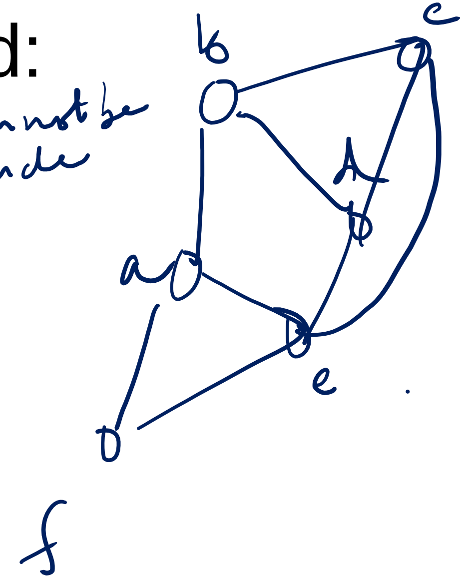
Complexity and Advanced Algorithms
Spring 2020

Fractional Independent Set (FIS)

- Let $G = (V, E)$ be an undirected graph.
- We say that a set $S \subseteq V$ is an independent set if no two vertices of S are mutual neighbors in G.
- The notion of an independent set is very popular in graph theory.
- Several variants are also studied:
  - Maximal Independent Set (MIS)
  - Maximum Independent Set
  - Fractional Independent Set

S cannot be expanded



$S = \{a, d\}$

Fractional Independent Set

- We now define an FIS.
- Let $G = (V, E)$ be an undirected graph. A set $S \subseteq V$ is called a (c,d)-fractional independent set of G if it satisfies:
  - S is an independent set
  - For every vertex v in S, degree(v) is at most d.
  - $|S|$ is at least $|V|/c$.

$|S| \geq |V| \cdot \frac{1}{c}$ $\quad c > 1$

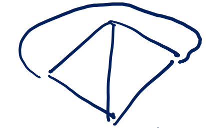
fraction

Fractional Independent Set

- We now define an FIS.
- Let G = (V, E) be an undirected graph. A set S $\subseteq$ V is called a (c,d)-fractional independent set of G if it satisfies:
  - S is an independent set
  - For every vertex v in S, degree(v) is at most d.
  - |S| is at least |V|/c.

- Not every graph may have an FIS.
- Planar graphs have an FIS.
  - We will see that today, along with a way to construct such an FIS.

Planar Graphs

$|V_d| \geq \frac{|V|}{c}$ K_4  $|E| = 6$

- Recall the theorem of Euler concerning planar graphs.

$|V| = 4$

$6 = 3 \times 4 - 6$

- Theorem (Euler): If G = (V, E) is planar with |V| at least 3, then |E| is at most 3|V| - 6.
- Using the above theorem, can show that in a planar graph G, there are lots of vertices of a degree at most d.

$d \geq 6$

- Theorem: Let G = (V, E) be a planar graph and d be an integer at least 6. Let Vd be the set of vertices of degree at most d. Then, |Vd| is at least |V|/c for some constant c.

V_d

V_d

$\rightarrow \sum \deg(v) = 2|E|$

$|V_d| \times d \leq 2|E|$

Planar Graphs

- Theorem: Let G = (V, E) be a planar graph and d be an integer at least 6. Let V_d be the set of vertices of degree at most d. Then, $|V_d|$ is at least $|V|/c$ for some constant c.

$V \setminus V_d = V_h$

- Proof: Let V_h be the complement of V_d.
- We will estimate an upper bound on the size of Vh as follows.
- Consider Σ_v degree(v) $\geq \Sigma_{v\in V_h}$ degree(v) $\geq (d+1)|V_h|$.

Planar Graphs

- Theorem: Let $G = (V, E)$ be a planar graph and d be an integer at least 6. Let V_d be the set of vertices of degree at most d. Then, $|V_d|$ is at least $|V|/c$ for some constant c.
- Proof: Let V_h be the complement of V_d.
- We will estimate an upper bound on the size of Vh as follows.
- Consider Σ_v degree(v) $\geq \Sigma_{v\in V_h}$ degree(v) $\geq$ $(d+1)|V_h|$.
- Using Euler's theorem, we get that $(d+1)|V_h| \leq$ $2|E|$ $2(3|V| - 6)$.

Planar Graphs

- Theorem: Let G = (V, E) be a planar graph and d be an integer at least 6. Let V_d be the set of vertices of degree at most d. Then, $|V_d|$ is at least $|V|/c$ for some constant c.
- Proof: Let V_h be the complement of V_d.
- We will estimate an upper bound on the size of Vh as follows.
- Consider Σ_v degree(v) $\geq \Sigma_{v\in V_h}$ degree(v) $\geq (d+1)|V_h|$.
- Using Euler's theorem, we get that $(d+1)|V_h| \leq 2(3|V| - 6)$.
- So, $|V_d| \geq |V| - |V_h| \geq |V|.(d - 5)/(d+1)$.

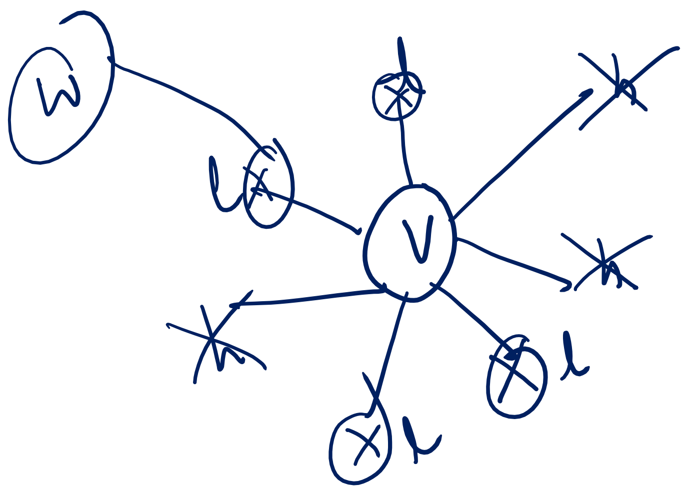
offline

d fixed

$f(d) = c$

Planar Graphs

- Theorem: Let G = (V, E) be a planar graph and d be an integer at least 6. Let V_d be the set of vertices of degree at most d. Then, $|V_d|$ is at least $|V|/c$ for some constant c.
- With d = 6, we get that $|V_6|$ is at least $|V|/7$.



$v \in S$

at most 6 other vertices are eliminated

Planar Graphs

$$|V_d| \geq |V| \cdot \frac{d-5}{d+1}$$

- Theorem: Let G = (V, E) be a planar graph and d be an integer at least 6. Let V_d be the set of vertices of degree at most d. Then, $|V_d|$ is at least $|V|/c$ for some constant c.
- With d = 6, we get that $|V_6|$ is at least $|V|/7$.
- Can be used to show that in a sequential setting, an FIS for a planar graph can be found.
  - Start with any vertex v.
  - If v has a degree at most 6, add v to the set S.
  - Remove all the neighbors of v.
  - Continue until there are more vertices.
- Can show that $|S|$ is at least $|V_6|/7$.

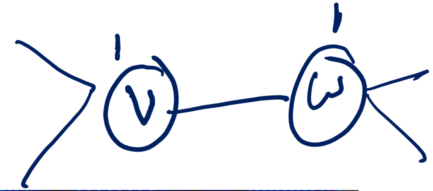
S is a (49, 6)-FIS

$|V_6| \geq |V|/7$

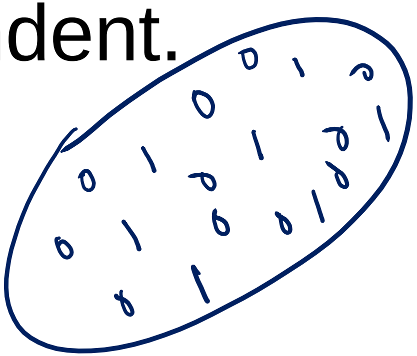
FIS

- The sequential algorithm is not efficient in parallel.
- Need a better approach where multiple nodes decide to join the FIS or not on their own.

Parallel FIS on Planar Graphs



- Consider each vertex of degree at most 6.
- For each such vertex, set label(v) = 1 with probability ½ and set label(v) = 0 with probability ~~0~~. 1/2 .
- Note that several vertices and their neighbors may choose their label as 1.
- So, the set of vertices with label set to 1 is not independent.



$$\text{Label}(V) = \begin{cases} \boxed{1} & \text{with prob } 1/2 \\ 0 & '' \end{cases}$$

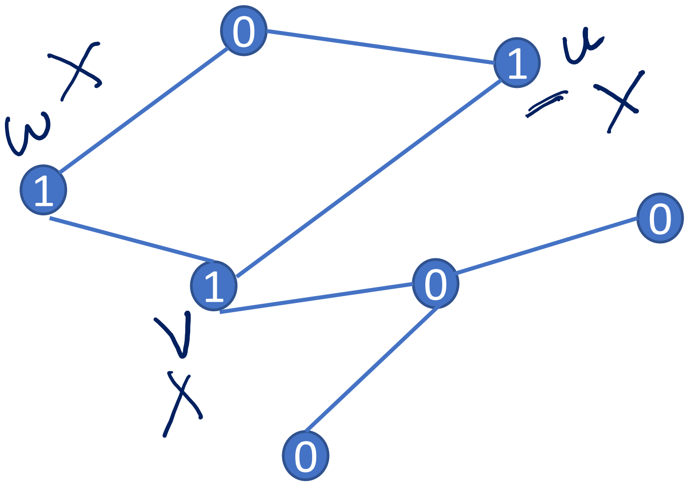
Parallel FIS on Planar Graphs

- Consider each vertex of degree at most 6.
- For each such vertex, set label(v) = 1 with probability ½ and set label(v) = 0 with probability 0.
- Note that several vertices and their neighbors may choose their label as 1.
- So, the set of vertices with label set to 1 is not independent.

Parallel FIS on Planar Graphs

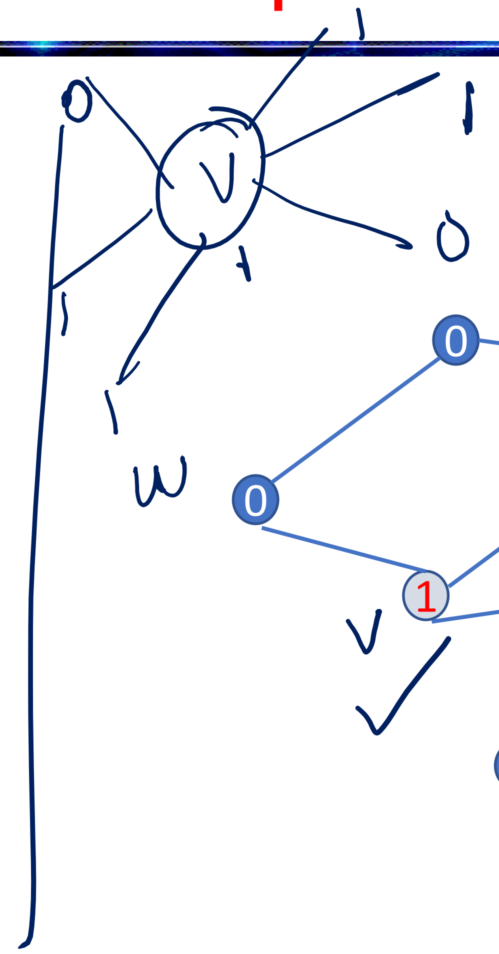
- Consider each vertex of degree at most 6.
- For each such vertex, set label(v) = 1 with probability ½ and set label(v) = 0 with probability 0.
- Note that several vertices and their neighbors may choose their label as 1.
- So, the set of vertices with label set to 1 is not independent.
- To make this set independent, we proceed as follows.

Parallel FIS on Planar Graphs

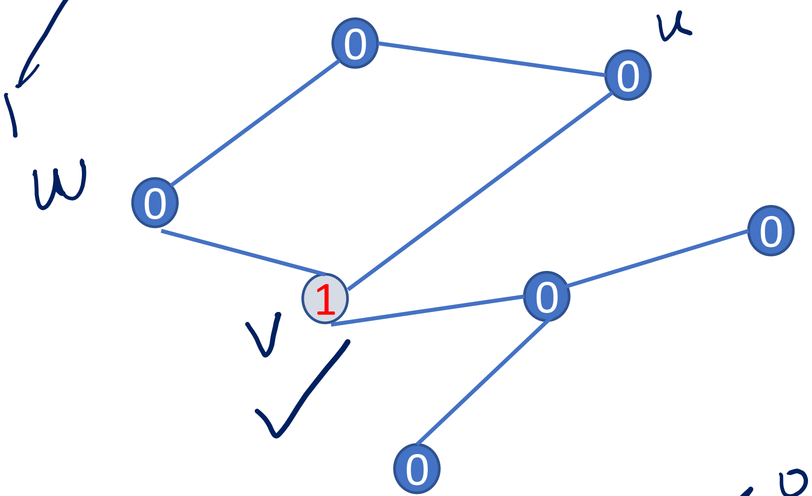


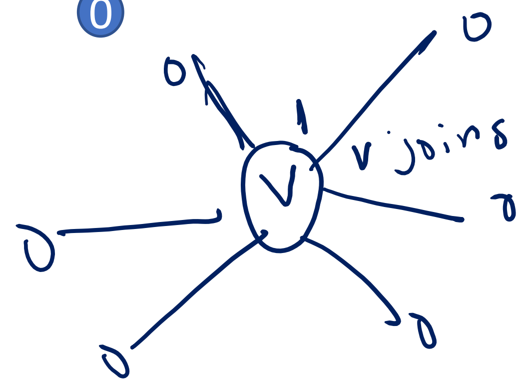
$\forall v \mid \deg(v) = 6$

$\Pr(v \in S) = \frac{1}{2^7}$.



v drops out





v joins S

Parallel FIS on Planar Graphs

- Consider each vertex of degree at most 6.
- For each such vertex, set label(v) = 1 with probability ½ and set label(v) = 0 with probability 0.
- Note that several vertices and their neighbors may choose their label as 1.
- So, the set of vertices with label set to 1 is not independent.
- To make this set independent, we proceed as follows.
- If a node v of degree at most 6 has label(v) = 1 and all its neighbors have label 0, then v enters a set S.
- Otherwise, v drops out.

