COMP 6651
Algorithm Design Techniques
Week 8

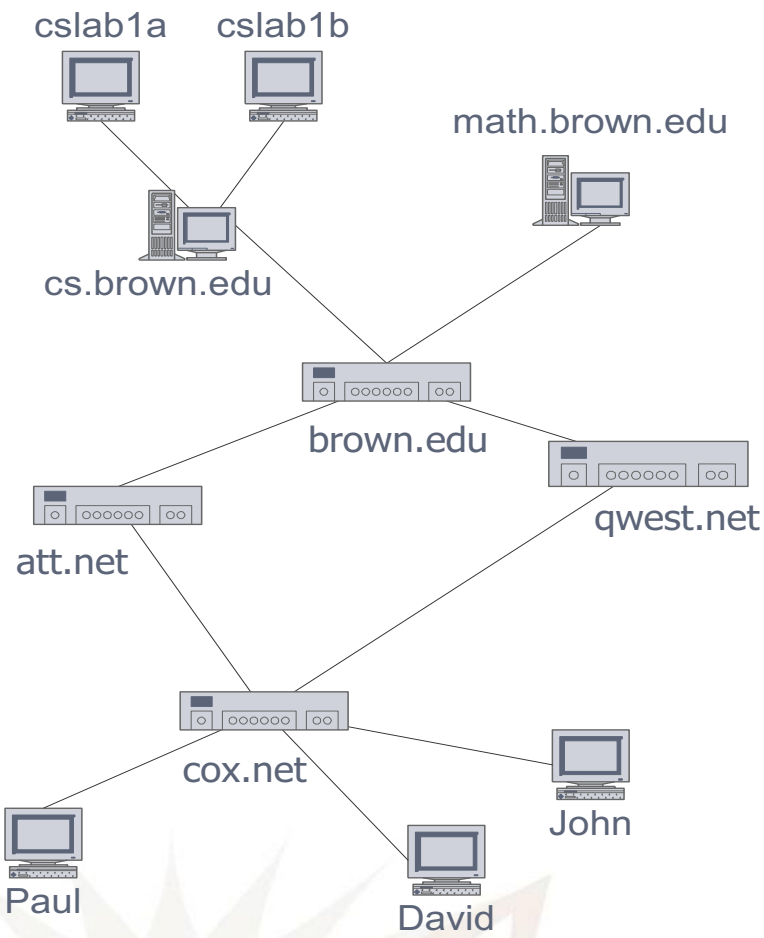
Graph Theory.

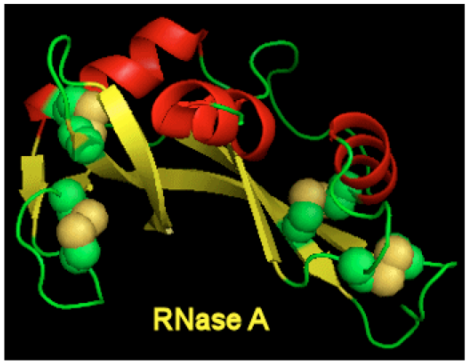
(some material is taken from web or other various sources with permission)

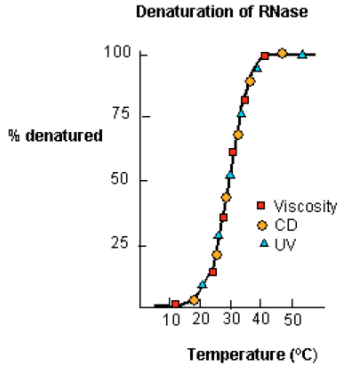
Graphs

- Graphs and graph algorithms are one of the most important and powerful data modeling tools in computer science
- Used in:
  - Computer graphics,
  - Computer vision
  - Image processing
  - Robotics
  - Probabilistic models (example Random Markov Fields)
  - A large number of combinatorial and optimization problems: Airlines and transportation logistics

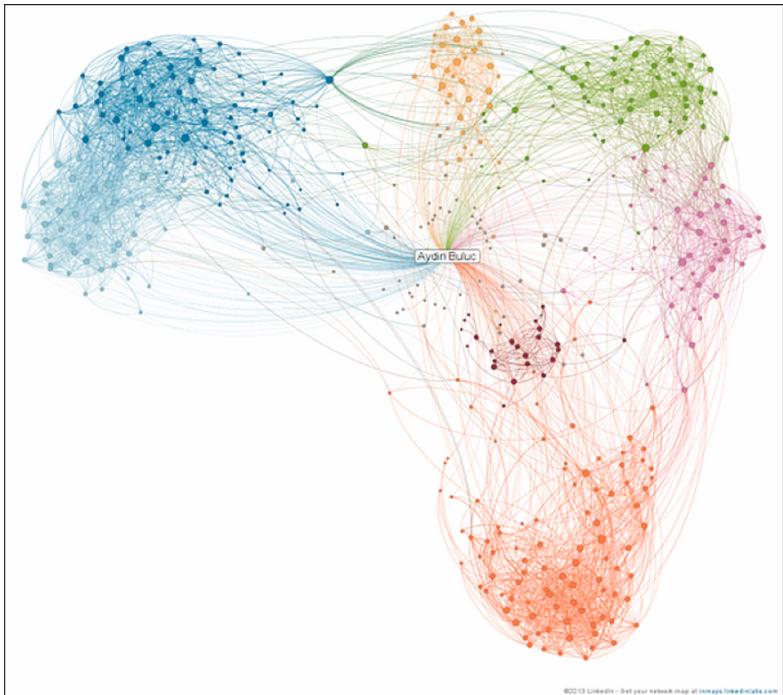
Graphs





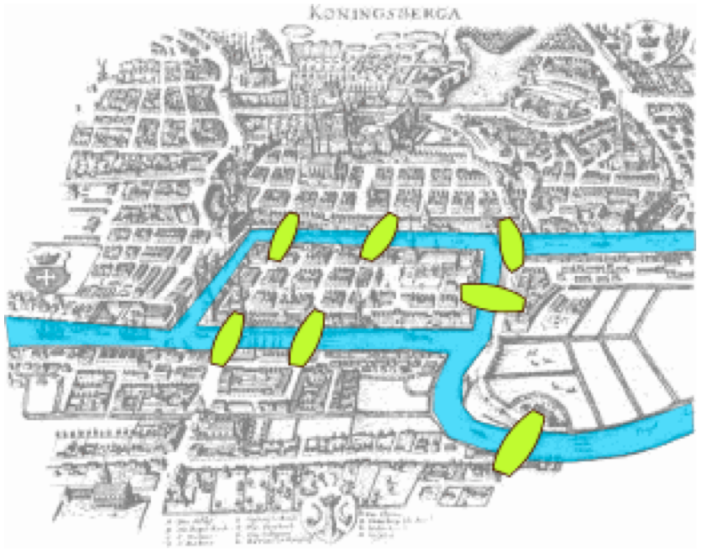


after Ginsburg and Carroll, Biochemistry 4, pg 2159 (1965)



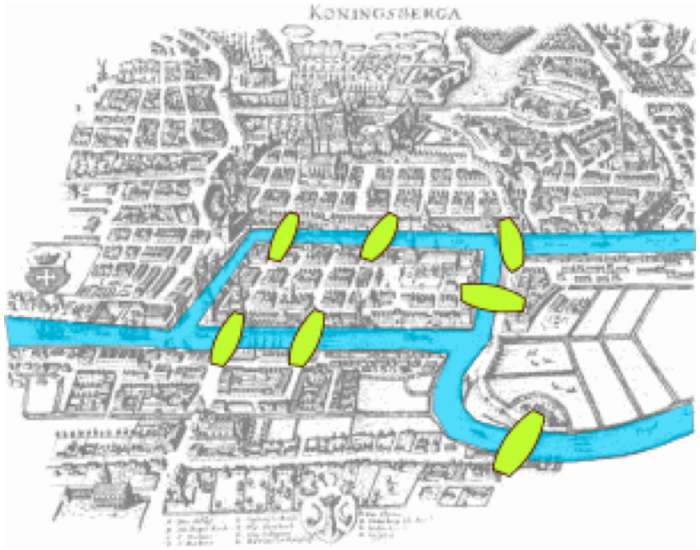
Seven Bridges of Königsberg

- Born 18th century
- Kaliningrad has 2 islands
- Problem was: devising a walk through the city that crosses each bridge exactly once
- Euler modeled the problem as a graph problem



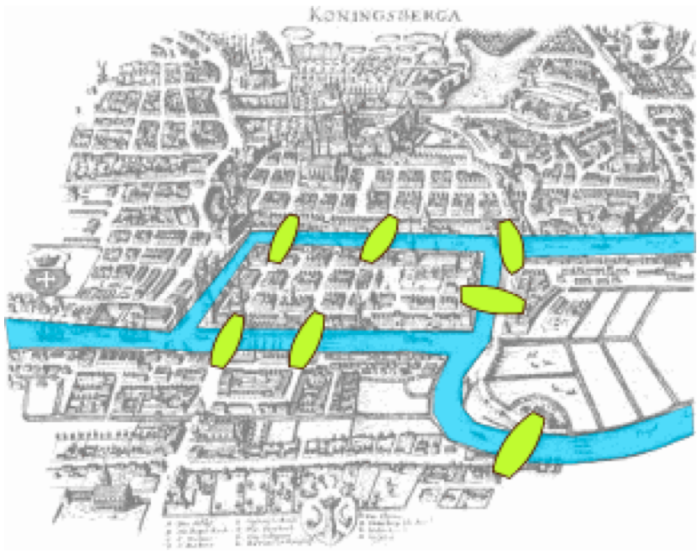
Seven Bridges of Königsberg

- Eulerian path
- Hamiltonian path
- Modern problems:
  GPS, uber
  - Cell phone towers positioning
  - Positioning of hospitals, finding optimal ambulance routes
  - Chemistry, biology (i.e. protein folding)

