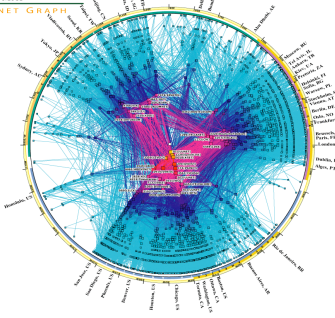
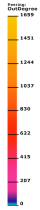


IPv4 INTERNET TOPOLOGY MAP
AS-Level INTERNET GRAPH



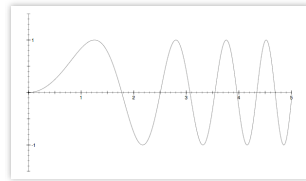
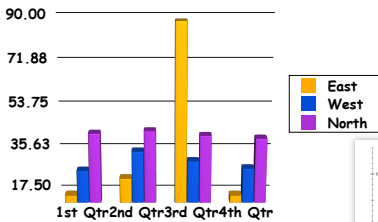
Graphs

Lecture 18
CS2110 – Fall 2008

Announcements

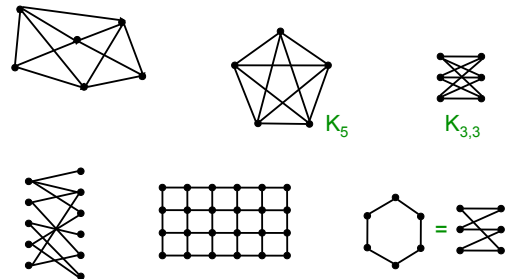
- Prelim 2
 - Tuesday, Nov 18, 7:30-9pm
 - Uris Auditorium
- Exam conflicts
 - Email Kelly Patwell ASAP
- Old exams are available for review on the course website

These are not Graphs



...not the kind we mean, anyway

These are Graphs



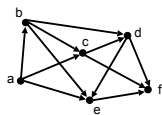
Applications of Graphs

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

Graph Definitions

- 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 $u, v \in V$
 - Usually require $u \neq v$ (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)



$$V = \{a,b,c,d,e,f\}$$

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

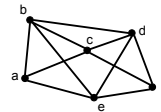
$$|V| = 6, |E| = 11$$

7

Example *Undirected* Graph

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

Example:



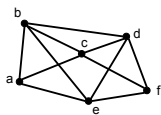
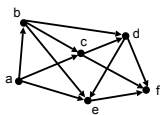
$$V = \{a,b,c,d,e,f\}$$

$$E = \{\{a,b\}, \{a,c\}, \{a,e\}, \{b,c\}, \{b,d\}, \{b,e\}, \{c,d\}, \{c,f\}, \{d,e\}, \{d,f\}, \{e,f\}\}$$

8

Some Graph Terminology

- 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

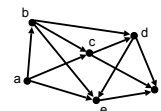


9

More Graph Terminology

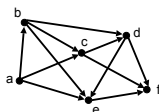


- A **path** is a sequence $v_0, v_1, v_2, \dots, v_p$ of vertices such that $(v_i, v_{i+1}) \in E, 0 \leq i \leq p-1$
- The **length** of a path is its number of edges
 - In this example, the length is 5
- A path is **simple** if it does not repeat any vertices
- A **cycle** is a path $v_0, v_1, v_2, \dots, v_p$ 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**



10

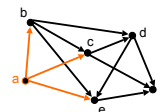
Is This a Dag?



- 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

11

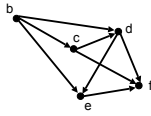
Is This a Dag?



- 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

12

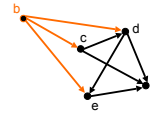
Is This a Dag?



- 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

13

Is This a Dag?



- 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

14

Is This a Dag?



- 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

15

Is This a Dag?



- 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

16

Is This a Dag?



- 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

17

Is This a Dag?



- 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

18

Is This a Dag?



- 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

19

Is This a Dag?



- 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

20

Is This a Dag?

