Euler Tour Technique

We have already seen this in the pointer jumping lecture

Application to problem 2: preorder tree traversal

Let us consider **the problem of numbering the vertices of a rooted tree in preorder (depth first search order).**

```
PREORDER.TRAVERSAL(nodeptr):
Begin
   if nodeptr≠null then
      nodecount ← nodecount + 1
      nodeptr.label ← nodecount
      PREORDER.TRAVERSAL(nodeptr.left)
      PREORDER.TRAVERSAL(nodeptr.right)
   endif
End
```

Where is the parallelism? The fundamental operation assigns a label to a node.

We cannot assign labels to the vertices in the right subtree of the left subtree, until we know how many vertices are on the left subtree of the left subtree, and so on.

The algorithm seems inherently sequential!

Can we parallelize this?

PARALLELIZATION OF THE TRAVERSAL

Instead of focusing on the vertices, let us look into the edges.

When we perform a preorder traversal, we systematically work our way through the edges of the tree.

- We pass along every vertex twice: one heading down from the parent to the child, and one going from the child to the parent.

Euler tour of the tree

- ***If we divide each tree edge into two edges, one corresponding to the downward traversal, and one corresponding to the upward traversal, then the problem of traversing a tree turns into the problem of traversing a single linked list.***

TARJAN AND VISHKIN (1984)

4 steps:

Euler tour of the tree

1. The algorithm constructs a singly linked list. Each vertex of the linked list corresponds to a downward or upward edge traversal.

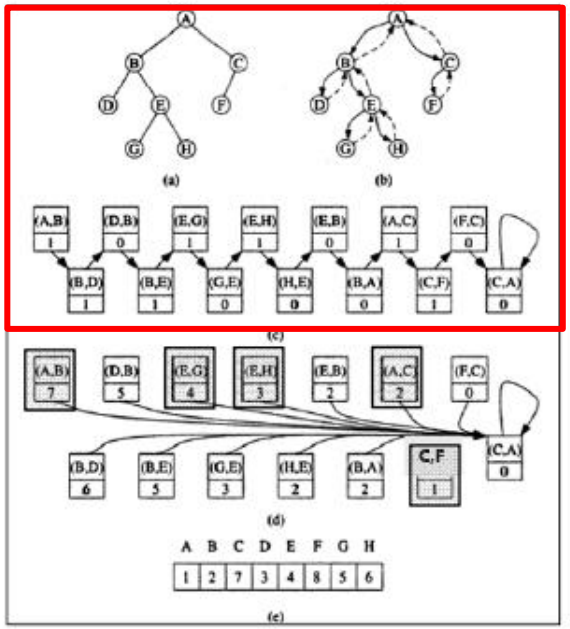
2. Algorithm assigns weights to the vertices of the newly created single linked list.
   - For vertices corresponding to downward edges, the weight is 1 (it contributes to node count).
   - For vertices corresponding to upward edges, the weight is 0 (it does not contribute to node count).

3. For each element of the singly-linked list, the rank of each element is determined (by pointer jumping).

4. The processors associated with the downward edges use the ranks they have computed to assign a preorder traversal number to their associated tree nodes (the tree node at the end of the downward edge).

EXAMPLE

Euler tour of the tree

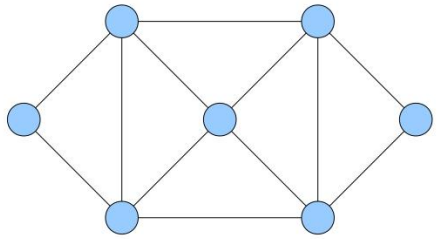


a) Tree
b) Double Tree Edges, distinguishing downward edges from upward edges.
c) Build linked list out of directed tree edges. Associate 1 with downward edges, and 0 with upward edges.
d) Use pointer jumping to compute total weight from each vertex to end of list.

The elements of the linked list which correspond to downward edges, have been shaded.
Processors managing these elements assign preorder values.
For example, (E,G) has a weight 4, meaning tree node G is 4th node from end of preorder traversal list.
The tree has 8 nodes, so it can compute that tree node G has label 5 in preorder traversal (=8-4+1)

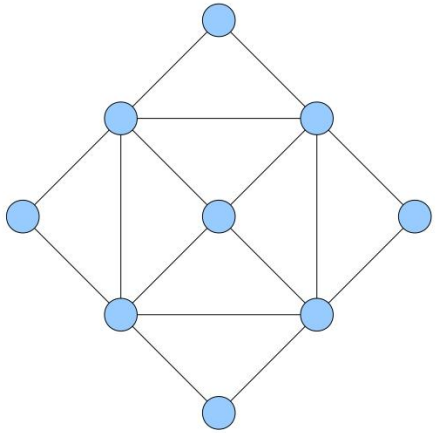
Euler Tours

- In a graph G, an ***Euler tour*** is a path through the graph that visits every edge exactly once.
- Mathematically formulates the “trace this figure without picking up your pencil or redrawing any lines” puzzles.



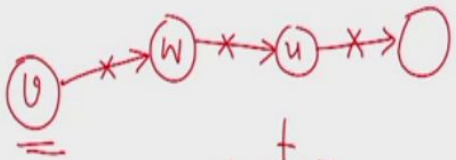
Euler Tours

- In a graph G, an ***Euler tour*** is a path through the graph that visits every edge exactly once.
- Mathematically formulates the “trace this figure without picking up your pencil or redrawing any lines” puzzles.



If all vertices are of even degrees, it's an Euler graph

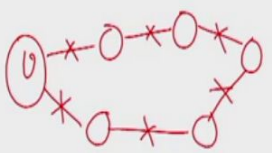
G has an Euler circuit

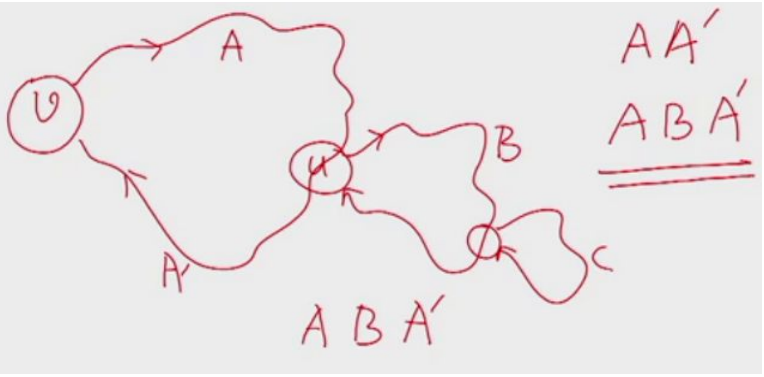


at every vertex, the incident edges can be paired off

G=(V,E) has an even degree at every vertex

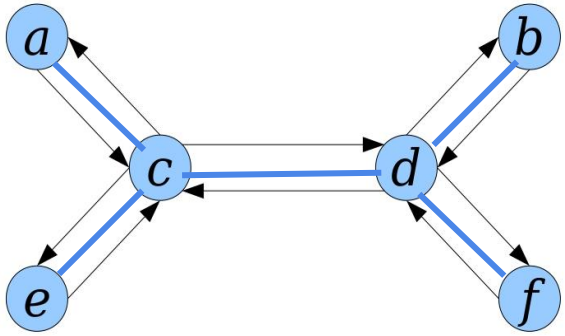
arbitrary vertex v, start walking in the graph.





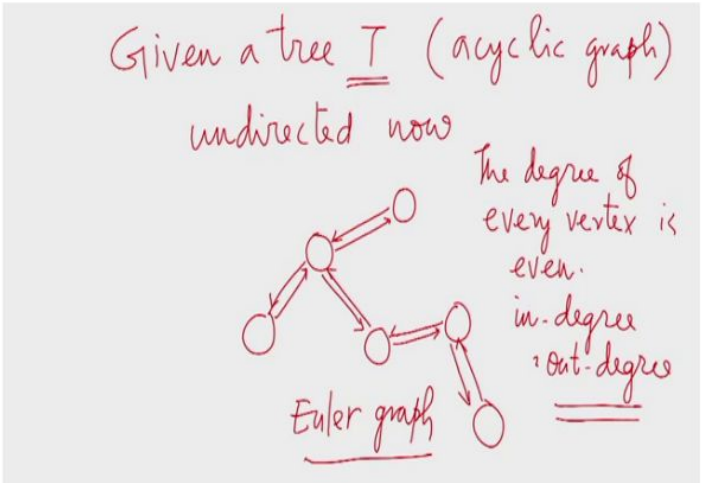
Euler Tours on Trees

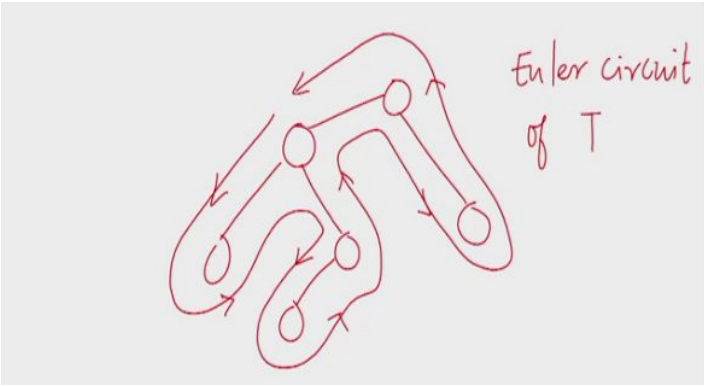
- Trees do not have Euler tours.



a c d b d f d c e c a

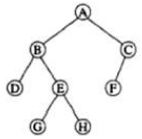
- ***Technique:*** replace each edge $\{u, v\}$ with two edges (u, v) and (v, u).
- Resulting graph has an Euler tour.





Special data structure for tree, finding successors

DATA STRUCTURE FOR THE TREE



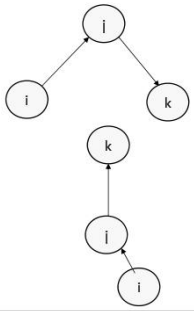
	A	B	C	D	E	F	G	H
parent	null	A	A	B	B	C	E	E
sibling	null	C	null	E	null	null	H	null
child	B	D	F	null	G	null	null	null

For every tree node, the data structure stores the node's parent, the node's immediate sibling to the right, and the node's leftmost child.