Parallel FIS on Planar Graphs

- We want to claim that S is a (c, 6)—FIS for some constant c.
  - Only for planar graphs of course.
- S is indeed an independent set.
- Moreover, the degree of any vertex in S is at most 6.
- So, we only have to find a suitable value for c.

$|S| \geq |V| \cdot \frac{1}{c}$

$\deg(v) \leq 6$

$\Pr(v \in S) \geq \frac{1}{7} \cdot \frac{1}{2}$

v

Are X_r's independent ?

Parallel FIS on Planar Graphs

$$X_r = \begin{cases} 1 & \text{if } u_r \in S \\ 0 & \text{o/w} \end{cases}$$

- For a vertex v with degree at most 6, note that Pr(v has label 1 and v is in S) is at least $1/2^7$.
  - For the event to occur, v has to pick 1 as its label, and the neighbors of v (of degree at most 6) have to pick 0 as their label.

$|S| \geq \frac{|V|}{c}$

- Let us use 1/128 as the actual probability of the above event going forward.

$|S| = X$, $E|S| \geq \frac{|V|}{7} \times \frac{1}{128}$

- Now, we can note that since $|V_6|$ is at least $|V|/7$, on expectation, the number of vertices in S is at least $|V|/(7\text{x}128)$.

$E X_r = 1 \times \frac{1}{128} = \frac{1}{128}$

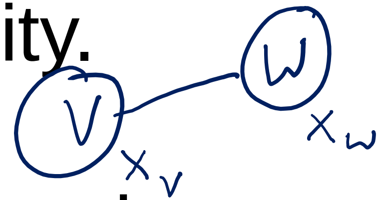
$V_6 = \{u_1, u_2, u_3 \cdots u_\ell\}$

$X = \sum_{r=1}^{\ell} X_r$, $EX = \frac{|V_6|}{128}$

Parallel FIS on Planar Graphs

$v \in S \Rightarrow w \notin S$?

- We wish that S has a large size not just in expectation, but also with high probability.



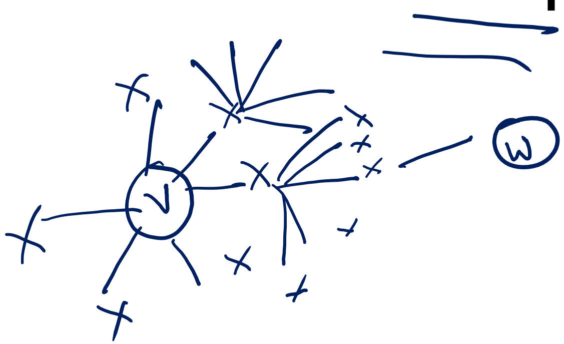
- We have E|S| is at least |V|/(7x128).
- It appears that we can use Chernoff bounds to show that the size of S is close to its expectation.
- But, the random variables that we use are not independent.

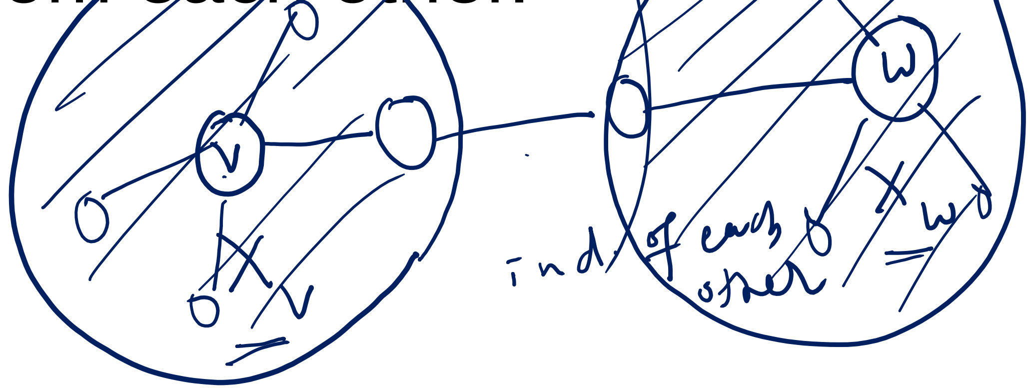
$x_v = 1 \Rightarrow x_w = 0$

- In particular, for two neighbors v and w of small degree, if v is in S then w cannot be in S.
- The events v in S, and w in S are therefore not always independent.

Parallel FIS in Planar Graphs

- There are several ways to deal with this lack of independence.
- One such way is to consider only a subset of random variables that are then independent of each other.
- In the present case, we will consider only vertices of degree at most 6 and are at least a distance of 3 apart from each other.





Parallel FIS in Planar Graphs

- There are several ways to deal with this lack of independence.
- One such way is to consider only a subset of random variables that are then independent of each other.
- In the present case, we will consider only vertices of degree at most 6 and are at least a distance of 3 apart from each other.
- The random variables corresponding to such vertices are independent.

Parallel FIS in Planar Graphs

$X = \sum_{v \in V'} X_v$ $\quad E X$

- Let V’ be the set of vertices of degree at most 6 .
- We observe that |V’| is at least |V|/36.
  6
- Now, define an indicator random variable for each v in V’ so that this RV takes value 1 if v is in S.
  $v \in V'$ $\quad X_v = \begin{cases} 1 & \text{if } v \in S \\ 0 & \text{o/w} \end{cases}$ $\quad$ are independ.
- Define X as the sum of these random variables.
- Note that EX is at least |V’|/(7x128), that is now at least |V|/(36x7x128).
  $\approx |V|/36000$ $\quad c = \frac{1}{72000}$
- Use Chernoff bounds to show that $\Pr(X \le EX/2)$ is at most exp{-EX.1/12} which is polynomially small.
  $\Rightarrow \{X \ge \frac{EX}{2}\}$ whp

Advanced Optimal Solutions

$1 \rightarrow 8 \rightarrow 5 \rightarrow 11 \rightarrow 2 \rightarrow 6 \rightarrow 10 \rightarrow 4 \rightarrow 3 \rightarrow 7 \rightarrow 12 \rightarrow 9$

- General technique suggests that we solve a smaller problem and extend the solution to the larger problem.
- To apply our technique we should use the pointer jumping based solution on a sub-list of size n/log n.
- How to identify such a sublist?

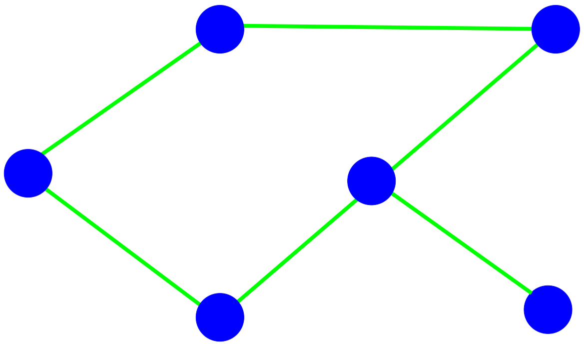
Advanced Solutions

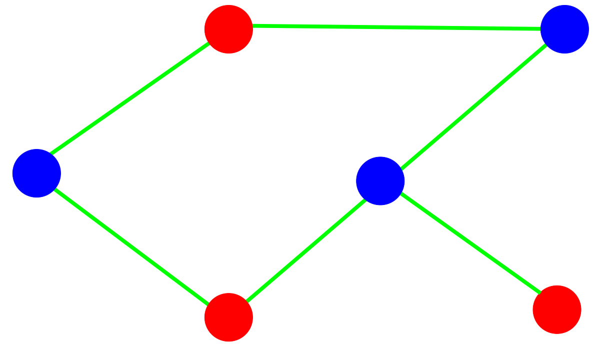
$$1 \rightarrow 8 \rightarrow 5 \rightarrow 11 \rightarrow 2 \rightarrow 6 \rightarrow 10 \rightarrow 4 \rightarrow 3 \rightarrow 7 \rightarrow 12 \rightarrow 9$$

$$1 \rightarrow 5 \rightarrow 11 \rightarrow 6 \rightarrow 4 \rightarrow 3 \rightarrow 12 \rightarrow 9$$

- Cannot pick equidistant as earlier.
- However, can pick independent nodes.
  - Removing independent nodes is easy!
  - Formally, an independent set of nodes.
  - Can extend the solution easily in such a case.

Advanced Solutions





- Formally, in a graph G = (V, E), a subset of nodes U $\subseteq$V is called an independent set if for ever pair of vertices u,v in U, (u,v) $\notin$ E.
- Linked lists (viewed as a graph) have the property that they have large independent sets.

Advanced Solutions

1 → 8 → 5 → 11 → 2 → 6 → 10 → 4 → 3 → 7 → 12 → 9

List with elements removed

1 →	5 →	11 →	6 →	4 →	3 →	12 →	9
(1)	(2)	1)	(2)	(2)	(1)	(2)	(1)

1 →	5 →	11 →	6 →	4 →	3 →	12 →	9
(1)	(3)	(4)	(6)	(8)	(9)	(11)	(12)

Ranked short list

8
(2)

reintroduce element 8

• Transfer current rank along with successor during removal.

Advanced Optimal Solutions

$1 \rightarrow 8 \rightarrow 5 \rightarrow 11 \rightarrow 2 \rightarrow 6 \rightarrow 10 \rightarrow 4 \rightarrow 3 \rightarrow 7 \rightarrow 12 \rightarrow 9$

$1 \rightarrow 5 \rightarrow 11 \rightarrow 6 \rightarrow 4 \rightarrow 3 \rightarrow 12 \rightarrow 9$

- Algorithm outline:
  - Remove an independent set of nodes in the linked list.
  - Rank the remaining list, and then
  - Rank the removed elements.

- Several algorithms use this technique with variations: Anderson-Miller, Hellman-JaJa, Reid-Miller,...

Advanced Solutions

- Issue 1: How to find a large independent set of nodes in parallel?
- Issue 2: How many iterations needed to reduce the size of the list?
- Issue 3: How to rank the removed elements?

Advanced Solutions

- Issue 1:
  - Use techniques from parallel symmetry breaking or fractional independent sets.
  - Can obtain an independent set of size ≥ n/c.
- Issue 2:
  - The naïve algorithm is slightly non-optimal (by a factor of O(log n)
  - Hence, reduce the size of the list from n to n/log n.
- Issue 3:
  - Bookkeep enough details during removal
  - Reintroduce in the reverse order.

Advanced Solutions

- Our algorithm outline:

```
Algorithm Rank(L)
L_1 = L;
For r iterations do
    Pick a fractional independent set U_i of nodes of size ≥ n/c
    L_i = Remove nodes in U_i from L_{i-1};
End-for

Rank the list L_r using pointer jumping.

For i = r down to 1 do
    Reinsert the nodes in U_i into L_i
End-for
End.
```

- Question: How many iterations required?

Back to List Ranking

- Each iteration of coloring the list can give an FIS of size at least n/c.
- We require that only n/log n nodes remain at the end.
- Hence, O(log log n) iterations are required.
  - $(n/c)^r = n/\log n$ at $r = O(\log \log n)$.

Back to List Ranking

- Time taken:
  - To shrink the list: Each iteration is O(1). At O(log log n) iterations, this takes O(log log n) time.
  - To rank the remaining list using pointer jumping: O(log n) time
  - To reintroduce the removed elements: Over r = O(log log n) iterations, O(log log n) time.
  - Total = O(log n).
- Work: O(n log log n)
  - Dominated by the work done in reintroducing the removed elements

Back to List Ranking

- A different way:
  - Reintroducing can be slowed down to make it optimal.
  - Use only n/log n processors, with each iteration taking O(log n) time.
  - Total time = O(log n).
  - Total work = O(n).
- Randomized algorithm since finding an FIS in randomized in nature.
- Deterministic variants exist too.