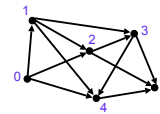


- Intuition:
 - If it's a dag, there must be a vertex with indegree zero – why?
- This idea leads to an algorithm
 - A digraph is a dag if and only if we can iteratively delete indegree-0 vertices until the graph disappears

21

Topological Sort

- 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

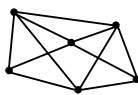


Useful in job scheduling with precedence constraints

22

Graph Coloring

- 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

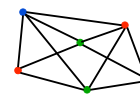


- How many colors are needed to color this graph?

23

Graph Coloring

- 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

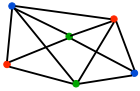


- How many colors are needed to color this graph?
 - 3

24

An Application of Coloring

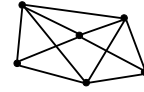
- Vertices are jobs
- Edge (u,v) is present if jobs u and v each require access to the same shared resource, and thus cannot execute simultaneously
- Colors are time slots to schedule the jobs
- Minimum number of colors needed to color the graph = minimum number of time slots required



25

Planarity

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

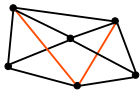


- Is this graph planar?

26

Planarity

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

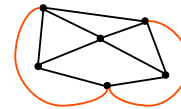


- Is this graph planar?
 - Yes

27

Planarity

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

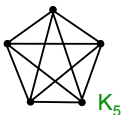


- Is this graph planar?
 - Yes

28

Detecting Planarity

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)

29

The Four-Color Theorem

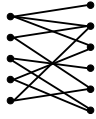
Every planar graph is 4-colorable
(Appel & Haken, 1976)



30

Bipartite Graphs

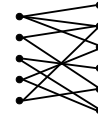
- A directed or undirected graph is **bipartite** if the vertices can be partitioned into two sets such that all edges go between the two sets



31

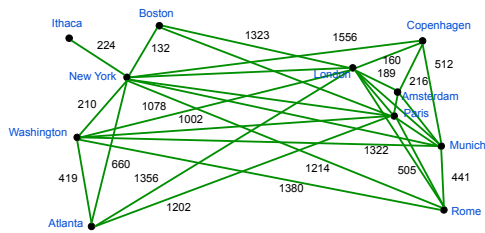
Bipartite Graphs

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



32

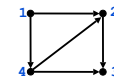
Traveling Salesperson



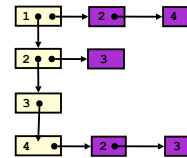
- Find a path of minimum distance that visits every city

33

Representations of Graphs



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

34

Adjacency Matrix or Adjacency List?

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)

35

Graph Algorithms

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

36

Depth-First Search

- 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

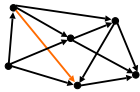
37

Depth-First Search



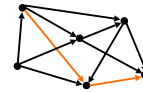
38

Depth-First Search



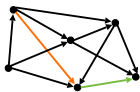
39

Depth-First Search



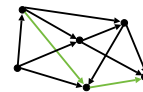
40

Depth-First Search



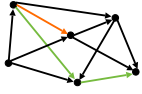
41

Depth-First Search



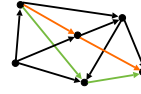
42

Depth-First Search



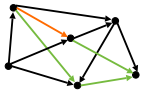
43

Depth-First Search



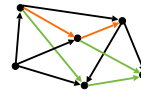
44

Depth-First Search



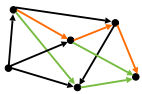
45

Depth-First Search



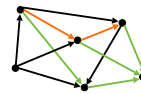
46

Depth-First Search



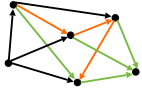
47

Depth-First Search



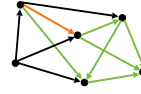
48

Depth-First Search



49

Depth-First Search



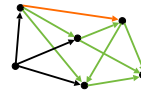
50

Depth-First Search



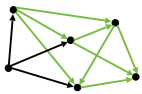
51

Depth-First Search



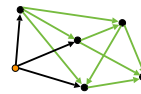
52

Depth-First Search



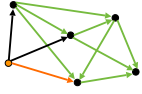
53

Depth-First Search



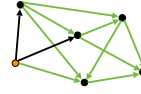
54

Depth-First Search



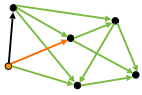
55

Depth-First Search



56

Depth-First Search



57

Depth-First Search



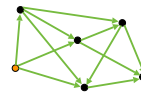
58

Depth-First Search



59

Depth-First Search



60

Depth-First Search



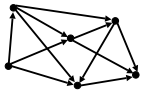
61

Breadth-First Search

- Same, except use a queue instead of a stack to determine which edge to explore next

