Intro to Data Structures

Lecture #26 – Graphs
November 30, 2014

Mark Stehlik

Outline for Today

- HW 6 issues
- 2/3 of course grade is behind you…
  - HW7 (7%)
  - Q7 (1.5%)
  - Final (25%)
- Graphs

HW6 issues

- Helper functions should be private
- *contains()* call in add method??
- if ((right - left == 1) || (right - left == -1) || (right - left == 0)) → Math.abs(right – left) <= 1
- *isBalanced()* needed to check subtrees as well!
- *mirrorTree()* was not supposed to damage/alter the original tree!!

HW6 issues

- No statistical output; mine looked like

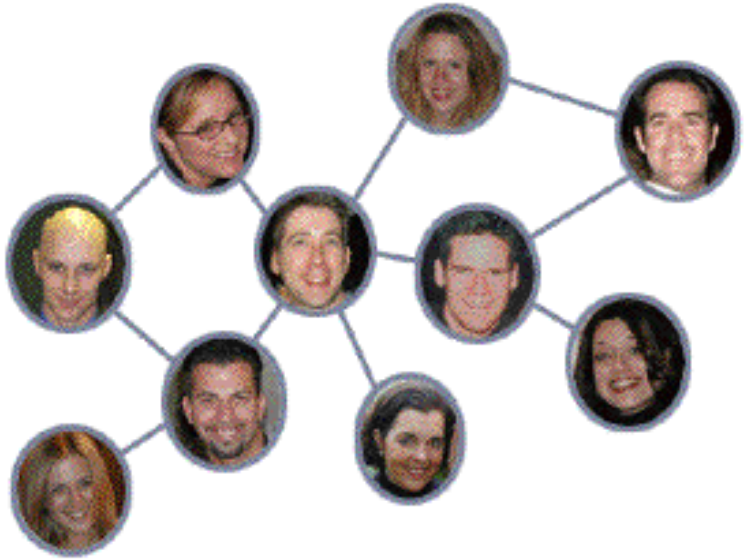
  Total number of words read: 172715
  Number of words inserted (length <= 7): 51913
  Number of nodes in the tree: 41121