Back to List Ranking

- Yet another way:
  - Use an optimal approach to finding an independent set.
  - Takes O(log n) time and O(n) work.
- Overall time and work:
  - Time = O(log n log log n)
  - Work = O(n)
  - Optimal!

Back to List Ranking

- In general, if one can spend O(t) time and O(n) work in each iteration of removing nodes, then
  - Time = O(t log log n + log n)
  - Work = O(n).
- Have to lower t to get O(log n) optimal list ranking.
- There are such algorithms.
  - Anderson-Miller is one such example.
- Further reading
  - Anderson Miller described in JaJa's book
  - Hellman-JaJa is another popular approach
    - Used by most practical papers in recent times.

After a long break, welcome again.

Have to make up two lectures and one exam.

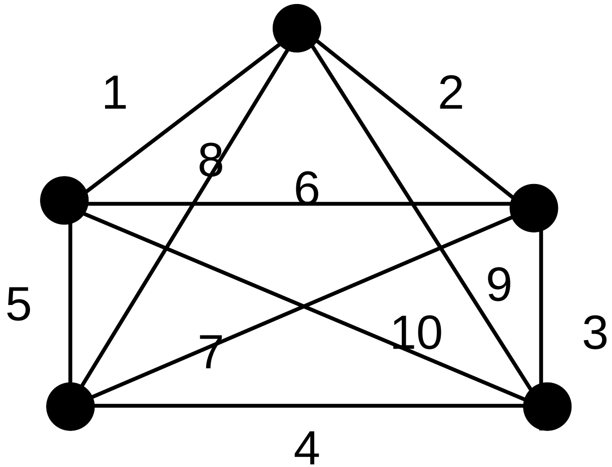
Lectures: What if we continue our classes till 4 PM for the next 6 lectures? With a small break after 1 hour.

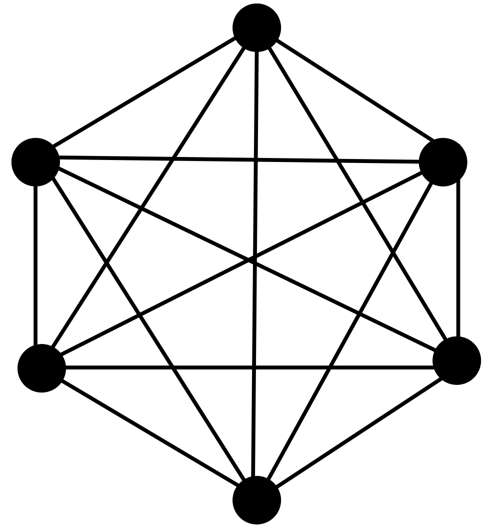
Exam: Will hold it some day after possibly 2 weeks from now. May be one hour exam, with only 15% weightage.

Tree Processing

- Now that we know how to process linked lists, let us consider tree algorithms.
- Problems we will consider:
  - Traversal
  - Expression evaluation
  - Least common ancestor, range minima

Traversal via Euler Tour

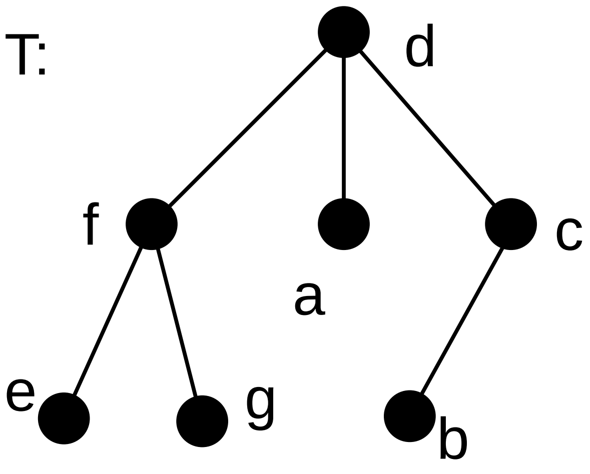




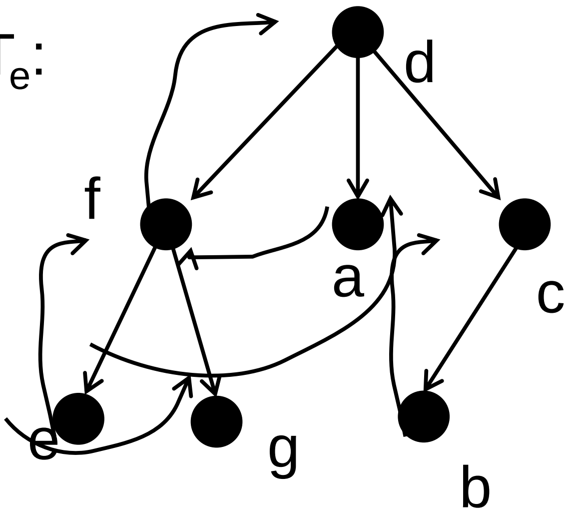
- Given a tree T = (V, E), we use an Euler tour as a primitive for tree traversal algorithms.
- Euler tour: Given a graph, an Euler tour is a cycle that includes every edge exactly once.
  - ➢ A directed graph G has an Euler tour if and only if for every vertex, its in-degree equals its out-degree.

Euler Tour of a Tree

T:



T_e:



- For a tree T = (V,E) to define an Euler tour, we make it a directed tree:
  - Define $T_e = (V_e, E_e)$ with $V_e = V$.
  - For each edge uv in E, add two directed edges (u,v) and (v,u) to E_e.
  - T_e has an Euler tour.

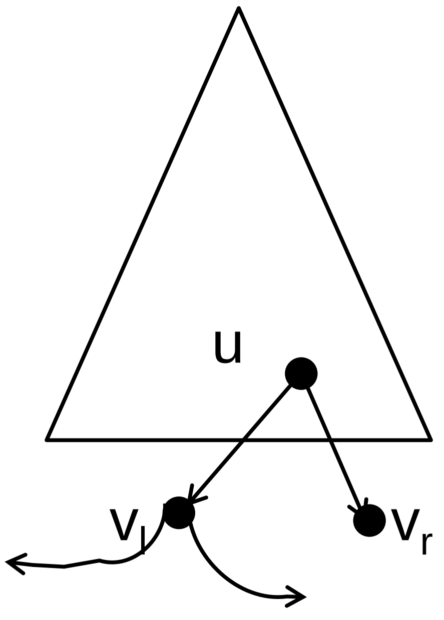
Defining an Euler Tour on a Tree

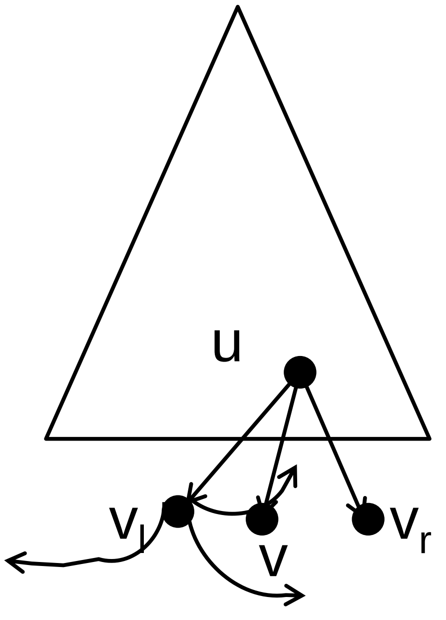
- Just have to define a successor. Here, successor for an edge.
- For a node u in T_e order its neighbors $v_1, v_2, \ldots, v_d$.
  - Can be done independently at each node.
  - For $e = (v_i, u)$, set $s(e) = e'$ where $e' = (u, v_{i+1})$.
    - Compute indices modulo the degree of u.

Euler Tour on a Tree

- Claim: The above definition of $s: E \rightarrow E$ is a tour.
- Proof: By induction. Let $n = 1$. Obviously true.
- For $n = 2$, at most one edge present. The tour is well defined according to s().
- Let the tour be well defined for $n = k$.
- Step: $n = k+1$.
  - Every tree has at least one leaf, say v.
  - $T' = T \setminus \{v\}$ has an Euler tour defined by $s':E(T') \rightarrow E(T')$ as $|V(T')| = k$.
  - We now extend this definition of s' to define a function $s: E(T) \rightarrow E(T)$.

Euler Tour on a Tree

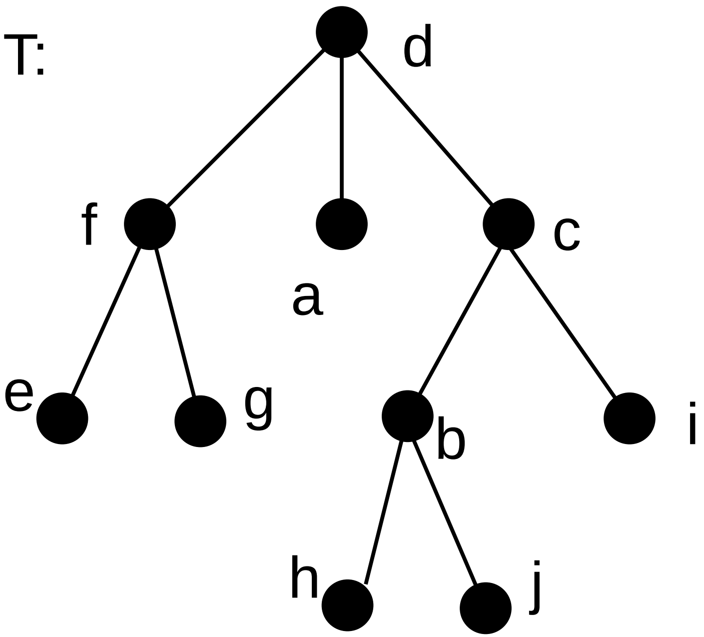




- Let u be the neighbor of v in T.
  - Let $N(u) = \{v_0, \ldots, v_{i-1}, v_i=v, v_{i+1}, \ldots v_d\}$.
- Set s(u, v) := (v,u). Set $s(v_{i-1},u) := (u,v)$.
- At all other edges e∈T, s(e):=s'(e).

Example

T:



$a \longrightarrow d$

$b \longrightarrow j \longrightarrow h \longrightarrow c$

$c \longrightarrow i \longrightarrow d \longrightarrow b$

$d \longrightarrow c \longrightarrow a \longrightarrow f$

$e \longrightarrow f$

$f \longrightarrow g \longrightarrow d$

$g \longrightarrow f$

$h \longrightarrow b$

$i \longrightarrow c$

$j \longrightarrow b$

Example

a → d

b → j → h → c

c → i → d → b

d → c → a → f

e → f

f → g → d

g → f

h → b

i → c

j → b

s(b,c) = (c,i)

s(c,i) = (i,c)

s(i,c) = (c,d)

s(c,d) = (d,a)

s(d,a) = (a,d)

s(a,d) = (d,f)

s(d,f) = (f,g)

s(f,g) = (g,f)

s(g,f) = (f,d)

s(f,d) = (d,c)

s(d,c) = (c,b)

s(c,b) = (b,j)

s(b,j) = (j,b)

s(j,b) = (b,h)

s(b,h) = (h,b)

s(h,b) = (b,c)

s(b,c) = (c,i)

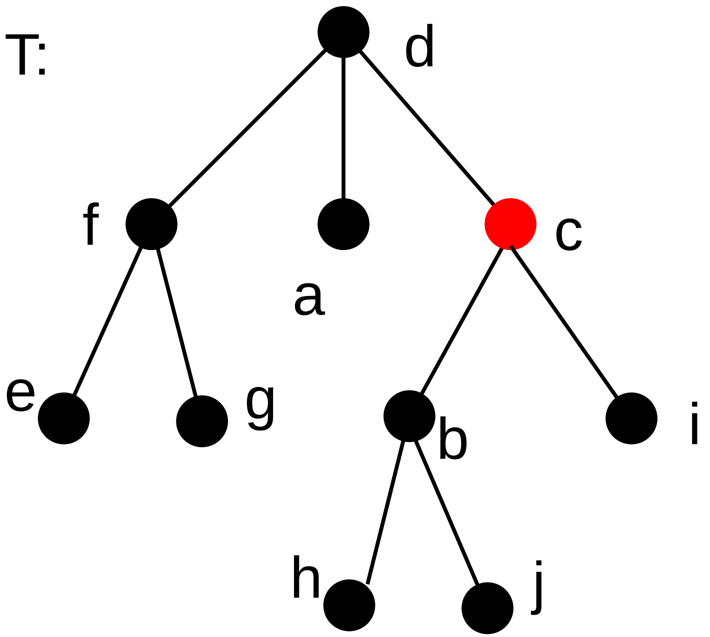
Euler Tour:

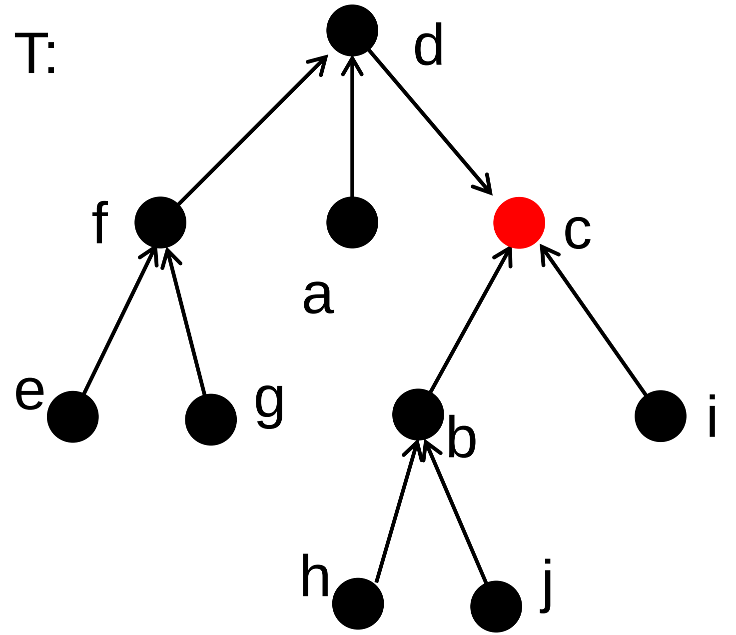
(b,c) → (c,i) → (i,c) → (c,d) → (d,a) → (a,d) → (d,f) → (f,g) → (g,f) → (f,d) → (d,c

Applications of Euler Tour to Traversal

- We now show why the Euler tour is an important construct for trees.
- Operations on a tree such as rooting, perorder and postorder traversal can be converted to routines on an Euler tour.