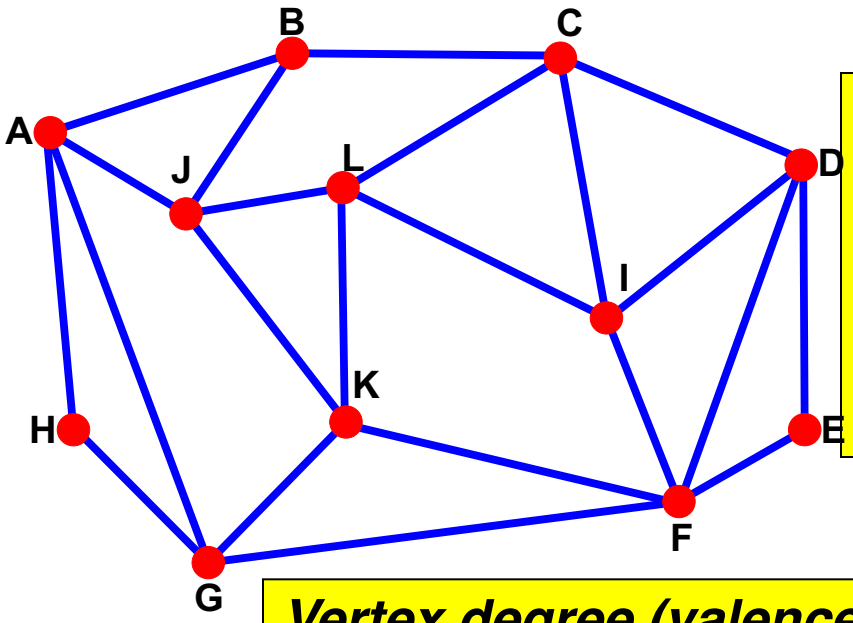


Seven Bridges of Königsberg

- Very well studied field in C&O
- In this class:
  - foundational concepts
  - classical algorithms



Standard Graph Definitions



G = <**V**,**E**>
V = vertices = {A,B,C,D,E,F,G,H,I,J,K,L}
E = edges = {(A,B),(B,C),(C,D)…(D,E),(E,F),(F,G), }

Vertex degree (valence) = number of edges incident on vertex deg(J) = 4, deg(H) = 2
k-regular graph = graph whose vertices all have degree *k*

Face: cycle of vertices/edges which cannot be shortened
F = faces = {(A,H,G),(A,J,K,G),(B,A,J),(B,C,L,J),(C,I,L),(C,D,I),(D,E,F),(D,I,F),(L,I,F,K),(L,J,K),(K,F,G)}

Connectivity

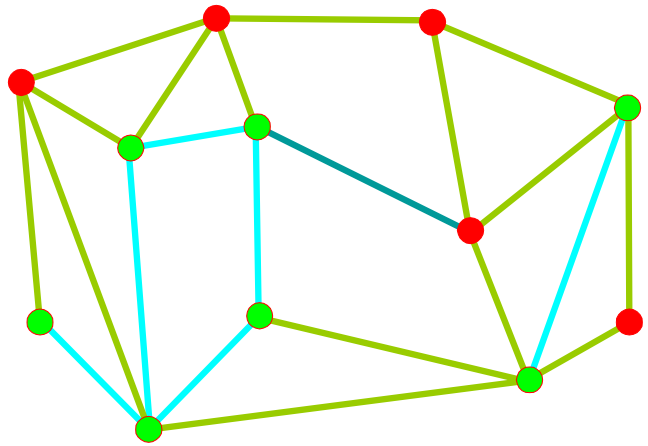
Graph is ***connected*** if there is a path of edges connecting every two vertices

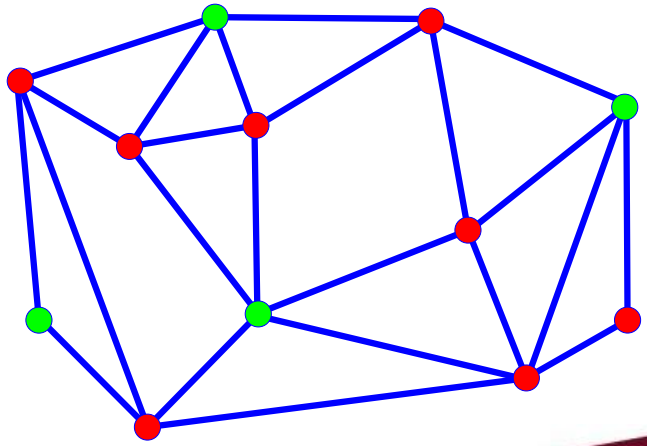
Graph is ***k-connected*** if between every two vertices there are *k* edge-disjoint paths

Graph **G'**=<**V'**,**E'**> is a ***subgraph*** of graph **G**=<**V**,**E**> if **V'** is a subset of **V** and **E'** is the subset of **E** incident on **V'**

Connected component of a graph: maximal connected subgraph

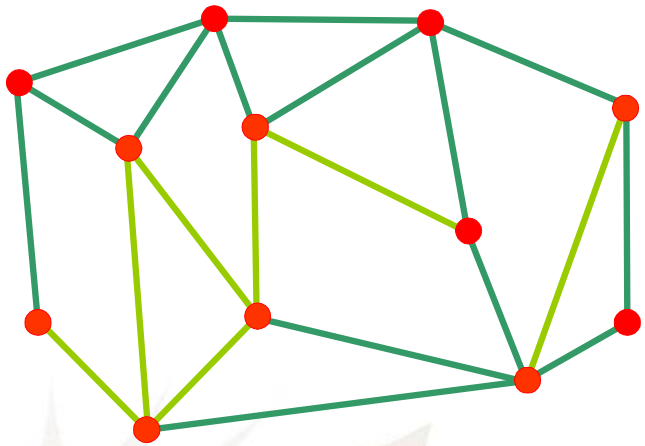
Subset **V'** of **V** is an ***independent*** set in **G** if the subgraph it induces does not contain any edges of **E**



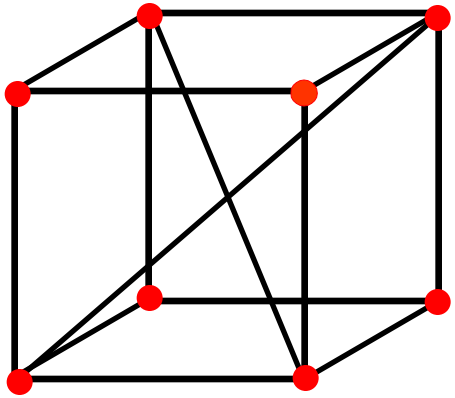


Graph Embedding

Graph is ***embedded*** in R^d if each vertex is assigned a position in R^d



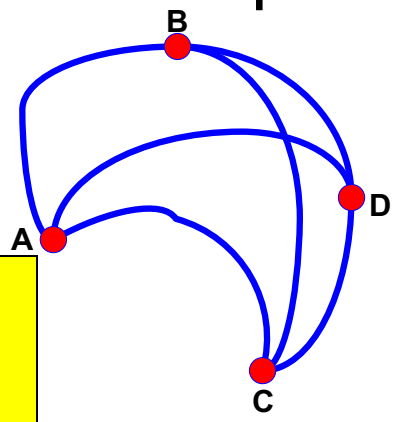
Embedding in R^2



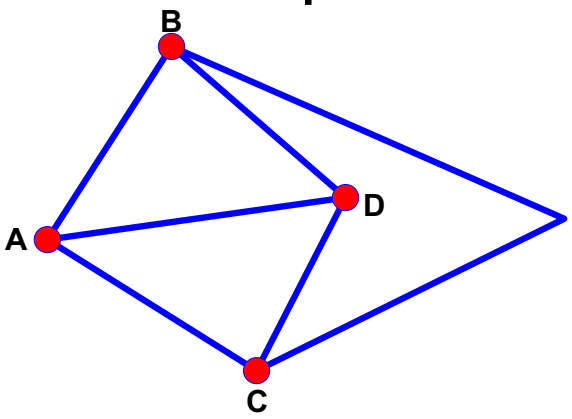
Embedding in R^3

Planar Graphs

Planar Graph

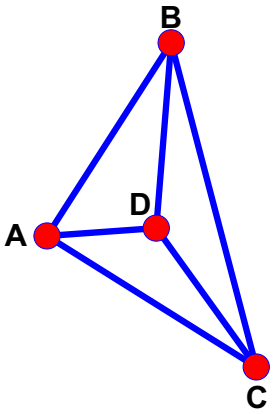


Plane Graph

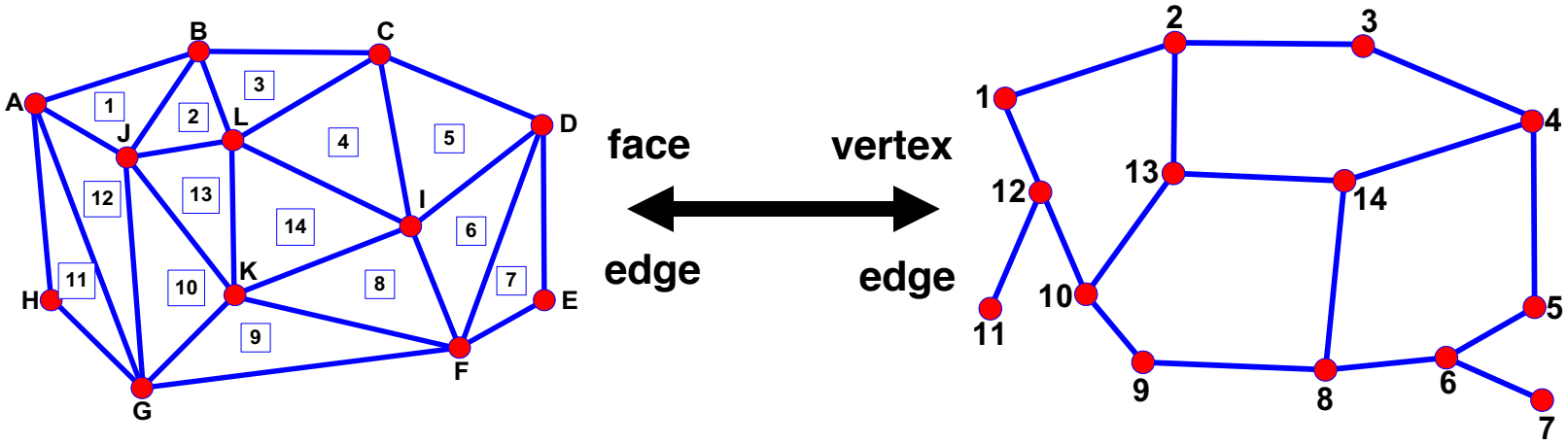


Planar graph: graph whose vertices and edges can be embedded in R^2 such that its edges do not intersect

