Introduction to Graph Theory, Part I

Tandy Warnow

October 17, 2018

Graphs

Tandy Warnow

Two lectures

- Graphs - basic terminology
- Trees - perhaps not what you think!
- Framing real-world problems as graph-theoretic problems: decision, optimization, and construction
- The basics of "NP-hardness"

Why graphs?

Graph algorithms are endemic – and many real world problems can be solved using off-the-shelf graph algorithms software.

Learning to communicate a real world problems as a graph-theoretic problem, and then figuring out what software to use to solve it, will make your life simpler!

On the other hand, many graph-theoretic problems are NP-hard – which in essence means you probably can't solve them exactly in polynomial time (more on this later in the course).

Knowing when a problem is NP-hard will also save you a lot of time (because you won't try to solve them).

Graph Terminology

A graph $G = (V, E)$ is an object that contains a vertex set V and an edge set E.

We also write $V(G)$ to denote the vertex set of G and $E(G)$ to denote the edge set of G.

Edges are pairs of vertices.

Most graphs we will discuss will not have directions on the edges.

However, **directed graphs** have directions on the edges (so an edge $u \to v$ from u to v is not the same as an edge $v \to u$ from v to u).

We often refer to directed graphs as **digraphs**.

Most graphs are finite (so $|V| < \infty$), but sometimes graphs are infinite.

In this class we'll only talk about finite graphs and unless otherwise specified the graphs will be undirected.

Notation for edges in undirected graphs

For undirected graphs, an edge between two vertices has no direction.

There are several ways to write an edge between vertices u and v in an undirected graph.

I tend to use (u, v) to denote the edge between u and v, and other instructors write uv. You may well find other UIUC professors writing edges using $\{u, v\}$!

1. $uv = vu$
2. $(u, v) = (v, u)$
3. $\{u, v\}$

Simple Graphs

A **self-loop** is an edge from a vertex to itself (i.e., edges (v, v)).

Parallel edges are multiple occurences of the same pair (i.e., (u, v) appearing more than once).

A **simple graph** is one that has no parallel edges and no self-loops.

Unless stated otherwise, all graphs in this class should be assumed to be finite, simple, undirected graphs.

Terminology

- For a given graph G, if $(v, w) \in E$, then v and w are said to be **adjacent** or **neighbors**.
- The **degree** of a vertex v in a graph is the number of its neighbors, and is denoted $deg(v)$.
- The **endpoints** of an edge $e = (v, w)$ are v and w.
- Two edges are said to be **incident** if they have a common endpoint.
- The **empty graph** has no vertices (and hence obviously no edges).
- Some people say "vertices" and some people say "nodes", and these refer to the same thing. (The singular version of "vertices" is "vertex".)

Terminology, cont.

- A **walk** in $G = (V, E)$ is a sequence of vertices $v_1, v_2, \ldots, v_k$ s.t. $(v_i, v_{i+1}) \in E$ for all $i = 1, 2, \ldots, k-1$; note that vertices can be repeated.
- A **path** in $G = (V, E)$ is a path in which vertices cannot be repeated.
- A graph is **connected** if you can go between any two vertices via some path.
- A **component** in a graph is a maximal subgraph that is connected.
- A **cycle** in a graph is a path that starts and ends at the same node, and doesn't repeat any other node.
- An **acyclic graph** is a graph that has no cycles.
- A **tree** is a connected acyclic graph.

Terminology, cont.

- A **clique** is a set of vertices that are pairwise adjacent.
- An **independent set** is a set of vertices where no two vertices are adjacent.
- A **proper vertex coloring** is an assignment of colors to the vertices so that no two adjacent vertices are assigned the same color.
- A **matching** in a graph is a subset of the edges that share no endpoints (equivalently, no two edges in the subset are incident with each other).
- A **dominating set** is a subset A of the vertices so that every other vertex is adjacent to at least one of the vertices in A.
- A **vertex cover** is a subset A of the vertices so that every edge is incident with at least one of the vertices in A.