Rooting a Tree



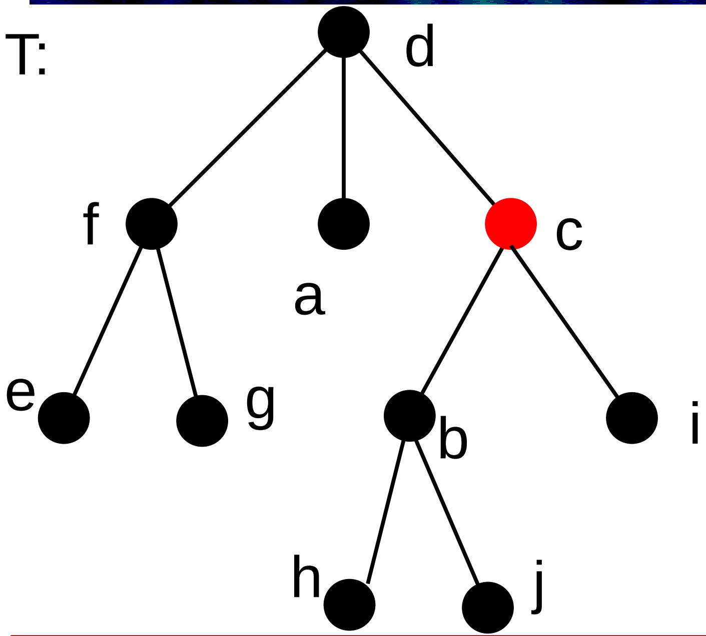


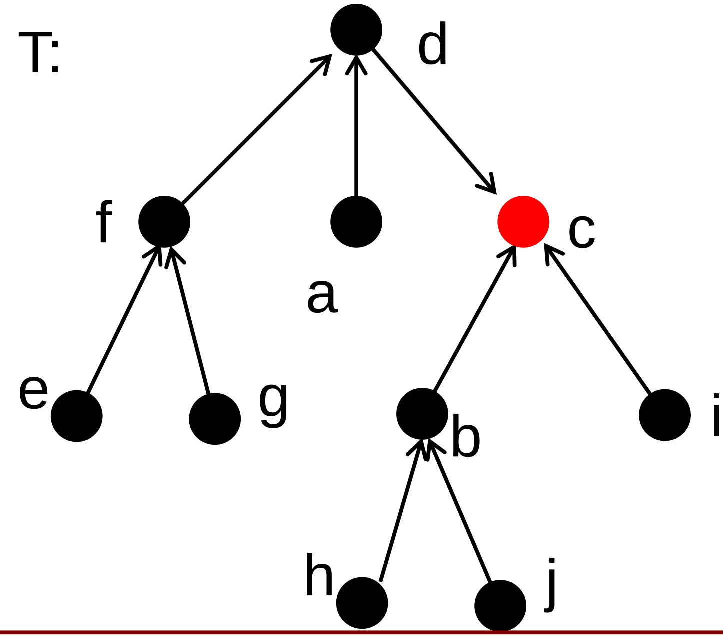
Euler Tour:

(b,c) → (c,i) → (i,c) → (c,d) → (d,a) → (a,d) → (d,f) → (f,g) → (g,f) → (f,d) → (d,

- Designate a node in a tree as the root.
- All edges are directed towards the root.

Rooting a Tree



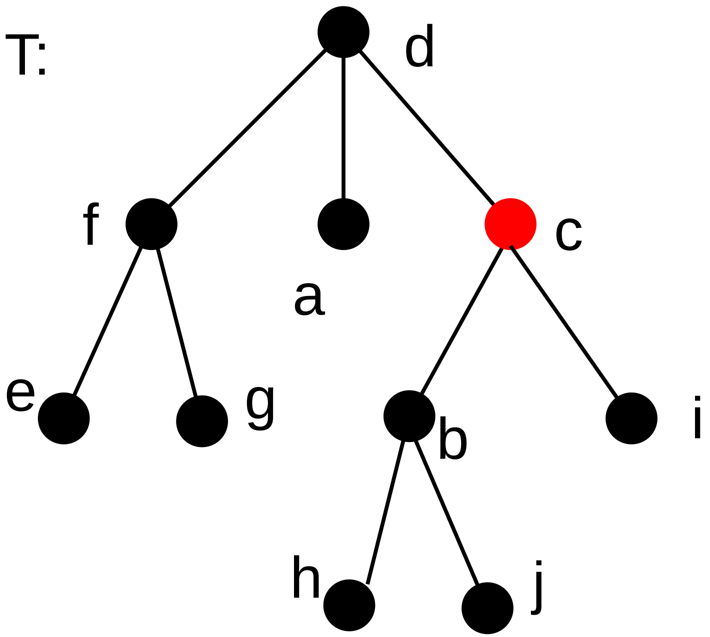


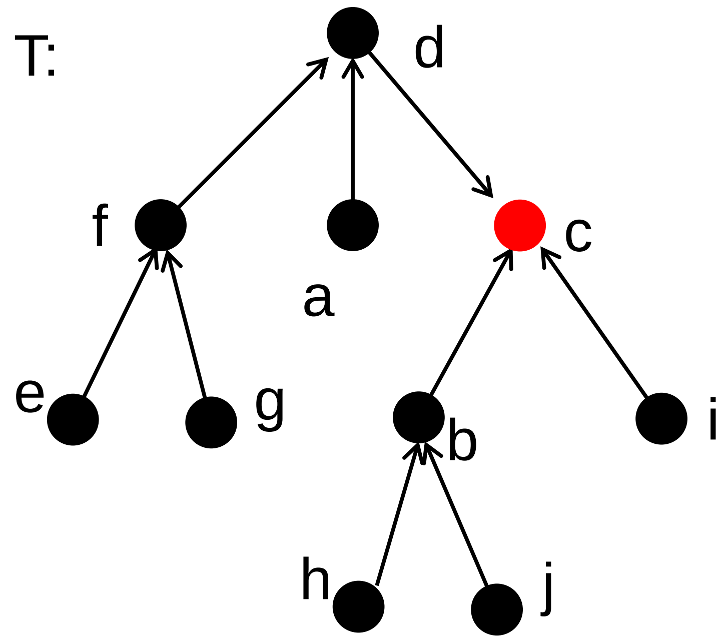
Euler Tour:

(b,c) → (c,i) → (i,c) → (c,d) → (d,a) → (a,d) → (d,f) → (f,g) → (g,f) → (f,d) → (d,

- Let (v_1, v_2, ..., v_d) be the neighbors of the root node r. In this case, say (i, d, b)
- Set s(e) = NULL for e = (v_d, r). In this case, s((b,c)).

Rooting a Tree



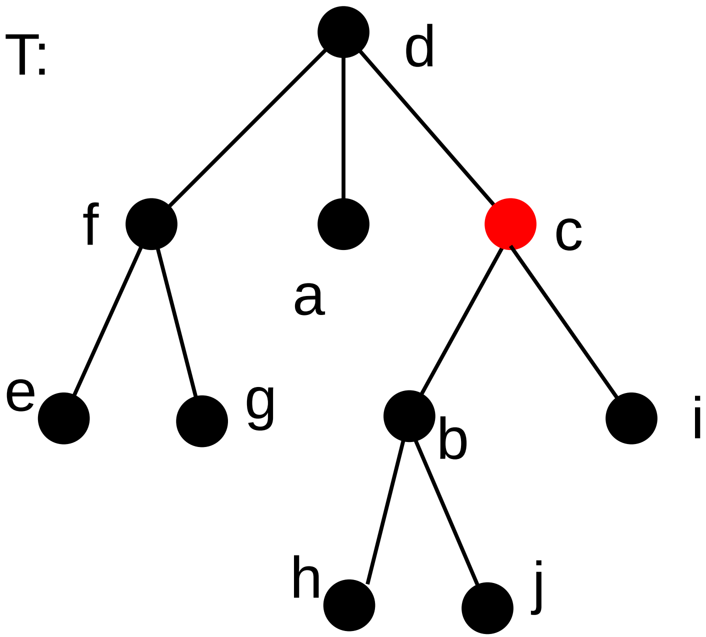


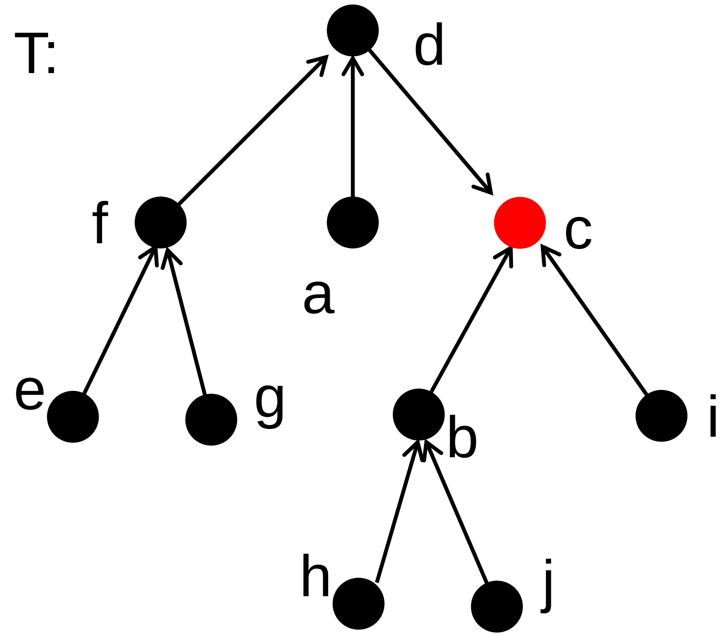
Euler Tour:

(c,i) → (i,c) → (c,d) → (d,a) → (a,d) → (d,f) → (f,g) → (g,f) → (f,d) → (d,c) → (c,b

- The edge (r,v_i) appears before (v_i,r).
- So, if u precedes v, then u = p(v). Orient the edge uv from v to u.

Rooting a Tree





Euler Tour:

(c,i) → (i,c) → (c,d) → (d,a) → (a,d) → (d,f) → (f,g) → (g,f) → (f,d) → (d,c) → (c,b

- So, if u precedes v, then u = p(v). Orient the edge uv from v to u.
- To know the above, use list ranking on the Euler path.

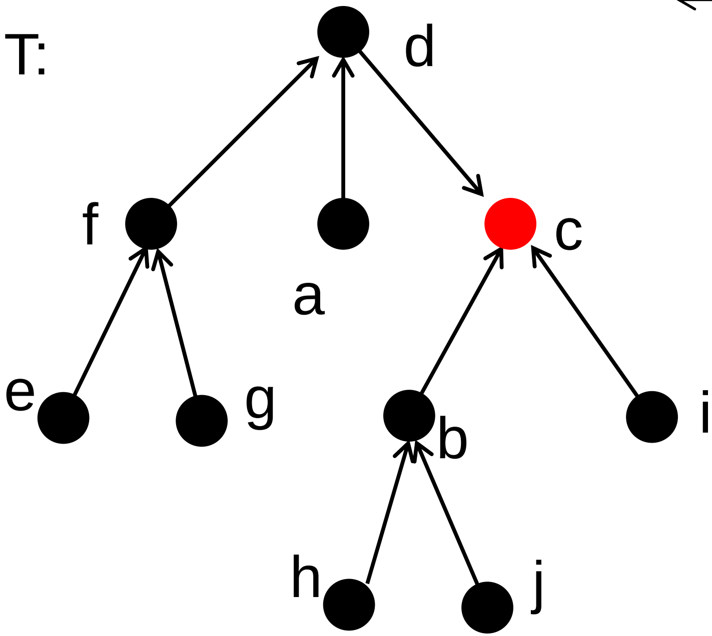
Preorder Traversal

ci-ic-cd-da-ad-df-fe-ef-fg-gf-fd-dc-cb-bj-jb-bh-hb-bc

Euler path Entering node d and its subtree

Euler path at node b and its s

T:



- Euler tour can be used to get a preoder number for every node.
- Associate meaning to the Euler tour.

