Graph Theory

And its applications in problem solving

Frankie Balch, Harriet Crockett, Orson Hart, Cade Simpson, Joe Wibberley

Exeter Mathematics Certificate

Introduction

Graph theory exists at the intersection of pure mathematics and computer science. We have been tasked with using Graph theory to solve several problems including: finding the shortest path between two points, finding the optimal flow through a network, and data compression algorithms such as the Huffman code. In order to accomplish this, we have been asked to use the coding language Haskell.

Main Objectives

1. Read and write simple programs in the functional programming language Haskell.
2. Describe the problem algorithms in the language of graph theory.
3. Apply, by hand, the problem algorithms.
4. Represent graphs in Haskell.
5. Represent at least one of the problem algorithms as a Haskell program.
6. Know the language of the order of complexity of algorithms.
7. Compute the order of complexity of at least one of the problem algorithms.

Huffman Coding

A line of text can be ordered by most common letter. For example:
Hello World can be written more simply as:

H	E	L	O	␣	W	R	D
1	1	3	2	1	1	1	1

Following this, the individual letters can be ordered into a graph, which shows the frequency of each letter within a given variable.

```
data Tree = Leaf Int | Node Char Tree Tree
   deriving Show
tsum :: Tree -> Int
tsum (Leaf x) = x
tsum (Node _ lt rt) = (tsum lt) + (tsum rt)

flatten :: Tree -> [Char]
flatten (Leaf _) = []
flatten (Node x lt rt) = (flatten lt) ++ [x] ++ (flatten rt)
join :: Char->Tree->Tree->Tree
join x lt rt = Node x lt rt

data Btree = Leave | Vert Int Btree Btree
  deriving Show

insertKey :: Btree->Int->Btree
insertKey Leave x = Vert x Leave Leave
insertKey (Vert y lt rt) x
  | x < y = Vert y (insertKey lt x) rt
  | otherwise = Vert y lt (insertKey rt x)

makeTree :: [Int]->Btree
makeTree = foldl insertKey Leave

flattenb :: Btree->[Int]
flattenb Leave = []
flattenb (Vert x lt rt) = (flattenb lt) ++ [x] ++ (flattenb rt)

treesort :: [Int]->[Int]
treesort = flattenb . makeTree
```

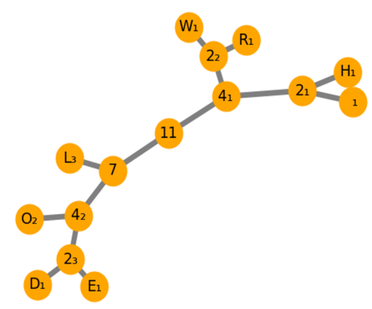

Figure 1: A Huffman graph using the text *"Hello World"*

Alternatively, this data can be conveyed within a tree, with ones on the left branches and zeros on the right branches. A binary value for a specific letter can be found by selecting the path to reach it and recording the values of the route at each bifurcation.

Uses of this particular type of data compression is used in several data types, including MP3, JPEG and ZIP files. It can be proven mathematically that Huffman coding is the optimal form of encoding.

Shortest Path Algorithm

The shortest path algorithm uses graph theory as a way to plot a journey of the least possible distance, while simultaneously allowing you to pass through a selected number of points. As a result, each destination can be described as a node and each distance an edge between. Such algorithms can be used in several mapping technologies such as Google Maps or Satellite Navigation within a car.

A common method for figuring out the shortest route between two points on a map is Dijkstra's Algorithm, which can be processed either manually by hand, or by using a digital representation of the graph. Other algorithms also exist to work with graphs that have negative values and other complications.

Optimal Flow Through a Network

This is similar to the shortest path algorithm, but for each edge there are two values, flow and capacity, so the method is different. Flow is what is being transported down the edges, capacity is how much flow can go down an edge. Therefore, flow can never exceed capacity.

It allows you to find the most efficient way of sending data or items from one place to another and is used in air traffic control and to send data.

