Analysis of Algorithms
CS 477/677

Instructor: Monica Nicolescu

Lecture 27

Final Exam

- Tuesday, May 14, 12:45-2:45pm, in class
- Exam structure:
  - TRUE/FALSE questions
  - short questions on the topics discussed in class
  - homework-like problems
- Questions will be from the entire semester, with emphasis on material after midterms

General Advice for Study

- **Understand** how the algorithms are working
  - Work through the examples we did in class
  - “Narrate” for yourselves the main steps of the algorithms in a few sentences
- Know **when** or **for what problems** the algorithms are applicable
- **Do not memorize** algorithms

Topics Covered after Midterms

- Dynamic programming (recurrence only)
- Greedy algorithms (greedy choice + proof)
- Graph algorithms
  - Search (BFS, DFS)
  - Topological Sort
  - Minimum spanning trees
  - Shortest paths (single source)
- NP-completeness (general questions only)
- Definitions
  - Do not say "An X is **when** [story here]"
  - Do say: "An X is **a/an** [noun phrase here]"
- Material prior to midterms
  - True/false, short questions/mini-problems, extra-credit/grad

Dynamic Programming

- Used for **optimization problems**
  - A set of choices must be made to get an optimal solution
  - Find a solution with the optimal value (minimum or maximum)
- Applicability:
  - Subproblems are not independent, i.e., subproblems share subsubproblems
  - A divide-and-conquer approach would repeatedly solve the common subproblems
  - Dynamic programming solves every subproblem just once and stores the answer in a table

Elements of Dynamic Programming

- Optimal Substructure
  - An optimal solution to a problem contains within it an optimal solution to subproblems
  - Optimal solution to the entire problem is build in a bottom-up manner from optimal solutions to subproblems
- Overlapping Subproblems
  - If a recursive algorithm revisits the same subproblems again and again ⇒ the problem has overlapping subproblems

Exercise

Give an $O(n^2)$ algorithm to find the longest montonically increasing sequence in a sequence of n numbers.

- Take an example: (5, 2, 8, 7, 3, 1, 6, 4)
- Define: s_i = the length of the longest sequence ending with the i-th character

	5	2	8	7	3	1	6	4
$s_0 = 0$	1	1	2	2	2	1	3	3

$$s_i = \max_{0<j<i,\ seq[j] < seq[i]} \{s_j\} + 1$$

s_i = 1 more than the greatest value for a previous number that is smaller than seq[i]

Greedy Algorithms

- Similar to dynamic programming, but simpler approach
  - Also used for optimization problems
- **Idea:**
  - When we have a choice to make, make the one that looks best right now in hope of getting a globally optimal solution
- **Problems:**
  - Greedy algorithms don’t always yield an optimal solution
- When the problem has certain general characteristics, greedy algorithms give optimal solutions

Correctness of Greedy Algorithms

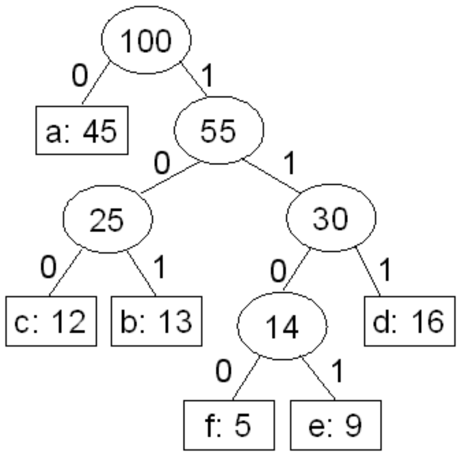
- Greedy Choice Property
  - A globally optimal solution can be arrived at by making a locally optimal (greedy) choice
- Optimal Substructure Property
  - Optimal solution to subproblem + greedy choice $\Rightarrow$ optimal solution for the original problem

Dynamic Programming vs. Greedy Algorithms

- Dynamic programming
  - We make a choice at each step
  - The choice depends on solutions to subproblems
  - Bottom up solution, from smaller to larger subproblems
- Greedy algorithm
  - Make the greedy choice and THEN
  - Solve the subproblem arising after the choice is made
  - The choice we make may depend on previous choices, but not on solutions to subproblems
  - Top down solution, problems decrease in size