Preorder Traversal

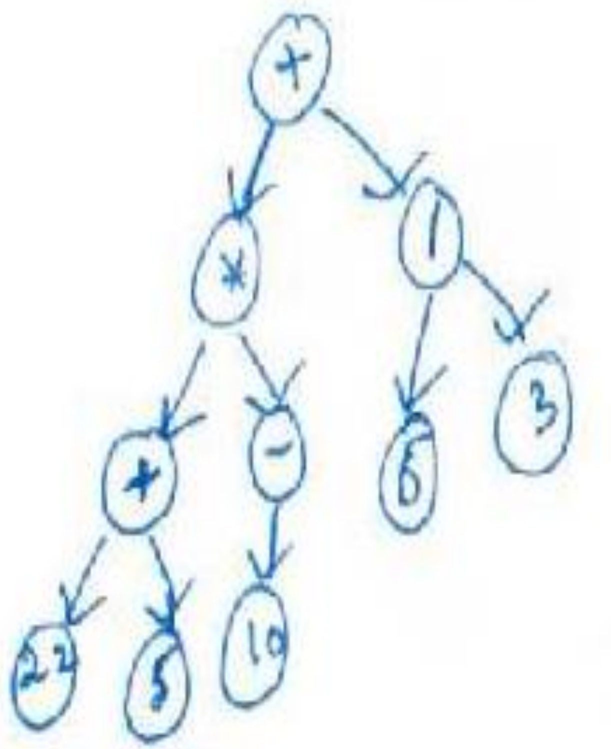
- In preorder traversal, a node is listed before any of the nodes in its subtree.
- In an Euler tour, nodes in a subtree are visited by entering those subtrees, and finally exiting to the parent.
- If we can therefore track the first occurrence of a node in the Euler path, then we can get the preorder traversal of the tree.

Postorder Traversal

- Similar rules can be designed for also postorder and inorder traversals.
- Inorder for binary trees only makes sense.
- Next we see how to process expression trees.

Expression Trees

- Expression trees are trees with operands at the leaf nodes, and operators at the internal nodes.
- Our interest is to evaluate the result of an expression represented by its expression tree.
- We would limit ourselves to binary operators.
  - Can also convert non-binary cases to the case of binary operators.
- However, the expression tree need not be balanced, or height in Q(log n).



Expression Tree Evaluation

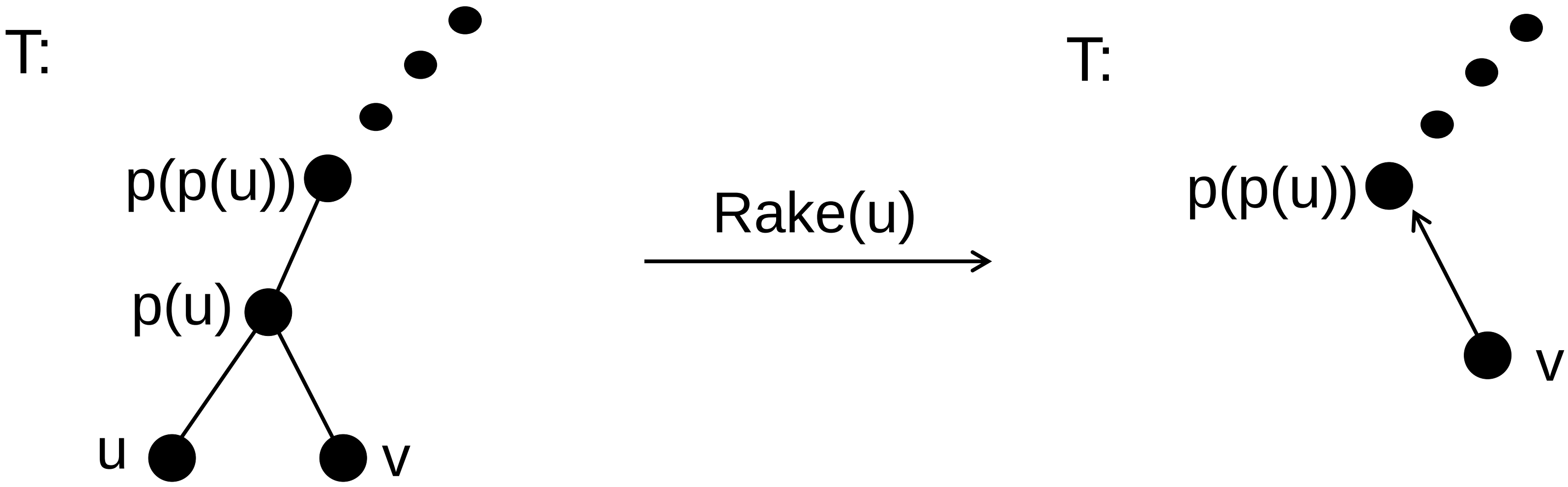
- Cannot directly apply the standard technique of
  - All the penultimate level nodes, then the next level etc.
    - Tree may not be balanced, and could have very few nodes at the penultimate level.
  - All leaves first cannot be processed at once
    - At an internal node, both operands may not be evaluated yet.

Two Steps

- A RAKE technique that contracts a tree
  - rake: 1.an agricultural implement with teeth or tines for gathering cut grass, hay, or the like or for smoothing the surface of the ground.

    2. any of various implements having a similar form, as a croupier's implement for gathering in money on a gaming table.
- Applying the rake technique to evaluate subexpressions.

The Rake Technique



- T = (V, E) be a rooted tree with r as the root and p() be the parent function
- One step of the rake operation at a leaf u with p(u) ≠ r involves:
  - Remove nodes u and p(u) from the tree.
  - Make the sibling of v as the child of p(p(u)).

The Rake Technique

- Why is this good?
- Can be applied simultaneously at several leaf nodes in parallel.
- Which ones?
  - All leaves which do not share the same parent, essentially non-siblings.

The Rake Technique

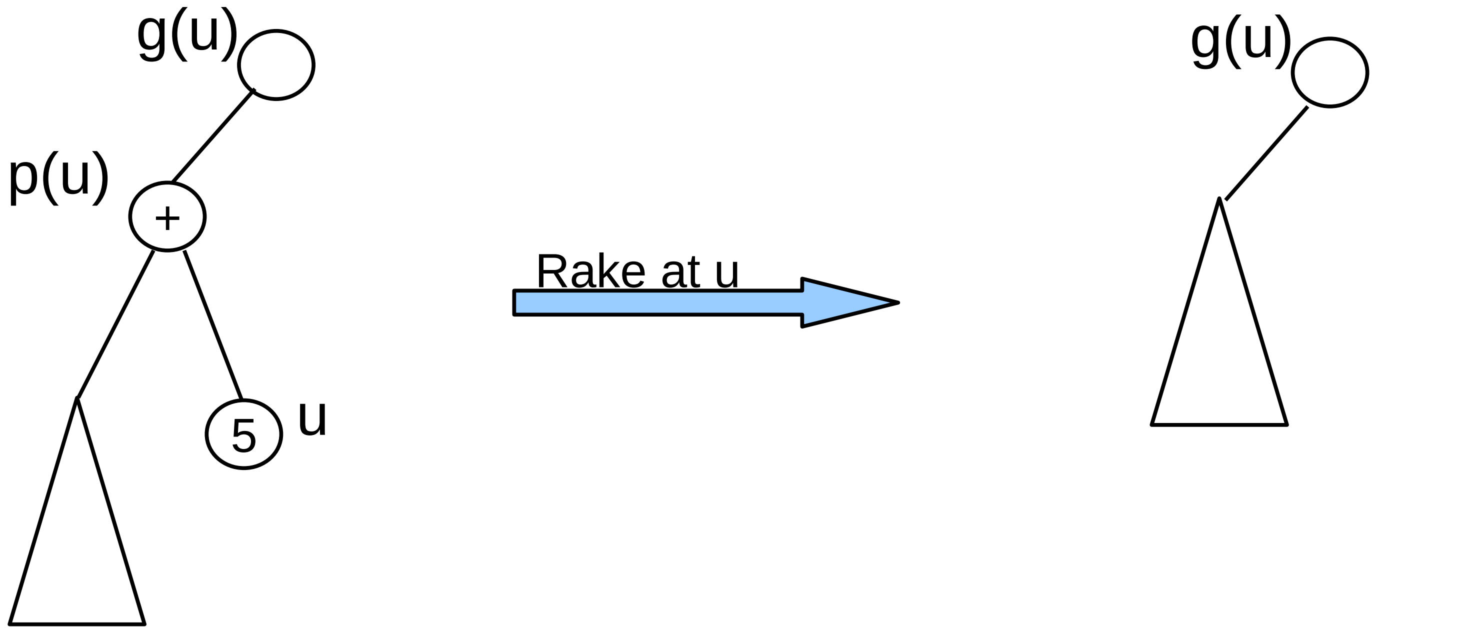
Algorithm ShrinkTree(T)
Step 1. Compute labels for the leaf nodes consecutively, excluding the leftmost and th
Step 2. for k iterations do
2.1 Apply the rake operation to all the odd numbered leaves that are lef
2.2 Apply the rake operation to all the odd
2.3 Update A to be the remaining (even) leaf nodes.
end-for
End Algorithm.

- So, we apply the Rake technique on all non-adjacent sibling nodes.
- The algorithm is as shown.

Time Analysis

- Observation: Each application of the rake at all the leaves as given in the algorithm reduces the number of leaves by a half.
- Each application of Rake at a leaf node is an O(1) operation.
- So the total time is O(log n).
- The number of operations is O(n).
  - Similar observations hold.

Applying Rake to Expression Evaluation

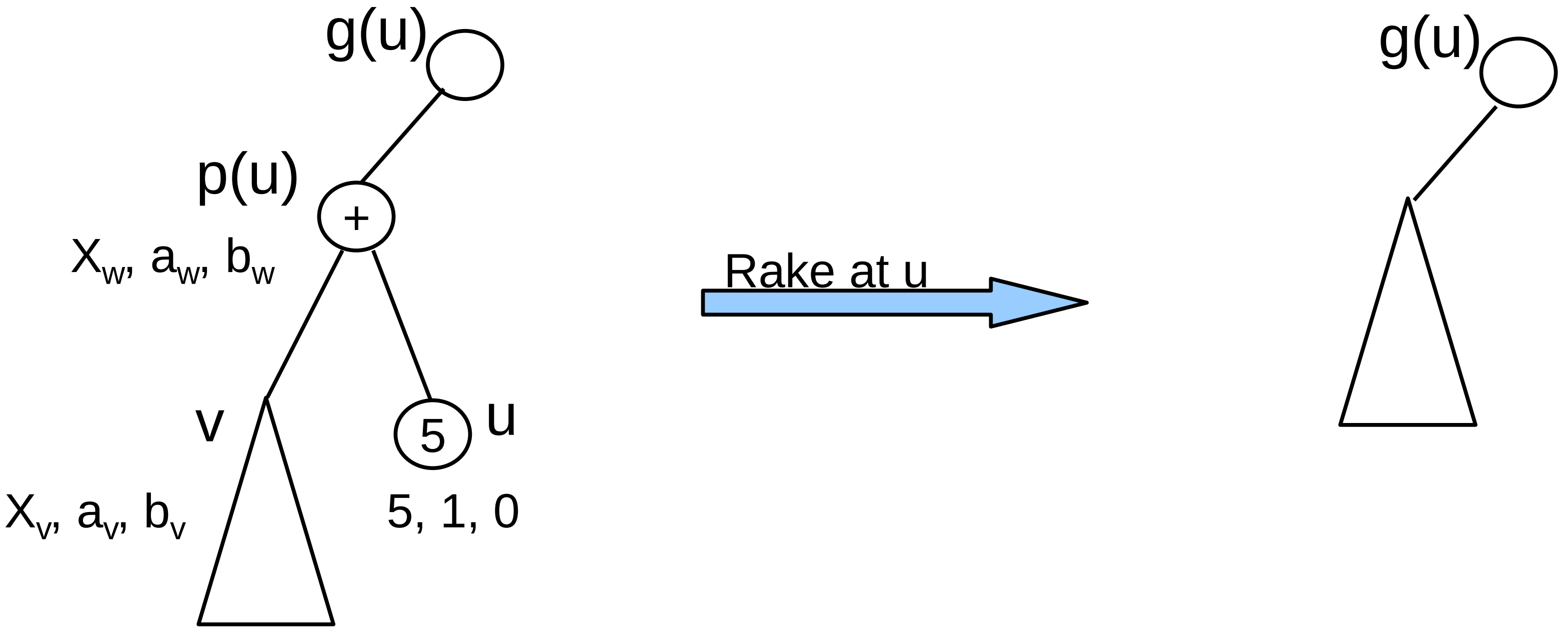


- Applying Rake means that we can process more than one leaf node at the same time.
- For expression evaluation, this may mean that an internal node with only one operand evaluated, also needs to be deleted.
    - Need to partially evaluate internal nodes.

