Graphs

Yu-Tai Ching
Department of Computer Science
National Chiao Tung University

- Real world problems can be modeled as a graph,
- First example, the Königsberg bridge problem.
- Figure 6.1, 4 lands interconnected by 7 bridges,
- starting from one of lands, walk through every bridge exactly once, and come back to the starting land.
- some other examples, find the shortest path. Chicago-to-Boston example,
  - model the road map as a graph
  - vertices represent intersections
  - edges represent road segments between intersections
  - edge weights represents road distances.
  - find the shortest path from a given intersection in Chicago (say Clark St. and Addison Ave.) to a given intersection in Boston (say, Brookline Ave. and Yawkey Way).

Definitions

- A graph G, consists of two sets V and E.
- V finite, nonempty set of vertices.
- E is a set of edges, each edge connects a pair of vertices, E may be empty.
- We use the notation $G = (V, E)$.
- $V(G)$ the set of vertices of G, $E(G)$ the set of edges of G.
- Undirected graph, the edge has no direction, the vertices are not ordered. (u, v) is the same as (v, u).
- A directed graph, edge is directional, $< u, v >$ is not the same as $< v, u >$.
- Figure 6.2, some examples.

Definitions

- There are no such edges (v, v) or $< v, v >$, self edge or self loop. There are no parallel edges.
- An n-vertex undirected graph, there are at most $n(n-1)/2$ edges, and it is called a complete graph.
- $(u, v) \in E(G)$, we say that u v are adjacent, and (u, v) incidents on vertices u and v.
- A path from u to v in G is a sequence of vertices u, i_1, i_2, $\cdots$, i_k, v, s.t. (u, i_1), (i_1, i_2), $\cdots$, (i_k, v) are edges in $E(G)$.
- If G is a directed graph, there is similar definition.
- The length is the number of edges on the path.
- A simple path is a path that vertices are distinct.

Definitions

- A cycle is a simple path in which the first and the last vertices are the same.
- G a undirected graph, u v are said to be connected iff there is a path from u to v.
- G a undirected graph, G is said to be ocnnected if every pair of distinct vertices, there is a path.
- A connected component, H, of an udirected graph G is a maximal connected subgraph, i.e., G contains no other subgraph that is both connected and properly contains H.
- A tree is a connected acyclic (has no cycle) graph.

Definition

- A directed graph G is said to be strongly connected iff for every pair of vertices v and v, there is a directed path from u to v and another directed path from v to u.
- A strongly connected component is a maximal subgraph that is strongly connected.
- Degree of a vertices is the number of edges incident to the vertex.
- A directed graph, there are in-degree and out-degree.
- A directed graph is called a digraph.

Graph Representation

- Adjacency Matrix: $G = (V, E)$, a graph with $n \geq 1$ vertices.
- The adjacency matrix of G is a 2-D $n \times n$ array a, $a[i][j] = 1$ iff $(i, j) \in E(G)$, ($< i, j > \in E(G)$ if G a directed graph.)
- if the graph is sparse, we still need $O(n^2)$ space for the graph.

Graph Representation

- Adjancy List: A row in the Adj. matrix becomes a list,
- nodes are nonzero entries.
- space depends on the number of edges, $O(n+e)$, n number of vertices, e number of edges.
- weighted edges, $a[i][j]$ keeps the weight (Adj. Matrix), or a node as the weight field(Adj. List).

Elementary Graph Operations

- Binary tree traversal: inorder, preorder, postorder traversals. visit the tree nodes systematically.
- Similar need in the graph: Given a graph $G = (V, E)$ and a node v in $V(G)$, we wish to visit all vertices in G that are reachable from v.
- There are two ways, the "depth-first search" and the "breadth-first search".

Depth-First Search

- Visiting the start vertex v.
- w is an unvisited vertex adjacent to v, Visiting the start vertex w.
- that means, as long as there is a way out (an unvisited vertex), visit that vertex and go deeper.
- Suppose that a vertex u is reached and all the vertices adjacent to u were visited, we back up to a visited vertex that has an unvisited vertex adjacent to it.
- Until there is no way out, check if there are unvisited vertices, if yes, we start dfs from that vertex.
- Need a stack so that you can back up.
- run time, $O(e)$ for adj. list and $O(n^2)$ for adj. matrix.

