Chapter 12

Graphs and their Representation

12.1 Graphs and Relations

Graphs (sometimes referred to as networks) offer a way of expressing relationships between pairs of items, and are one of the most important abstractions in computer science.

Question 12.1. *What makes graphs so special?*

What makes graphs special is that they represent relationships. As you will (or might have) discover (discovered already) relationships between things from the most abstract to the most concrete, e.g., mathematical objects, things, events, people are what makes everything interesting. Considered in isolation, hardly anything is interesting. For example, considered in isolation, there would be nothing interesting about a person. It is only when you start considering his or her relationships to the world around, the person becomes interesting. Challenge yourself to try to find something interesting about a person in isolation. You will have difficulty. Even at a biological level, what is interesting are the relationships between cells, molecules, and the biological mechanisms.

Question 12.2. *Trees captures relationships too, so why are graphs more interesting?*

Graphs are more interesting than other abstractions such as tree, which can also represent certain relationships, because graphs are more expressive. For example, in a tree, there cannot be cycles, and multiple paths between two nodes.

Question 12.3. *What do we mean by a "relationship"? Can you think of a mathematical way to represent relationships.*

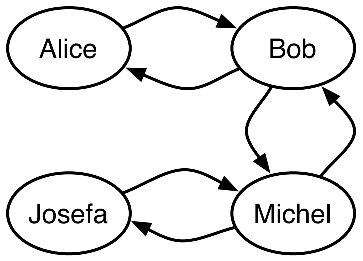


Figure 12.1: Friendship relation {(Alice, Bob), (Bob, Alice), (Bob, Michel), (Michel, Bob), (Josefa, Michel), (Michel, Josefa)} as a graph.

What we mean by a relationship, is essentially anything that we can represent abstractly by the mathematical notion of a relation. A *relation* is defined as a subset of the Cartesian product of two sets.

Exercise 12.4. *You can represent the friendship relation(ship) between people as a subset of the Cartesian product of the people, e.g,* {*(Alice, Bob), (Bob, Alice), (Josefa, Michel), (Michel, Josefa), (Bob, Michel), (Michel,Bob), . . .* }.

Now what is cool about graphs is that, they can represent any mathematical relation.

Question 12.5. *Can you see how to represent a (mathematical) relation with a graph?*

To represent a relation with a graph, we construct a graph, whose vertices represent the domain and the range of the relationship and connect the vertices with edges as described by the relation. Figure 12.1 shows an example. We will clarify what we mean by vertices and edges momentarily.

In order to be able to use graph abstractions, we need to set up some definitions and introduce some terminology. Please see Section 2.4 to review the basic definitions and concepts involving graphs. In the rest of this chapter, we assume familiarity with basic graph theory and discuss representation techniques for graphs as well as applications of graphs.

12.2 Representing Graphs

To choose an efficient and fast representation for graphs, we need to determine first the kinds of operations that we intend to support. For example we might want to perform the following operations on a graph $G = (V, E)$:

(1) Map over the vertices $v \in V$.

(2) Map over the edges $(u, v) \in E$.

(3) Map over the neighbors of a vertex $v \in V$, or in a directed graph the in-neighbors or out-neighbors.

(4) Return the degree of a vertex $v \in V$.

(5) Determine if the edge (u, v) is in E.

(6) Insert or delete vertices.

(7) Insert or delete edges.

Representing graphs for parallel algorithms. To enable parallel algorithm design, in this course , we represent graphs by using the abstract data types that we have seen such as sequences, sets, and tables. This strategy allows us to select the best implementation (data structure) that meets the needs of the algorithm at the lowest cost. In the discussion below, we mostly consider directed graphs. To represent undirected graphs one can, for example, keep each edge in both directions, or in some cases just keep it in one direction. In this chapter, we do not associate data or weights with the edges and leave that for Chapter 16. For the following discussion, consider a graph $G = (V, E)$ with n vertices and m edges.

Edge Sets. The simplest representation of a graph is based on its definition as a set of vertices V and a set of directed edges $A \subseteq V \times V$. If we use the set ADT, the keys for the edge set are simply pairs of vertices. The representation is similar to the edge list representation, but it abstracts away from the particular data structure used for the set—the set could be implemented as a list, an array, a tree, or a hash table.