Partial Evaluation

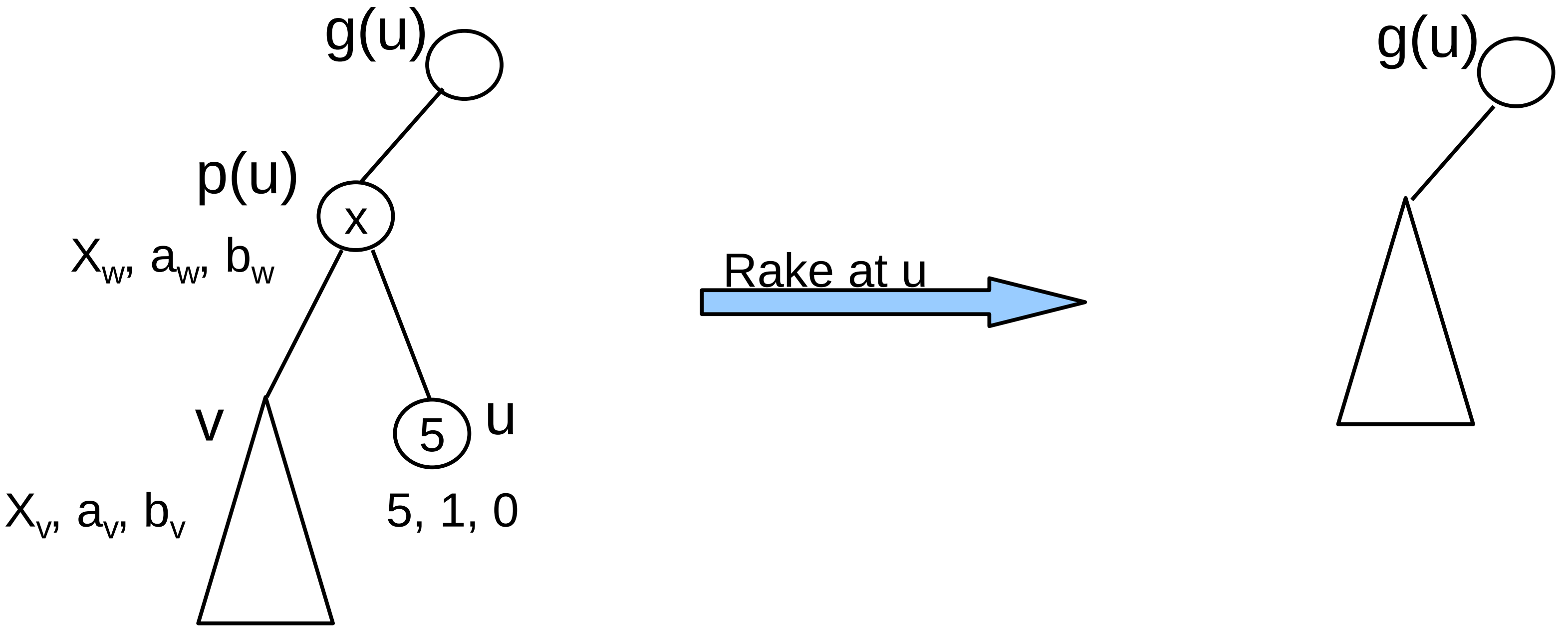
- Transfer the impact of applying the operator at p(u) to the sibling of u.
- Associate with each node u labels a_u and b_u so that $R_u = a_u X_u + b_u$.
  - X_u is the result of the subexpression, possibly unknown, at node u.
- Adjust the labels a_u and b_u during any rake operation appropriately.
- Initially, at each leaf node u, X_u equals the operand, $a_u = 1$, and $b_u = 0$.

Adjusting Labels



- Prior to rake at u, contribution of p(u) to g(u) is $a_w X_w + b_w$.
- $X_w = (a_u X_u + b_u) + (a_v X_v + b_v) = a_v X_v + (a_u X_u + b_u + b_v)$
- Therefore, adjust a_v and b_v as $a_w a_v$ and $a_w(a_u X_u + b_u + b_v)$.

Adjusting Labels



- Prior to rake at u, contribution of p(u) to g(u) is $a_w X_w + b_w$.
  - $X_w = (a_u X_u + b_u) \times (a_v X_v + b_v)$
- Therefore, adjust a_v and b_v as:
  - $a_v = a_w a_v (a_u X_u + b_u)$, $b_v = b_w + a_w b_v (a_u X_u + b_u)$.

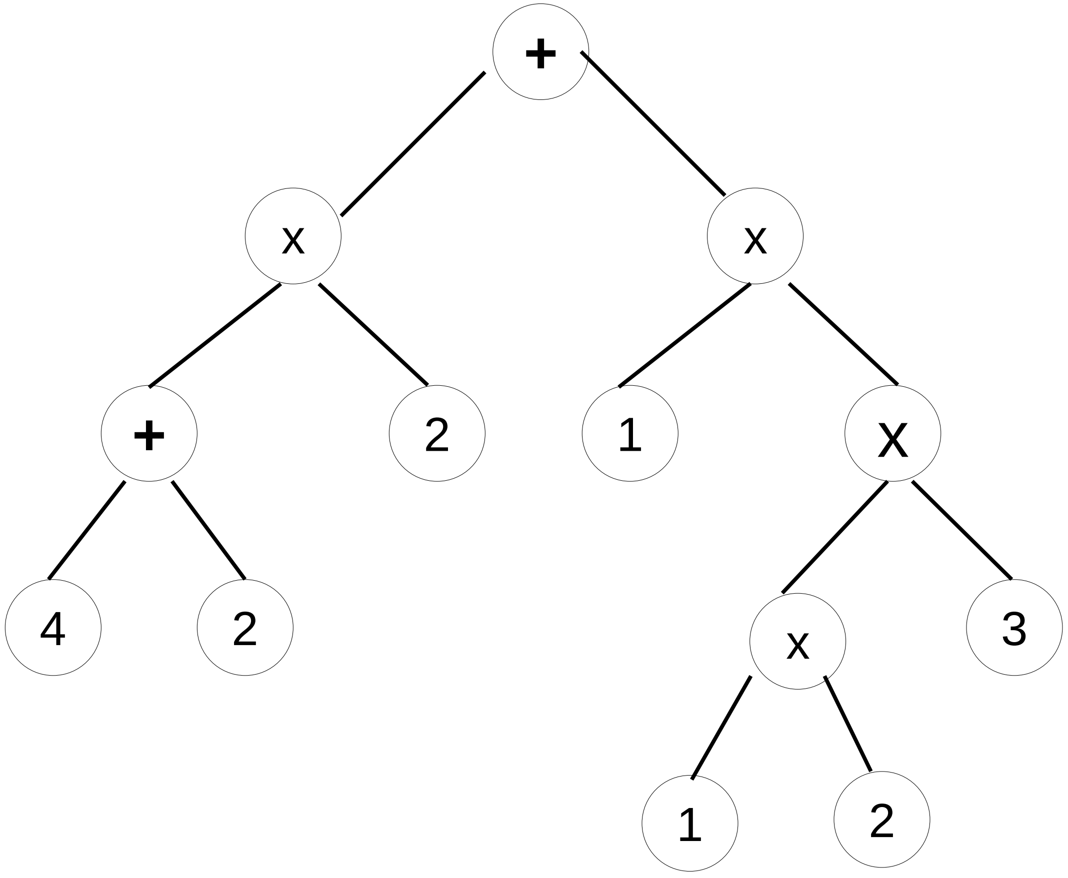
Adjusting Labels

- For other operators, proceed in a similar fashion.
- HW Problem.

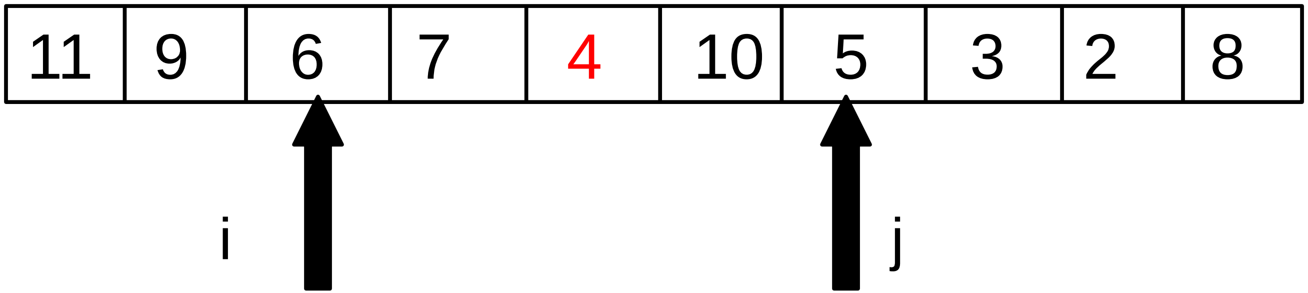
Expression Evaluation

- Parallel algorithm has the following main steps:
  - Rake the expression tree
  - Set up and adjust labels while raking
  - Stop when the tree has only three nodes, one operator and two operands as children.
  - Evaluate this three node tree.
- Theorem: Expression evaluation of an n-node expression tree can be done in parallel on an EREW PRAM using O(log n) time and O(n) operations.