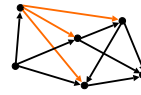
62

Breadth-First Search



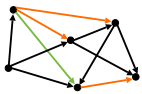
63

Breadth-First Search



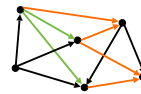
64

Breadth-First Search



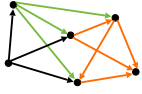
65

Breadth-First Search



66

Breadth-First Search



67

Breadth-First Search



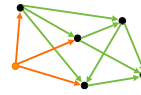
68

Breadth-First Search



69

Breadth-First Search



70

Breadth-First Search



71

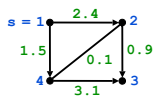
Shortest Paths

Suppose you have a US Airways route map with intercity distances. You want to know the shortest distance from Ithaca to every city served by US Airways.

This is known as the *single-source shortest path problem*.

72

Shortest Paths



Digraph with edge weights

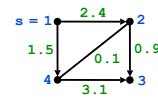
	1	2	3	4
1	0	2.4	∞	1.5
2	∞	0	0.9	∞
3	∞	∞	0	∞
4	∞	0.1	3.1	0

Corresponding matrix

Single-source shortest path problem: Given a graph with edge weights $w(u,v)$ and a designated vertex s , find the shortest path from s to every other vertex (length of a path = sum of edge weights)

73

Shortest Paths

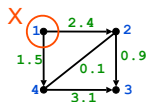


• Let $d(s,u)$ denote the distance (length of shortest path) from s to u . In this example,

- $d(1,1) = 0$
- $d(1,2) = 1.6$
- $d(1,3) = 2.5$
- $d(1,4) = 1.5$

74

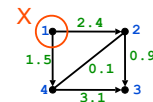
Dijkstra's Algorithm



- Let $X = \{s\}$
 - X is the set of nodes for which we have already determined the shortest path
- For each node $u \notin X$, define $D(u) = w(s,u)$
 - $D(2) = 2.4$
 - $D(3) = \infty$
 - $D(4) = 1.5$

75

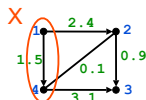
Dijkstra's Algorithm



- Find $u \notin X$ such that $D(u)$ is minimum, add it to X
 - at that point, $d(s,u) = D(u)$
- For each node $v \notin X$ such that $(u,v) \in E$, if $D(u) + w(u,v) < D(v)$, set $D(v) = D(u) + w(u,v)$
 - $D(2) = 2.4$
 - $D(3) = \infty$
 - $D(4) = 1.5$

76

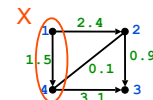
Dijkstra's Algorithm



- Find $u \notin X$ such that $D(u)$ is minimum, add it to X
 - at that point, $d(s,u) = D(u)$ $u = 4$
- For each node $v \notin X$ such that $(u,v) \in E$, if $D(u) + w(u,v) < D(v)$, set $D(v) = D(u) + w(u,v)$
 - $D(2) = 2.4$
 - $D(3) = \infty$
 - $D(4) = 1.5 = d(1,4)$

77

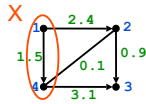
Dijkstra's Algorithm



- Find $u \notin X$ such that $D(u)$ is minimum, add it to X
 - at that point, $d(s,u) = D(u)$ $u = 4$
- For each node $v \notin X$ such that $(u,v) \in E$, if $D(u) + w(u,v) < D(v)$, set $D(v) = D(u) + w(u,v)$
 - $D(2) = ~~2.4~~ 1.6$
 - $D(3) = ~~\infty~~ 4.6$
 - $D(4) = 1.5 = d(1,4)$