Huffman Codes

- Technique for data compression
- **Idea**:
  - Represent each character as a binary string
  - Variable length code: assign short codewords to frequent characters
  - Represent the codes as full binary trees whose leaves are the given characters



$$B(T) = \sum_{c \in C} f(c) d_T(c)$$

- Constructing an optimal prefix code (Huffmann code)
  - Start with a set of |C| leaves
  - At each step, merge the two least frequent objects: the frequency of the new node = sum of two frequencies

Adj. List - Adj. Matrix Comparison

Graph representation: adjacency list, adjacency matrix

Comparison	Better
Faster to test if (x, y) exists?	matrices
Faster to find vertex degree?	lists
Less memory on sparse graphs?	lists ($m+n$) *vs.* n^2
Faster to traverse the graph?	lists ($m+n$) *vs.* n^2

Adjacency list representation is better for most applications

BFS vs. DFS

BFS

- **Input**:
  - A graph G = (V, E) (directed or undirected)
  - A **source** vertex $s \in V$
- **Idea**:
  - Explore the edges of G to "discover" every vertex reachable from s, taking the ones closest to s first
- **Output**:
  - d[v] = distance (smallest # of edges, or shortest path) from s to v, for all $v \in V$
  - BFS tree

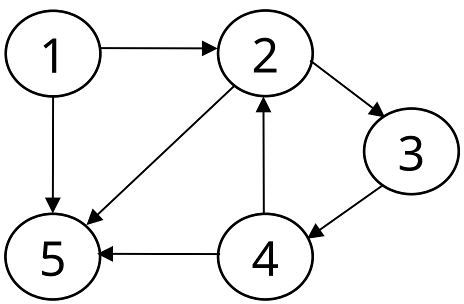
DFS

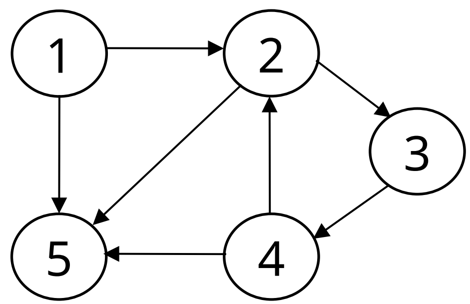
- **Input:**
  - A Graph G = (V, E) (directed or undirected)
  - No source vertex given!
- **Idea**:
  - Explore the edges of G to "discover" every vertex in V starting at the most recently visited node
  - Search may be repeated from multiple sources
- **Output:**
  - 2 **timestamps** on each vertex: d[v], f[d]
  - DFS forest

Example - Question

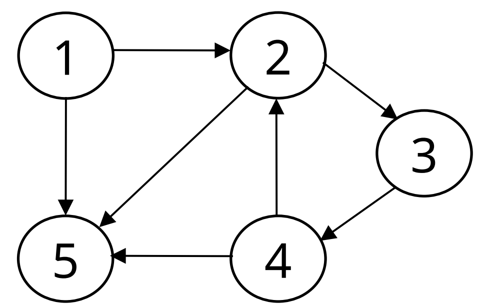
- G = (V, E). True or false?

  All DFS forests (for traversal starting at different vertices) will have the same number of trees.





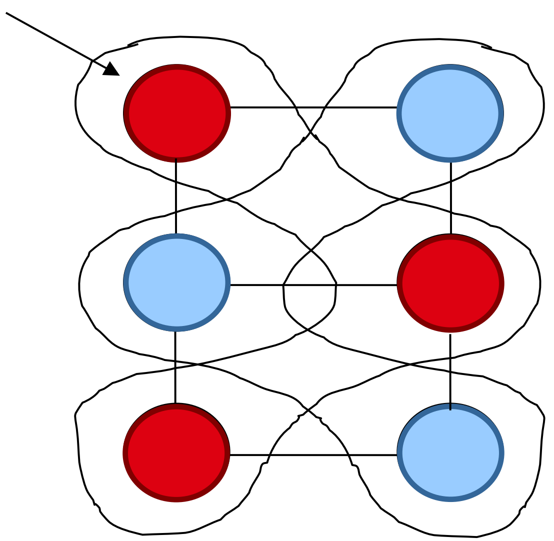
{1, 2, 5}, {3, 4}



{2, 5}, {1}, {4}, {3}

Exercise

- Show how you can detect whether a graph is bipartite.
- Idea:
  - Nodes in the same set will never be adjacent to each other