- Many of yours looked like

- Test, test, test…

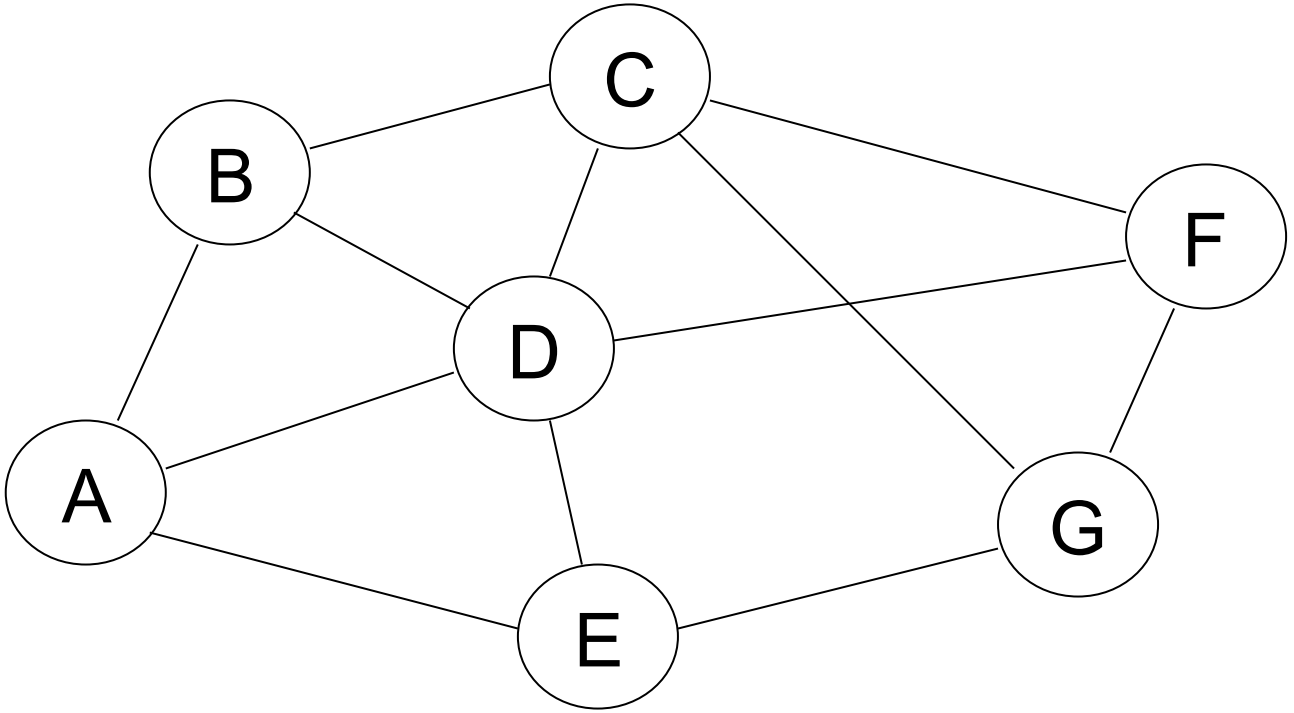
Graphs

- From this…



Graphs

- To this…



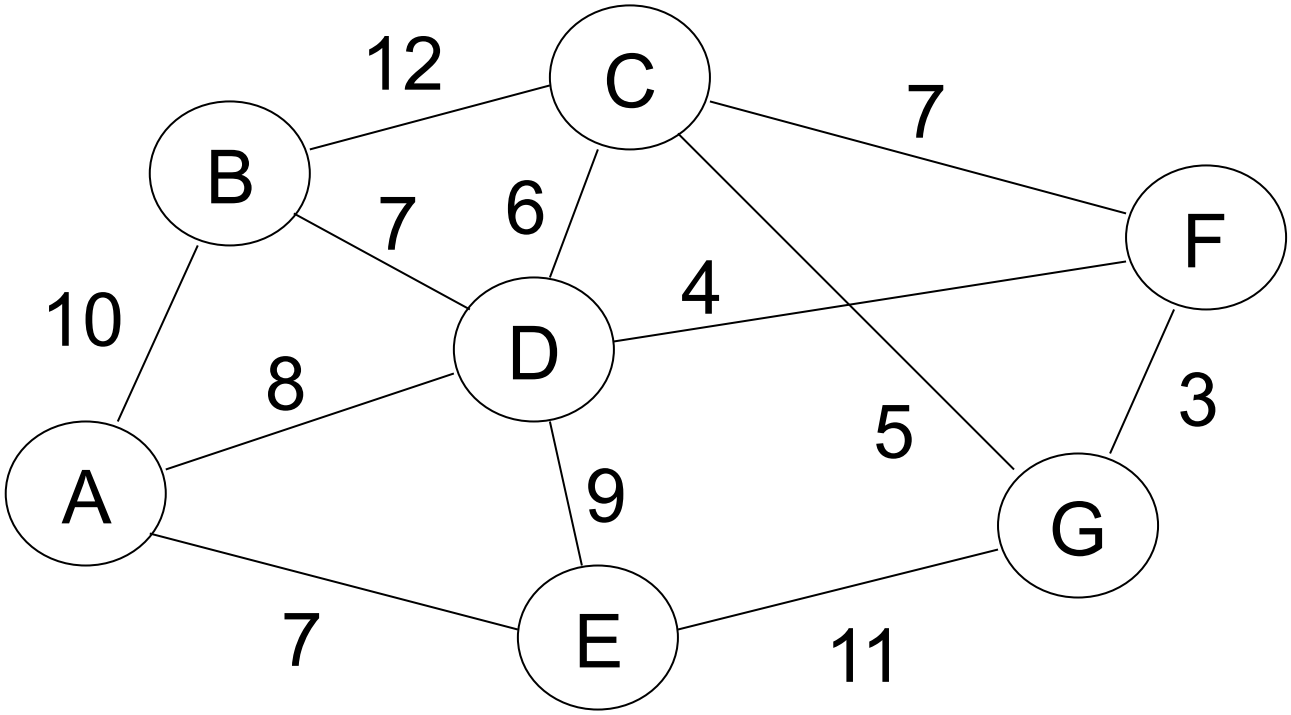
Graphs

- A graph is a collection of vertices (nodes) and edges (connecting the nodes); G = <V, E>
- Edges can be
  - directed/undirected
  - non-weighted/weighted (have different values)
- A *path* is a sequence of vertices connected by edges; the length of the path is the number of edges in the path
- *degree*: number of edges incident to a vertex