VERTEX COLORING

Definition: A proper **vertex coloring** of a graph is an assignment of colors to the vertices so that no two adjacent vertices get the same color. Thus, a proper vertex coloring is a function $f : V \to \{1, 2, \ldots, k\}$ such that $f(v) = f(w) \Rightarrow (v, w) \notin E$.

Decision problem:

- Input: Graph $G = (V, E)$ and integer k
- Question: Does G have a proper k-coloring?

Optimization problem: Find the minimum number of colors needed to vertex-color an input graph G. (This number is called the **chromatic number**.)

Construction problem: Find a vertex coloring of input graph G that uses the minimum number of colors.

Note: Vertex Coloring is NP-complete (one of Karp's original 21 problems)

CLIQUE

Definition: A **clique** in $G = (V, E)$ is a subset V_0 of the vertices of G such that every pair of vertices in V_0 are adjacent. In other words, $V_0 \subseteq V$ such that $\forall \{v, w\} \subseteq V_0, (v, w) \in E$.

Decision problem:

- Input: Graph $G = (V, E)$ and integer k
- Question: Does G have a clique of size k?

Optimization problem: Find the size of the largest clique in the input graph G.
Construction problem: Find the largest clique in the input graph G.

NOTE: CLIQUE is NP-complete (one of Karp's original 21 problems)

MAXIMUM INDEPENDENT SET

Definition: An **independent set** in a graph $G = (V, E)$ is a subset of the vertices so that no two vertices in the subset are adjacent. In other words, $V_0 \subseteq V$ such that $\forall \{v, w\} \subseteq V_0, (v, w) \notin E$.

Decision problem:

- Input: Graph $G = (V, E)$ and integer k
- Question: Does G have an independent set of size k?

Optimization problem: Find the size of the largest independent set in the input graph G.

Construction problem: Find the largest independent set in the input graph G.

NOTE: MAXIMUM INDEPENDENT SET is NP-hard (easy reduction from CLIQUE).

MAXIMUM MATCHING

Definition: A **matching** in a graph is a subset of the edges that do not share any endpoints. In other words, $E_0 \subseteq E$ such that $\forall (u, v) \in E_0$ and $\forall (w, x) \in E_0$, if $u = w$ then $v = x$.

Decision problem:

- Input: Graph $G = (V, E)$ and integer k
- Question: Does G have a matching of size k?

Optimization problem: Find the size of the largest matching in the input graph G. This is called the **matching number** of G.
Construction problem: Find the largest matching in the input graph G.

NOTE: MAXIMUM MATCHING can be solved in polynomial time!

VERTEX COVER

Definition: A **vertex cover** in a graph is a set V_0 of vertices so that every edge in the graph has at least one of its endpoints in V_0. In other words, a vertex cover is a set $V_0 \subseteq V$ so that $\forall (v, w) \in E, v \in V_0$ or $w \in V_0$.

Decision problem:

- Input: Graph $G = (V, E)$ and integer k
- Question: Does G have a vertex cover of size k?

Optimization problem: Find the size of the smallest vertex cover in the input graph G.

Construction problem: Find the smallest vertex cover in the input graph G.

NOTE: VERTEX COVER is NP-complete (one of Karp's original problems)

DOMINATING SET

Definition: A **dominating set** in a graph in G is a subset V_0 of the vertices so that every other vertex in G is adjacent to at least one element of V_0. In other words, $V_0 \subseteq V$ such that $\forall v \in V - V_0, \exists w \in V_0$ so that $(v, w) \in E$.

Decision problem:

- Input: Graph $G = (V, E)$ and integer k
- Question: Does G have a dominating set of size k?

Optimization problem: Find the size of the smallest dominating set in the input graph G.
Construction problem: Find the smallest domiating set in the input graph G.

NOTE: Dominating Set is NP-complete.

Formulating real world problems as graph problems

Consider the following problems, all based on the set of people in this class, and under the assumption where you know who is friends with whom (and you assume friendship is symmetric).