Representing the node this way keeps the amount of data stored a constant for each tree node and simplifies the tree traversal.

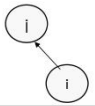
HANDLING UPWARD EDGES

Edge (i,j), such that parent(i)=j



If sibling[i]≠NULL
succ[(i,j)]←(j,sibling[i])

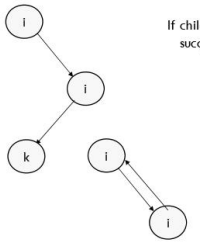
Else If parent[i]≠NULL
succ[(i,j)]←(j,parent[i])



Else
succ[(i,j)]←(i,j)
The edge is at the end of the tree traversal, so we put a loop at the end of the element list.

HANDLING DOWNWARD EDGES

Edge (i,j), such that parent[i]≠j.



If child[j]≠NULL
succ[(i,j)]←(j,child[j])

else
succ[(i,j)]←(j,i)

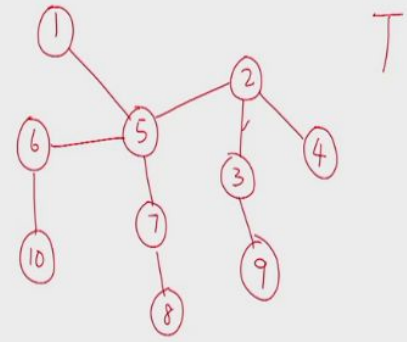
ie. j is a leaf and the successor is the edge back from the child to the parent.

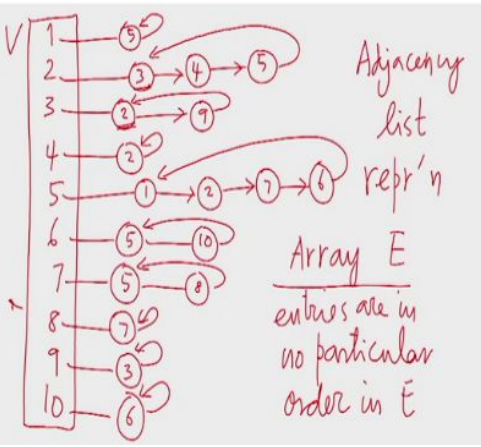
Finding Euler Circuit of tree from adjacency lists

T is given in adjacency list representation

Array of V adjacency lists

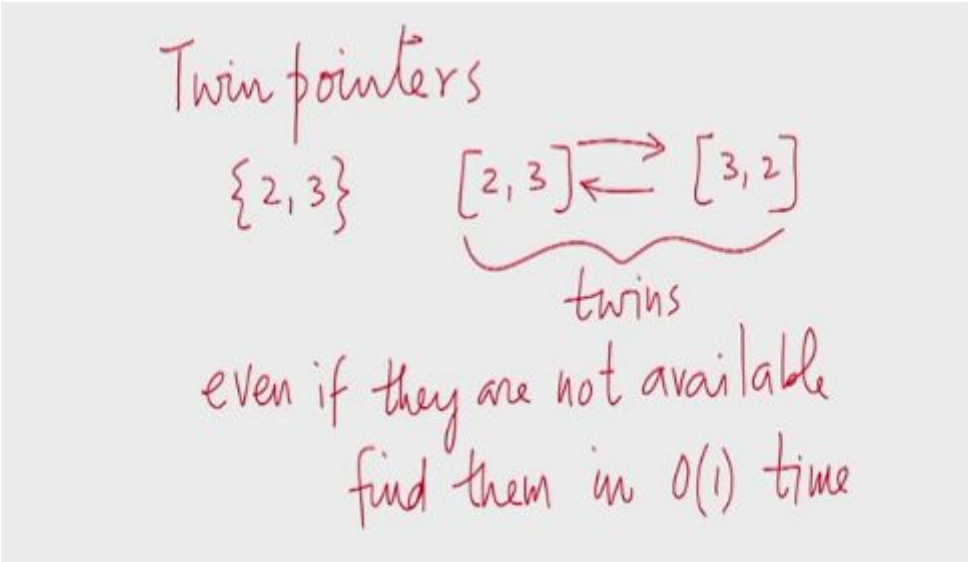
$\forall v \in V$, we have a circular linked list of the edges outgoing from v

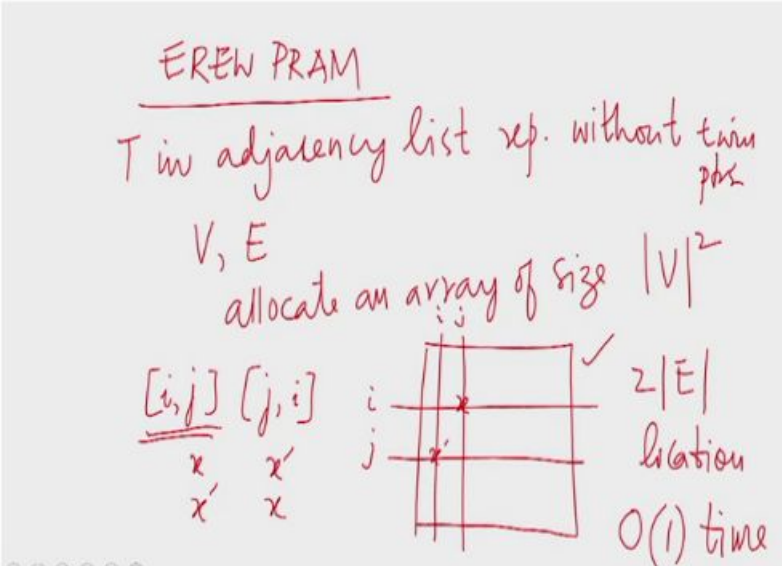




Adding twin pointers to the adjacency lists in parallel

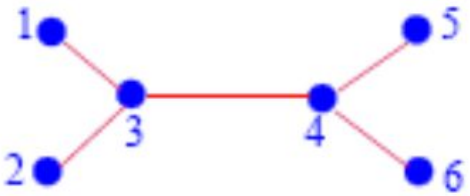
In an adjacency list representation of an undirected simple graph G = (V, E), each edge (u, v) has two adjacency list entries: [v] in the adjacency list of u, and [u] in the adjacency list of v. These are called twins of each other. A twin pointer is a pointer from an adjacency list entry to its twin.



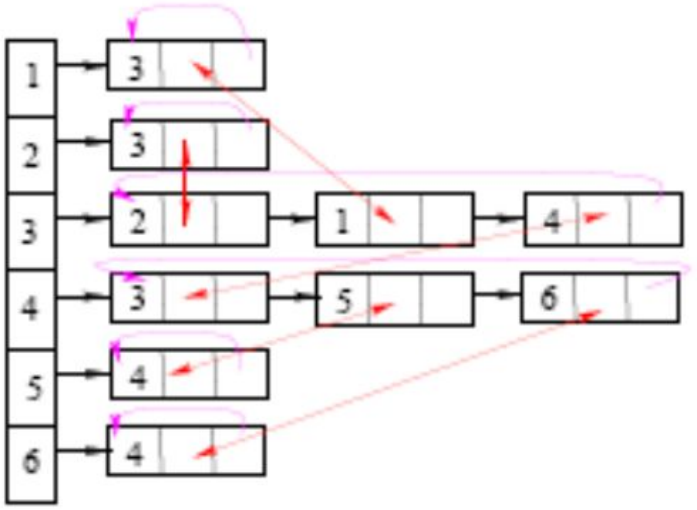


Adding twin pointers with |E| processors takes O(1) time.

Sample circular adjacency lists with twin pointers



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



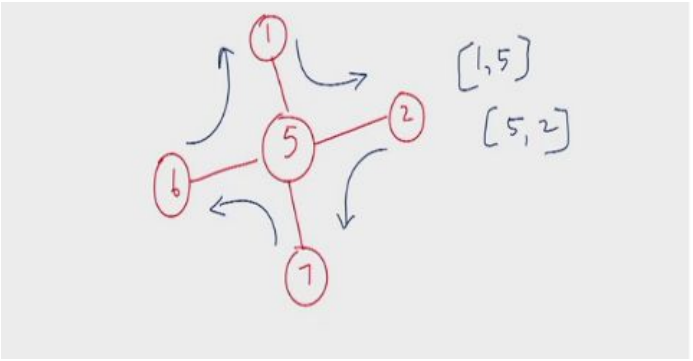
Compute Euler circuit in parallel, using circular adjacency lists with twin pointers

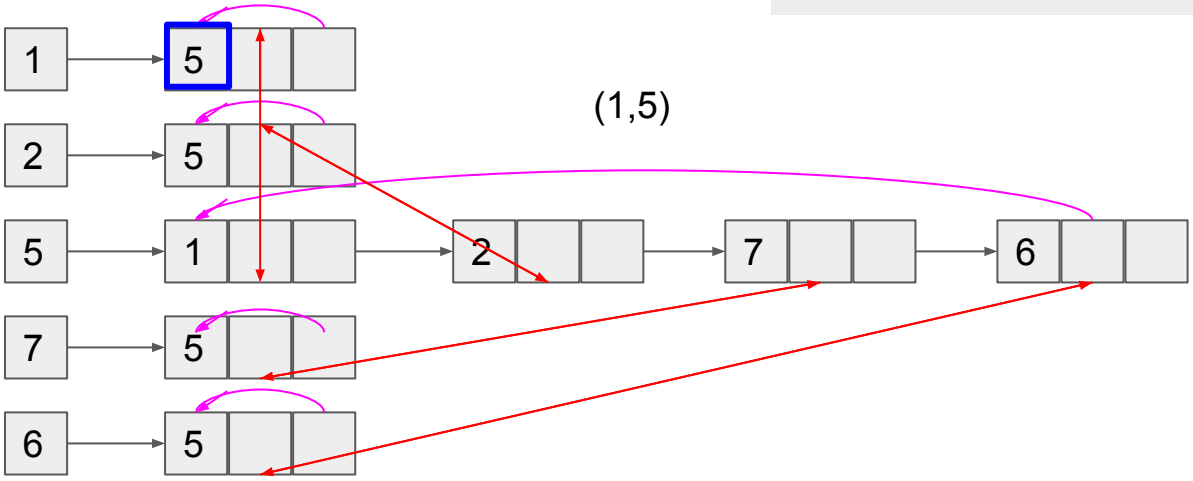
Euler circuit of T

for each adjacency list entry [i,j]