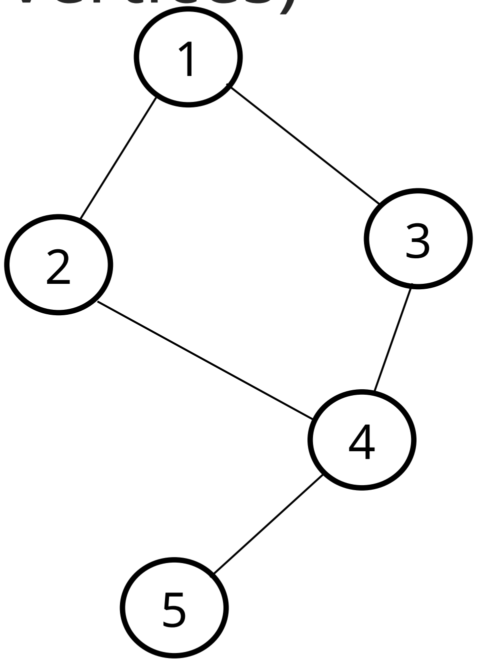


While doing a BFS, color nodes in two colors (red, blue), always alternating from a parent to a child

The graph is bipartite if all adjacent nodes have different colors

Exercise

- Give an O(n) algorithm to test if an **undirected graph** has a cycle (n = # of vertices)



- **Idea:**
  - The graph has a cycle if it has more than n – 1 edges
  - Checking this takes O(n)

Sample Problem

- Give an $\Theta(n+m)$ algorithm that tests whether an **undirected graph** is connected. The graph is given in adjacency list representation and has n vertices and m edges.
  - Run DFS using any vertex as a start vertex and then check whether all nodes were visited.
  - The graph is connected if and only if all nodes were visited, that is, if and only if all nodes are reachable from the start vertex.

Properties of DFS

- Descendant – ancestor relationships
  - Relations on discovery – finish times for descendant, parent (ancestor) nodes
- Parenthesis Theorem
- White Path Theorem

Review these properties!

Topological Sort

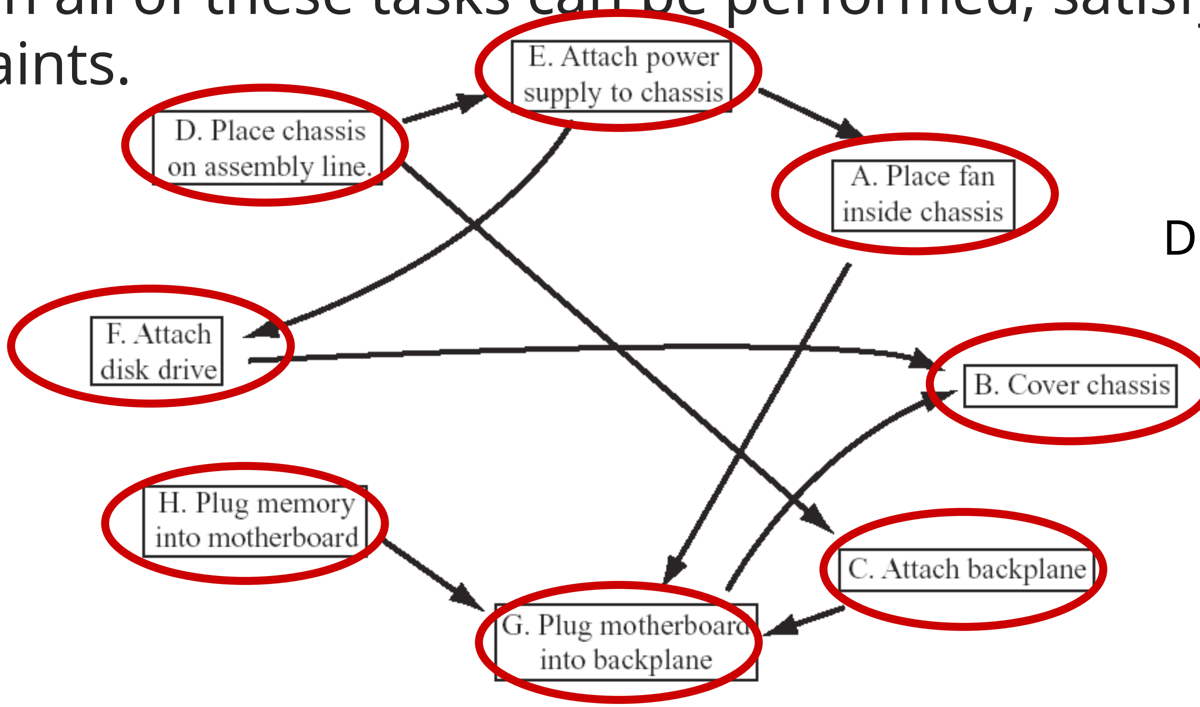
Topological sort of a directed acyclic graph G = (V, E): a linear order of vertices such that if there exists an edge (u, v), then u appears before v in the ordering.

