Lecture 17: Graphs

Readings

• Chapter 28 Graphs
• Chapter 29 Graph Implementations

These are not Graphs

These are Graphs

...not the kind we mean, anyway

Applications of Graphs

• Communication networks; social networks
• Routing and shortest path problems
• Commodity distribution (network flow)
• Traffic control
• Resource allocation
• Numerical linear algebra (sparse matrices)
• Geometric modeling (meshes, topology, …)
• Image processing (e.g., graph cuts)
• Computer animation (e.g., motion graphs)
• Systems biology
• …

Applications of Graphs

• Communication networks; social networks
• Routing and shortest path problems
• Commodity distribution (network flow)
• Traffic control
• Resource allocation
• Numerical linear algebra (sparse matrices)
• Image processing (e.g., graph cuts)
• Systems biology
• Geometric modeling (meshes, topology, …)
• …
Applications of Graphs

- Communication networks, social networks
- Routing and shortest path problems
- Congestion distribution (network flow)
- Traffic control
- Resource allocation
- Numerical linear algebra (sparse matrices)
- Image processing (e.g., graph cuts)
- System biology
- Geometric modeling (meshes, topology, ...)


http://semiengineering.com/wp-content/uploads/2014/03/Fig13_Sparse_Matrix_reordering.png


Subdivision Level 0

Subdivision Level 1
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** or **node**
- An element of \(E\) is called an **edge** or **arc**
- \(|V| = \text{size of } V\), often denoted \(n\)
- \(|E| = \text{size of } E\), often denoted \(m\)

**Example Directed Graph (Digraph)**

\[ V = \{a, b, c, d, e, f\} \]
\[ E = \{(a, b), (a, c), (a, e), (b, c), (b, d), (b, e), (c, d), (c, f), (d, f), (e, f)\} \]
\[ |V| = 6, |E| = 11 \]

**Example Undirected Graph**

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

**Example:**

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

**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

**More Graph Terminology**

- A **path** is a sequence \(v_0, v_1, v_2, ..., v_p\) of vertices such that \((v_i, v_j) \in E\), \(0 \leq i, j \leq p\)
- 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 \(v_0, v_1, v_2, ..., v_p\) such that \(v_p = v_0\)
- 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?**

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

Example of Topological Sort

- Starcraft II build order: Roach Rush

  Possible Topological Sorts
  1. Hatch, SPool, RWarren, Gas, Roaches
  2. Hatch, SPool, Gas, RWarren, Roaches
  3. Hatch, Gas, SPool, RWarren, Roaches

  Timing is everything though :)

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?

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.

Planarity

- A graph is **planar** if it can be embedded in the plane with no edges crossing.
- Is this graph planar?
Planarity

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

- Is this graph planar?
  - Yes

Detecting Planarity

- Kuratowski’s Theorem

- 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)

Four-Color Theorem:

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

Another 4-colored planar graph

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

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

- Find a path of minimum distance that visits every city

Representations of Graphs

- Adjacency List
  - Uses space $O(m+n)$
  - Can iterate over all edges in time $O(m+n)$
  - Better for sparse graphs (fewer edges)

- 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)

Graph Algorithms

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

Adjacency Matrix or Adjacency List?

- Definitions
  - $n$ = 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)

Readings

- Chapter 28 Graphs
- Chapter 29 Graph Implementations