Euler_next[i,j]

= Adj_next[twin[i,j]]





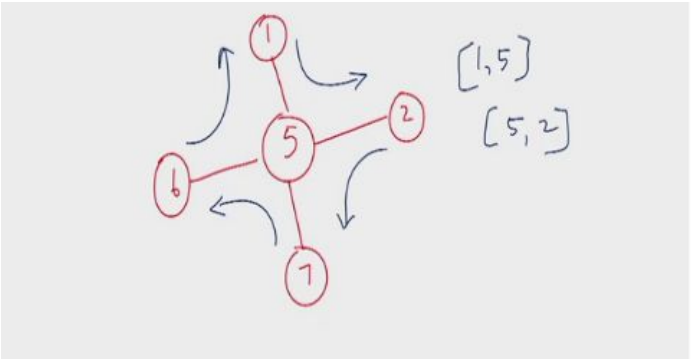
Compute Euler circuit in parallel, using circular adjacency lists with twin pointers

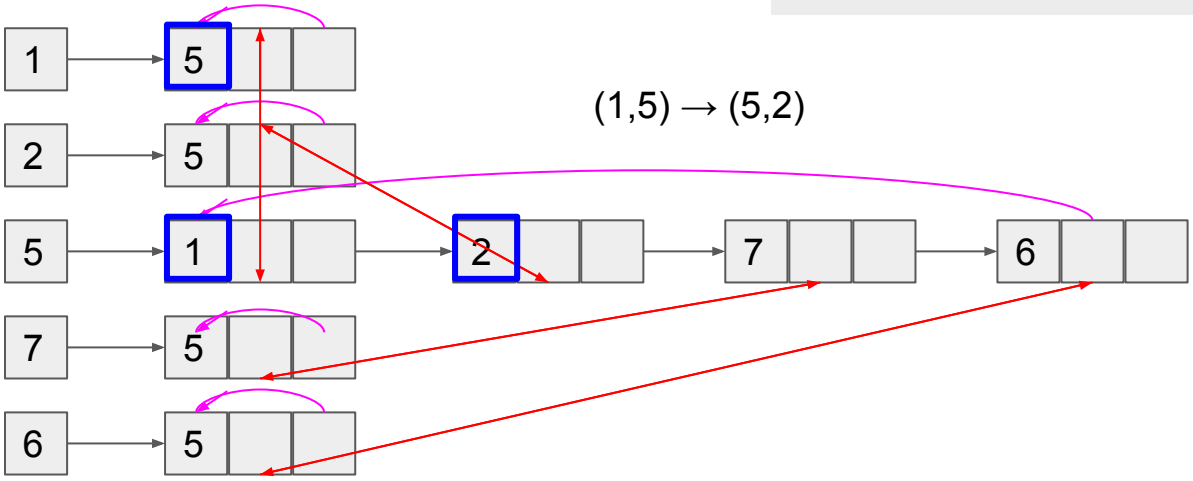
Euler circuit of T

for each adjacency list entry [i,j]

Euler_next[i,j]

= Adj_next[twin[i,j]]





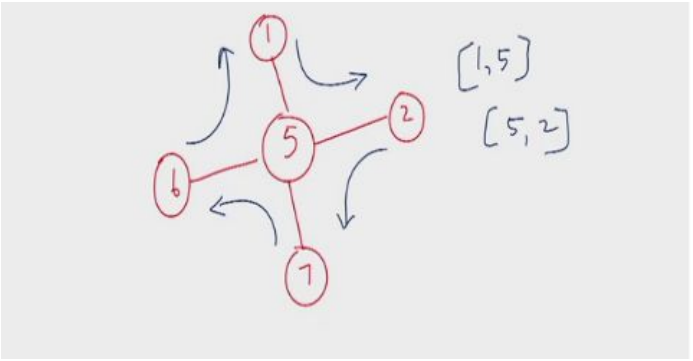
Compute Euler circuit in parallel, using circular adjacency lists with twin pointers

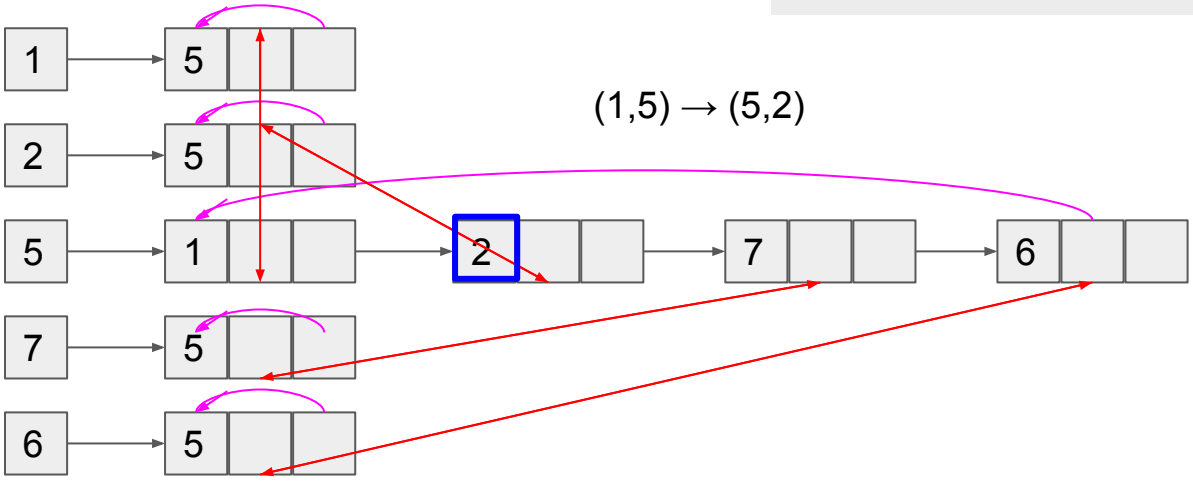
Euler circuit of T

for each adjacency list entry [i,j]

Euler_next[i,j]

= Adj_next[twin[i,j]]





(1,5) → (5,2)

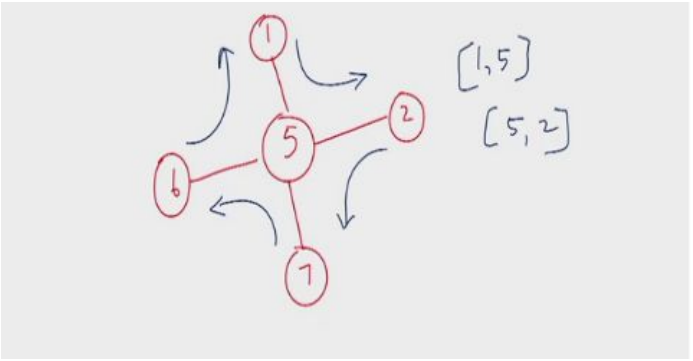
Compute Euler circuit in parallel, using circular adjacency lists with twin pointers

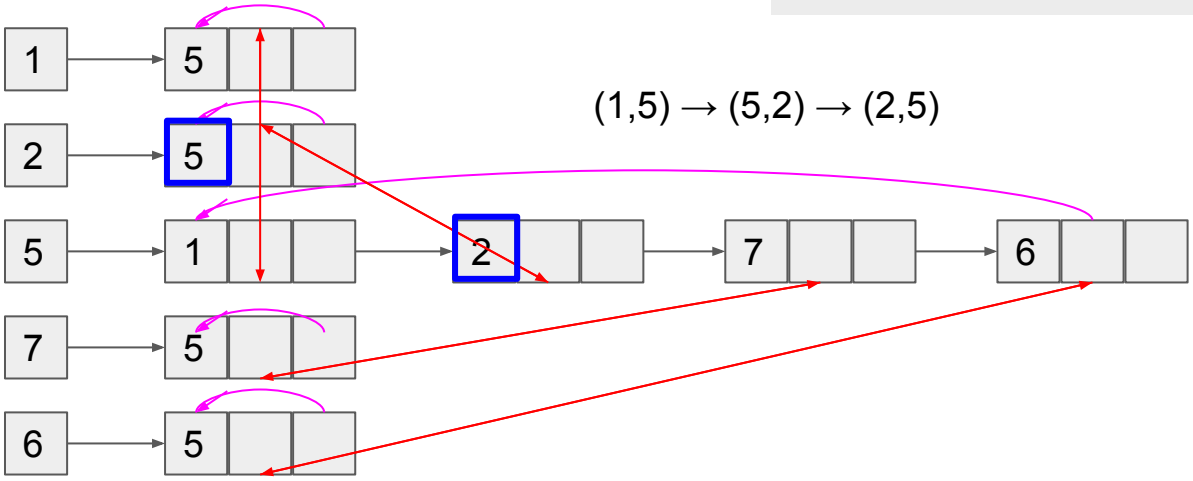
Euler circuit of T

for each adjacency list entry [i,j]

Euler_next[i,j]

= Adj_next[twin[i,j]]





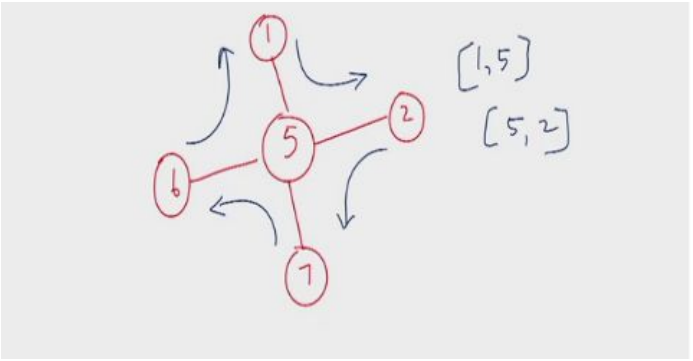
Compute Euler circuit in parallel, using circular adjacency lists with twin pointers