TOPOLOGICAL-SORT(V, E)

1. Call DFS(V, E) to compute finishing times f[v] for each vertex v
2. When each vertex is finished, insert it onto the front of a linked list
3. Return the linked list of vertices

Sample Problem

- In the following graph, boxes represent tasks that must be performed in the assembly of a computer, and arrows represent constraints that one task must be performed before another (for instance, the disk drive must be attached before the chassis can be covered). Write down a sequence in which all of these tasks can be performed, satisfying all constraints.

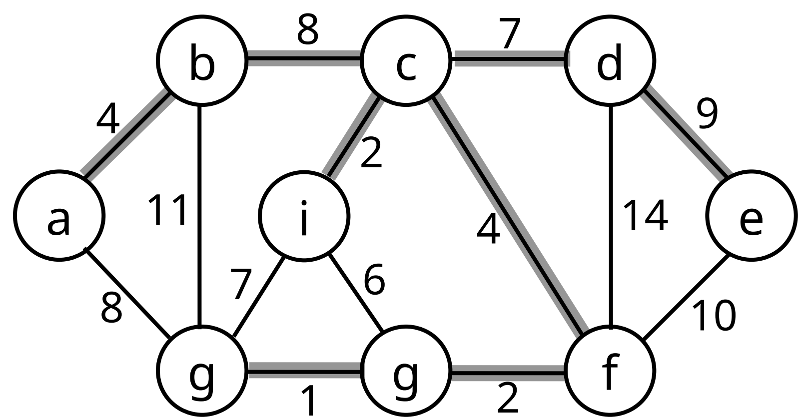


D E A F H C G B

Minimum Spanning Trees

Given:

- A connected, undirected, weighted graph G = (V, E)

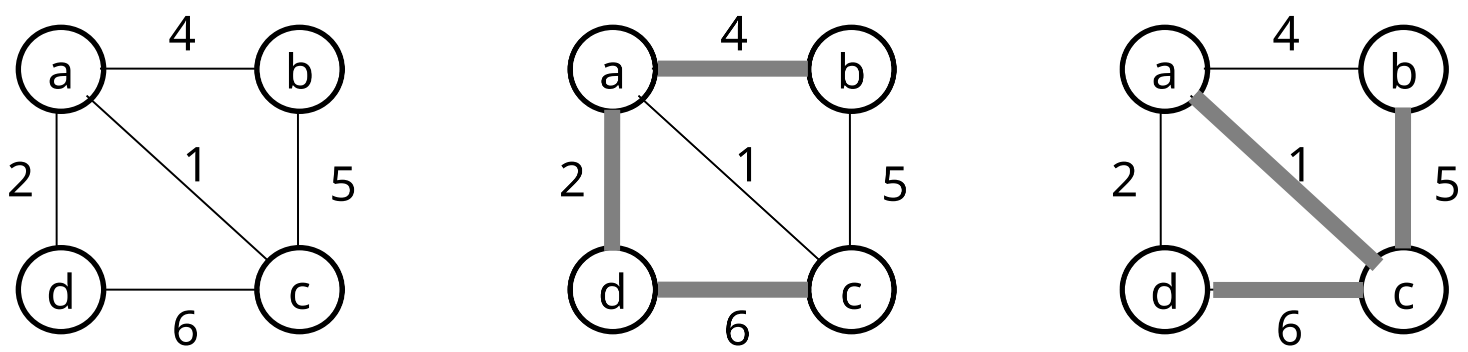


A minimum spanning tree:

1. T connects all vertices
2. $w(T) = \Sigma_{(u,v)\in T}\, w(u, v)$ is minimized

Exercise

- True or False?
  - If all the weights of a connected, weighted graph are distinct, then distinct spanning trees of G have distinct weights



