CSCI 189 Assignment 9 Problem Set

This assignment provides an introduction to graphs, trees, matrices, and their uses in computer science. The compression of JPEG image data, along with its Huffman encoding, provides an interesting practical application of these mathematical concepts.

Summary of Concepts in Sections 5.1-5.4, 4.5 and Haskell

Matrices

A *matrix* is a rectangular arrangement of values, with m rows and n columns. Below are three matrices A, B, and C:

$$A = \begin{bmatrix} 1 & 0 & 6 \\ 3 & -1 & 5 \end{bmatrix} \quad B = \begin{bmatrix} 4 & 5 \\ 2 & 0 \\ 6 & 7 \end{bmatrix} \quad C = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad D = \begin{bmatrix} 4 & 5 & 1 \\ 2 & 0 & 3 \end{bmatrix}$$

A has $m = 2$ rows and $n = 3$ columns, and so it's a 2x3 matrix. B and C are 3x2 and 3x3 matrices respectively. An element of a matrix a_{ij} is identified by its row and column number. For example, in the matrix A above, the element $a_{23} = 5$.

A *square* matrix is one for which $m = n$, and the *diagonal* of a square matrix contains the elements a_{ii} for all $i \in \{1, \ldots, n\}$. Matrix C above is square, and all the elements of its diagonal $c_{ii} = 1$. A *symmetric* matrix is a square matrix for which $c_{ij} = c_{ji}$ for all $i, j \in \{1, \ldots, n\}$. Matrix C above is also symmetric.

A matrix can be multiplied by a single value, called *scalar multiplication*, by simply multiplying each element by that value. Two matrices can be added or subtracted, provided that they have the same number of rows and columns. Their sum/difference is that matrix whose elements are the sums/differences of corresponding elements in the two original matrices. Here are two examples, using matrices A and D above.

$$5A = 5 \begin{bmatrix} 1 & 0 & 6 \\ 3 & -1 & 5 \end{bmatrix} = \begin{bmatrix} 5 & 0 & 30 \\ 15 & -5 & 25 \end{bmatrix}$$

$$A + D = \begin{bmatrix} 1 & 0 & 6 \\ 3 & -1 & 5 \end{bmatrix} + \begin{bmatrix} 4 & 5 & 1 \\ 2 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 5 & 5 & 7 \\ 5 & -1 & 8 \end{bmatrix}$$

Two matrices can be multiplied together, provided that the first has the same number of columns as the second has rows. The number of rows and columns in the product matrix is the same as the number of rows in the first matrix and the number of columns in the second.

Each entry p_{ij} in the product matrix, say P, is the sum of the products of corresponding elements in the ith row of the first matrix and the jth column in the second. That is, if A is an $m \times n$ matrix, B is an $n \times p$ matrix, then their product P is an $m \times p$ matix in which each entry $p_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$.

Below, for example, is the product P of the two matrices A and B defined above:

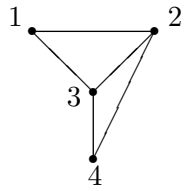
$$P = A \cdot B = \begin{bmatrix} 1 & 0 & 6 \\ 3 & -1 & 5 \end{bmatrix} \times \begin{bmatrix} 4 & 5 \\ 2 & 0 \\ 6 & 7 \end{bmatrix} = \begin{bmatrix} 1 \cdot 4 + 0 \cdot 2 + 6 \cdot 6 & 1 \cdot 5 + 0 \cdot 0 + 6 \cdot 7 \\ 3 \cdot 4 - 1 \cdot 2 + 5 \cdot 6 & 3 \cdot 5 - 1 \cdot 0 + 5 \cdot 7 \end{bmatrix} = \begin{bmatrix} 40 & 47 \\ 40 & 50 \end{bmatrix}$$

Note that P is a 2x2 matrix, since A has 2 rows and B has 2 columns.

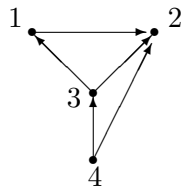
Matrix C above is an example of an *identity* matrix, which is any $n \times n$ square matrix with 1s on the diagonal and 0s everywhere else. An identity matrix has the unique property that, when multiplied by any other $n \times n$ matrix, returns that matrix as a result.

Graphs

A *graph* is a nonempty set of *nodes* and *arcs* (arcs are sometimes called *edges*). Below is a graph with 4 nodes and 5 edges.



A *directed graph* is a graph whose edges all have a "direction," as indicated by arrows rather than lines. Below is a directed graph that has the same nodes and edges as the above (undirected) graph.



A *path* between two nodes n_1 and n_2 on a graph is a series of edges that one can take to reach n_2 starting from n_1. For example, there are three paths from node 4 to node 1 in the undirected graph above: 4-3-1, 4-2-1, 4-2-3-1, and 4-3-2-1. In the directed graph, there is only one path from node 4 to node 1: 4-3-1.

A graph is *connected* if there is a path from every node to every other node. The undirected graph above is connected. A *cycle* is a path from a node to itself. For example, 4-2-1-3-4 is a cycle in the undirected graph above. A graph with no cycles is called *acyclic*.

A graph is *complete* if every pair of nodes has an edge connecting them. The above graphs are not complete, since nodes 1 and 4 have no edge connecting them.

A *planar* graph is one that can be drawn so that all of its edges intersect only at the nodes (i.e.,

none of its edges intersect elsewhere). The graph on page 365, Exercise 29 is an example of a non-planar graph.