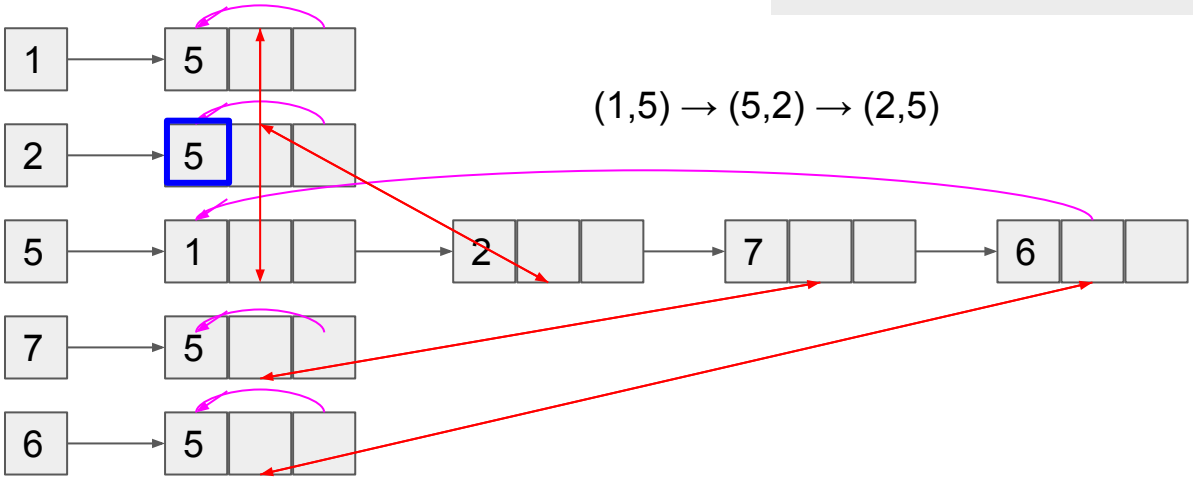
Euler circuit of T

for each adjacency list entry [i,j]

Euler_next[i,j]

= Adj_next[twin[i,j]]





(1,5) → (5,2) → (2,5)

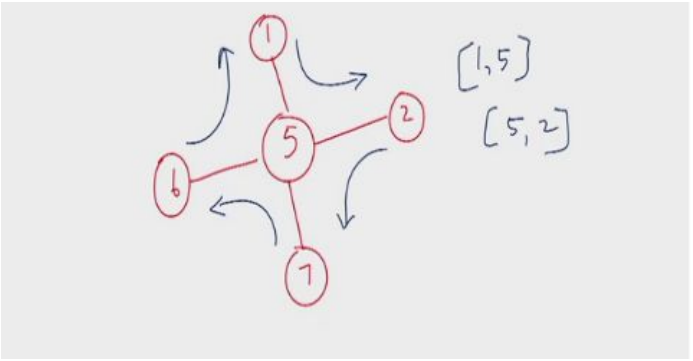
Compute Euler circuit in parallel, using circular adjacency lists with twin pointers

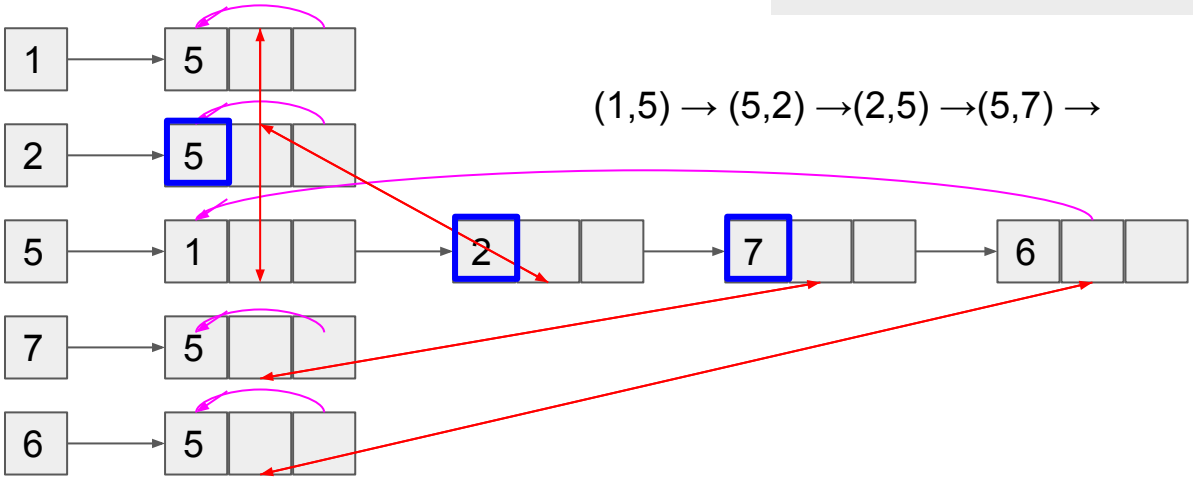
Euler circuit of T

for each adjacency list entry [i,j]

Euler_next[i,j]

= Adj_next[twin[i,j]]





(1,5) → (5,2) →(2,5) →(5,7) →

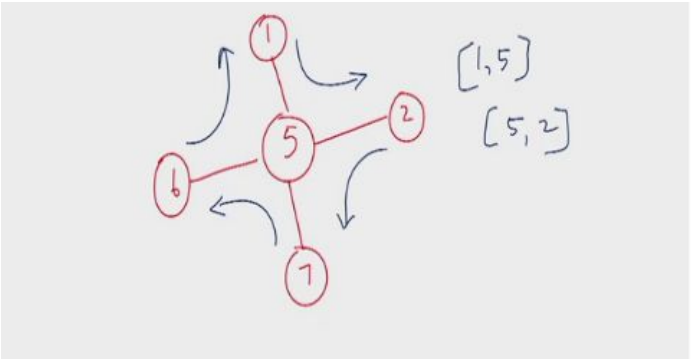
Compute Euler circuit in parallel, using circular adjacency lists with twin pointers

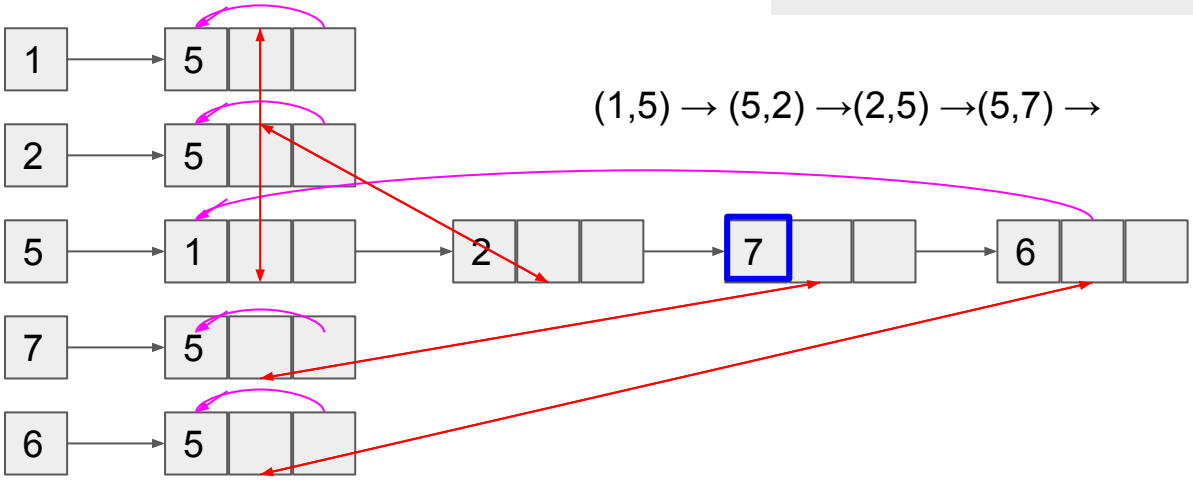
Euler circuit of T

for each adjacency list entry [i,j]

Euler_next[i,j]

= Adj_next[twin[i,j]]





(1,5) → (5,2) →(2,5) →(5,7) →

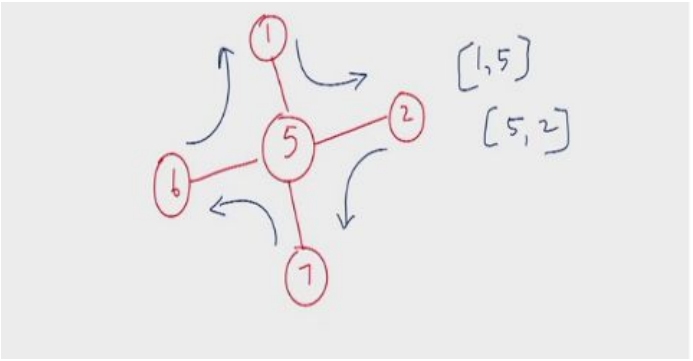
Compute Euler circuit in parallel, using circular adjacency lists with twin pointers

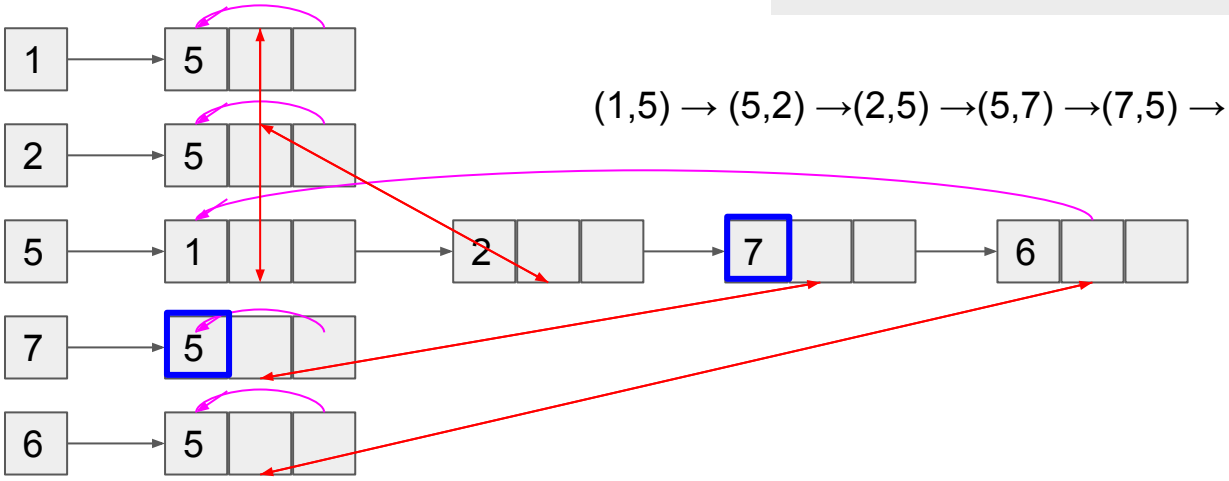
Euler circuit of T

for each adjacency list entry [i,j]

