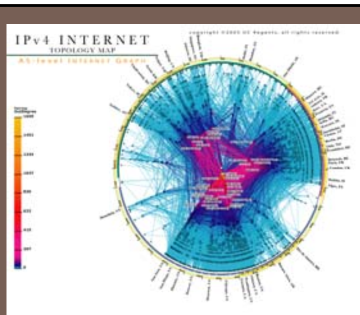


GRAPHS

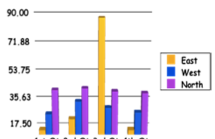
Lecture 19
CS2110 – Spring 2013

Announcements

2

- Prelim 2: Two and a half weeks from now
  - Tuesday, April16, 7:30-9pm, Statler
- Exam conflicts?
  - We need to hear about them and can arrange a makeup
  - It would be the same day but 5:30-7:00
- Old exams available on the course website

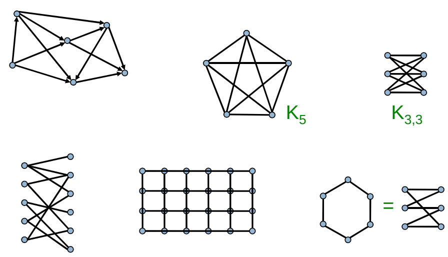
These are not Graphs



...not the kind we mean, anyway

These are Graphs

4



Applications of Graphs

5

- Communication networks
- Routing and shortest path problems
- Commodity distribution (flow)
- Traffic control
- Resource allocation
- Geometric modeling
- ...

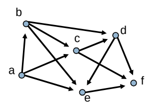
Graph Definitions

6

- A directed graph (or digraph) is a pair (V, E) where
  - V is a set
  - E is a set of ordered pairs (u,v) where u,v □ V
    - Usually require u v (i.e., no self-loops)
- An element of V is called a vertex (pl. vertices) or node
- An element of E is called an edge or arc
- |V| = size of V, often denoted n
- |E| = size of E, often denoted m

Example Directed Graph (Digraph)

7



V = {a,b,c,d,e,f}
E = {(a,b), (a,c), (a,e), (b,c), (b,d), (b,e), (c,d), (c,f), (d,e), (d,f), (e,f)}

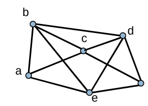
|V| = 6, |E| = 11

Example *Undirected* Graph

8

An *undirected graph* is just like a directed graph, except the edges are *unordered pairs* (*sets*) {u,v}

Example:

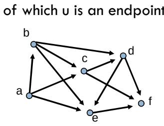


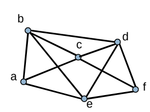
V = {a,b,c,d,e,f}
E = {{a,b}, {a,c}, {a,e}, {b,c}, {b,d}, {b,e}, {c,d}, {c,f}, {d,e}, {d,f}, {e,f}}

Some Graph Terminology

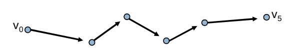
9

- Vertices u and v are called the source and sink of the directed edge (u,v), respectively
- Vertices u and v are called the endpoints of (u,v)
- Two vertices are adjacent if they are connected by an edge
- The outdegree of a vertex u in a directed graph is the number of edges for which u is the source
- The indegree of a vertex v in a directed graph is the number of edges for which v is the sink
- The degree of a vertex u in an undirected graph is the number of edges of which u is an endpoint

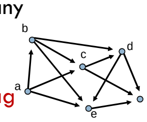




More Graph Terminology

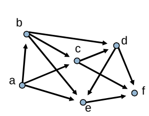


- A path is a sequence $v_0,v_1,v_2,\dots,v_p$ of vertices such that $(v_i,v_{i+1}) \in E$, $0 \le i \le p-1$
- The length of a path is its number of edges
  - In this example, the length is 5
- A path is simple if it does not repeat any vertices
- A cycle is a path $v_0,v_1,v_2,\dots,v_p$ such that $v_0 = v_p$
- A cycle is simple if it does not repeat any vertices except the first and last
- A graph is acyclic if it has no cycles
- A directed acyclic graph is called a dag



Is This a Dag?

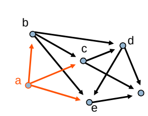
11



- Intuition:
  - If it's a dag, there must be a vertex with indegree zero – why?
- This idea leads to an algorithm
  - A digraph is a dag if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is This a Dag?

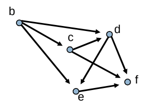
12



- Intuition:
  - If it's a dag, there must be a vertex with indegree zero – why?
- This idea leads to an algorithm
  - A digraph is a dag if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is This a Dag?