Every planar graph can be drawn as a ***straight-line plane graph***

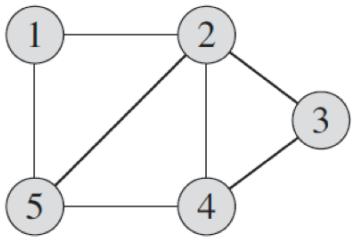


Duality

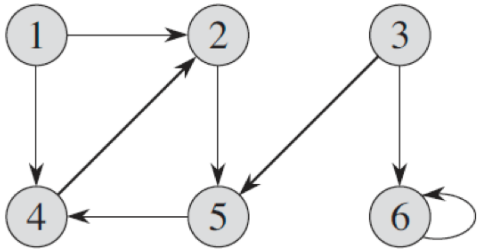


All topological properties of a graph are preserved in its dual

Directed vs. Undirected Graphs



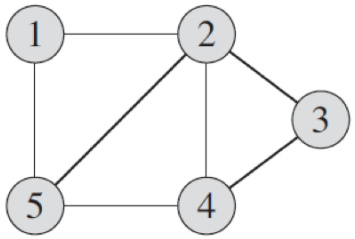
(a)



(a)

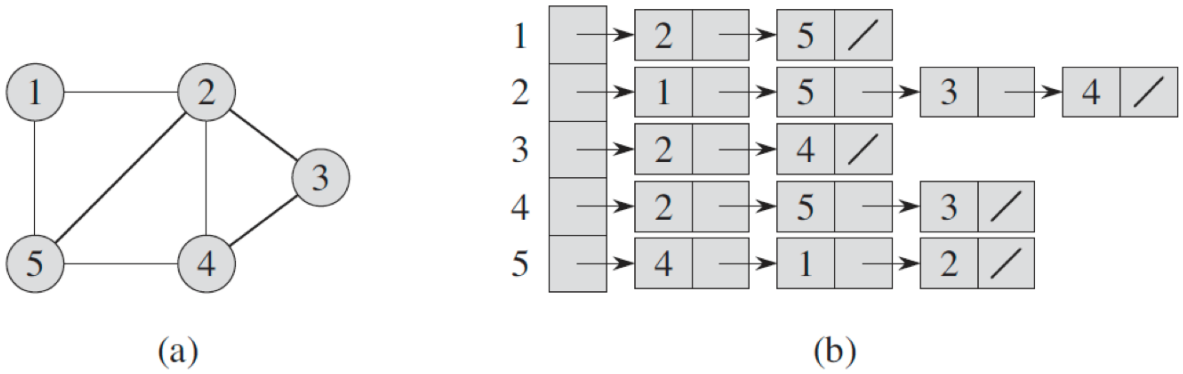
Can also attach a weight to the edge

Directed vs. Undirected Graphs

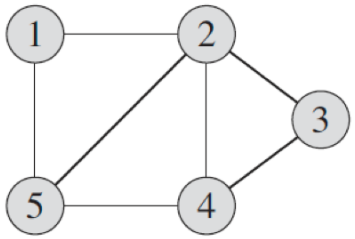


(a)

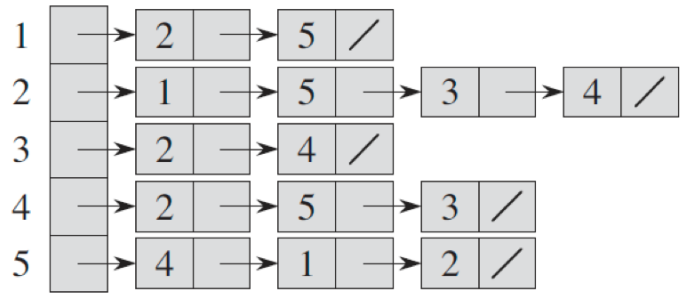
Directed vs. Undirected Graphs



Directed vs. Undirected Graphs



(a)

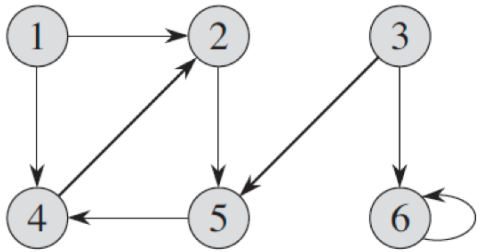


(b)

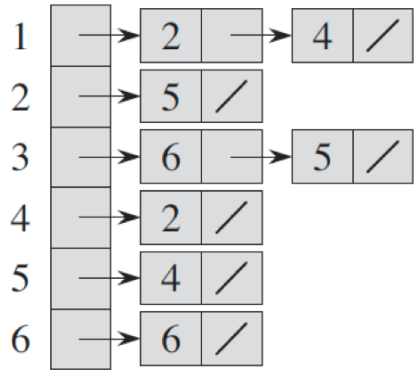
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)

Directed vs. Undirected Graphs



(a)



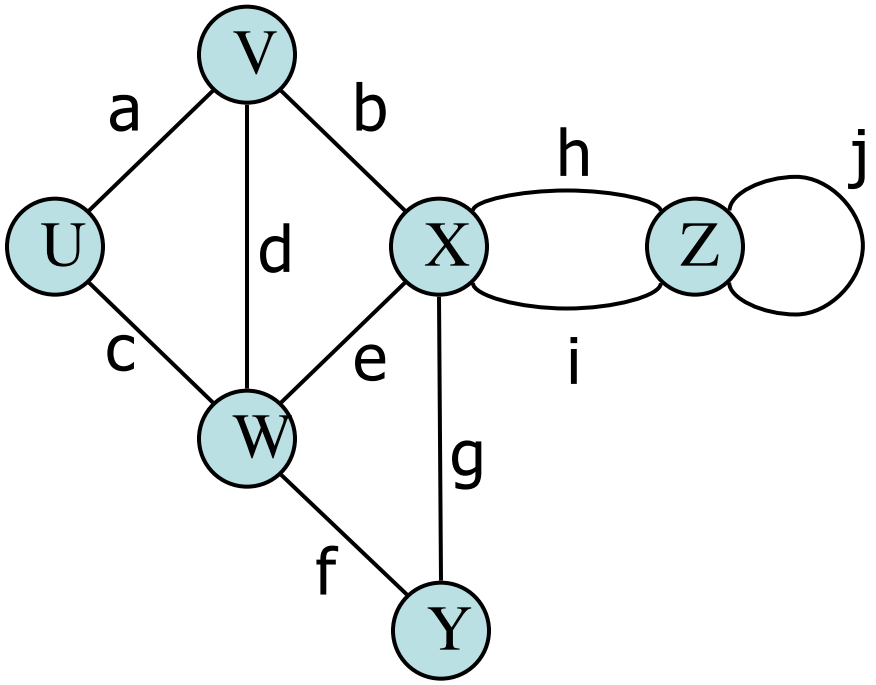
(b)

	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

(c)

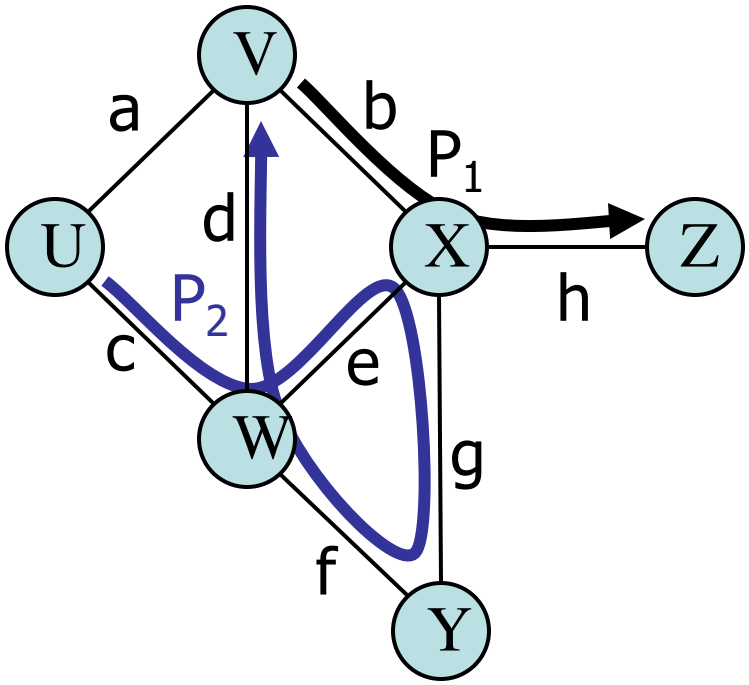
Terminology

- End vertices (or endpoints) of an edge
  - U and V are the endpoints of a
- Edges incident on a vertex
  - a, d, and b are incident on V
- Adjacent vertices
  - U and V are adjacent
- Degree of a vertex
  - X has degree 5
- Parallel edges
  - h and i are parallel edges
- Self-loop
  - j is a self-loop



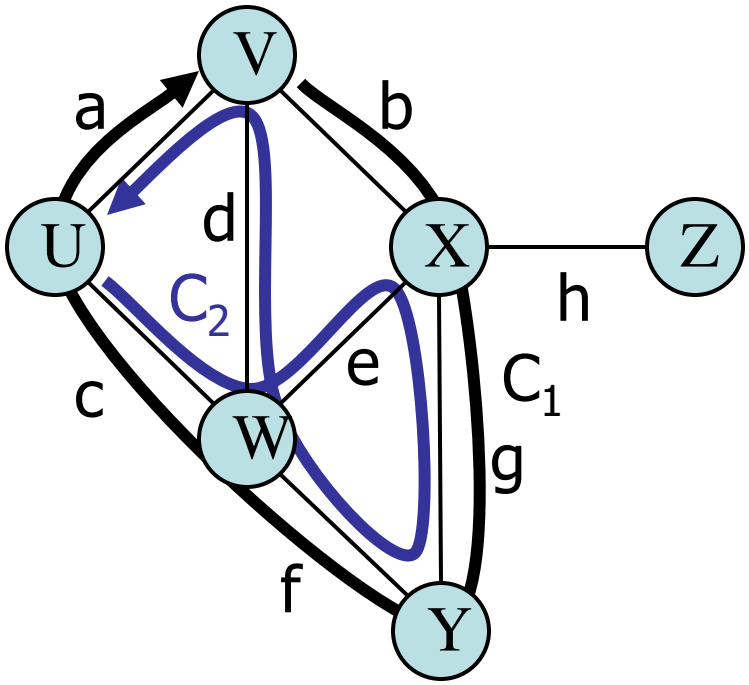
Terminology (cont.)

- Path
  - sequence of alternating vertices and edges
  - begins with a vertex
  - ends with a vertex
  - each edge is preceded and followed by its endpoints
- Simple path
  - path such that all its vertices and edges are distinct
- Examples
  - P_1=(V,b,X,h,Z) is a simple path
  - P_2=(U,c,W,e,X,g,Y,f,W,d,V) is a path that is not simple



Terminology (cont.)