Graphs have many applications in computer science. For example, the nodes may represent cities on a map and the edges may represent roads connecting the cities. A classical problem is the "shortest path" problem, in which each edge has a numerical length and the problem is to find the shortest path between any two cities. Another graph problem is in computer chip design, where the goal is to design circuits for which no two wires cross in the same plane - i.e., planar graphs.

Thus, it is important to consider how graphs can be represented by a computer program. One representation scheme is called the "adjacency matrix," which is an $n \times n$ matrix (n is the number of nodes in the graph). The ij^{th} entry in this matrix contains a 1 or a 0 depending on whether or not nodes i and j have an edge connecting them. Here is an adjacency matrix for the undirected graph shown above:

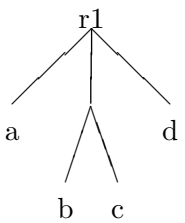
$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

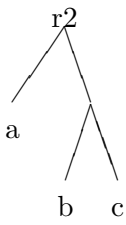
The adjacency matrix for an undirected graph is always symmetric (i.e., the ij^{th} entry is the same as the ji^{th} entry), since an edge that connects nodes i and j is the same edge that connects nodes j and i. However, the adjacency matrix for a directed graph is usually not symmetric, since its ij^{th} entry indicates an edge from node i to node j, taking into account the direction of the edge. Below is the adjacency matrix for the directed graph shown above:

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

Trees and Decision Trees

A *tree* is an acyclic connected graph with one node designated as its *root*. Below are two trees with roots r1 and r2.





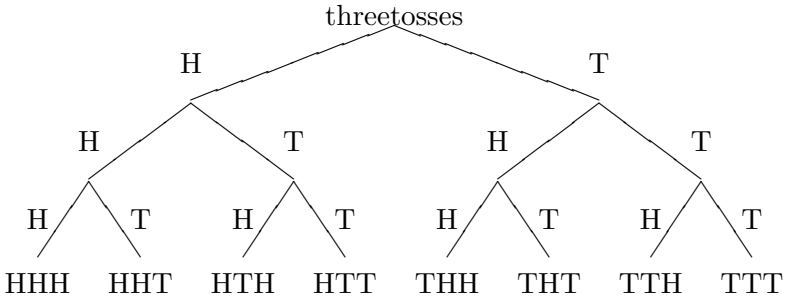
A tree can be defined recursively as follows:

1. A single node r is a tree, with that node as its root.
2. If $T_1, T_2, \ldots, T_n$ are trees, then so is the tree with new root r connected by an edge to each of the roots $r_1, r_2, \ldots, r_n$ of these trees.

The node r is called the *parent* of each of the roots $r_1, r_2, \ldots, r_n$, who are known as r's *children*. Nodes on a tree that have no children are called the *leaves* of that tree. The *depth* of a node in a tree is the number of edges between it and the root. The *height* of a tree is the maximum depth of all the tree's nodes. For example, both trees above have height 2. The children of root r1 are three trees (the tree a, the tree with children b and c, and the tree d). The depth of nodes r1, a, and b are 0, 1, and 2, respectively.

A *binary tree* is a tree in which no node has more than two children. A *complete* binary tree is one in which the nodes at the maximum depth have no children (i.e., they are leaves), and all others have exactly two children. For example, the tree above with root r2 is a binary tree, but it is not complete.

Trees occur often in computer science. Many examples appear on pages 372-380 in your text. *Decision trees* are trees in which the arcs represent actions and the nodes represent outcomes. For example, below is a decision tree with root "threetosses" that models all the possible outcomes for three tosses of a coin.



Huffman Codes and JPEG images

Computers store, process and transmit enormous quantities of data. Various schemes have been designed to *compress* this data so that it doesn't take up so much space. Especially important is the need to compress the data that appears in digital photographic images.

A process called "Huffman coding" is widely used for data compression. This process depends on the fact that some particular values (whether they are colors in an image or letters in the alphabet) occur much more frequently in data than others.

For instance, the normal digital coding for a text file is the "ASCII code," which requires 8 bits per character. Therefore, to store a text of 1000 characters using the ASCII code, 8000 bits are required. Alternatively, if the coding for the character "e" is encoded using fewer than 8 bits, and the coding for the character "z" (which occurs much less frequently than "e"), the whole text can be "compressed" to much less than 8000 bits. This is the idea behind the Huffman coding algorithm, which is fully discussed on pages 399-400 of your text.

JPEG images are digital images that are stored using a compression technique developed by the "Joint Photographic Experts Group." This compression technique uses Huffman coding to reduce the number of bits to represent an image. An image is stored as a rectangular array of "pixels" (short for "picture elements"), each of which has an x-y coordinate and a color.

For instance, a typical computer screen has an array of 1024x768 pixels, and each pixel's color typically ranges over a spectrum of $2^{24} = 16,777,216$ different colors. Without image compression, this totals $1024 \times 768 \times 24 = 18,874,368$ bits to hold a single screen image.

JPEG image compression allows tremendous savings in space for storing images. It has two parts - one that reduces number of bits to represent each pixel's color and one that uses Huffman coding to reduce the number of bits for encoding the colors that occur most frequently (such as "white" or "black"). Many images that are compressed using JPEG recover 80 percent or more of the original image size, which is significant.

Problems to be handed in

Section 4.5 (p 329) Exercises 5bcj, 6b, 8, 10b.

Section 5.1 (p 362) Exercises 2, 10, 15abe, 40, 70ab, 76.

Section 5.2 (p 381) Exercises 2, 6, 17, 39, 41.

Section 5.3 (p 394) Exercises 3, 8, 15, 17.

Section 5.4 (p 405) Exercises 2, 5, 8, 14, 15.