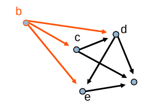
13



- Intuition:
  - If it's a dag, there must be a vertex with indegree zero – why?
- This idea leads to an algorithm
  - A digraph is a dag if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is This a Dag?

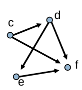
14



- Intuition:
  - If it's a dag, there must be a vertex with indegree zero – why?
- This idea leads to an algorithm
  - A digraph is a dag if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is This a Dag?

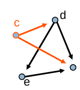
15



- Intuition:
  - If it's a dag, there must be a vertex with indegree zero – why?
- This idea leads to an algorithm
  - A digraph is a dag if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is This a Dag?

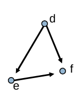
16



- Intuition:
  - If it's a dag, there must be a vertex with indegree zero – why?
- This idea leads to an algorithm
  - A digraph is a dag if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is This a Dag?

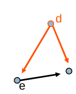
17



- Intuition:
  - If it's a dag, there must be a vertex with indegree zero – why?
- This idea leads to an algorithm
  - A digraph is a dag if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is This a Dag?

18



- Intuition:
  - If it's a dag, there must be a vertex with indegree zero – why?
- This idea leads to an algorithm
  - A digraph is a dag if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is This a Dag?

19

- Intuition:
  - If it's a dag, there must be a vertex with indegree zero – why?
- This idea leads to an algorithm
  - A digraph is a dag if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is This a Dag?

20

- Intuition:
  - If it's a dag, there must be a vertex with indegree zero – why?
- This idea leads to an algorithm
  - A digraph is a dag if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Is This a Dag?

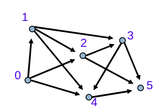
21

- Intuition:
  - If it's a dag, there must be a vertex with indegree zero – why?
- This idea leads to an algorithm
  - A digraph is a dag if and only if we can iteratively delete indegree-0 vertices until the graph disappears

Topological Sort

22

- We just computed a topological sort of the dag
  - This is a numbering of the vertices such that all edges go from lower- to higher-numbered vertices

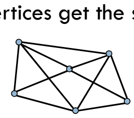


- Useful in job scheduling with precedence constraints

Graph Coloring

23

- A coloring of an undirected graph is an assignment of a color to each node such that no two adjacent vertices get the same color

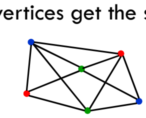


- How many colors are needed to color this graph?

Graph Coloring

24

- A coloring of an undirected graph is an assignment of a color to each node such that no two adjacent vertices get the same color

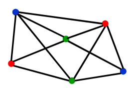


- How many colors are needed to color this graph?
  - 3

An Application of Coloring

25

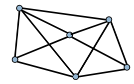
- Vertices are jobs
- Edge (u,v) is present if jobs u and v each require access to the same shared resource, and thus cannot execute simultaneously
- Colors are time slots to schedule the jobs
- Minimum number of colors needed to color the graph = minimum number of time slots required



Planarity

26

- A graph is planar if it can be embedded in the plane with no edges crossing

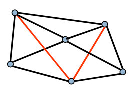


- Is this graph planar?

Planarity

27

- A graph is planar if it can be embedded in the plane with no edges crossing

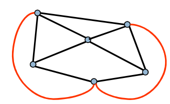


- Is this graph planar?
  - Yes

Planarity

28

- A graph is planar if it can be embedded in the plane with no edges crossing

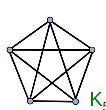


- Is this graph planar?
  - Yes

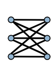
Detecting Planarity

29

- Kuratowski's Theorem



K_5



$K_{3,3}$

- A graph is planar if and only if it does not contain a copy of K_5 or $K_{3,3}$ (possibly with other nodes along the edges shown)

The Four-Color Theorem

30

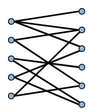
Every planar graph is 4-colorable

(Appel & Haken, 1976)

Bipartite Graphs

31

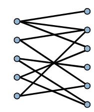
- A directed or undirected graph is bipartite if the vertices can be partitioned into two sets such that all edges go between the two sets



Bipartite Graphs

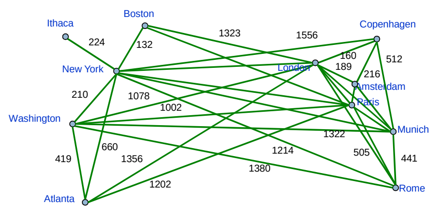
32

- The following are equivalent
  - G is bipartite
  - G is 2-colorable
  - G has no cycles of odd length



Traveling Salesperson

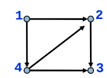
33



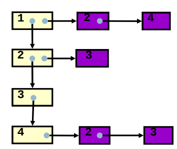
- Find a path of minimum distance that visits every city

Representations of Graphs

34



Adjacency List



Adjacency Matrix

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

Adjacency Matrix or Adjacency List?