- Cycle
  - circular sequence of alternating vertices and edges
  - each edge is preceded and followed by its endpoints
- Simple cycle
  - cycle such that all its vertices and edges are distinct
- Examples
  - C_1=(V,b,X,g,Y,f,W,c,U,a,↵) is a simple cycle
  - C_2=(U,c,W,e,X,g,Y,f,W,d,V,a,↵) is a cycle that is not simple



Properties

Property 1

$\sum_v \deg(v) = 2m$

Proof: each edge is counted twice

Property 2

In an undirected graph with no self-loops and no multiple edges

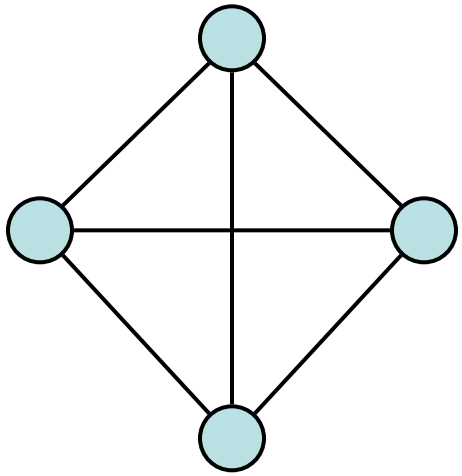
$m \leq n\,(n-1)/2$

Proof: each vertex has degree at most $(n-1)$

What is the bound for a directed graph?

Notation

n	number of vertices
m	number of edges
$\deg(v)$	degree of vertex v



Example

- $n = 4$
- $m = 6$
- $\deg(v) = 3$

Planar Graphs

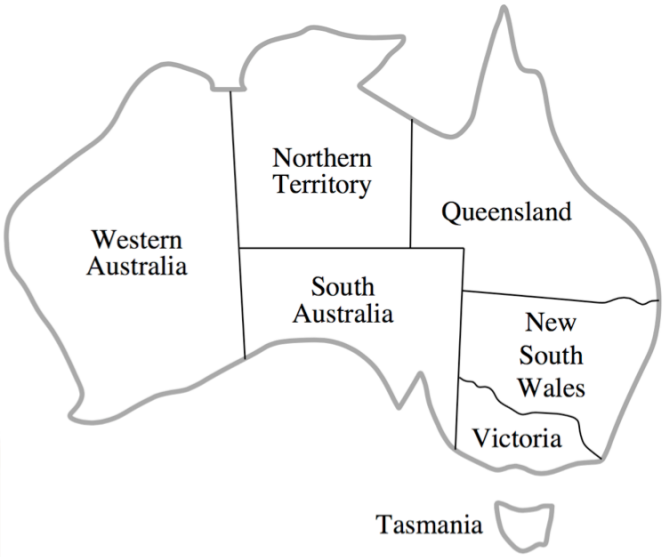
- A graph that has a planar embedding
- Complete graphs: K3, K4, K5
- Complete Bipartite graphs: K2,2 K2,3 K3,3
- On the board
- Theorem (Kuratowski).
  - A graph is planar if and only if it contains no subdivision of K3,3 or K5.
- Euler formula
  - v – e + f = 2
  - |e|<=3|v|-6
  - Average degree of a vertex <=6

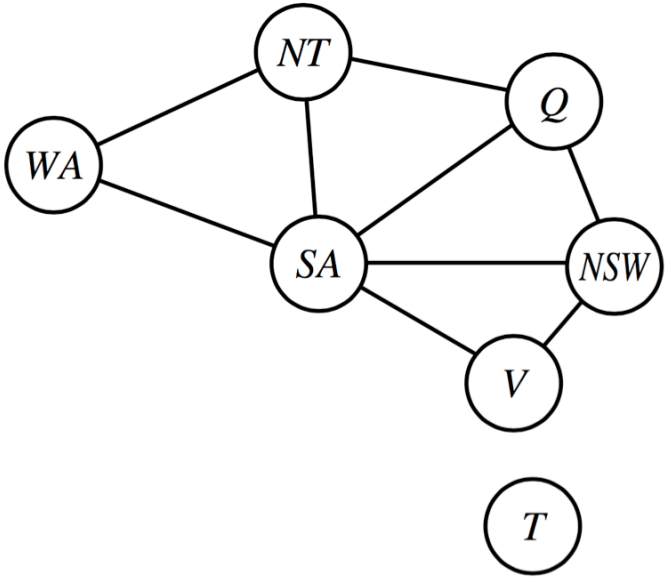
Planar Graphs

- Map coloring problem:
  I have a map where countries are represented as polygons with sharing edges (and assume edges at water borders)
- Want to color the countries s.t. no adjacent countries share the color?
- How many colors do I need?

Planar Graphs

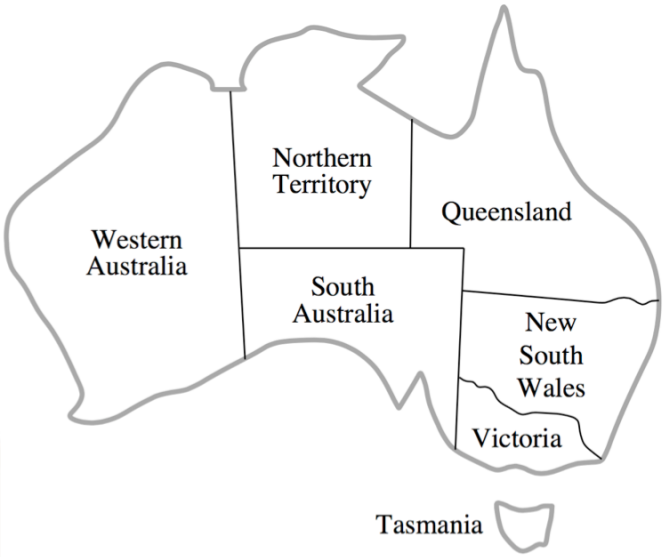
- Map coloring problem:
  How do I formalize it in terms of graphs?

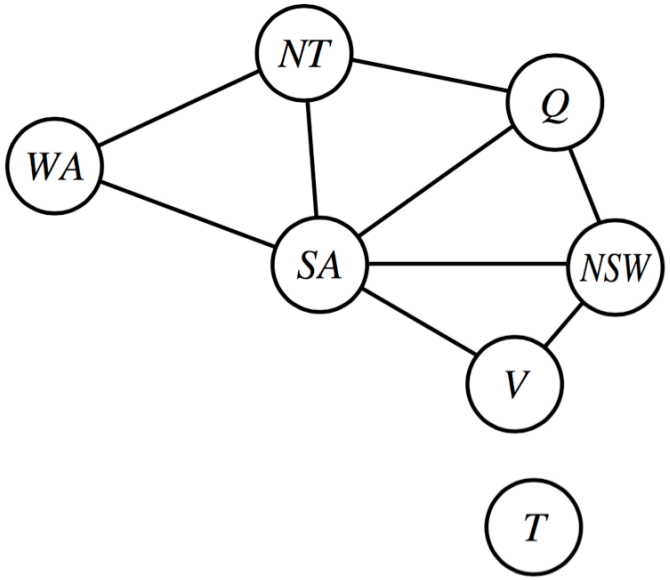




Planar Graphs

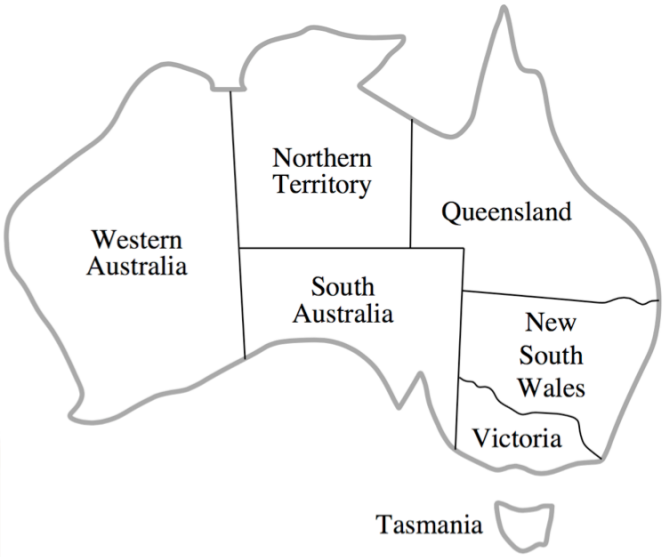
- Map coloring problem:
  only need 4 colors (posed in 18th century, solved in 1977 with the help of computer simulations)