1. You want to find a set of people in this class so that everyone in the class is friends with someone in the set, and the set is as small as possible.
2. You want to partition the set of people into subsets so that every two people in any subset are friends, and make the number of subsets as small as possible.
3. You want to pair people off in the class so that they will study together. You want to have as large a number of people able to be in study groups, but you have the following rules: study groups must be pairs of friends and no person can be in two study groups.
4. You want to have a party and invite as many people as you can to it, subject to (a) they are all your friends and (b) they all friends with each other.

Formulating real world problems as graph problems

You want to find a set of people in this class so that everyone in the class is friends with someone in the set, and the set is as small as possible.
Challenge: describe this as a graph problem.

Solution:
The graph $G = (V, E)$ is defined by:

- Let V denote all the people in the class.
- Put an edge between v and w if v and w are friends.

We are looking for the smallest $V_0 \subseteq V$ such that $\forall v \in V \setminus V_0$, $\exists w \in V_0$ such that $(v, w) \in E$.

Formulating real world problems as graph problems

You want to find a set of people in this class so that everyone in the class is friends with someone in the set, and the set is as small as possible.

The graph $G = (V, E)$ is defined by:

- Let V denote all the people in the class.
- Put an edge between v and w if v and w are friends.

We are looking for the smallest $V_0 \subseteq V$ such that $\forall v \in V \setminus V_0$, $\exists w \in V_0$ such that $(v, w) \in E$.

Questions:

- Does a solution always exist?
- What graph problem is this?
- How hard is it to solve this problem?

Describing a real world problem

You want to partition the set of people into subsets so that every two people in any subset are friends, and make the number of subsets as small as possible.

Solution: The graph $G = (V, E)$ is defined by

- V is the set of people in the class
- E contains (v, w) if and only if v and w are friends.

We are looking for a partition of V into a small number of sets so that every one of the sets is a clique (which means that all pairs of vertices in any of the sets are adjacent).
In other words, we want to write $V = V_1 \cup V_2 \cup \ldots \cup V_k$, where V_i is a clique in G and where k is minimized.

Questions to you:

- Does a solution always exist?
- Does it have to be unique?
- What graph problem does this resemble?
- How hard is it to solve the problem?

Things to note

When you formulate a real world problem as a graph problem, you have to:

- Describe the graph precisely. What are the vertices? What are the edges? Do the edges have weights? Use correct terminology (don't be sloppy about language).
- Once the graph is defined, the problem (whether a decision problem, optimization problem, or construction problem) is then defined only in terms of the graph and not in terms of the original problem.

The power in making this formulation is that there are many algorithms (and software) for most natural graph problems, and so you can use those programs to solve your problem.

Many graph problems are NP-hard, but sometimes you have extra structure in your problem that allows you to solve the problem in polynomial time. (For example, MIN VERTEX COLORING is NP-hard, but solvable in polynomial time on trees.)

Class Exercise

You are trying to pick a jury of 12 people from a group of people. The group has already been screened so that everyone is legally acceptable. However, you cannot have any two people on the jury who know each other. So the question is whether or not it's even possible to do this.

Question: Formulate this as a graph problem.

Cycles and circuits

- A **cycle** is a "closed path" (i.e., a sequence of vertices that begins and ends at the same vertex). Thus, a cycle can be written as $v_1, v_2, \ldots, v_k, v_{k+1}$ so that $(v_i, v_{i+1}) \in E$ for all $i = 1, 2, \ldots, k$, $v_{k+1} = v_1$, and otherwise there are no repeated vertices.
- An **acyclic** graph is one that has no cycles.
- A **circuit** in a graph G is a "closed walk" (i.e., a walk that begins and ends at the same vertex). Thus, a circuit can be written as a sequence of vertices $v_1, v_2, \ldots, v_k, v_{k+1}$ so that $(v_i, v_{i+1}) \in E$ for all $i = 1, 2, \ldots, k$, $v_{k+1} = v_1$, and where a vertex can appear more than once.

Terminology, cont.