Graphs (facts)

- An undirected graph with n vertices will have at most ? edges
  - (n choose 2) = n * (n-1) /2
- A directed graph with n vertices will have at most ? Edges
  - n * (n-1)

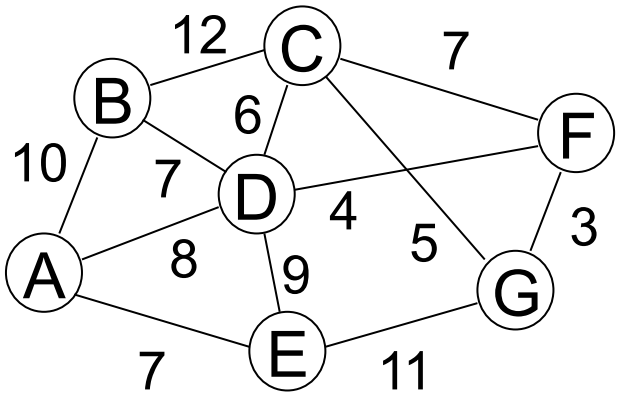
A weighted graph



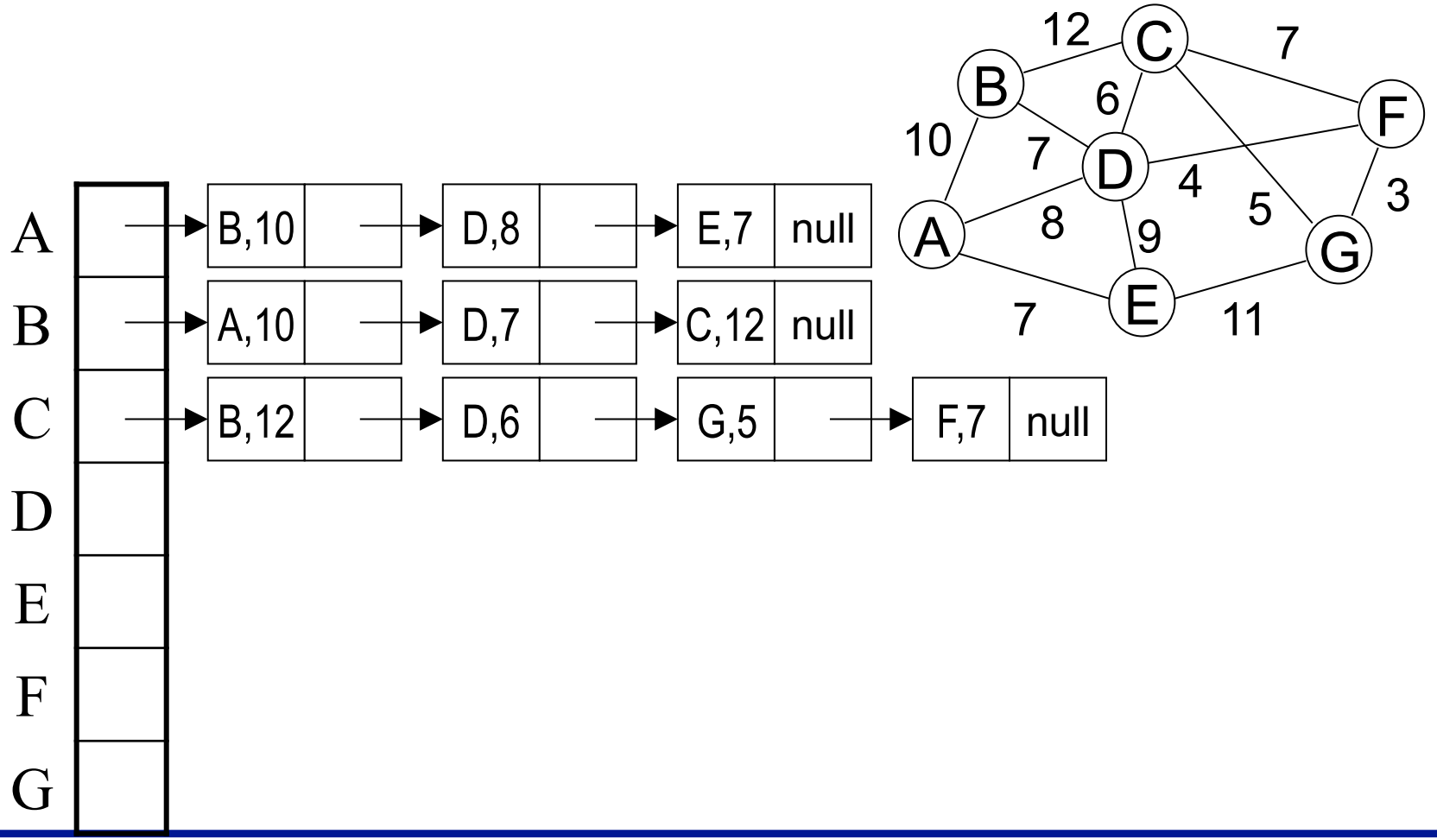
What does this look like?