Breadth-First Search

- Put the start vertex v into a queue.
- if the queue is not empty, dequeue to get u,
- inqueue every one not visited yet into the queue.
- until the queue is empty.
- Adj. list $O(e)$, Adj. matrix $O(n^2)$.

Connected Components

- The same as the "equivalence classes"
- Find the connected component component that includes u,
- Start dfs or bfs from u.
- Find all the connected components, apply the dfs or bfs iteratively until all the components are found.
- Adj. list: to find a connected components, $O(e)$, to find all the connected components $O(n+e)$.

Spanning Tree

- Suppose that G is connected, dfs or bfs in G partition the edges into two sets, T (for tree edges) and N (for nontree edges.)
- T: edges used in graph traversal (to visit an unvisited vertex).
- N: the set fo remaining edges.
- T and all the vertices in G form the "spanning tree" of G, the tree spans the graph.
- The tree: formed by dfs is the dfs-tree; fromed by bfs is the bfs-tree. (Figure 6.17)

Biconnected Components

- G is a undirected, connected graph

Definition: A vertex v is a "articulation point" iff the deletion of v, together with deletion of all edges incident to v, leaves behind a graph that has at lease two connected components. (Figure 6.20 (a), 1, 3, 5, and 7 are articulation).

Definition: A "biconnected graph" is a connected graph that has no articulation points.

If a computer network that is biconnected, then any pair of computers still can talk to each other even one of the computer is down. The network is much more reliable.

Definition of Biconnected Component

A “biconnected component” of a connected graph G is a maximal biconnected subgraph H of G. By maximal, we mean that G contains no other subgraph that is both biconnected and properly contains H.

- 6.20 (a) contains 6 biconnected components.
- Two biconnected components of the same graph share at most one vertex (removing that breaks the graph.)
- Biconnected componets can be obtained by running the dfs to establish the dfs spanning tree.

Computing the Biconnected Components

- Figure 6.21 (a), a dfs tree rooted at 3. (a), non-tree edges are shown in broken lines.
- depth-first number, dfn: the number that the vertex is discovered.
- in dfs tree, if u is the ancester of v then dfn of u is less than the dfn of v.
- Broken edges: non-tree edges, called back edges, it goes from a node to its ancester (form a loop if the back edge is included).
- It can be shown that there are no "cross edges".

Conditions that a vertex is articulation

- Root is articulation iff the root has more than one children.
- Any other vertex u is articulation point iff it has at least one child w such that it is not possible to reach an ancestor of u using a path composed solely of w, decendents of w, and a single back edge.

Computing the Articulation Points

- Define the value *low* for each vertex.
- $low(w)$ is the lowest depth-first number that can be reached from w using a path of decendants followed by one back edge.

$$
\begin{aligned}
low(w) = \min\{ \quad & dfn(w), \\
& \min\{low(x) | x \text{ is a child of } w\}, \\
& \min\{dfn(x) | (w, x) \text{ is a back edge}\}\}
\end{aligned}
$$

Computing the Articulation Points

- u is an articulation point iff
  - u is the root of the spanning tree and has two or more children
  - u is not the root and u has a child w s.t. $low(w) \geq dfn(u)$.

dfn and *low* for the spanning tree in Fig. 6.21

vertex	0	1	2	3	4	5	6	7	8	9
dfn	5	4	3	1	2	6	7	8	10	9
low	5	1	1	1	1	6	6	6	10	9

Minimum Spanning Tree

- Given a weighted, undirected graph, the minimum cost spanning tree is a spanning tree of least cost.
- Greedy algorithm is used to solve the problem. Greedy approach: contruct the optimal solution in stages, at each stage, we make the decision that is the best at that time.
- If the problem can be solved by using greedy approach, this kind of decision making leads to the global optimal solution.

Kruskal's Algorithm

- Build a MST by adding edges to T one at a time.
- By the greedy approach, choose the edge with the least weight.
- Need to know if there is a cycle formed. If there is, drop the selected edge and choos the next one.
- Since G is connected, we need $n-1$ edges to connect the vertices in the graph. The algorithm stops when there are $n-1$ edges selected.
- Figure 6.23.
- computing time, and some implementation details.