Euler_next[i,j]

= Adj_next[twin[i,j]]





(1,5) → (5,2) →(2,5) →(5,7) →(7,5) →

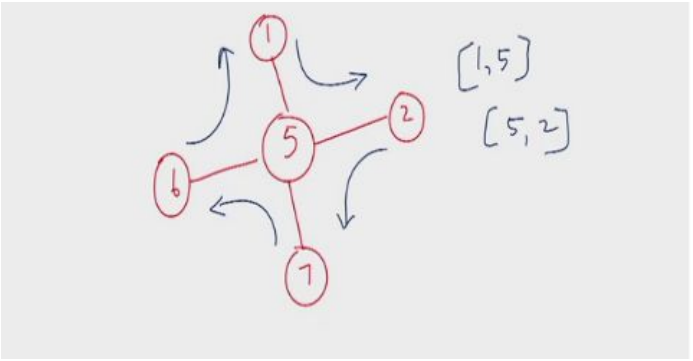
Compute Euler circuit in parallel, using circular adjacency lists with twin pointers

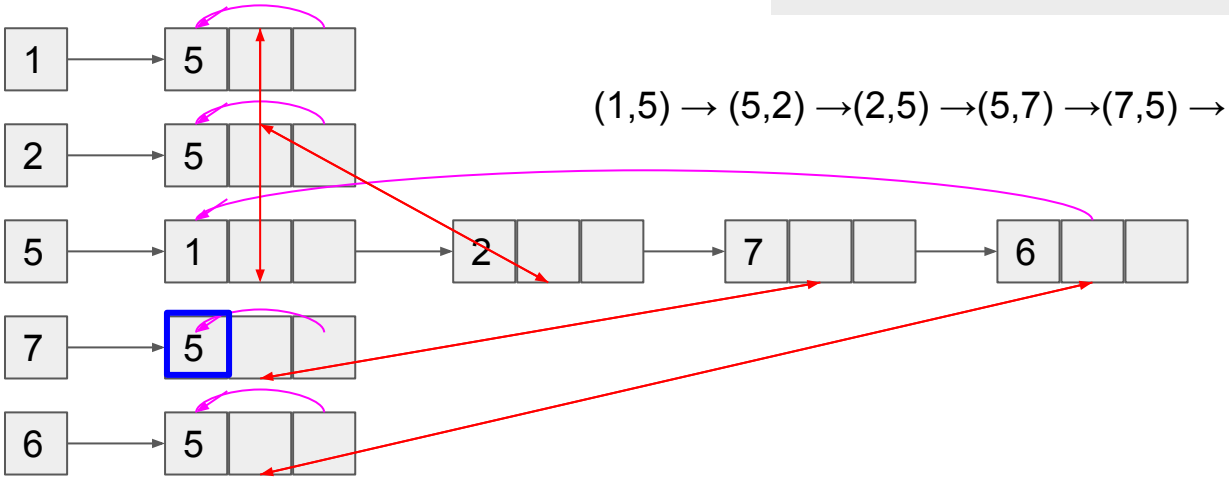
Euler circuit of T

for each adjacency list entry [i,j]

Euler_next[i,j]

= Adj_next[twin[i,j]]





(1,5) → (5,2) →(2,5) →(5,7) →(7,5) →

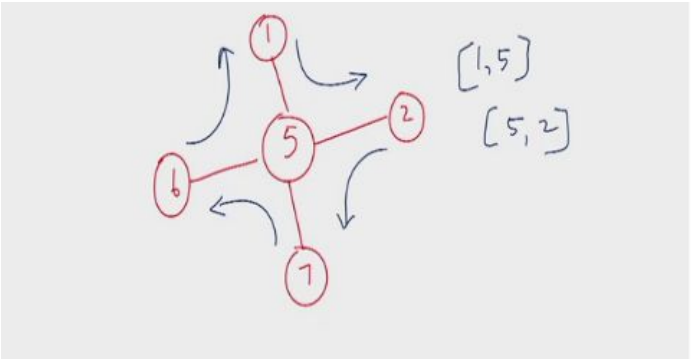
Compute Euler circuit in parallel, using circular adjacency lists with twin pointers

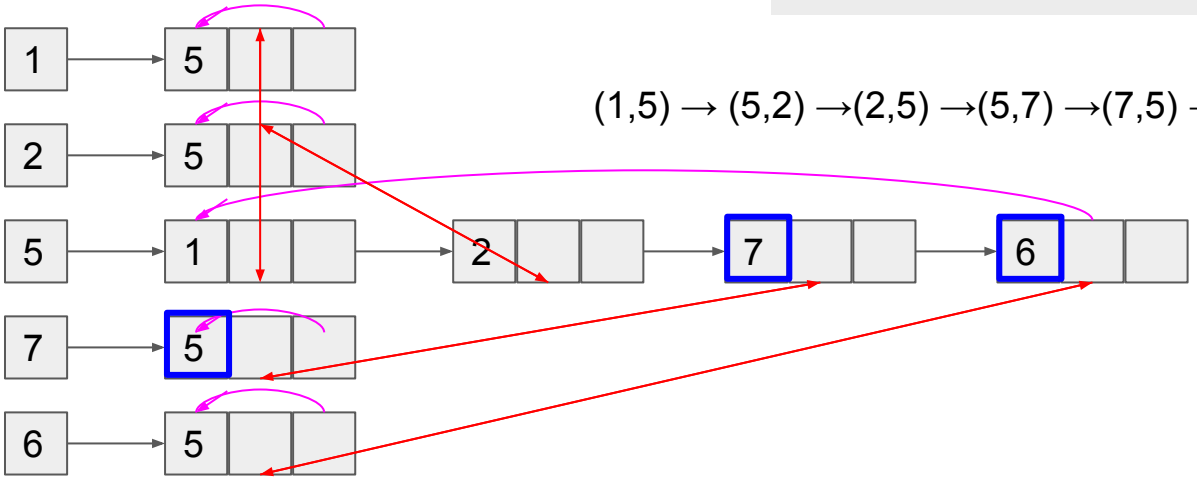
Euler circuit of T

for each adjacency list entry [i,j]

Euler_next[i,j]

= Adj_next[twin[i,j]]





(1,5) → (5,2) →(2,5) →(5,7) →(7,5) →(5,6) →

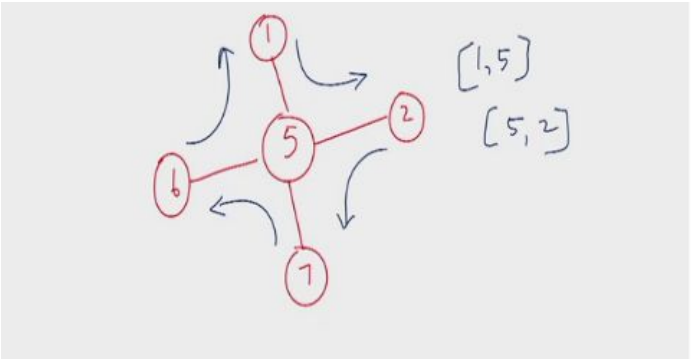
Compute Euler circuit in parallel, using circular adjacency lists with twin pointers

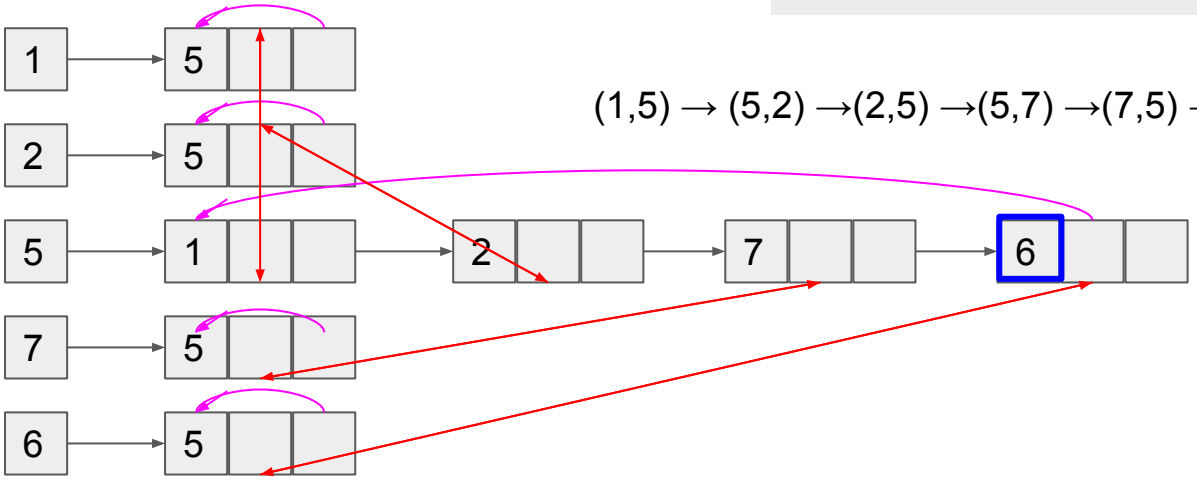
Euler circuit of T

for each adjacency list entry [i,j]

Euler_next[i,j]

= Adj_next[twin[i,j]]





