Graphs (continued)

Tuesday, December 1, 2015 11:36 AM

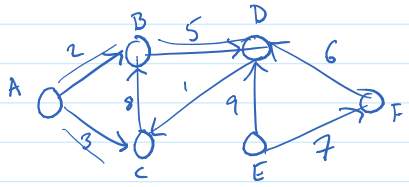
Final Saturday Dec 12 12:30 pm

Last SI sessions this week

Recall:

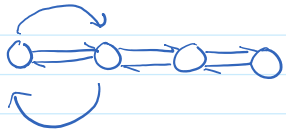
A graph is a pair $\langle V, E \rangle$, with a function $w: E \to \text{Double}$. w is called the "weight function".

An edge in E consists of two nodes $u, v \in V$ denoted as $\langle u, v \rangle$ meaning an edge from u to v



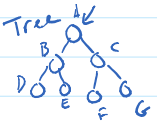
Linked list (Singly)
directed, unweighted

Linked list (doubly)
undirected, unweighted



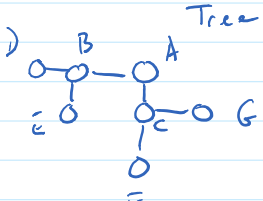
rooted Tree
Acyclic, unweighted

rooted Tree



not a tree

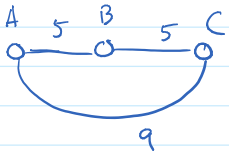
Tree

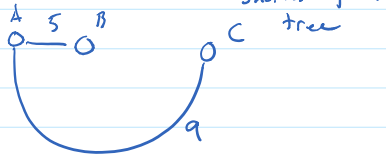


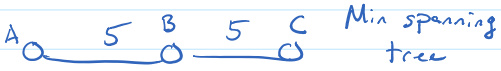
A 5 B 5 C

Min Spanning Tree

Shortest Path (Tree)







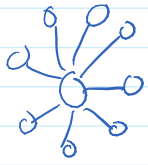
Representations

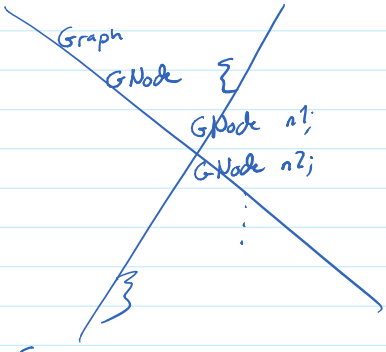
LL

```
Node {
    Node next;
    Node prev;
}
```

Tree

```
TreeNode{
    TreeNode parent;
    TreeNode left, right;
}
```






```
Graph
    GNode<E, N> {
        LinkedList<GNode> neighbors;
        N name;
        E element;
    }
```

```
BTree {
    Node root;
    Node {
        Node left, right, parent;
    }
}
```

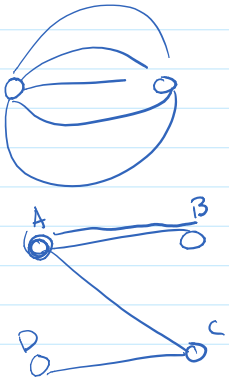
HashMap<K, V> } java API
HashSet<E> } java.util.

```
Graph {
    Map<String, GNode> nodes;
    GNode {
        Set<GNode> neighbors;
```

Adjacency List representation

```
Map<String, GNode> nodes;
GNode {
    Set<GNode> neighbors;
}
```

representation



adj. list of A
B B C

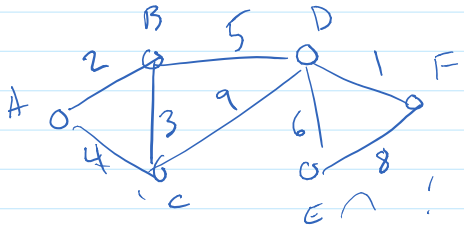
Adjacency Matrix

$n^2 - n\sqrt{2} = \Theta(n^2)$

	A	B	C	D	E	F
A	F					
B	T	F				
C	T	T	F			
D	F	T	T	F		
E	F	F	F	T	F	
F	F	F	F	T	T	F

	A	B	C	D	E
B	1	-			
C	1	1	-		
D	0	1	1		
E	0	0	0	1	
F	0	0	0	1	1

bool[][]
int[][]
Integer[][] → null means no edge



Weighted
Undirected

	A	B	C	D	E
B	2				
C	4	3			
D	-	5	9		
E	-	-	-	6	
F	-	-	-	1	8

$|E| = \#$ of edges

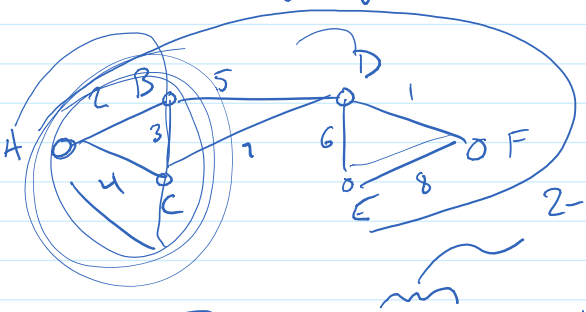
$|E| = O(f(|V|))$

$|E| = O(|V|^2)$

$|E|$ = # of edges
$|V|$ = # of vertices

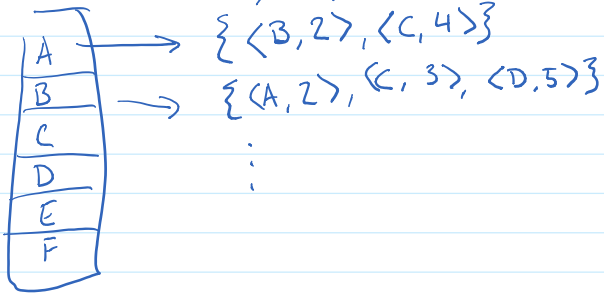
As a rule of thumb:
If $|E| \ll |V|^2$, use an adjacency list

$\ll$ - "much smaller"



A B C D E F
A B D F E C

2-Tuple




```
Tuple<W extends Comparable<W>> {
    String name;
    W weight;
}
```

DFS
BFS

```
BFS (Node n):
    Queue<Node> q;
    Set<Node> visited;
    add n to q & visited
    while q is not empty:
        dequeue a node
        print(node)
        for (Node neighbor : node.neighbors):
            if neighbor in visited, do nothing
            else, add to queue & visited
```