Question 12.6. *What is the cost of performing graph operations using this representation?*

Consider, for example, the tree-based cost specification for sets given in Chapter 11. For m edges this would allow us to determine if an arc (u, v) is in the graph with $O(\log m)$ work using a find, and allow us to insert or delete an arc (u, v) in the same work. We note that we will often use $O(\log n)$ instead of $O(\log m)$, where n is the number of vertices. This is OK to do since $m \leq n^2$, which means that $O(\log m)$ implies $O(\log n)$.

Although edge sets are efficient for finding, inserting, or deleting an edge, they are not efficient if we want to identify the neighbors of a vertex v. For example, finding the set of out edges requires a filter based on checking if the first element of each pair matches v:

$$\{(x, y) \in E \mid v = x\}$$

For m edges this requires $\Theta(m)$ work and $O(\log n)$ span, which is not efficient in terms of work. Indeed just about any representation of sets would require at least $O(m)$ work.

Adjacency Tables. To more efficiently access neighbors we will use adjacency tables, which are a generalization of adjacency lists and adjacency arrays. The *adjacency table* representation is a table that maps every vertex to the set of its (out) neighbors. This is simply an edge-set table.

Question 12.7. *What is the cost of accessing the neighbors of a vertex?*

In this representation, accessing the out neighbors of a vertex v is cheap since it just requires a lookup in the table. Assuming the tree cost model for tables, this can be done in $O(\log n)$ work and span.

Question 12.8. *Can you give an algorithm for finding an edge (u, v)?*

One can determine if a particular arc (u, v) is in the graph by first pulling out the adjacency set for u and then using a find to determine if v is in the set of neighbors. These both can be done in $O(\log n)$ work and span. Similarly inserting an arc, or deleting an arc can be done in $O(\log n)$ work and span. The cost of finding, inserting or deleting an edge is therefore the same as with edge sets. Note that in general, once the neighbor set has been pulled out, we can apply a constant work function over the neighbors in $O(d_G(v))$ work and $O(\log d_G(v))$ span.

Adjacency Sequences. A special case of adjacency tables are adjacency sequences, where we use sequences to represent both tables and sets. Recall that a sequence is a table with a domain taken from $\{0, \ldots, n-1\}$. Sequences allow for fast random access, requiring only $O(1)$ work to access the i^{th} element rather than $O(\log n)$. This allows, for example, accessing a vertex at less cost. Certain other operations, such as subselecting vertices, however, is more expensive. Because of the reduced cost of access, we sometimes use adjacency sequences, which have type `((int seq) seq)`, to represent a graph.

Costs. The cost of edge sets and adjacency tables is summarized with the following cost specification.

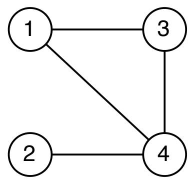


Figure 12.2: An undirected graph.

Cost Specification 12.9 (Graphs)**.** *The cost for various graph operations assuming a tree-based cost model for tables and sets and an array-based cost model for sequences. Assumes the function being mapped uses constant work and span. All costs are big-O.*

	edge set		***adj table***		***adj seq***	
	work	span	work	span	work	span
$(u, v) \overset{?}{\in} G$	$\log n$	$\log n$	$\log n$	$\log n$	n	$\log n$
map over edges	m	$\log n$	m	$\log n$	m	1
find neighbors	m	$\log n$	$\log n$	$\log n$	1	1
map over neighbors	$d_G(v)$	$\log n$	$d_G(v)$	$\log n$	$d_G(v)$	1
$d_G(v)$	m	$\log n$	$\log n$	$\log n$	1	1

12.2.1 Traditional Representations for Graphs

Traditionally, graphs are represented by using one of the four standard representations, which we review briefly below. Of these representations, edge lists and adjacency lists can be viewed as implementations of edge sets and adjacency tables, by using lists to implement sets. For the following discussion, consider a graph $G = (V, E)$ with n vertices and m edges. As we consider different representations, we illustrate how the graph shown in Figure 12.2 is represented using each one.

Adjacency matrix. Assign a unique label from 0 to $n - 1$ to each vertex and construct an $n \times n$ matrix of binary values in which location (i, j) is 1 if $(i, j) \in E$ and 0 otherwise. Note that for an undirected graph the matrix is symmetric and 0 along the diagonal. For directed graphs the 1s can be in arbitrary positions.