35

- n = number of vertices
- m = number of edges
- d(u) = degree of u = number of edges leaving u
- Adjacency Matrix
  - Uses space $O(n^2)$
  - Can iterate over all edges in time $O(n^2)$
  - Can answer "Is there an edge from u to v?" in O(1) time
  - Better for dense graphs (lots of edges)

- Adjacency List
  - Uses space O(m+n)
  - Can iterate over all edges in time O(m+n)
  - Can answer "Is there an edge from u to v?" in O(d(u)) time
  - Better for sparse graphs (fewer edges)

Graph Algorithms

36

- Search
  - depth-first search
  - breadth-first search
- Shortest paths
  - Dijkstra's algorithm
- Minimum spanning trees
  - Prim's algorithm
  - Kruskal's algorithm

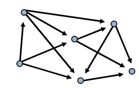
Depth-First Search

37

- Follow edges depth-first starting from an arbitrary vertex r, using a stack to remember where you came from
- When you encounter a vertex previously visited, or there are no outgoing edges, retreat and try another path
- Eventually visit all vertices reachable from r
- If there are still unvisited vertices, repeat
- O(m) time

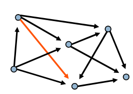
Depth-First Search

38



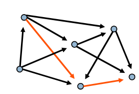
Depth-First Search

39



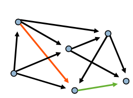
Depth-First Search

40



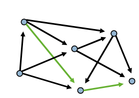
Depth-First Search

41



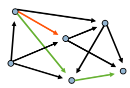
Depth-First Search

42



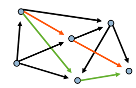
Depth-First Search

43



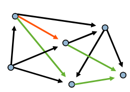
Depth-First Search

44



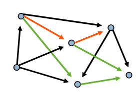
Depth-First Search

45



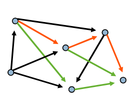
Depth-First Search

46



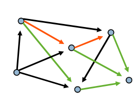
Depth-First Search

47



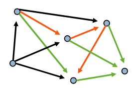
Depth-First Search

48



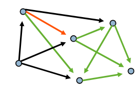
Depth-First Search

49



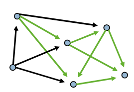
Depth-First Search

50



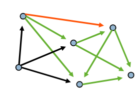
Depth-First Search

51



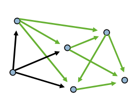
Depth-First Search

52



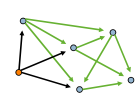
Depth-First Search

53



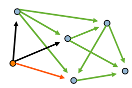
Depth-First Search

54



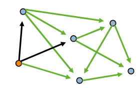
Depth-First Search

55



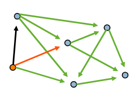
Depth-First Search

56



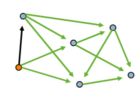
Depth-First Search

57



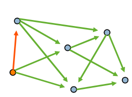
Depth-First Search

58



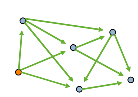
Depth-First Search

59



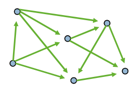
Depth-First Search

60



Depth-First Search

61



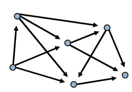
Breadth-First Search

62

- Same, except use a queue instead of a stack to determine which edge to explore next
  - Recall: A stack is last-in, first-out (LIFO)
  - A queue is first-in, first-out (FIFO)

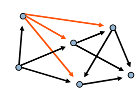
Breadth-First Search

63



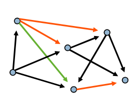
Breadth-First Search

64



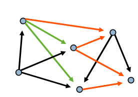
Breadth-First Search

65



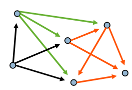
Breadth-First Search

66



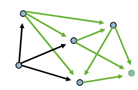
Breadth-First Search

67



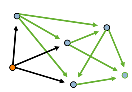
Breadth-First Search

68



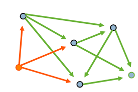
Breadth-First Search

69



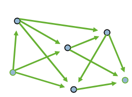
Breadth-First Search

70



Breadth-First Search

71



Summary

72

- We've seen an introduction to graphs and will return to this topic next week on Tuesday
  - Definitions
  - Testing for a dag
  - Depth-first and breadth-first search
- On Thursday Ken and David will be out of town.
  - Dexter Kozen will do a lecture on induction
  - We use induction to prove properties of graphs and graph algorithms, so the fit is good