(1,5) → (5,2) →(2,5) →(5,7) →(7,5) →(5,6) →

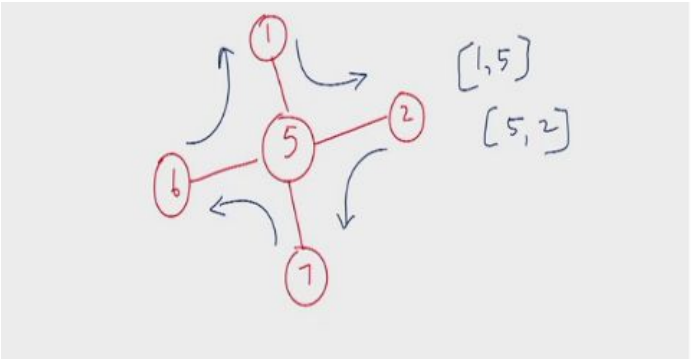
Compute Euler circuit in parallel, using circular adjacency lists with twin pointers

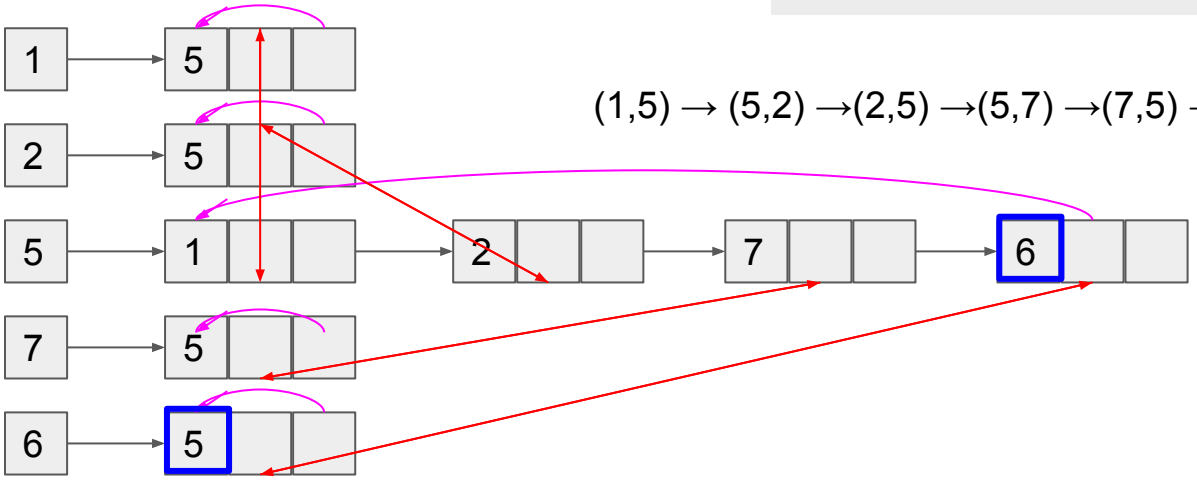
Euler circuit of T

for each adjacency list entry [i,j]

Euler_next[i,j]

= Adj_next[twin[i,j]]





(1,5) → (5,2) →(2,5) →(5,7) →(7,5) →(5,6) →(6,5)

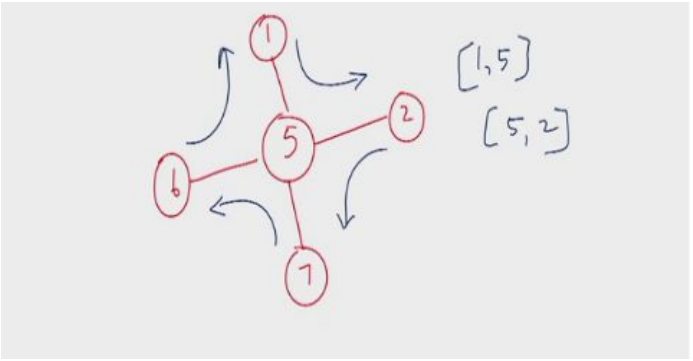
Compute Euler circuit in parallel, using circular adjacency lists with twin pointers

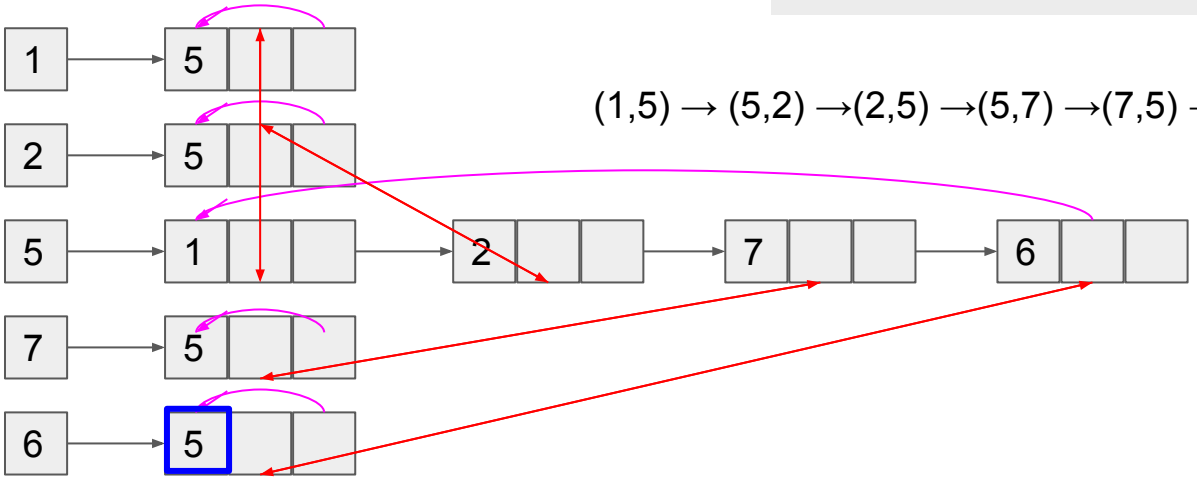
Euler circuit of T

for each adjacency list entry [i,j]

Euler_next[i,j]

= Adj_next[twin[i,j]]





(1,5) → (5,2) →(2,5) →(5,7) →(7,5) →(5,6) →(6,5)

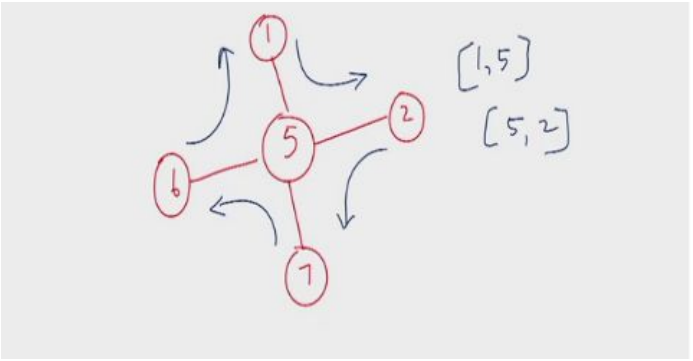
Compute Euler circuit in parallel, using circular adjacency lists with twin pointers

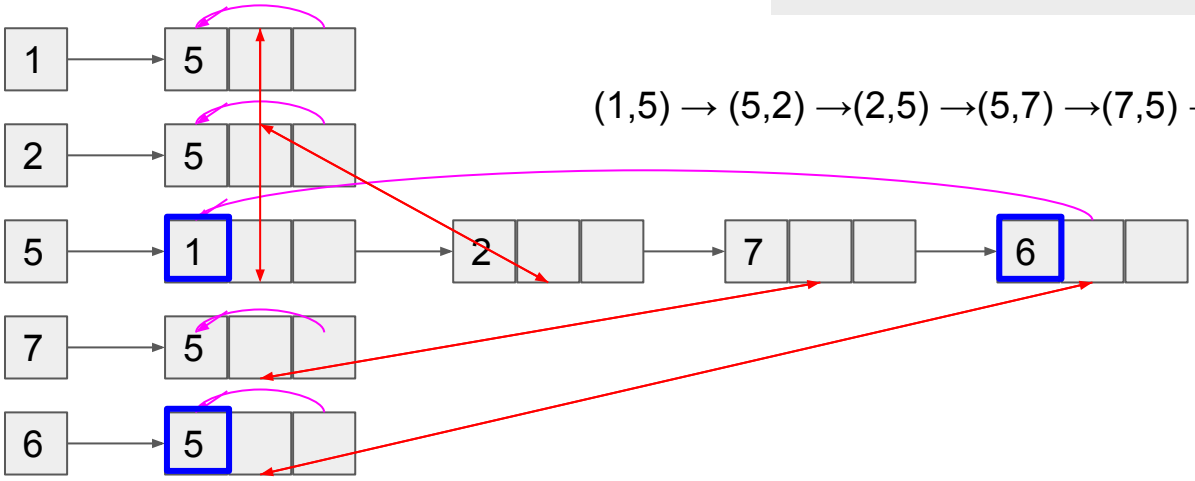
Euler circuit of T

for each adjacency list entry [i,j]

Euler_next[i,j]

= Adj_next[twin[i,j]]





(1,5) → (5,2) →(2,5) →(5,7) →(7,5) →(5,6) →(6,5) → (5,1)

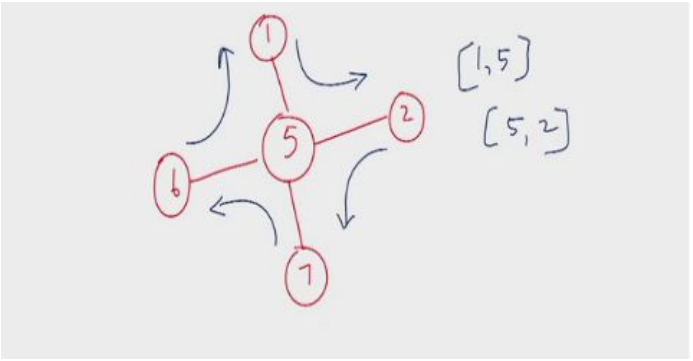
Compute Euler circuit in parallel, using circular adjacency lists with twin pointers