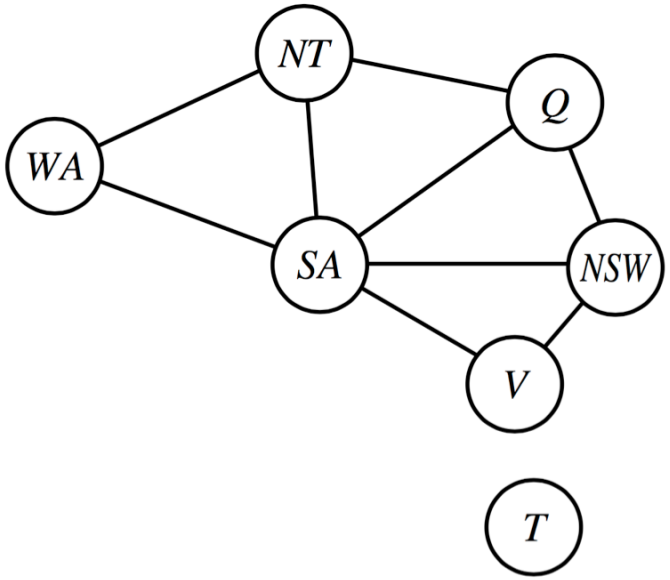




Planar Graphs

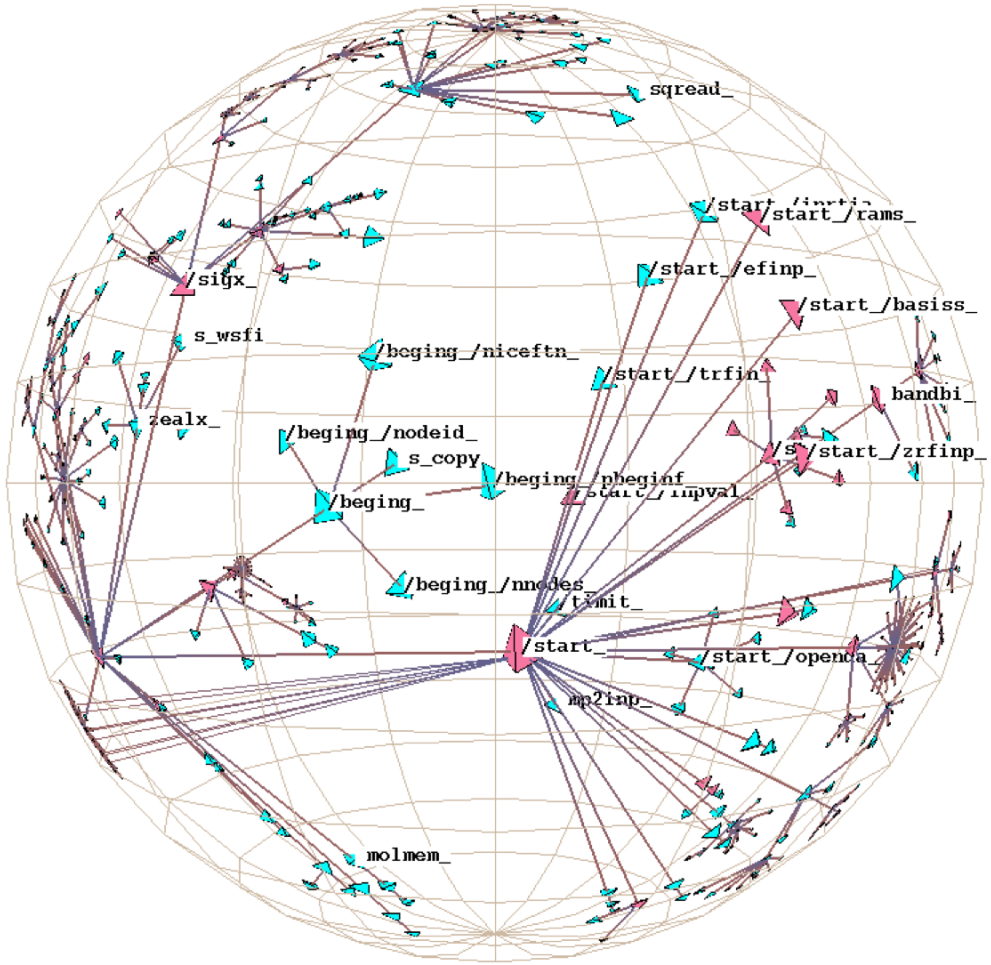
- However, we will prove in class the 6 coloring theorem!!!





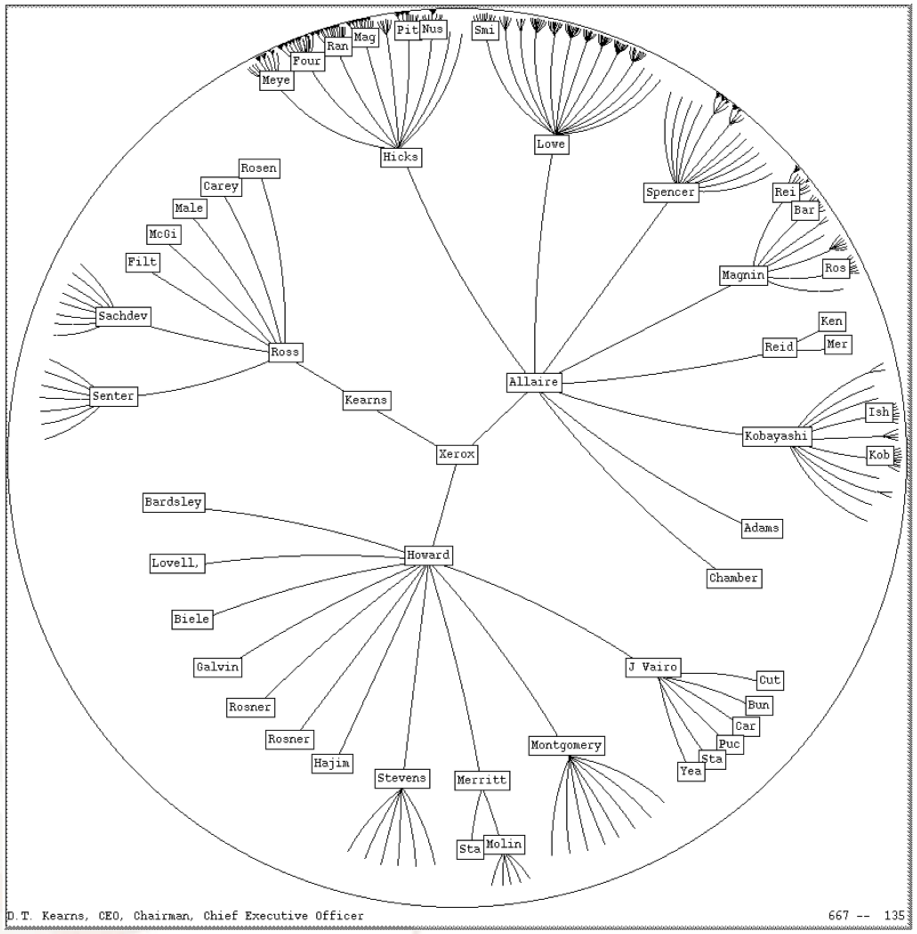
Other embeddings Graphs

- Sphere



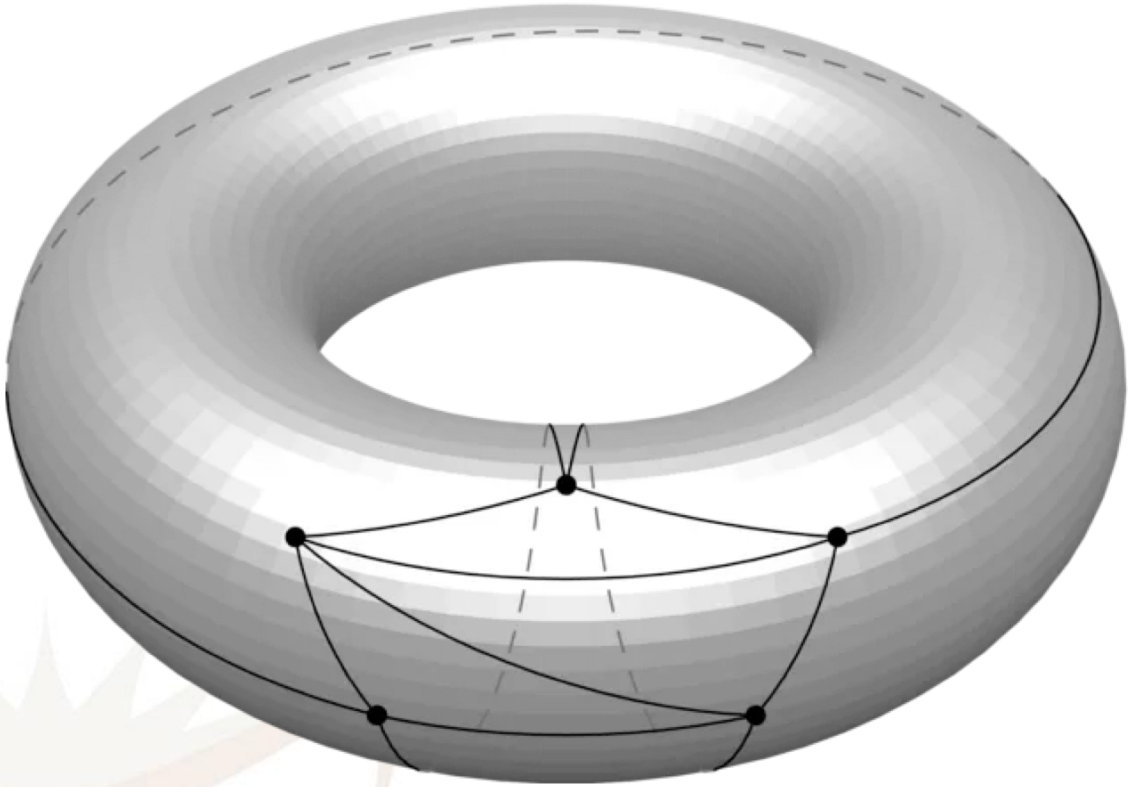
Other embeddings Graphs

- Hyperplane (non-Euclidian geometry) **Lobachevsky**



Other embeddings Graphs

- K5 is not planar
- Can I embed it on any 2D surface?

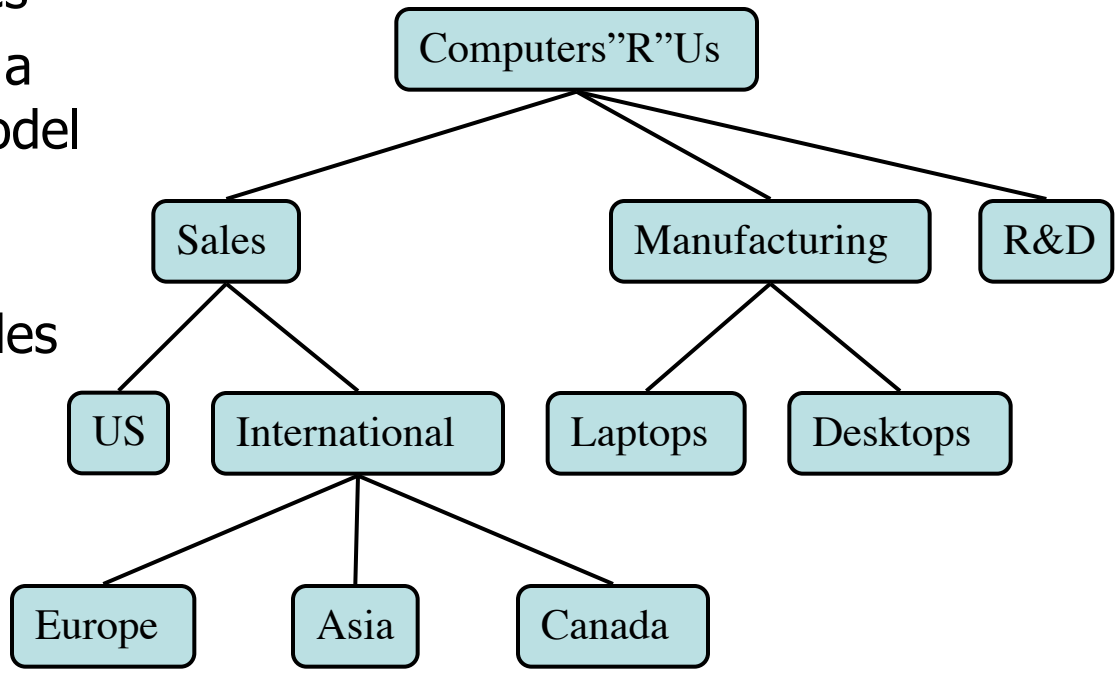


Graph Ordering

- What if you need to enumerate the vertices?
- (from neighbour to neighbor)