Example 12.10. *Using an adjacency matrix, the graph in Figure 12.2 is represented as follows.*

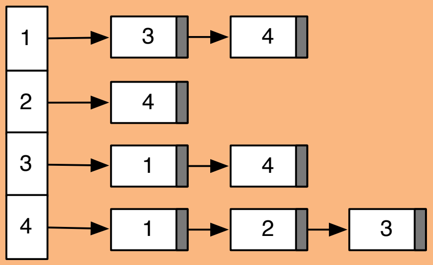
$$\begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

Question 12.11. *What are the advantages and disadvantages of this representation?*

The disadvantage of adjacency matrices is their space demand of $\Theta(n^2)$. Graphs are often sparse, with far fewer edges than $\Theta(n^2)$.

Adjacency list. Assign a unique label from 0 to $n-1$ to each vertex and construct an array A of length n where each entry $A[i]$ contains a pointer to a linked list of all the out-neighbors of vertex i. In an undirected graph with edge $\{u, v\}$ the edge will appear in the adjacency list for both u and v.

Example 12.12. *Using adjacency lists, the graph Figure 12.2 is represented as follows.*

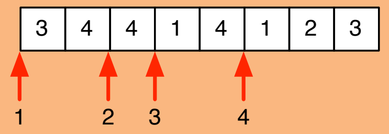


Question 12.13. *What are the advantages and disadvantages of this representation?*

Adjacency lists are not well suited for parallelism since the lists require that we traverse the neighbors of a vertex sequentially.

Adjacency array. Similar to an adjacency list, an adjacency array keeps the neighbors of all vertices, one after another, in an array `adj`; and separately, keeps an array of indices that tell us where in the `adj` array to look for the neighbors of each vertex.

Example 12.14. *Using an adjacency array, the graph Figure 12.2 is represented as follows.*



Question 12.15. *What are the advantages and disadvantages of this representation?*

The disadvantage of this approach is that it is not easy to insert new edges.

Edge list. A list of pairs $(i, j) \in E$. As with adjacency lists, this representation is not good for parallelism.

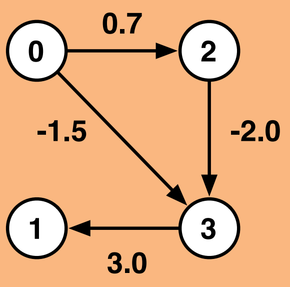
12.3 Weighted Graphs and Their Representation

Many applications of graphs require associating weights or other values with the edges of a graph. Such graphs can be defined as follows.

Definition 12.16 (Weighted and Edge-Labeled Graphs)**.** *An* edge-labeled graph *or a* weighted graph *is a triple $G = (E, V, w)$ where $w \colon E \to L$ is a function mapping edges or directed edges to their labels (weights) , and L is the set of possible labels (weights).*

In a graph, if the data associated with the edges are real numbers, we often use the term "weight" to refer to the edge labels, and use the term "weighted graph" to refer to the graph. In the general case, we use the terms "edge label" and edge-labeled graph. Weights or other values on edges could represent many things, such as a distance, or a capacity, or the strength of a relationship.

Example 12.17. *An example directed weighted graph.*



We described three different representations of graphs suitable for parallel algorithms: edge sets, adjacency tables, and adjacency sequences.

Question 12.18. *Can you see how we can extend these representations to support edge values?*

We can extend each of these representations to support edge-labeling by separately representing the function from edges to labels using a table (mapping) that maps each edge (or arc) to its value. This representation allows looking up the edge value of an edge $e = (u, v)$ by using a table lookup. We call this an *edge table*.

Example 12.19. *For the weighted graph in Example 12.17, the edge table is:*

$$W = \{(0,2) \mapsto 0.7,\ (0,3) \mapsto -1.5,\ (2,3) \mapsto -2.0,\ (3,1) \mapsto 3.0\}$$

A nice property of an edge table is that it works uniformly with all representations of the structure of a graph, and is clean since it separates the edge labels from the structural information. However keeping the edge table separately creates redundancy, wasting space and possibly requiring extra work to access the edge labels.

Question 12.20. *Can you eliminate this redundancy?*

The redundancy can be avoided by storing the edge values directly with the edge information.

For example, instead of using edge sets, we can use edge tables (mapping edges to their values). Similarly, when using adjacency tables, we can replace each set of neighbors with a table mapping each neighbor to the label of the edge to that neighbor. Finally, we can extend an adjacency sequences by creating a sequence of neighbor-value pairs for each out edge of a vertex. This is illustrated in the following example.