Correctness of the Kruskal's Algorithm

Theorem 6.1:

Let G be an weighted, undirected, connected graph. Kruskal's algorithm generates a minimum-cost spanning tree.
Proof (a) Kruskal's algorithm generates a spanning tree, (b) and the cost is the least.
Assume that the graph is connected. Kruskal's algorithm adds edges one by one and always prevents forming cycle. When there are $n-1$ edges added, n vertices are connected and a spanning tree is formed.
The spanning tree has the least possible cost. Let T be the spanning tree established by using the Kruskal's algorithm and U be a minimum spanning tree. We try to argue that cost(T) should be the same as cost(U), i.e., T must be minimum spanning tree.

If T is the same as U we are done.
If T is not the same as U, there must be some edges in T but not in U. Let e be one of those having the least weight. If we put e into U, there must be a cycle formed. And in this cycle there must be an edge e' in U but not in T.
$w(e') \geq w(e)$ otherwise $w(e') < w(e)$ so that Kruskal's algorithm selected e' before selecting e. And e' should be in T.
What if we put e into U and remove e' from U? We should get another tree U' and the cost of U' is less than or equal to the cost of U. Since U is optimal, we conclude $\text{cost}(U') = \text{cost}(U)$. U' must be also optimal.
We use this kind of "cut and paste" argument to trnasform U to T and conclude T must be optimal. Thus T is minimum spanning tree.

Implementation Details

- Find the minimum weight edge.
- Form cycle?

- Use directed graph to model real world problems.
- AOV network, activity-on-Vertex, topological sort.
- AOE network, activity-on-Edge, critical path calculation, (introduce the problem, but will not talk about the algorithm.)

Course Number	Course Name	Prerequsities
C1	Programming I	None
C2	Discrete Math	None
C3	Data Structure	C1, C2
C4	Calculus	None
C5	Calculus	c4
C6	Linear Algebra	C5
C7	Analysis Of Algorithms	C3, C6
C8	Assembly Language	C3
C9	Operating System	C7, C8
C10	Programming Language	C7
C11	Compiler Design	C10
C12	Artificial Intelligence	C7
C13	Computational Theory	C7
C14	Parallel Computing	C13
C15	Numerical Analysis	C5

Definition: A Directed Graph G, vertices: tasks or activities, edges: precedence relations between tasks, an activity-on-vertex network or AOV network. (Figure 3.6(b)), courses network from Figure 3.6 (a).

Definition: Vertex i in an AOV network G is a predecessor of vertex j iff there is a direct path from vertex i to vertex j. i is immediate predecessor of j iff $< i, j >$ is an edge in G. If i is a predecessor of j, then j is a successor of i. If i is an immediate predecessor of j, then j is an immediate successor or i.

Definition: A relation $\cdot$ is transitive iff $\forall$ triple i, j, k, $i \cdot j$ and $j \cdot k$ implies $i \cdot k$. A relation $\cdot$ is irreflexive on the set S if for no element $x \in S$, $x \cdot x$. A precedence relation that is both transitive and irreflexive is a partial order.

- Figure 6.36, an AOV network.
- Given an AOV network, determine whether or not the precedence relation defined by the edges is irreflexive.
- Identical to determine whether or not the network contains any directed cycles.
- Whether or not the graph is acyclic graph (direct graph with no directed cycle).

A topological order is a linear ordering of the vertices of a graph such that for any vertices *i* and *j*, if *i* is a predecessor of *j* in the network, then *i* precedes *j* in the linear ordering.

- C1, C2, C4, C5, C3, C6, C8, C7, C10, C13, C12, C14, C15, C11, C9
- C4, C5, C2, C1, C6, C3, C8, C15, C7, C9, C10, C11, C12, C13, C14
- Any vertex that does not have a predecessor, the vertex can be outputed to the linear order. Figure 6.37.
- Adjacency list representation that also keep track of the number of predecessor. 6.38.

- Tasks: represented by directed edges.
- Events: represented by vertices. Events signal the completion of certain activities.
- Activities represented by edges leaving a vertex cannot be started until the event at that vertex has occurred.
- An event occurs only when all activities entering it have been completed.
- Figure 6.39 (a), an AOE network. A weighted directed graph, weight associated with an edge is the time required to perform that activity.
- The longest path, the critical path, the least possible time to complete the project.
- The critical path is the path must be paid attention to. To reduce the time required, try to reduce the time required from the critical path.