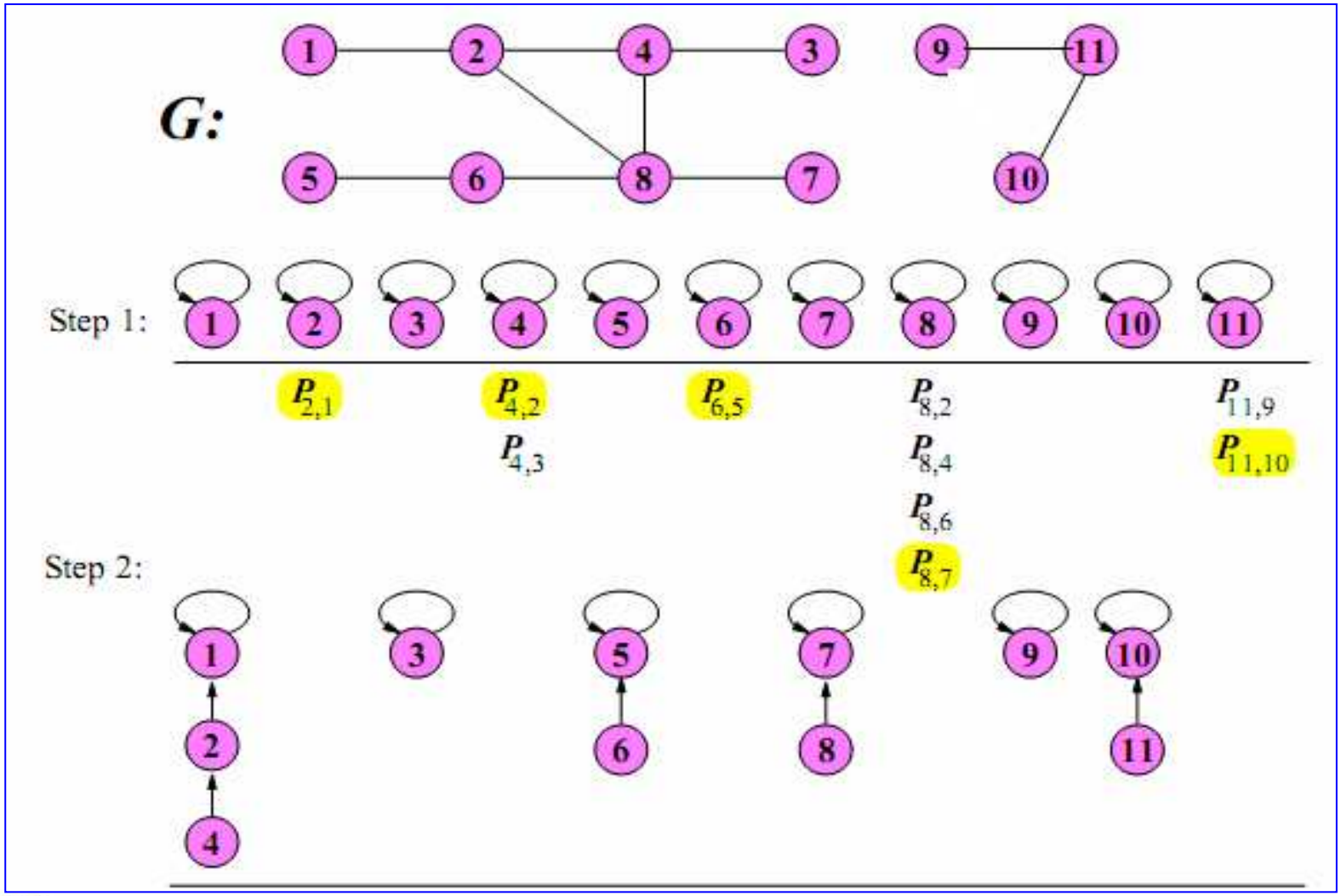
SV algorithm

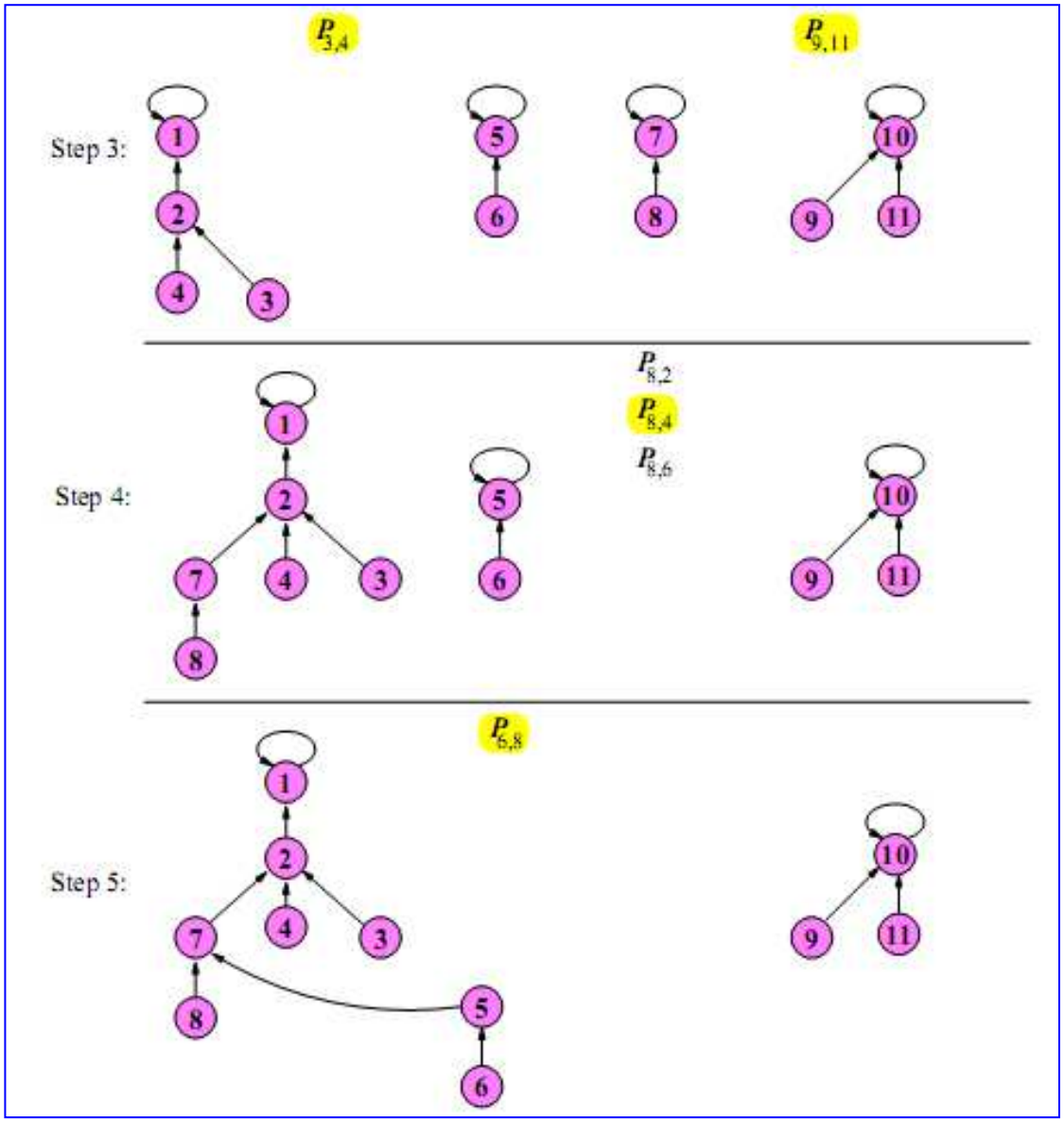
- Problems with Unconditioning Grafting:
  - After the first iteration, all the trees are star-shaped, so it is easy to occur the cycle when performing Unconditioning grafting.
  - In the following steps, not all trees are star-shaped, so the above phenomenon does not occur.
- Resolutions:
  - Performing Unconditioning grafting from the second iteration step.
  - In the first step, performing Unconditioning grafting with single nodes (only 1 node).

Problem with Unconditioning grafting



Illustration




$P_{3,4}$
$P_{9,11}$
Step 3:
1
2
4
3
5
6
7
8
9
10
11
$P_{8,2}$
$P_{8,4}$
$P_{8,6}$
Step 4:
1
2
7
4
3
8
5
6
9
10
11
$P_{6,8}$
Step 5:
1
2
7
4
3
8
5
6
9
10
11

13.3 Write a parallel graph-based program

Write parallel graph-based program

- Choose a graph-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

ĐẠI HỌC BÁCH KHOA

25 YEARS ANNIVERSARY SOICT

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