Cost 12

Cost 12

Minimum Spanning Trees

- Kruskal's algorithm
  - Start with each vertex being its own component
  - Repeatedly merge two components into one by choosing the light edge (minimum cost) that connects them
  - During the algorithm the MST is a forest of trees
- Prim's algorithm
  - The edges added to the MST always form a single tree
  - Repeatedly add light edges that connect vertices from outside the MST to vertices in the MST

Variants of Shortest Paths

- **Single-source shortest path**
  - G = (V, E) ⇒ find a shortest path from a given source vertex s to each vertex v ∈ V
  - Belmann-Ford algorithm
  - Single-source shortest paths in acyclic graphs
  - Dijkstra's algorithm
- Know when the algorithms are applicable and how they work

Relaxation

- **Relaxing** an edge (u, v) = testing whether we can improve the shortest path to v found so far by going through u

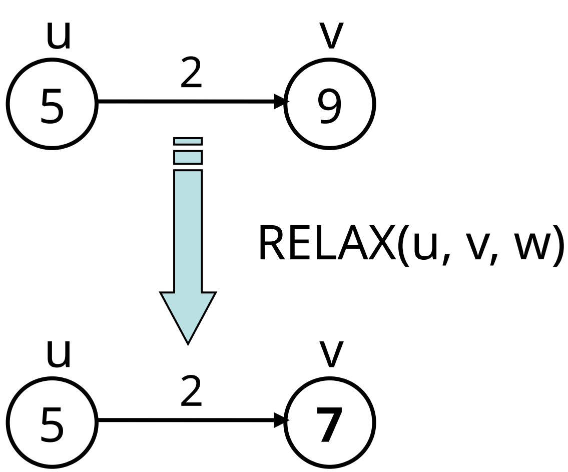
    If d[v] > d[u] + w(u, v)

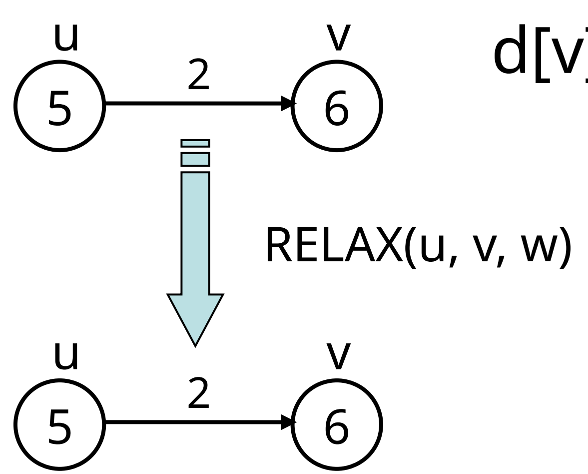
    we can improve the shortest path to v

    ⇒ update d[v] and π[v]

After relaxation:

d[v] ≤ d[u] + w(u, v)