Graph representation (Adjacency Matrix)

	A	B	C	D	E	F	G
A	0	10	0	8	7	0	0
B	10	0	12	7	0	0	0
C	0	12	0	6	0	7	5
D							
E							
F							
G							



Graph representation (Adjacency List)



Graph representation cost/benefits

- Adjacency matrix
  - Advantage?
    - O(1) access to edge
  - Disadvantage?
    - O($|V|^2$) space
- Adjacency list
  - Advantage
    - O(|V| + |E|) space (dominated by O(|E|)
    - good if not too many edges (sparse graph)
  - Disadvantage?
    - O(|E|) access to edge

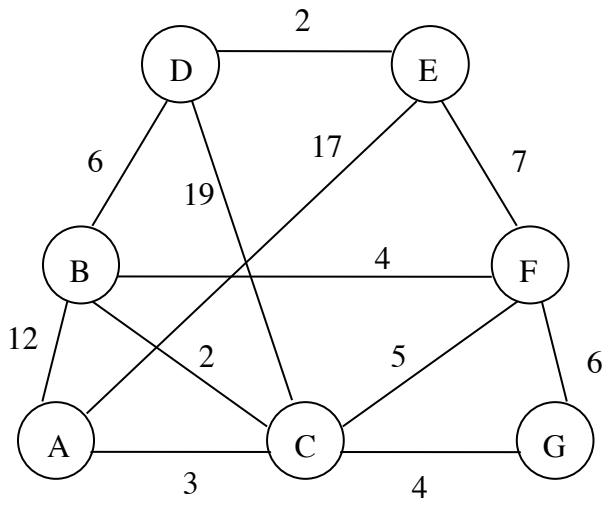
Graph algorithms (a selection)

- Traversals
  - Breadth-first
  - Depth-first
- Others
  - Minimum Spanning Tree (Prim; Kruskal)
    - Tree -> no cycles
    - Spanning -> connects all nodes in the graph
    - Minimum -> lowest overall cost
  - Single-source Shortest Path (Dijkstra)
    - Shortest path from the source to all other nodes in the graph

Graphs (Dijkstra's algorithm)

- Assign every node an initial distance (0 to the source, ∞ for all others); mark all nodes as unvisited
- While there are unvisited nodes:
  - select unvisited node with smallest distance (current)
  - consider all unvisited neighbors of current node:
    - compute distance to each neighbor from current node
    - if less than current distance, replace with new distance
  - mark current node as visited (and is never evaluated again)

Dijkstra example



	A	B	C	D	E	F	G
	0	∞	∞	∞	∞	∞	∞
A->		12	3	∞	17	∞	∞
C->		5		22	17	8	7
B->				11	17	8	7
G->				11	17	8	
F->				11	15		
D->					13		
E	0	5	3	11	13	8	7