Example

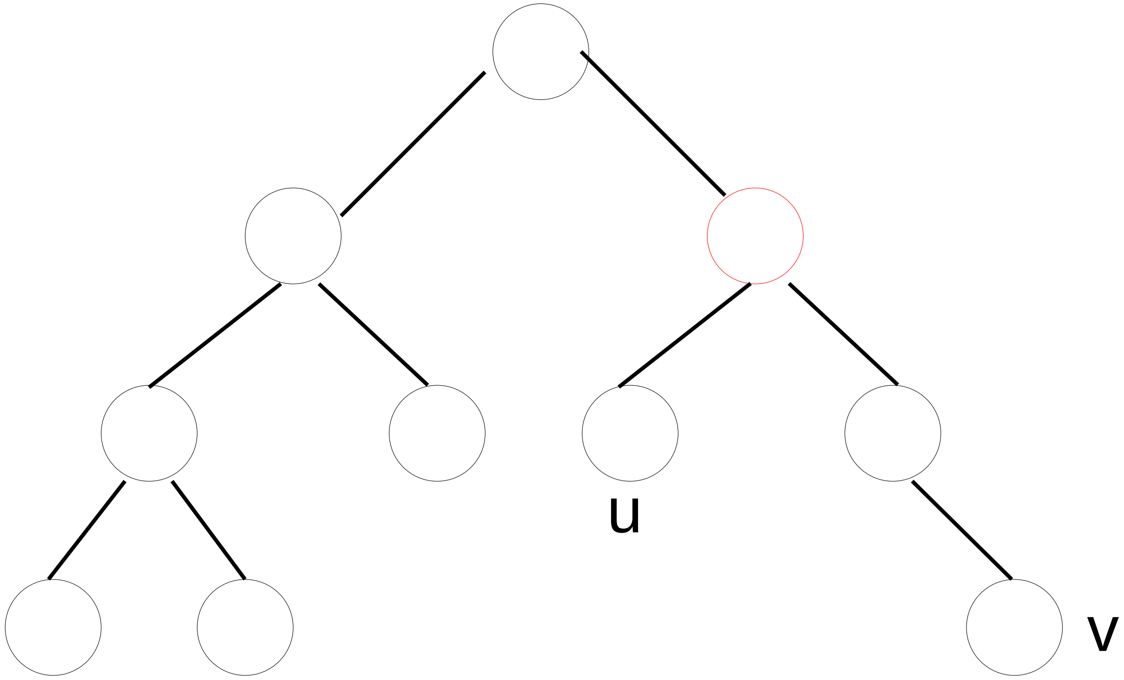


Range Minima



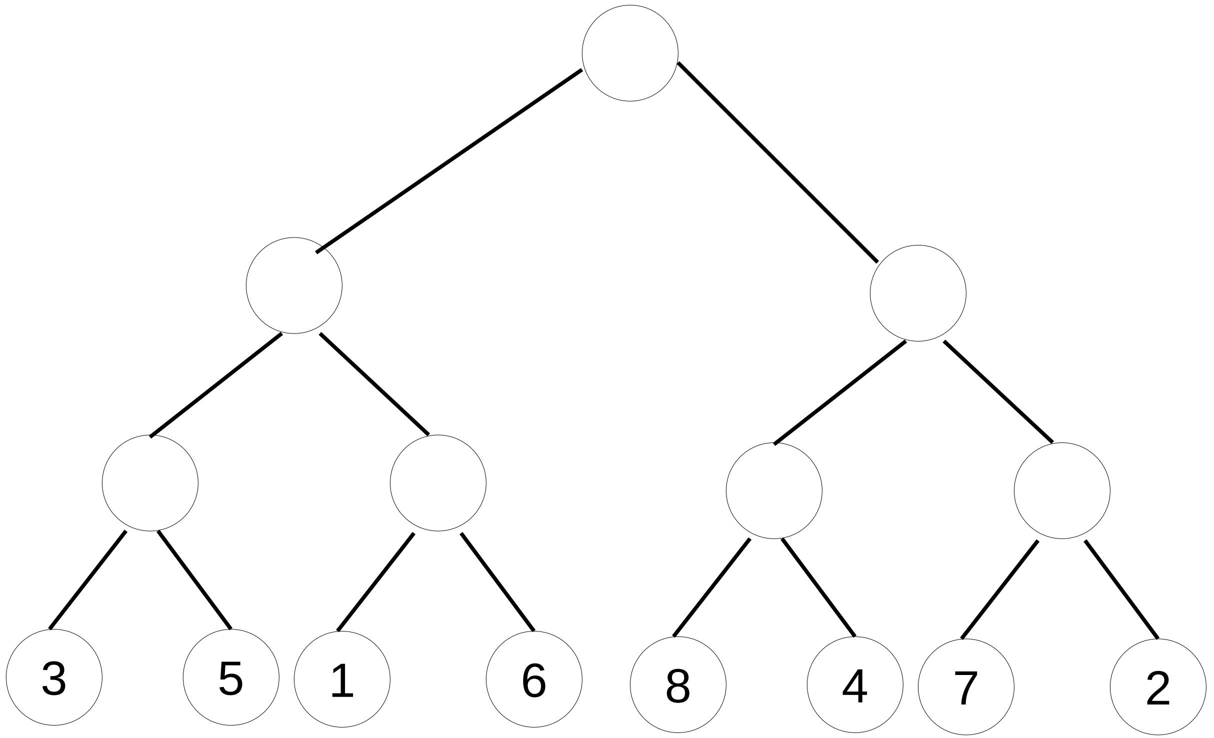
- Several geometric problems take the following flavor.
- Given a set of points S in a one dimensional space, preprocess S into a data structure D(S) so that:
  - Given a range of indices [i,j], report the element of S of least value between indices i and j

Applications of Range Minima



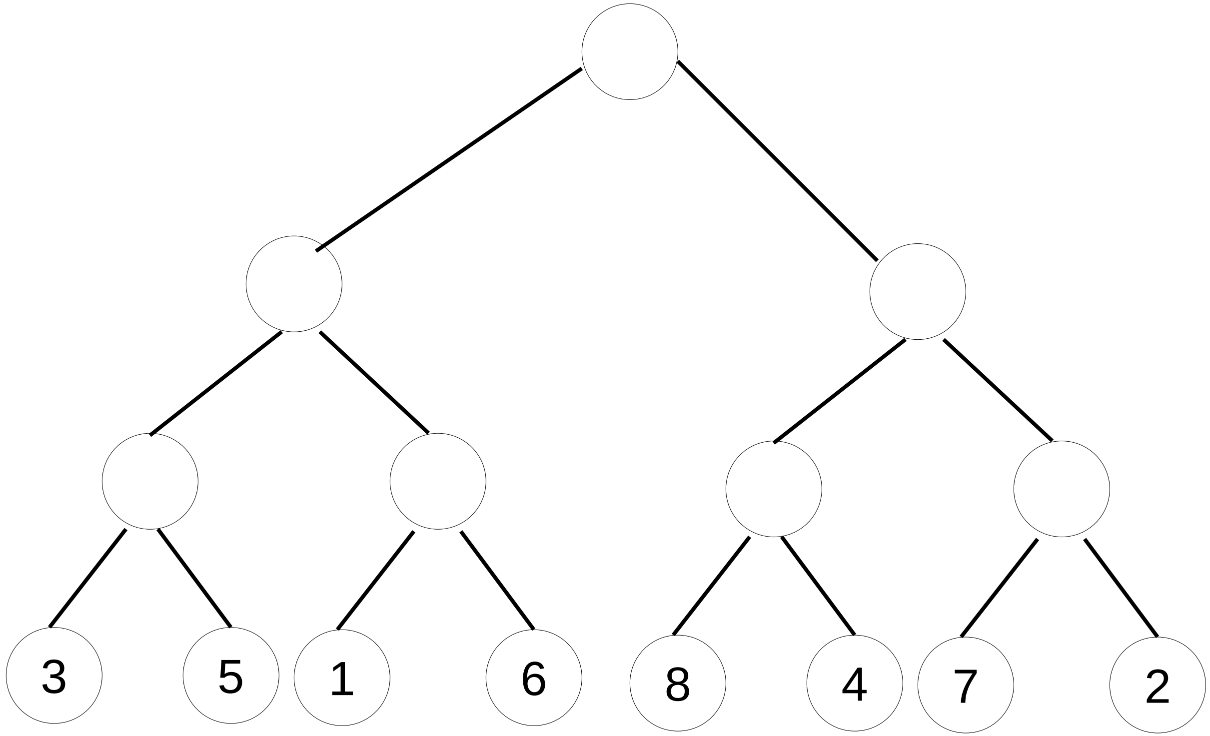
- Range minima can be used to solve the problem of finding the least common ancestor of two nodes in a tree.
  - Called as an LCA query, and is useful in several other settings.

Range Minima – Preprocessing



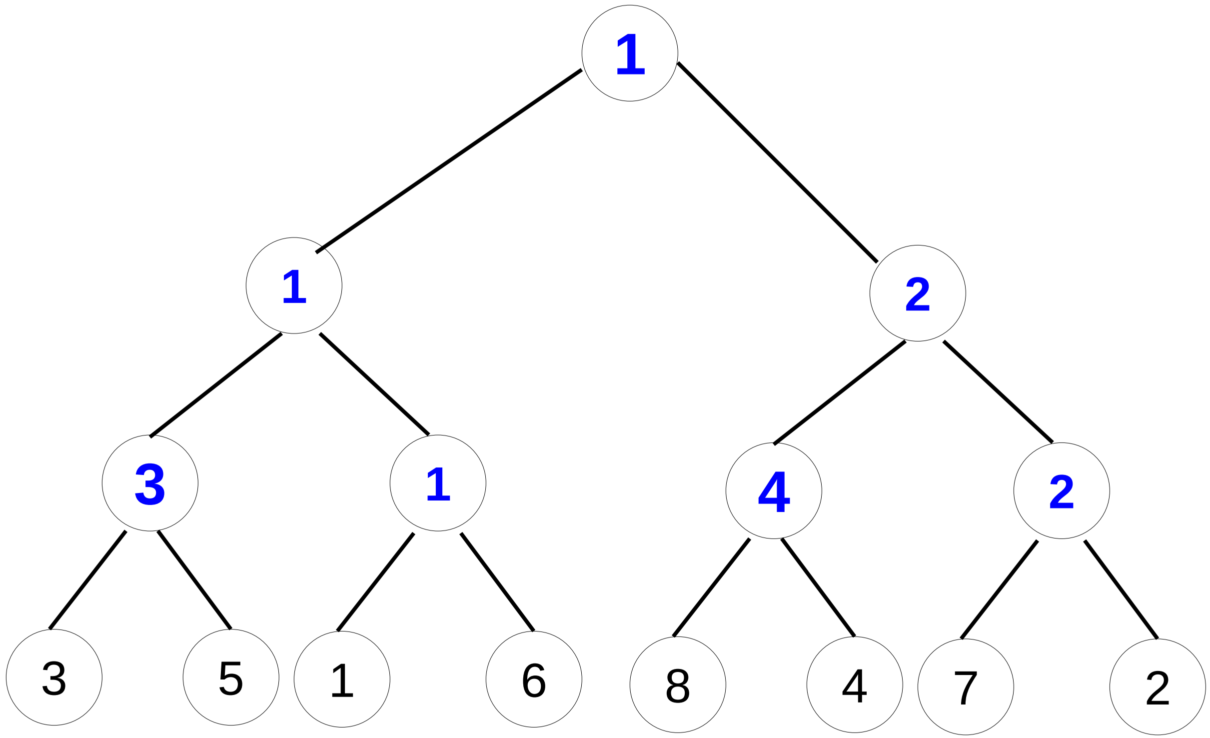
- Let n be the number of elements, and $n = 2^k$.
- Consider a full binary tree on the n elements.

Range Minima – Preprocessing



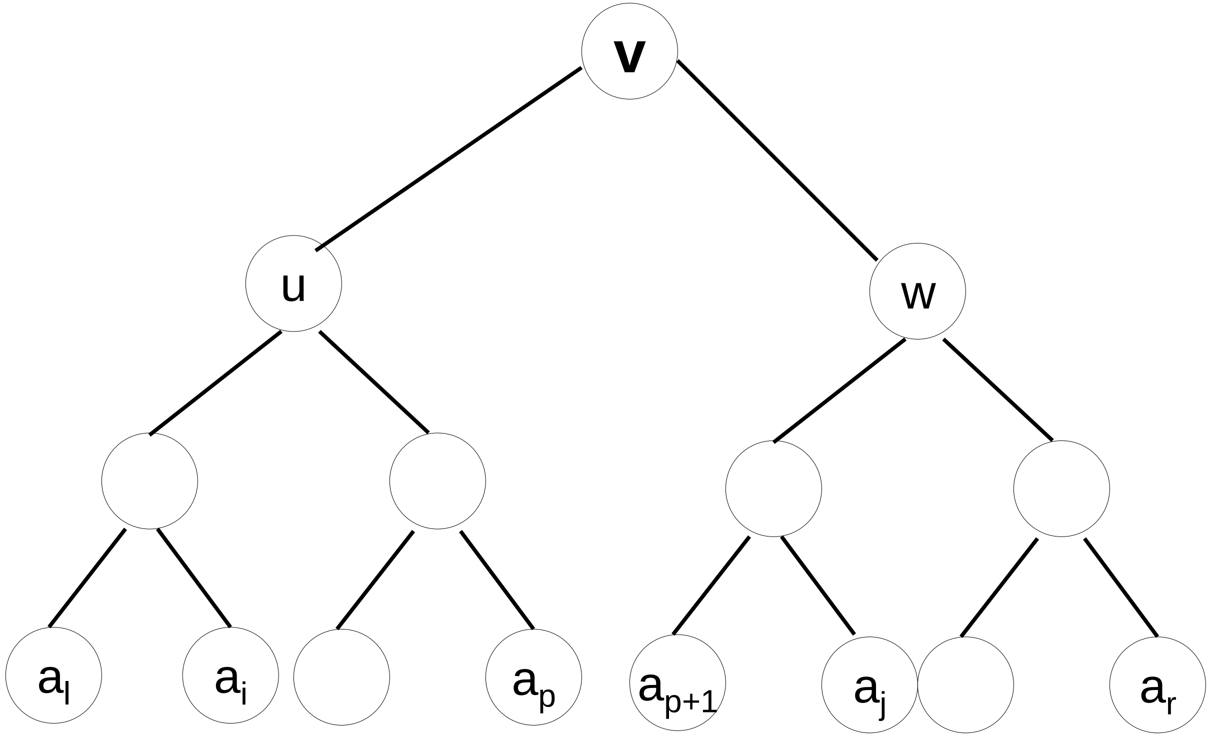
- Given two indices i and j, let v be the node in the tree that corresponds to the LCA of i and j.
  - ➢ No circular reasoning here. On a full binary tree, LCA is easy to find.