What is a Tree

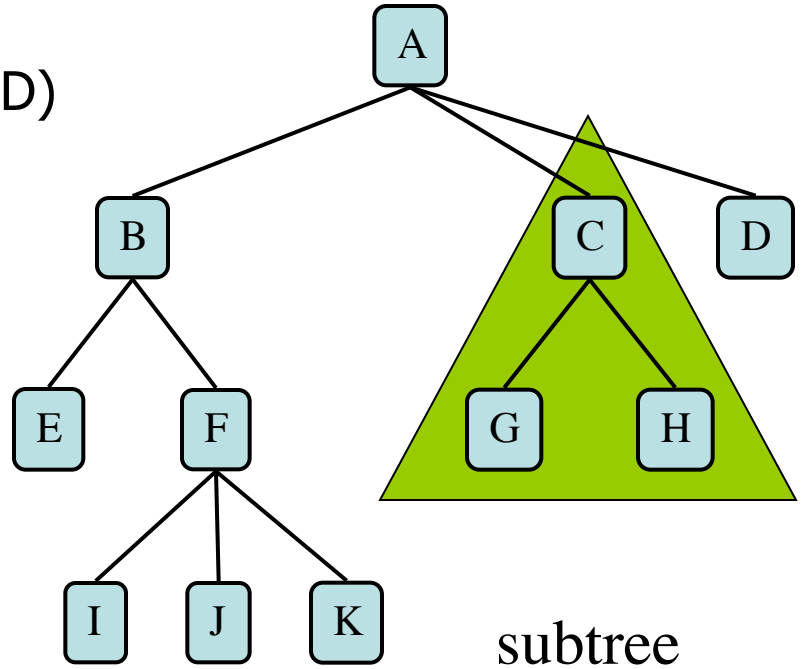
- A graph with no cycles
- In computer science, a tree is an abstract model of a hierarchical structure
- A tree consists of nodes with a parent-child relation
- Applications:
  - Organization charts
  - File systems
  - Programming environments



Tree Terminology

- Root: node without parent (A)
- Internal node: node with at least one child (A, B, C, F)
- External node (a.k.a. leaf): node without children (E, I, J, K, G, H, D)
- Ancestors of a node: parent, grandparent, grand-grandparent, etc.
- Depth of a node: number of ancestors
- Height of a tree: maximum depth of any node (3)
- Descendant of a node: child, grandchild, grand-grandchild, etc.

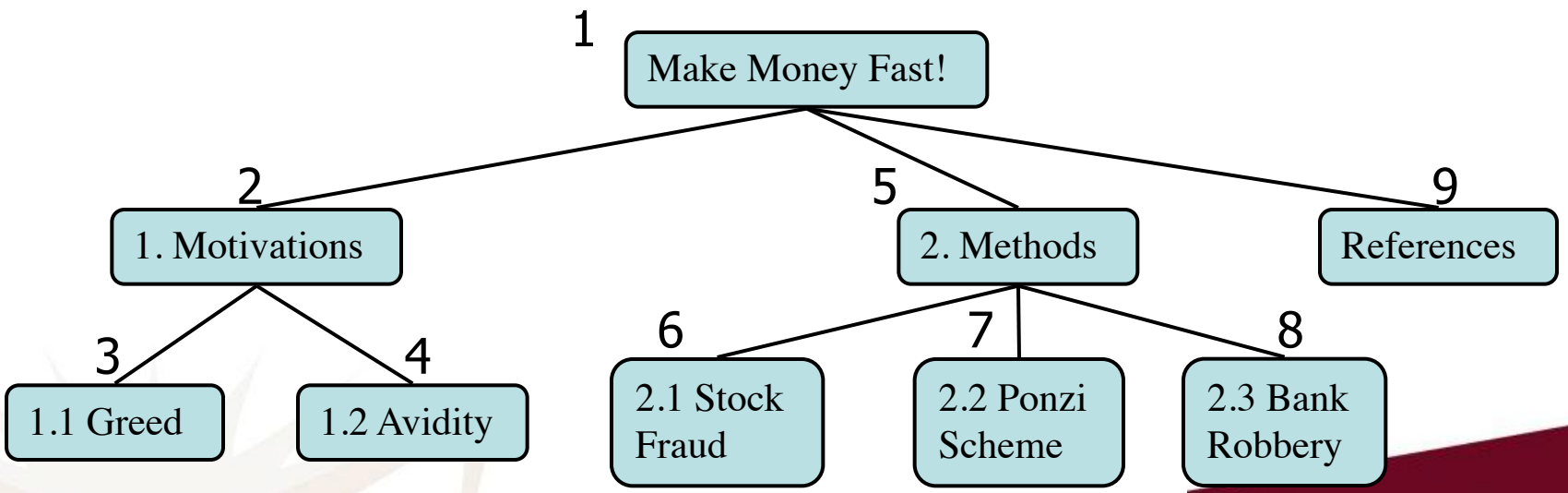
- Subtree: tree consisting of a node and its descendants



Preorder Traversal

- A traversal visits the nodes of a tree in a systematic manner
- In a preorder traversal, a node is visited before its descendants
- Application: print a structured document

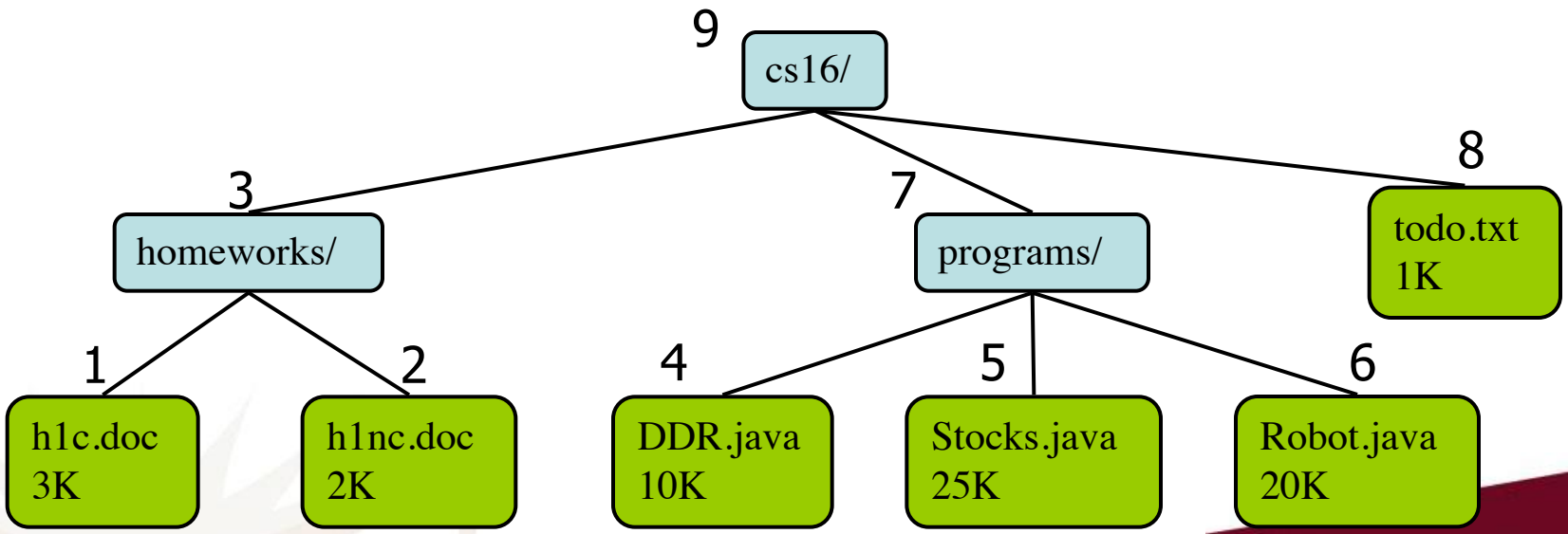
Algorithm ***preOrder*****(*****v*****)**
    visit**(*****v*****)**
    for each child ***w*** of ***v***
        preorder **(*****w*****)**



Postorder Traversal

- In a postorder traversal, a node is visited after its descendants
- Application: compute space used by files in a directory and its subdirectories

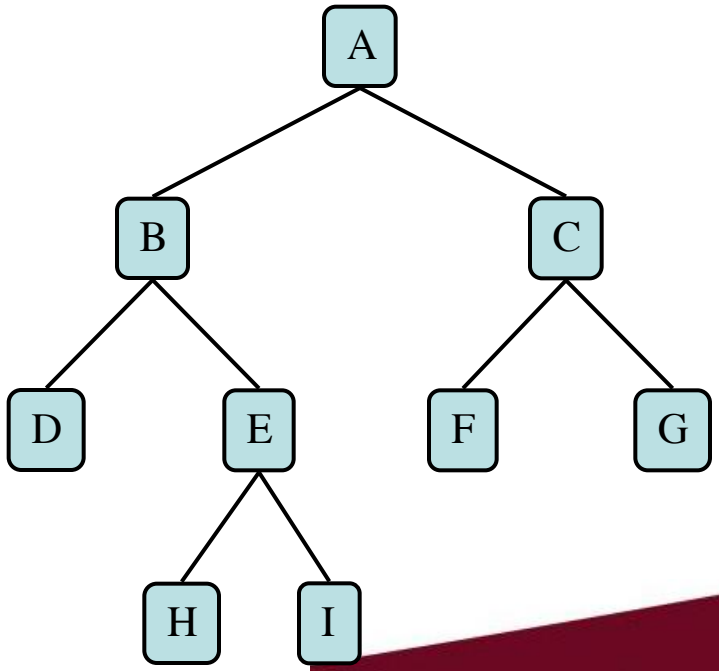
Algorithm ***postOrder*****(*****v*****)**
  for each child ***w*** of ***v***
    postOrder **(*****w*****)**
  visit**(*****v*****)**