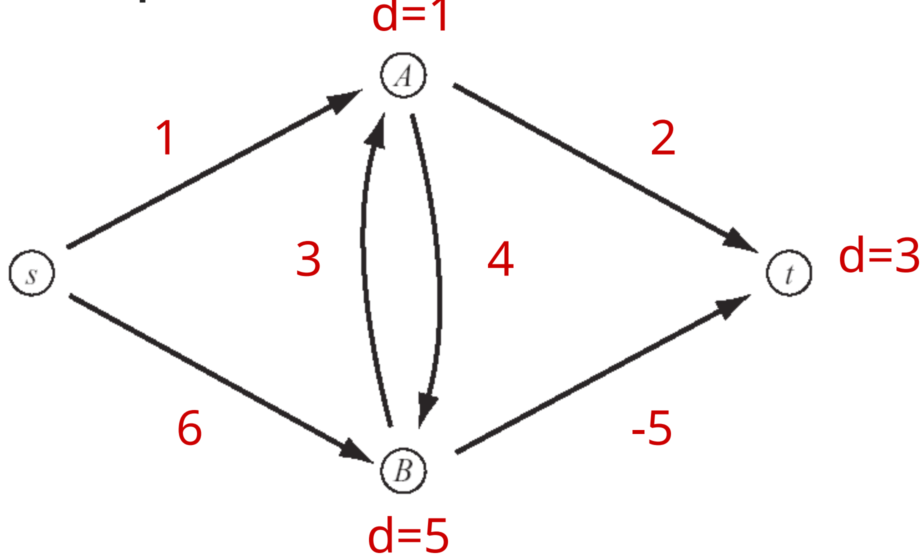


Single Source Shortest Paths

- Bellman-Ford Algorithm
  - Allows negative edge weights
  - TRUE if no negative-weight cycles are reachable from the source s and FALSE otherwise
  - Traverse all the edges |V – 1| times, every time performing a relaxation step of each edge
- Single-Source Shortest Paths in DAGs
  - Topologically sort the vertices of the graph
  - Relax the edges according to the order given by the topological sort
- Dijkstra’s Algorithm
  - No negative-weight edges
  - Repeatedly select a vertex with the minimum shortest-path estimate d[v] – uses a queue, in which keys are d[v]

Sample Problem

(a) Write down lengths for the edges of the following graph, so that Dijkstra's algorithm would not find the correct shortest path from *s* to *t*.



Dijkstra's algorithm will visit the vertices in the order s – A – t

d[t] = 3 and d[B] = 5

However, the path s-A-B-t is the shortest.

Sample Problem

(b) Which of the shortest path algorithms described in class would be most appropriate for finding paths in the graph of part (a) with the weights you gave? Explain your answer.

- Bellman-Ford, because it can handle graphs with negative edge weights and cycles.
- We can't use the DAG algorithm because this graph is not a DAG.

True/False Questions

(a) True False

- Merge sort, Quicksort and Insertion sort are comparison-based sorting algorithms.

(b) True False

- In a red-black tree, if a node is black than both its children are red.

(c) True False

- A reverse-sorted array (i.e., decreasing order) is always a max-heap.

True/False Questions

(d) True False

- In a directed graph with positive edge weights, the edge with minimum weight belongs to the shortest paths tree for any source vertex.

(e) True False

- Given any weighted directed graph with all distinct edge weights and any specified source vertex, the shortest paths tree is unique.

True/False Questions

(f) True False

- After running DFS in a graph G = (V, E), a vertex v ($\in$ V) is a proper descendant of another vertex u ($\in$ V) $\Longleftrightarrow$ $d[v] < d[u] < f[u] < f[v]$

(g) True False

- The problem of determining an optimal order for multiplying a chain of matrices can be solved by a greedy algorithm, since it displays the optimal substructure and overlapping subproblems properties.

True/False Questions

(h) True False

- Kruskal’s algorithm for finding a minimum spanning tree of a weighted, undirected graph is an example of a dynamic programming algorithm.

True/False Questions

(i) TRUE FALSE

The depths of nodes in a red-black tree can be efficiently maintained as fields in the nodes of the tree.

- No, because the depth of a node depends on the depth of its parent
- When the depth of a node changes, the depths of all nodes below it in the tree must be updated
- Updating the root node causes n - 1 other nodes to be updated

Sample Questions

- What do we mean by the running time of an algorithm?
  - The running time of an algorithm is the number of (primitive) operations it performs before coming to a halt. The running time is expressed as a function of the input size n.
- List at least three methods for solving recurrences.
  - Masters method, iteration, substitution

Sample Questions

Describe the characteristic feature of randomized algorithms. What is the main advantage of using such algorithms?

- The behavior is determined in part by values produced by a random-number generator: the algorithm generates its own randomness.
- Advantage: No particular input can consistently elicit worst case behavior.

Sample Questions

What does dynamic programming have in common with divide-and-conquer, and what is the principal difference between the two techniques?

- Common: Both divide the initial problem into subproblems.
- Difference: Dynamic programming is used when the subproblems are not independent.