Range Minima – Preprocessing



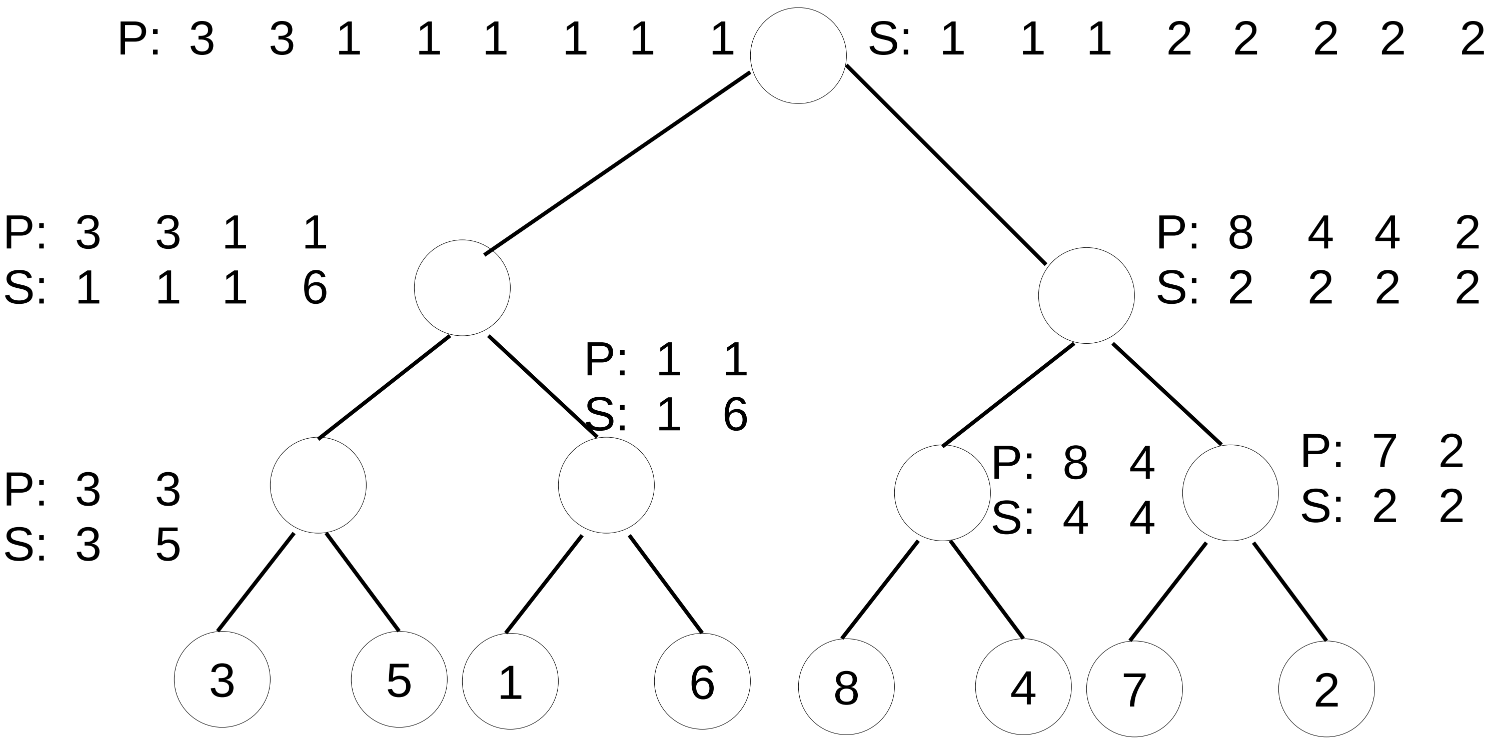
- Does not suffice if at every internal node, we just store the minima of the elements in that subtree.

Range Minima – Preprocessing



- Let u and w be the left and right child of v.
  - Also, let $A_v = (a_l, a_{l+1},\ldots,a_i, a_{i+1},\ldots a_j, a_{j+1},\ldots a_r)$.
- The required minima is the minimum of $\min\{a_i, a_{i+1},\ldots, a_p\}$ and $\min\{a_{p+1}, a_{p+2}, \ldots, a_j\}$.

Range Minima – Preprocessing

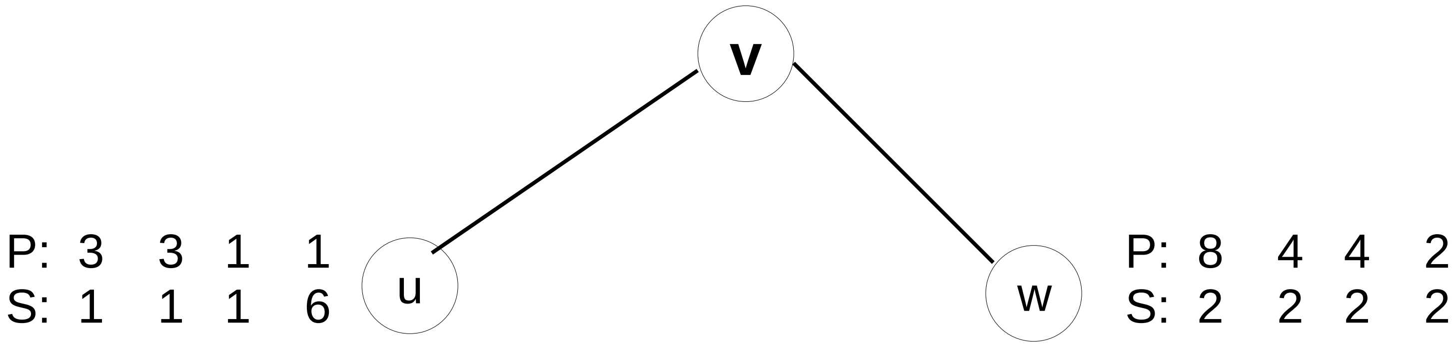


- For each node u, suppose we store the suffix and prefix minima of nodes in A_u.
- The required answer can be computed quickly.

Range Minima – Preprocessing

- For each node u, let P_u and S_u be the prefix and suffix minima arrays of elements in the subtree at node u.

Range Minima – Preprocessing



- Given a node v with children u and w, can actually compute P_v and S_v from P_u, P_w, and S_u, S_w respectively quickly.
  - How?

Range Minima – Preprocessing

- Given a node v with children u and w, can actually compute P_v and S_v from P_u, P_w, and S_u, S_w respectively quickly.
  - Let P_v be the concatenation of P_u and P_w.
  - The P_w part of P_v may change depending on the last element in P_u.
  - Similar rules apply for $S_v = S_u o\ S_w$

Range Minima – Preprocessing

- Theorem: Given n elements in an array A, the array can be preprocessed in O(log n) parallel time and O(nlog n) operations so that range minima queries can be answered in O(1) time.
  - Requires CREW model.
    - Where do we require concurrent read?
  - Can be made to use O(n) operations. Use standard technique 1.
  - On the CRCW model, can actually reduce the parallel time to O(log log n) in O(n) operations.

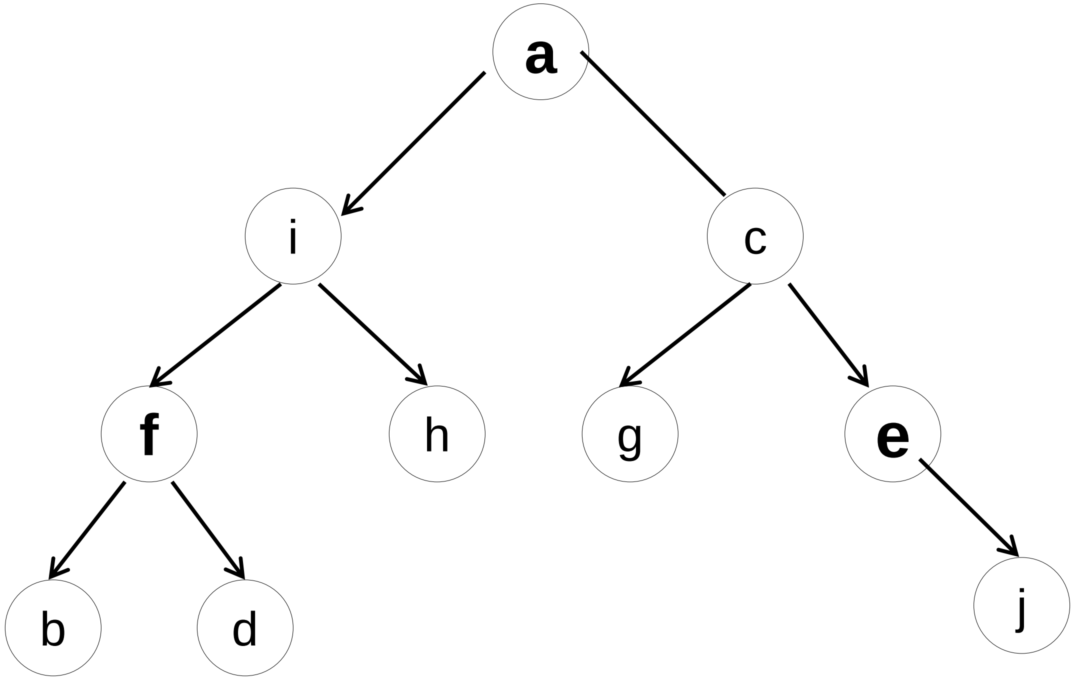
From Range Minima to LCA

- For a tree T rooted at r, let P be its Euler path with the edge (u,v) replaced by v.
- Compute the level of every node in the tree, with root at level 0.
  - Call this as the array Level[ ].
- Compute the leftmost and the rightmost occurrence of each node in P.
  - Call them as L(v) and R(v) for a node v.

From Range Minima to LCA

- Theorem: Let u and v be two vertices in T and L and R are given. Then,
  - $L(u) < L(v) < R(u)$ if and only if u is an ancestor of v.
  - u and v do not share an ancestor-descendant relationship iff $R(u) < L(v)$ or $R(v) < L(u)$.
  - If $R(u) < L(v)$ then the vertex with the minimum **Level** in the range $[R(u), L(v)]$ is the LCA of u and v.
    - Therefore preprocess the Level array for range minima.

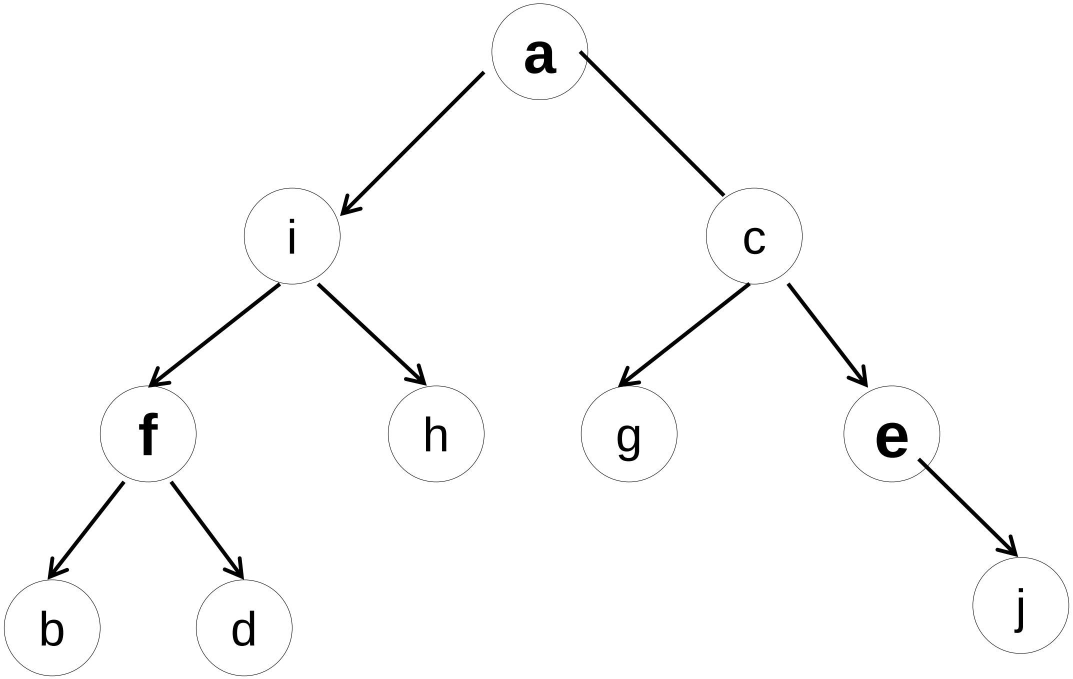
Example



(a,i) →(i,f) →(f,b)→ (b,f)→ (f,d)→ (d,f)→ (f,i)→ (i,h)→ (h,i)→ (i,a)→ (a,c)→ (c,g)→ (g,c)→ (c

P: i →f →b→ f→ d→ f→ i→ h→ i→ a→ c→ g→c→ e→ j→ e→ c→ a

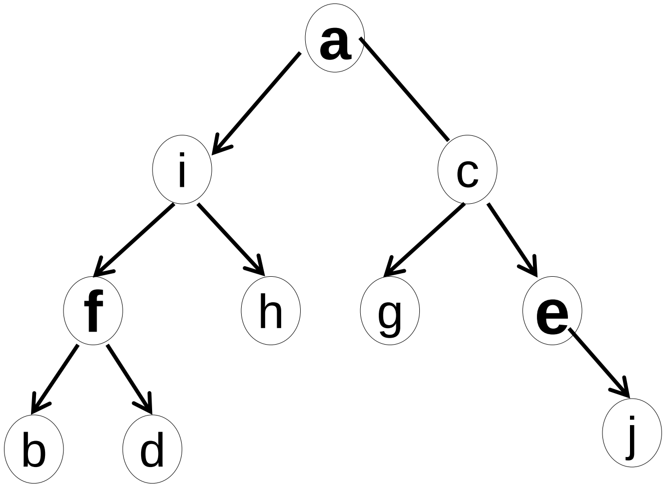
Example



P: a →i →f →b→ f→ d→ f→ i→ h→ i→ a→ c→ g→c→ e→ j→ e→ c→ a

Level **0 1 2 3 2 3 2 1 2 1 0 1 2 1 2 3 2 1 0**

Example



LCA of nodes g and j : R(g) = 12, L(j) = 15, $RM_{12,15}$(Level) = 1, LCA(g,j) = c

P: a → i → f → b → f → d → f → i → h → i → a → c → g → c → e → j → e → c → a

Level **0 1 2 3 2 3 2 1 2 1 0 1 2 1 2 3 2 1 0**

Graph Algorithms