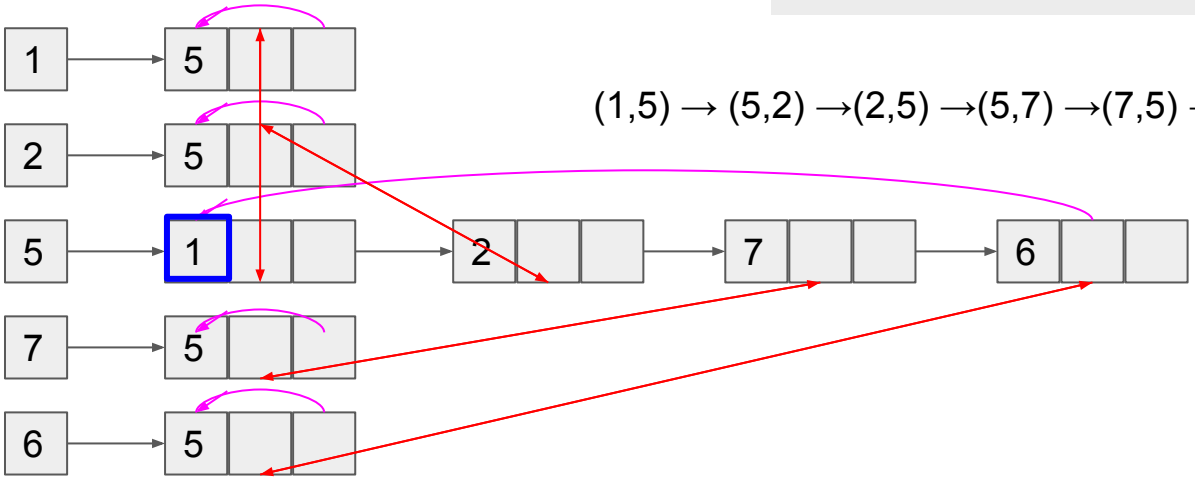
Euler circuit of T

for each adjacency list entry [i,j]

Euler_next[i,j]

= Adj_next[twin[i,j]]





(1,5) → (5,2) →(2,5) →(5,7) →(7,5) →(5,6) →(6,5) → (5,1)

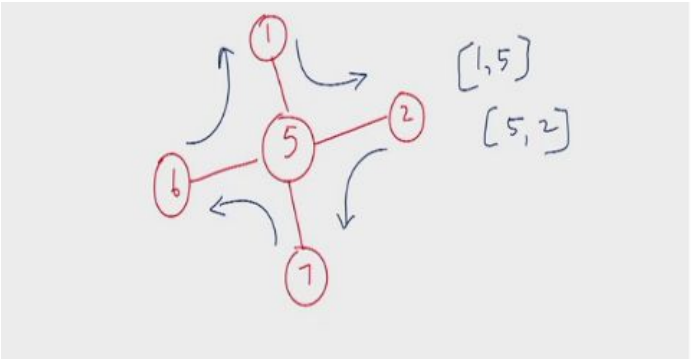
Compute Euler circuit in parallel, using circular adjacency lists with twin pointers

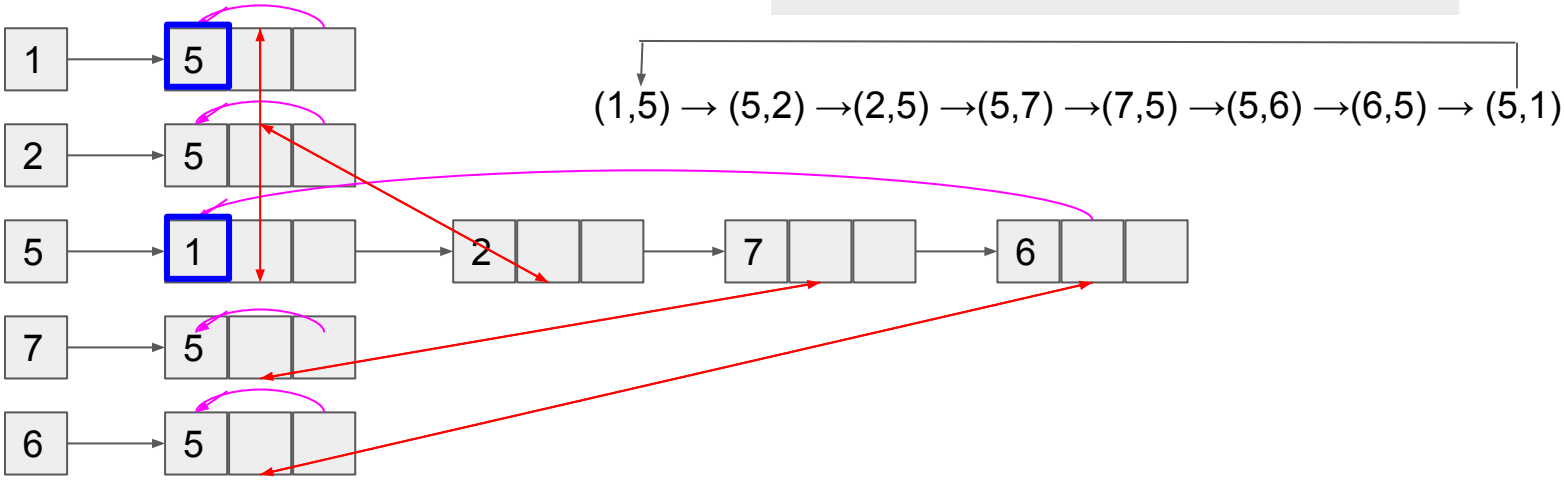
Euler circuit of T

for each adjacency list entry [i,j]

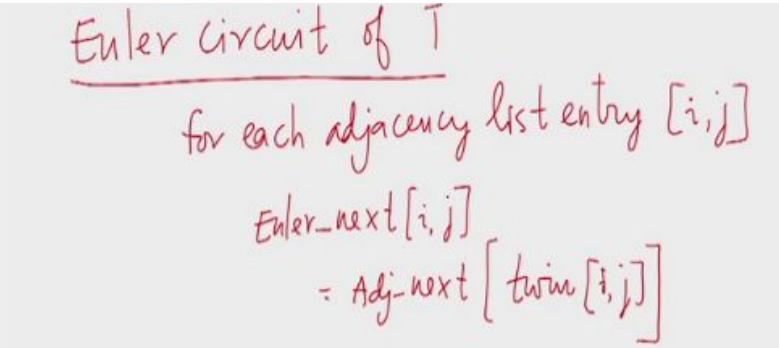
Euler_next[i,j]

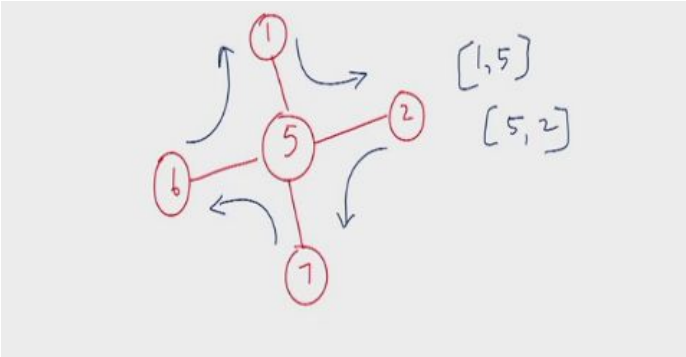
= Adj_next[twin[i,j]]

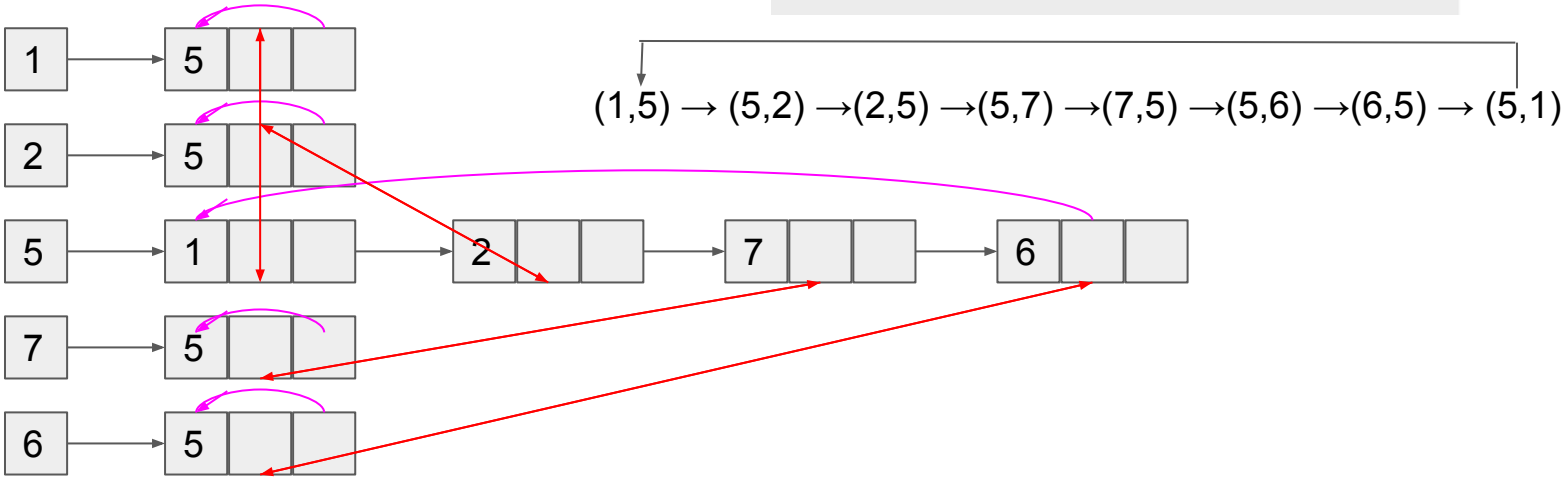




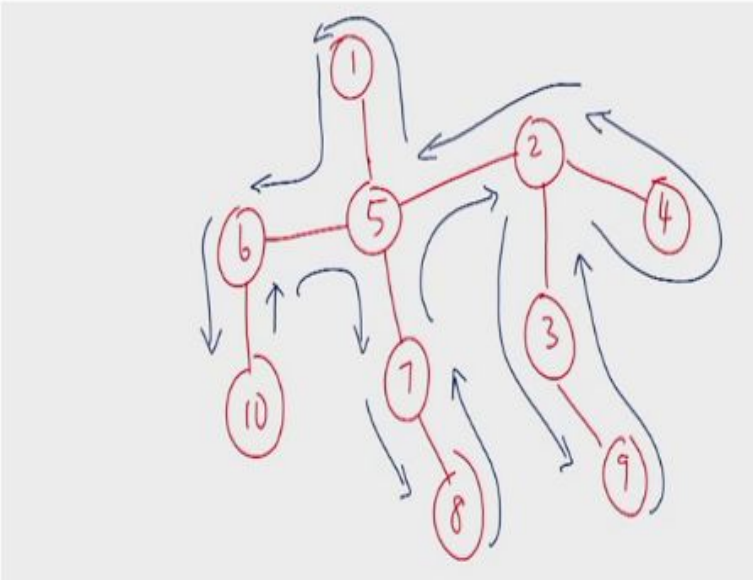
Compute Euler circuit **in parallel**, using circular adjacency lists with twin pointers