Exercise

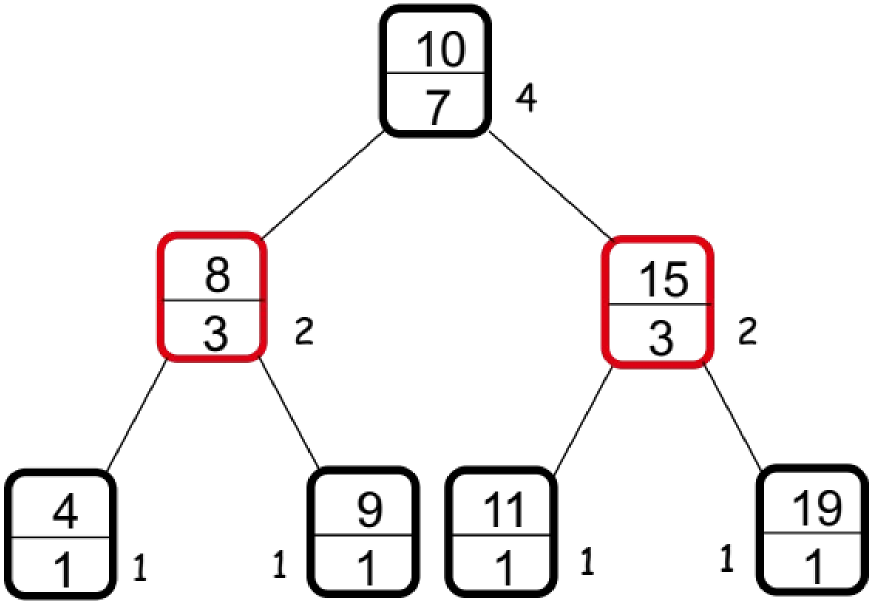
- In an OS-tree, the size field can be used to compute the rank' of a node x, in the subtree for which x is the root. If we want to store this rank in each of the nodes, show how can we maintain this information during insertion and deletion.

Insertion

- add 1 to rank'[x] if z is inserted within x's left subtree
- leave rank'[x] unchanged if z is inserted within x's right subtree

Deletion

- subtract 1 from rank'[x] whenever the deleted node y had been in x's left subtree.



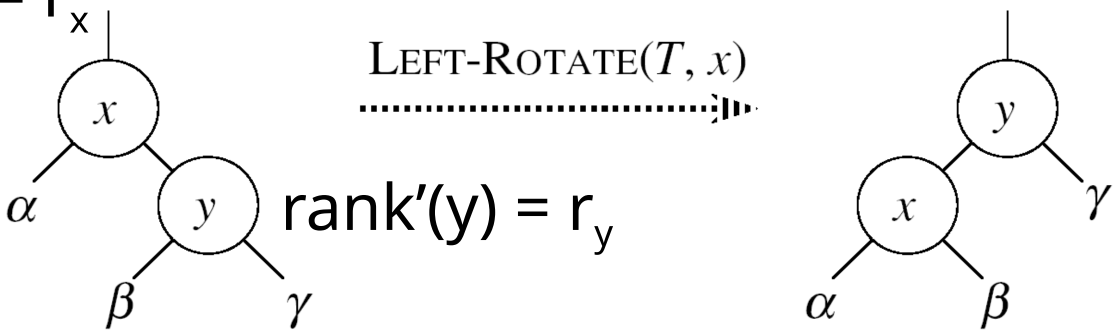
rank'[x] = size[left] + 1

Exercise (cont.)

- We also need to handle the rotations that occur during insertion and deletion

rank’(x) = r_x

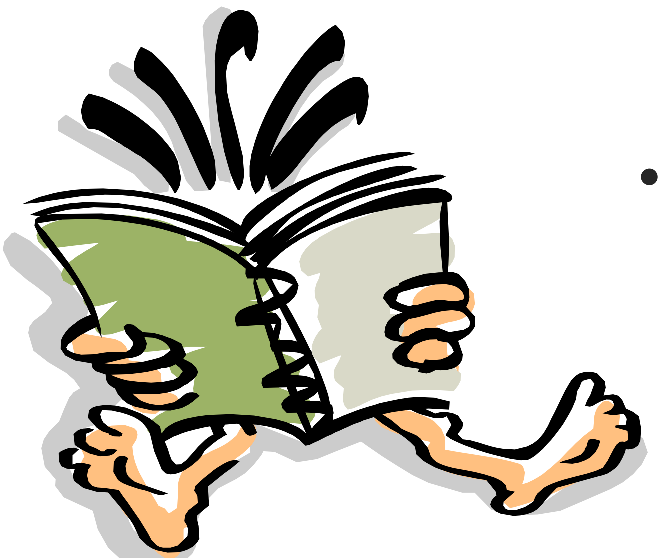
rank’(y) = r_y + rank’(x)



rank’(y) = r_y

rank’(x) = r_x

Readings



- All topics covered during the semester