Example 12.21. *For the weighted graph in Example 12.17, the adjacency table representation is*

$$G = \{1 \mapsto \{2 \mapsto 0.7, 3 \mapsto -1.5\}, 3 \mapsto \{3 \mapsto -2.0\}, 4 \mapsto \{1 \mapsto 3.0\}\},$$

and the adjacency sequence representation is

$$G = \langle\, \langle\, (2, 0.7),\ (3, -1.5)\, \rangle\, , \langle\ \rangle\, , \langle\, (3, -2.0)\, \rangle\, , \langle\, (1, 3.0)\, \rangle\, \rangle\, .$$

12.4 Applications of Graphs

Since they are powerful abstractions, graphs can be very important in modeling data. In fact, many problems can be reduced to known graph problems. Here we outline just some of the many applications of graphs.

1. *Social network graphs: to tweet or not to tweet.* Graphs that represent who knows whom, who communicates with whom, who influences whom or other relationships in social structures. An example is the twitter graph of who follows whom. These can be used to determine how information flows, how topics become hot, how communities develop, or even who might be a good match for who, or is that whom.

2. *Transportation networks.* In road networks vertices are intersections and edges are the road segments between them, and for public transportation networks vertices are stops and edges are the links between them. Such networks are used by many map programs such as Google maps, Bing maps and now Apple IOS 6 maps (well perhaps without the public transport) to find the best routes between locations. They are also used for studying traffic patterns, traffic light timings, and many aspects of transportation.

3. *Utility graphs.* The power grid, the Internet, and the water network are all examples of graphs where vertices represent connection points, and edges the wires or pipes between them. Analyzing properties of these graphs is very important in understanding the reliability of such utilities under failure or attack, or in minimizing the costs to build infrastructure that matches required demands.

4. *Document link graphs.* The best known example is the link graph of the web, where each web page is a vertex, and each hyperlink a directed edge. Link graphs are used, for example, to analyze relevance of web pages, the best sources of information, and good link sites.

5. *Protein-protein interactions graphs.* Vertices represent proteins and edges represent interactions between them that carry out some biological function in the cell. These graphs can be used, for example, to study molecular pathways—chains of molecular interactions in a cellular process. Humans have over 120K proteins with millions of interactions among them.

6. *Network packet traffic graphs.* Vertices are IP (Internet protocol) addresses and edges are the packets that flow between them. Such graphs are used for analyzing network security, studying the spread of worms, and tracking criminal or non-criminal activity.

7. *Scene graphs.* In graphics and computer games scene graphs represent the logical or spacial relationships between objects in a scene. Such graphs are very important in the computer games industry.

8. *Finite element meshes.* In engineering many simulations of physical systems, such as the flow of air over a car or airplane wing, the spread of earthquakes through the ground, or the structural vibrations of a building, involve partitioning space into discrete elements. The elements along with the connections between adjacent elements forms a graph that is called a finite element mesh.

9. *Robot planning.* Vertices represent states the robot can be in and the edges the possible transitions between the states. This requires approximating continuous motion as a sequence of discrete steps. Such graph plans are used, for example, in planning paths for autonomous vehicles.

10. *Neural networks.* Vertices represent neurons and edges the synapses between them. Neural networks are used to understand how our brain works and how connections change when we learn. The human brain has about 10^{11} neurons and close to 10^{15} synapses.

11. *Graphs in quantum field theory.* Vertices represent states of a quantum system and the edges the transitions between them. The graphs can be used to analyze path integrals and summing these up generates a quantum amplitude (yes, I have no idea what that means).

12. *Semantic networks.* Vertices represent words or concepts and edges represent the relationships among the words or concepts. These have been used in various models of how humans organize their knowledge, and how machines might simulate such an organization.

13. *Graphs in epidemiology.* Vertices represent individuals and directed edges the transfer of an infectious disease from one individual to another. Analyzing such graphs has become an important component in understanding and controlling the spread of diseases.

14. *Graphs in compilers.* Graphs are used extensively in compilers. They can be used for type inference, for so called data flow analysis, register allocation and many other purposes. They are also used in specialized compilers, such as query optimization in database languages.

15. *Constraint graphs.* Graphs are often used to represent constraints among items. For example the GSM network for cell phones consists of a collection of overlapping cells. Any pair of cells that overlap must operate at different frequencies. These constraints can be modeled as a graph where the cells are vertices and edges are placed between cells that overlap.

16. *Dependence graphs.* Graphs can be used to represent dependences or precedences among items. Such graphs are often used in large projects in laying out what components rely on other components and used to minimize the total time or cost to completion while abiding by the dependences.