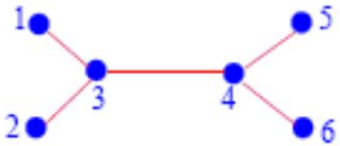




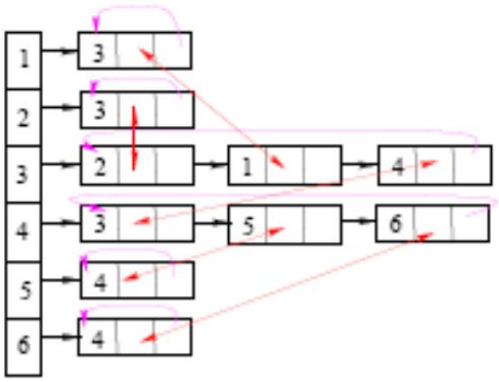


Much larger trees, many possible Euler circuits as adjacency lists have vertices in any arbitrary order



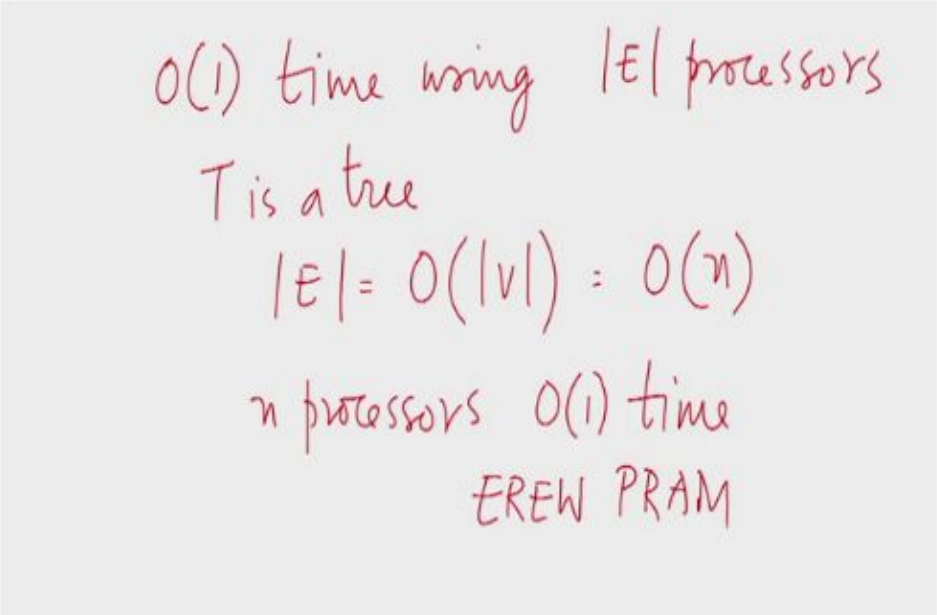


v	adj(v)
1	3
2	3
3	2, 1, 4
4	3, 5, 6
5	4
6	4

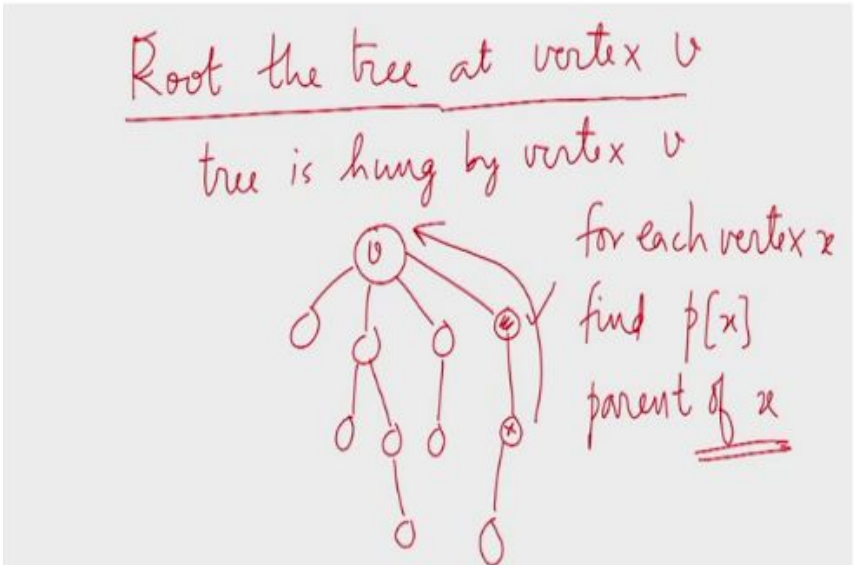


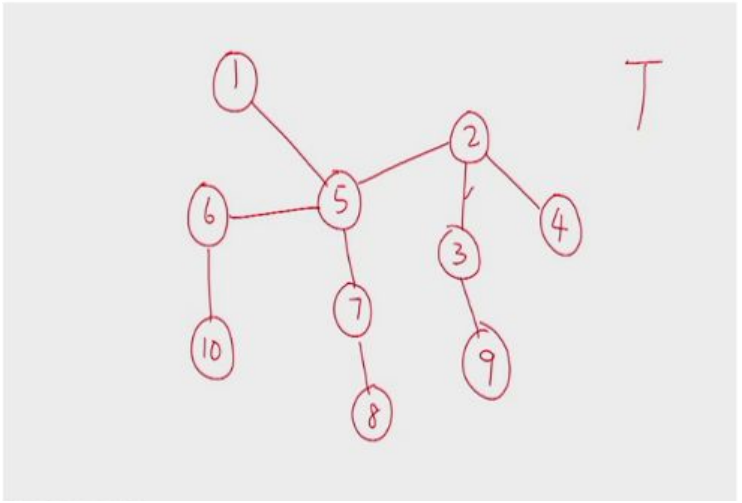
(1,3) → (3,4) →(4,5) →(5,4) →(4,6) →(6,4) →(4,3) → (3,2) → (2,3) → (3,1)

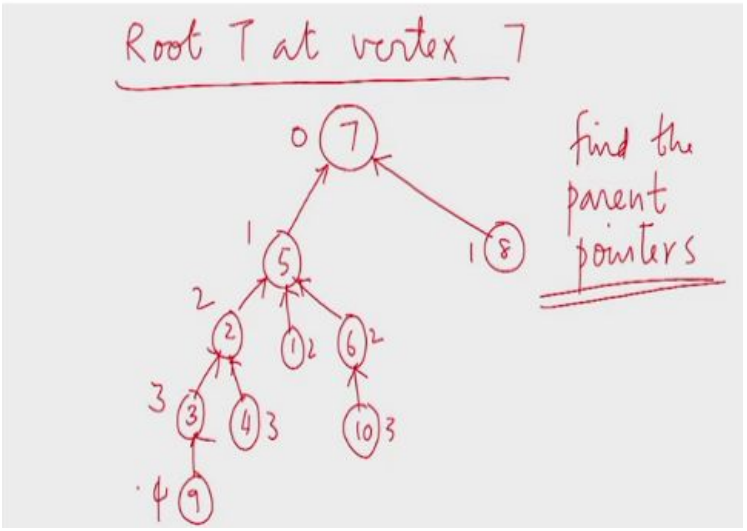
So we can compute Euler circuit of any tree in O(1)



The notion of a “rooted” tree

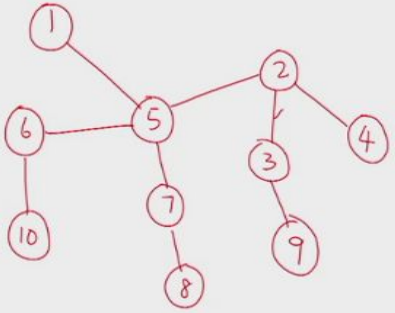






Each node's parent in "rooted" tree using Euler circuit

Take the Euler circuit
break the circuit by deleting
one incoming edge of v
Now the ckt has become a list
Rank the list
Use the ranks ?



15 11 78 17 93 5
56 12 ~~87~~ 32 6
6 10 13 75 1 24 7
10 6 14 52 2 42 8
6 5 15 23 3 25 9
5 7 16 39 4 51 10

v = 7

Compare [i,j] with [j,i]
if rank [i,j] < rank [j,i]
i = p(j)

Each node's level in "rooted" tree using Euler circuit

The levels of the nodes of T
(T is rooted at vertex v)

level (v) = 0
level (w) = 1 where w is a child of v
level (w) = 2 —— " is a grandchild of v

15 11 78 17 93 5
56 12 ~~87~~ 32 6
6 10 13 75 1 24 7
10 6 14 52 2 42 8
6 5 15 23 3 25 9
5 7 16 39 4 51 10

v = 7

compare [i,j] with [j,i]
if rank [i,j] < rank [j,i]
i = p(j)

5 7 2 1

5-1
2-2
6-2

75 1 1
52 1 2
23 1 3
39 1 4
93 -1 3
32 -1 2
24 1 3
42 -1 2

25 -1 1
51 -1 2
15 -1 1
56 1 2
6 10 1 3
10 6 -1 2
6 5 -1 1
5 7 -1 0
78 1 1

for parent-child edges : 1
child parent edges : -1

Euler circuit for trees can compute for each node

- **Preorder labels**
- Inorder labels
- Postorder labels
- **Parent**
- **Level**
- Number of descendants