# Week 1

Undirected Graphs

Set of vertices connected pairwise by edges

**Examples**

* Protein-protein interaction network
* Internet
* Science clickstreams (scientific social map)
* Facebook
* Communication patterns in corporate
* FCC lobbying coalitions

**Path**: Sequence of vertices connected by edges

**Cycle**: Path whose first and last vertices are the same

**Euler Tour**: A cycle that uses each edge exactly once

**Hamilton Tour**: A cycle that uses each vertex exactly once

Two vertices are connected if there is a path between them

Length of a path

Degree of a vertex

Shortest path between two vertices

Self-loops

Parallel edges

Undirected Graph API

Set of edges graph representation

Adjacency matrix representation

Adjacency list graph representation: maintain vertex-indexed array of lists (‘bags’)

Graphs

Paths

Depth First Search

Maze exploration: vertex (intersection), edge (passage)

Mark a vertex ‘v’ as visited; recursively visit all unmarked vertices w adjacent to v

Breadth First Search

Not a recursive algorithm

Repeat until queue (FIFO) is empty

* Remove vertex v from the queue
* Add to queue all unmarked vertices adjacent to v and mark them

**Application**

* Shortest path problem (that is, determine fewest number of edges) from vertex s to all other vertices in a graph in time proportional to E + V.
* Routing: fewest number of hops in a communication network
* Kevin Bacon numbers: vertices are for actors and movies.

Connected Components

A connected component is a maximal set of connected vertices.

Running time: linear

**Goal**: Partition the vertices into connected components

Initialize all vertices v as unmarked. For each unmarked vertex v, run DFS to identify all vertices discovered as part of the same component

Graphical Challenges

1. Is a graph bi-partite? It is possible to divide the vertices into two sub-sets such that every edge connects a vertex in one subset with a vertex in another subset.

**Example**: Dating graph

**Solution**: Simple DFS based approach

1. Finding a cycle in a graph

**Solution**: Simple DFS based approach

1. Seven Bridges of Konigsberg

Yes, if and only if: connected and all vertices have even degree

1. Find a cycle that uses every edge exactly once
2. Find a cycle that visits every vertex exactly once / travelling sales person problem

It is a classical NP-complete problem. Nobody knows an efficient algorithm for this problem

Hamiltonian cycle

1. Are two graphs identical except for vertex names?

Graph isomorphism is a longstanding open problem

1. Can you lay out a graph in the plane without crossing edges?

Linear time DFS based planarity algorithm

Directed Graphs

Set of vertices connected pairwise by directed edges

|  |  |  |
| --- | --- | --- |
| **Application** | **Vertex** | **Edge** |
| **Road network** | Intersection | One way street |
| **Political Blogosphere** | Blog | Interlinks (red & blue) |
| **Overnight interbank loan graph** | Bank | Overnight loan |
| **Implication Graph** | Variable | Logical implication |
| **Combinational Circuit** | Logical gate | Wire |
| **WordNet Graph** | Synset | Hypernym relationship |
| **Web** | Web page | Hyperlink |
| **Program Control-Flow Analysis** | Basic block of instructions | Jump  (dead-code elimination, infinite loop detection) |
| **Mark-sweep Garbage collector** | Object | Reference  (Roots: objects directly accessible by a program.  Reachable objects: indirectly accessible) |
|  |  |  |

Is there a directed path from s to t?

What is the shortest directed path from s to t?

Can you draw a digraph so that all edges point upwards?

Is there a directed path between all pairs of vertices?

What is the importance of a web page? [**Page rank**]

Digraph API

Only one edge for each directed link

Directed Search

Find all vertices reachable from s along a directed path

**Depth first search**

It is a digraph algorithm, same as used for undirected graph.

* Reachability
* Path finding
* Topological sort
* Direct cycle detection
* 2-satisfiability
* Directed Euler path
* Strongly connected components

**Breadth first Search**

It is a digraph algorithm, same as used for undirected graph. It computes shortest path from s to all other vertices in the digraph in time proportional to “E + V”.

**Multiple-source shortest paths**

Use BFS, but by initialize by enqueue-ing all source vertices

**Web crawler**

Crawl web, starting from some root web page

* Choose root web page as source
* Maintain a Queue of website to explore
* Maintain a site of discovered websites
* De-queue the next website and en-queue websites to which it links

Topological sort

Given a set of tasks (vertex) to be completed with precedence constraints (edges)

Works on a DAG (Directed Acyclic Graph); Redraw DAG so that all edges point upwards

Run depth first search, return vertices in reverse post order

Directed cycle

**Cyclic inheritance**: Java compiler does cycle detection

**Spreadsheet recalculation**: Microsoft Excel

Strongly-connected components

Vertices v & w are strongly connected if there is a directed path from v to w and a directed path from w to v.

A strong component is a maximal subset of strongly connected vertices

|  |  |  |  |
| --- | --- | --- | --- |
| **Application** | **Vertex** | **Edge** | **Connectivity paradigm** |
| **Food Web Graph** | Species | From Producer to consumer | Subset of species with common energy flow |
| **Software module dependency graph** | Software module | From module to dependency | Subset of mutually interacting modules so that we can package strong components together |

Strong components in G are same as G reserve

Kosaraju-Sharir Algorithm

Compute topological order (reverse post-order) in kernel DAG; and run DFS, considering vertices in reverse topological order.

* Phase I: Run DFS on reverse graph to compute reverse post-order
* Phase II: Run DFS on graph, considering vertices in order given by first DFS
* Running time: O (E + V)

# Week 2

Minimum Spanning Tree

Undirected graph G with positive edge weights connected

A spanning tree of G is a sub-graph T that is connected and acyclic

**Goal**: Find minimum weight spanning tree

**Applications**:

* Bicycle routes in a city
* Arrangement of nuclei in the epithelium for cancer research (medical image research)
* Dithering
* Real time face detection

Greedy Algorithm

**Assumptions**: Edge weights are distinct; graph is connected

A cut in a graph is a partition of its vertices into two non-empty sets.

A crossing edge connects a vertex in one set with a vertex in the other

**Cut Theory**

Given any cut, the crossing edge of minimum weight is in the MST

**Algorithm**

* All edges colored gray
* Find cut with no black crossing edges; color its minimum weight edge black
* Repeat until V-1 edges are colored black

What if edge weights are not all distinct?

It would mean multiple MSTs

What if graph is not connected?

Compute minimum spanning forest = MST of each component

Edge Weighted Graph API

Edge abstraction is needed for weighted edges.

Kruskal’s Algorithm

Consider edges in ascending of order of weight; Add next edge to a tree T unless that edge would create a cycle. [Use MinPQ to put edges in sorted order and UnionFind to determine cycles].