- A **Hamiltonian Path** is a path that covers every vertex exactly once.
- An **Eulerian Walk** is a walk that goes through every edge exactly once.

Eulerian Graphs

A graph $G = (V, E)$ is **Eulerian** if it has a circuit that covers every edge exactly once. Note – vertices can be repeated, but not edges. Such a circuit is called an *Eulerian Circuit*.

Theorem: A connected simple graph $G = (V, E)$ is Eulerian if and only if every vertex in G has even degree. (Note we assume G is simple and finite.)
Proof: (Will be done later in the semester.)
Notes:

- Determining if a graph G is Eulerian can be performed in polynomial time.
- We also are interested in graphs that have *Eulerian walks*, i.e., walks that cover every edge exactly once.
- Finding Eulerian Circuits and Eulerian Walks is used in Assembling Genomes!

Travelling salesman

The Travelling Salesman is generally stated as an optimization problem:

- Input: Complete graph $G = (V, E)$ with positive weights on the edges
- Output: Hamilton circuit in G (circuit that visits every vertex exactly once) and has total minimum weight

However we can formulate this as a decision problem, as follows:

- Input: Graph $G = (V, E)$ with positive weights on the edges, and bound B
- Output: Does there exist a circuit in G that visits every vertex at least once, and has total weight at most B?

And of course we could make it into a construction problem.

Note: The decision problem for Travelling Salesman is NP-complete.

Trees: connected acyclic graphs

Thus, trees are just undirected graphs that are connected and acyclic. Nothing more special than that.

- If a tree $T = (V, E)$ has n vertices, then it has $n - 1$ edges
- Every tree can be 2-colored.
- The maximum clique size is 2.
- Generally NP-hard problems become polynomial on trees.

Representations of graphs

We can represent a graph $G = (V, E)$, where $V = \{v_1, v_2, \ldots, v_n\}$, in several ways.

The two most popular ways are **adjacency matrices** and **adjacency lists**.

Adjacency Matrices

$G = (V, E)$ is undirected and does not have any parallel edges (nor weights on the edges), and $V = \{v_1, v_2, \ldots, v_n\}$.

The adjacency matrix M for G is $n \times n$ where

- $M[i, j] = 1$ if $(v_i, v_j) \in E$
- $M[i, j] = 0$ if $(v_i, v_j) \notin E$

If the graph is simple (so no self-loops), then $M[i, i,] = 0$ for all $i = 1, 2, \ldots, n$.
For undirected graphs, $M[i, j] = M[j, i]$; in other words, the adjacency matrix for an undirected graph is symmetric.

Adjacency Matrices

Extensions:

- If G has non-zero weights on the edges, then we could let $M_{i,j}$ denote the weight of the edge (v_i, v_j).
- For directed graphs, we distinguish between edges from v_i to v_j and from v_j to v_i; hence, we can get asymmetric matrices.

Note that this representation inherently requires $\Theta(n^2)$ space, even for graphs that don't have many edges.

Given an adjacency matrix, checking if an edge (v_i, v_j) takes $O(1)$ time.

See https://en.wikipedia.org/wiki/Adjacency_matrix.

Adjacency List

In an adjacency list, for each vertex in the graph, you have a list of its neighbors.

If the graph $G = (V, E)$ is undirected, then
$List(x) = \{w \in V | (x, w) \in E\}$.

If the graph $G = (V, E)$ is directed, then
$List(x) = \{w \in V | x \rightarrow w \in E\}$.

Note that an adjacency list requires $\Theta(m)$ space, where m is the number of edges.

This can be much more space efficient than an adjacency matrix for sparse graphs, but some operations take more time (e.g., checking if an edge is present). See

https://en.wikipedia.org/wiki/Adjacency_list.

Practice problems

You should be able to do the following tasks:

- Know various graph-theoretic terms (tree, degree, path, cycle, clique, matching, etc.) and be able to answer questions based on these terms
- Define decision, optimization, and construction problems defined in today's lecture
- Find solutions to these decision problems on small graphs (with at most 6 vertices)
- Formulate a real world problem as a graph problem