78

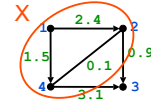
Dijkstra's Algorithm



- Find $u \notin X$ such that $D(u)$ is minimum, add it to X
 - at that point, $d(s,u) = D(u)$
- For each node $v \notin X$ such that $(u,v) \in E$, if $D(u) + w(u,v) < D(v)$, set $D(v) = D(u) + w(u,v)$
 - $D(2) = \cancel{2.4} \quad 1.6$
 - $D(3) = \cancel{4.6} \quad 4.6$
 - $D(4) = 1.5 = d(1,4)$

79

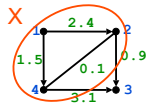
Dijkstra's Algorithm



- Find $u \notin X$ such that $D(u)$ is minimum, add it to X
 - at that point, $d(s,u) = D(u) \quad u = 2$
- For each node $v \notin X$ such that $(u,v) \in E$, if $D(u) + w(u,v) < D(v)$, set $D(v) = D(u) + w(u,v)$
 - $D(2) = \cancel{2.4} \quad 1.6 = d(1,2)$
 - $D(3) = \cancel{4.6} \quad 4.6$
 - $D(4) = 1.5 = d(1,4)$

80

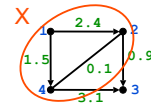
Dijkstra's Algorithm



- Find $u \notin X$ such that $D(u)$ is minimum, add it to X
 - at that point, $d(s,u) = D(u) \quad u = 2$
- For each node $v \notin X$ such that $(u,v) \in E$, if $D(u) + w(u,v) < D(v)$, set $D(v) = D(u) + w(u,v)$
 - $D(2) = \cancel{2.4} \quad 1.6 = d(1,2)$
 - $D(3) = \cancel{4.6} \quad \cancel{4.6} \quad 2.5$
 - $D(4) = 1.5 = d(1,4)$

81

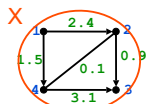
Dijkstra's Algorithm



- Find $u \notin X$ such that $D(u)$ is minimum, add it to X
 - at that point, $d(s,u) = D(u)$
- For each node $v \notin X$ such that $(u,v) \in E$, if $D(u) + w(u,v) < D(v)$, set $D(v) = D(u) + w(u,v)$
 - $D(2) = \cancel{2.4} \quad 1.6 = d(1,2)$
 - $D(3) = \cancel{4.6} \quad \cancel{4.6} \quad 2.5$
 - $D(4) = 1.5 = d(1,4)$

82

Dijkstra's Algorithm



- Find $u \notin X$ such that $D(u)$ is minimum, add it to X
 - at that point, $d(s,u) = D(u) \quad u = 3$
- For each node $v \notin X$ such that $(u,v) \in E$, if $D(u) + w(u,v) < D(v)$, set $D(v) = D(u) + w(u,v)$
 - $D(2) = \cancel{2.4} \quad 1.6 = d(1,2)$
 - $D(3) = \cancel{4.6} \quad \cancel{4.6} \quad 2.5 = d(1,3)$
 - $D(4) = 1.5 = d(1,4)$

83

Dijkstra's Algorithm

Proof of correctness – show that the following are invariants of the loop:

- For $u \in X$, $D(u) = d(s,u)$
- For $u \in X$ and $v \notin X$, $d(s,u) \leq d(s,v)$
- For all u , $D(u)$ is the length of the shortest path from s to u such that all nodes on the path (except possibly u) are in X

Implementation:

- Use a **priority queue** for the nodes not yet taken – priority is $D(u)$

84

Complexity

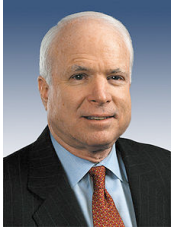
- Every edge is examined once when its source is taken into X
- A vertex may be placed in the priority queue multiple times, but at most once for each incoming edge
- Number of insertions and deletions into priority queue = $m + 1$, where $m = |E|$
- Total complexity = $O(m \log m)$

85

Conclusion

- There are faster but much more complicated algorithms for single-source, shortest-path problem that run in time $O(n \log n + m)$ using something called *Fibonacci heaps*
- It is important that all edge weights be nonnegative – Dijkstra's algorithm does not work otherwise, we need a more complicated algorithm called *Warshall's algorithm*
- Learn about this and more in CS4820

86



Don't forget to vote!