Binary Trees

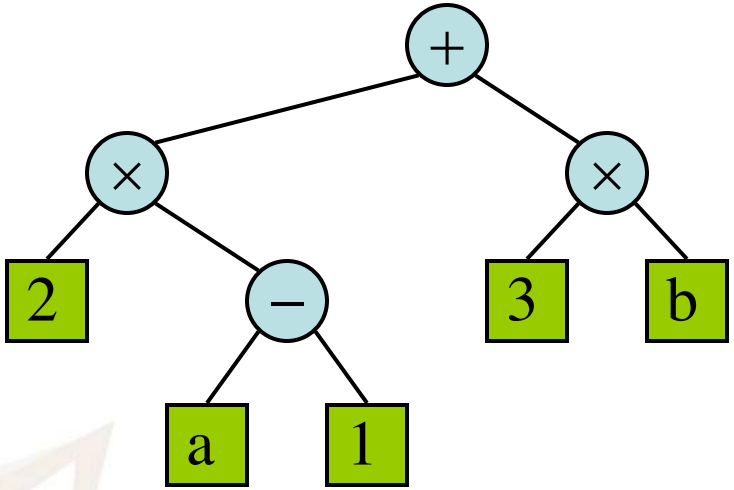
- A binary tree is a tree with the following properties:
  - Each internal node has at most two children (exactly two for proper binary trees)
  - The children of a node are an ordered pair
- We call the children of an internal node left child and right child
- Alternative recursive definition: a binary tree is either
  - a tree consisting of a single node, or
  - a tree whose root has an ordered pair of children, each of which is a binary tree

- Applications:
  - arithmetic expressions
  - decision processes
  - searching



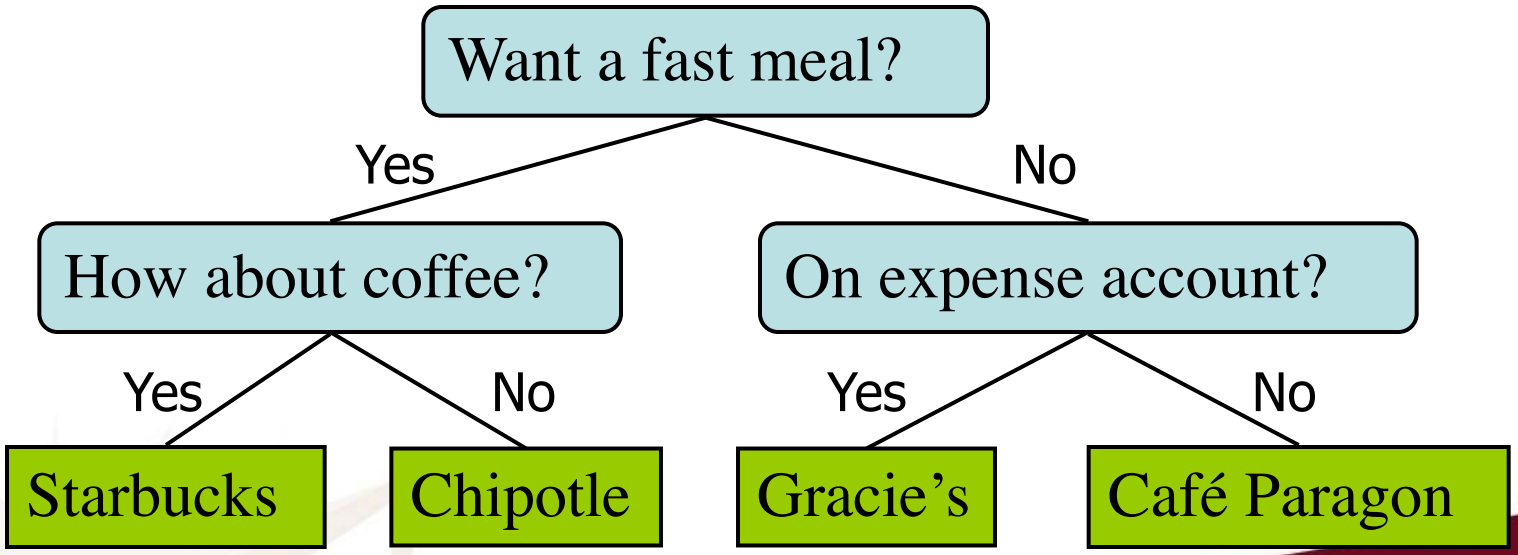
Arithmetic Expression Tree

- Binary tree associated with an arithmetic expression
  - internal nodes: operators
  - external nodes: operands
- Example: arithmetic expression tree for the expression $(2 \times (a - 1) + (3 \times b))$



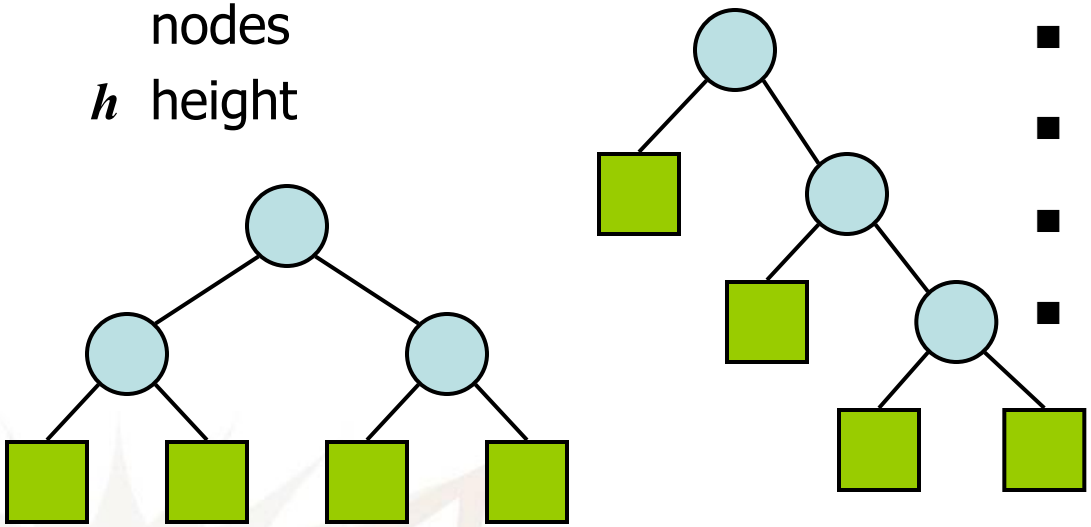
Decision Tree

- Binary tree associated with a decision process
  - internal nodes: questions with yes/no answer
  - external nodes: decisions
- Example: dining decision



Properties of Proper Binary Trees

- Notation
  - n number of nodes
  - e number of external nodes
  - i number of internal nodes
  - h height

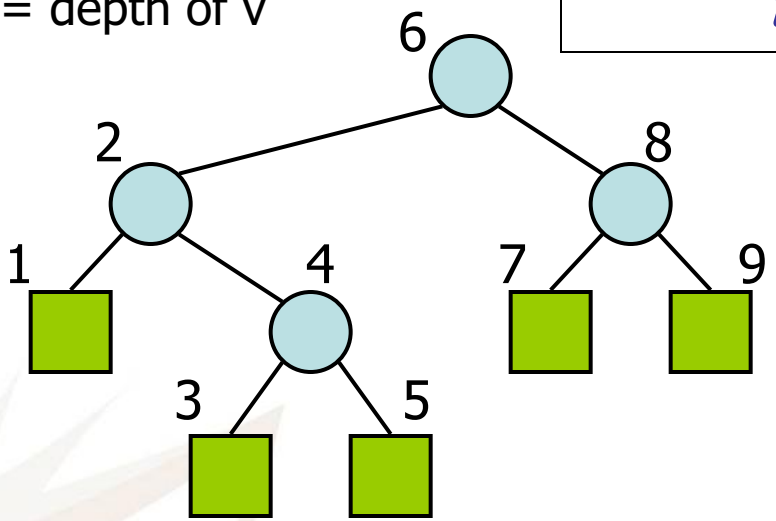


- Properties:
  - $e = i + 1$
  - $n = 2e - 1$
  - $h \leq i$
  - $h \leq (n - 1)/2$
  - $e \leq 2^h$
  - $h \geq \log_2 e$
  - $h \geq \log_2 (n + 1) - 1$

Inorder Traversal

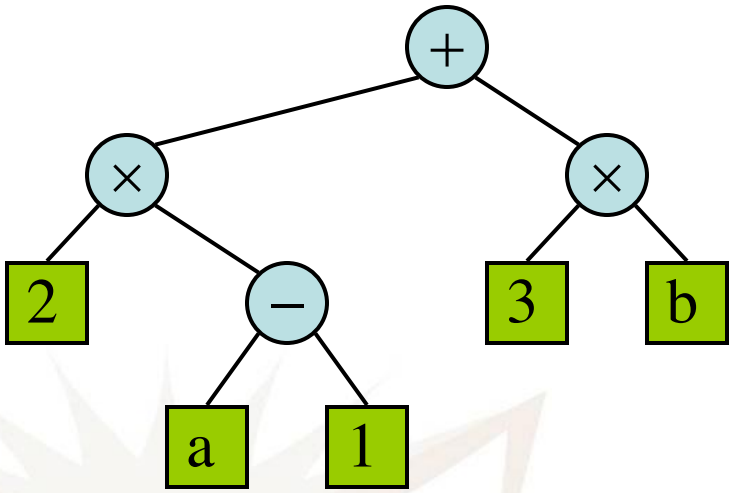
- In an inorder traversal a node is visited after its left subtree and before its right subtree
- Application: draw a binary tree
  - x(v) = inorder rank of v
  - y(v) = depth of v

Algorithm ***inOrder***(v)
  if $left\ (v) \neq$ **null**
    $inOrder\ (left\ (v))$
  $visit(v)$
  if $right(v) \neq$ **null**
    $inOrder\ (right\ (v))$



Print Arithmetic Expressions

- Specialization of an inorder traversal
  - print operand or operator when visiting node
  - print "(" before traversing left subtree
  - print ")" after traversing right subtree



Algorithm ***printExpression***(v)

if $left\,(v) \neq$ **null**

$print($"(" $)$

$inOrder\,(left(v))$

$print(v.element\,())$

if $right(v) \neq$ **null**

$inOrder\,(right(v))$

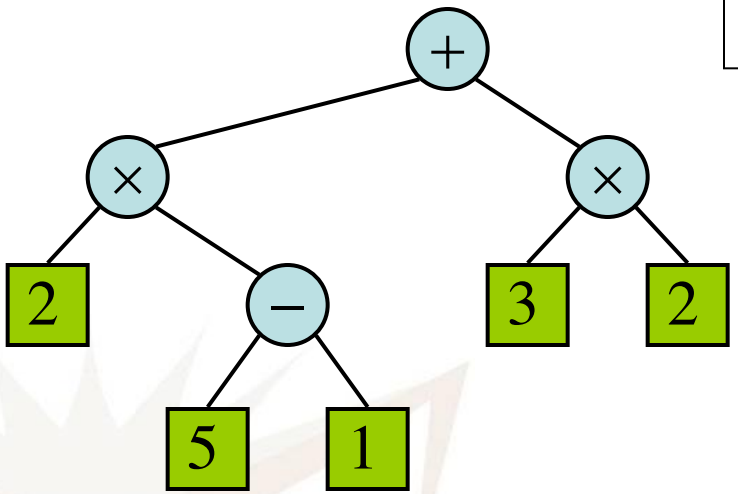
$print\,($")" $)$

$((2 \times (a - 1)) + (3 \times b))$

Evaluate Arithmetic Expressions

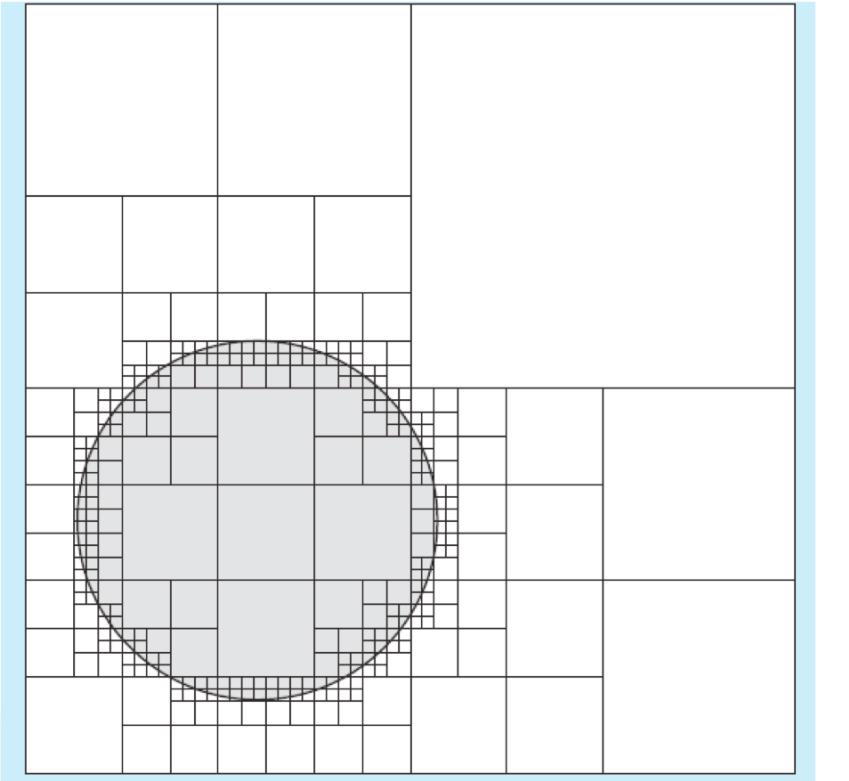
- Specialization of a postorder traversal
  - recursive method returning the value of a subtree
  - when visiting an internal node, combine the values of the subtrees

Algorithm *evalExpr*(*v*)
  if *isExternal* (*v*)
    return *v.element* ()
  else
    $x \leftarrow$ *evalExpr*(*left*(*v*))
    $y \leftarrow$ *evalExpr*(*right*(*v*))
    $\lozenge \leftarrow$ operator stored at *v*
    return $x \lozenge y$



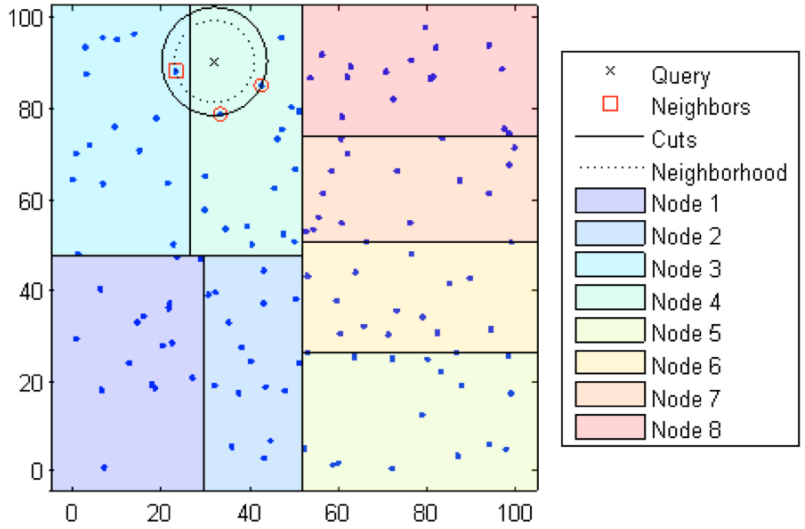
Space Partitioning

Quad-tree/Oct-tree



Space Partitioning

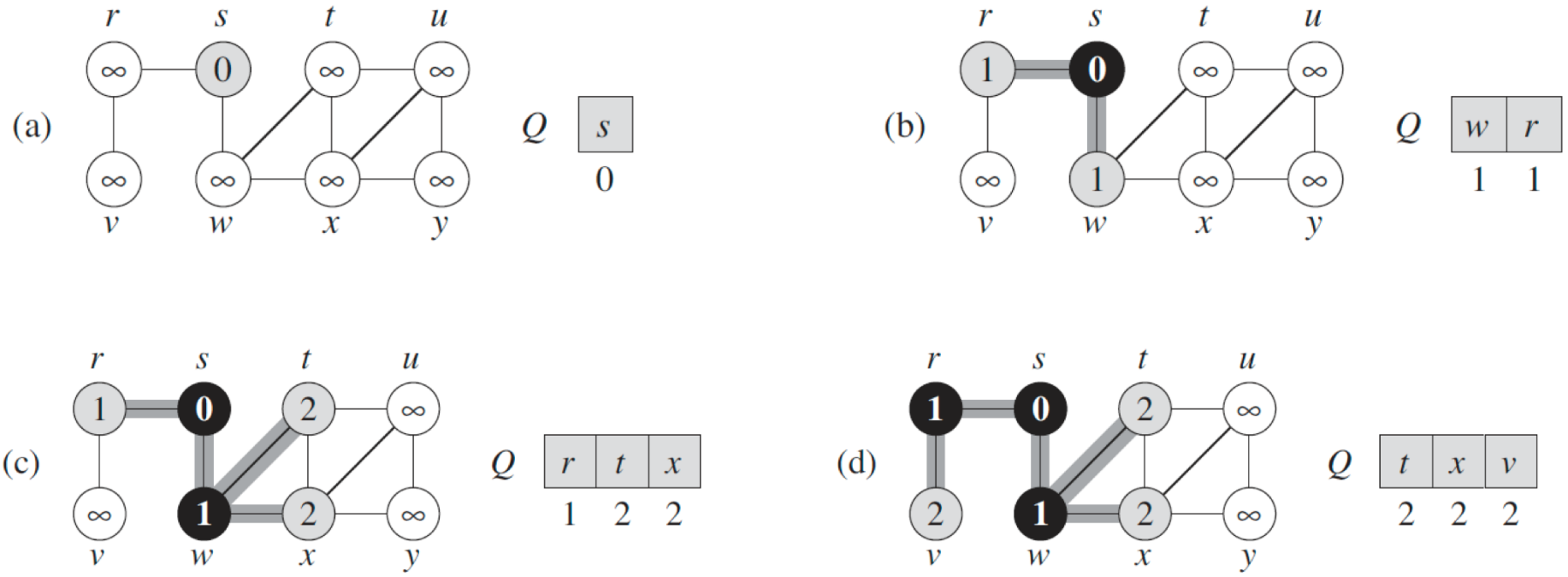
Kd-tree



Breadth-first Search Exploration of a Graph

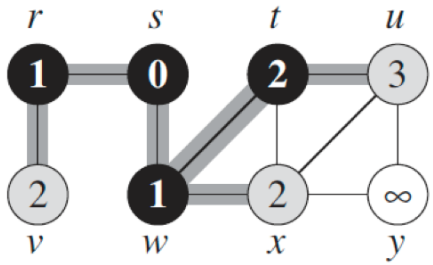
```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2       u.color = WHITE
 3       u.d = ∞
 4       u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11       u = DEQUEUE(Q)
12       for each v ∈ G.Adj[u]
13            if v.color == WHITE
14                 v.color = GRAY
15                 v.d = u.d + 1
16                 v.π = u
17                 ENQUEUE(Q, v)
18       u.color = BLACK
```

Breadth-first Search Exploration of a Graph



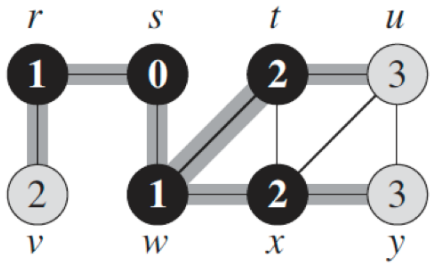
Breadth-first Search Exploration of a Graph

(e)



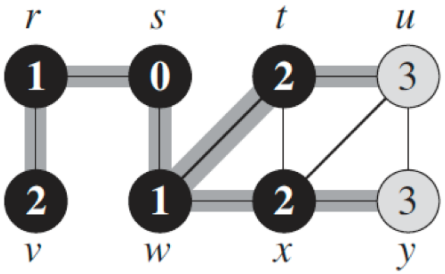
Q | x | v | u |
2 2 3

(f)



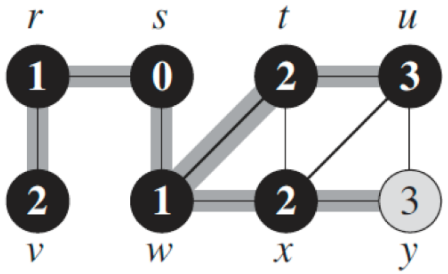
Q | v | u | y |
2 3 3

(g)



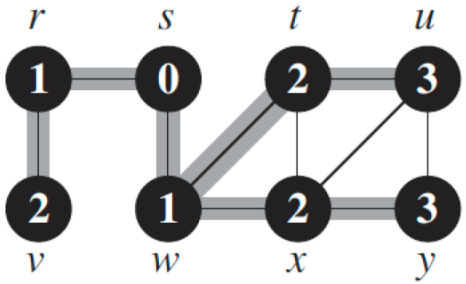
Q | u | y |
3 3

(h)



Q | y |
3

(i)



Q $\emptyset$

Shortest Path

- $\delta(u, v)$ - shortest distance (edge count) between nodes u and v
- Theorem: BFS trees discovers the shortest distance between the source vertex and all other vertices in the graph
- If the BFS trees is rooted at s, $\delta(s\ u)\ \forall\ u \in V$ is the distance induced by the BFS tree (in the tree the path form s to u is unique by definition of a tree)