Finding optimal flow through a network by hand is long and often tedious due to the need of working out the optimal flow of each edge individually, but it can be coded to work quickly, efficiently, and accurately.

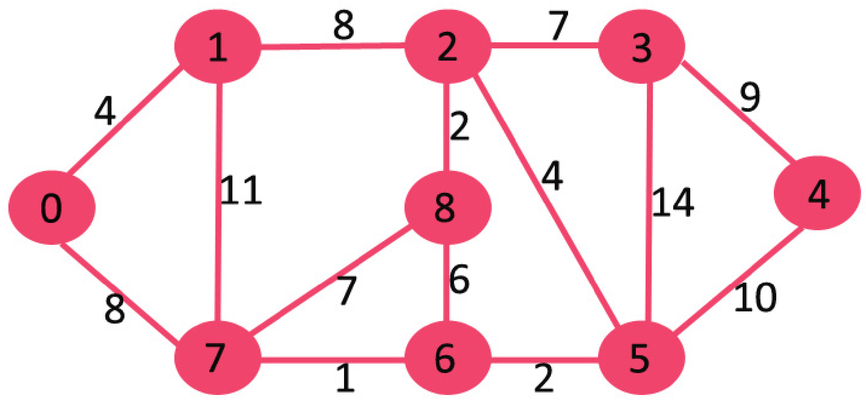

Figure 2: An example of a network graph.

Big O Notation

Big O notation is a mathematical concept that is frequently applied to computer science, being used to determine program efficiency, with relation to the amount of processing power that is required to run said program. It is divided into categories of intensity, which state roughly how much power a section of code requires. It operates on the idea that in a function, the variable with the largest exponential will eventually make all other parts of the function essentially negligible.

For instance, in the function $f(x) = (x^2 + 5x + 2)$, after x becomes larger than 5, the $+2$ becomes negligible, and after x is even greater, the $5x$ also becomes insignificant, as the x^2 exponentially becomes the largest, and only important, part of the function. In this way, after a certain limit of x is reached, the entire function could be considered as: $f(x) = x^2$. In Computer Science, this concept can be used to analyse the processing power required to compute a section of code. The notation for this $O(Value)$. So in the case of our original function, the expression for it in Big O notation would be: $O(x^2)$.

With this in mind, a programming function that took the same amount of time and processing power to run irregardless of the size of the input data could be shown as $O(1)$. A program section that takes time and power linearly proportional to the size of the input data would be represented as: $O(N)$. A program that is exponentially proportional in its power requirement to the size of its input data size, for instance a function that prints the whole data set for each value within it. This would be shown as $O(N^x)$ where x is any possible exponent.

In terms of Huffman Coding, Big O Notation can be applied to show the order of complexity. If the characters are already ordered by frequency, then $O(N)$ applies, where the processing time and power are linearly proportional to the length of the text to be encoded. However, should the characters need to be ordered, then $O(n * log(n))$ applies.

Haskell

The optimal language for coding such problems is Haskell. Haskell, named for famous logician Haskell Curry, is a purely functional coding language with a strong emphasis on mathematics. This makes it ideal for problems centred around graph theory, which is a subsection of pure mathematics. In particular, we have been asked to use Glasgow Haskell Coding Interface (*GCHI*) in order to accomplish this

Conclusions

- A selection of letters and numbers can be compressed most efficiently using a huffman graph or tree.
- The optimal route between two points can only be known if all possible routes are known
- If paths have flow and capacity you must first calculate the maximum flow before you can find the shortest path.
- Program complexity can be analysed with Big O Notation.

References

[1] Robb Bell. A beginners guide to big o notation. *https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation*, 2009.

[2] Introductory Haskell documents. Adobe acrobat documents. *"Extended Project", "Suggestions"*, 2018.

[3] Advanced Algorithms Lecture. Pdf document. *http://www.inf.unibz.it/ zini/AA/Lectures/aa-2013-lecture-08.pdf"*, 2013.

[4] Shortest Path Problem. Pdf document. *https://www.pearsonschoolsandfecolleges.co.uk/secondary/Mathematics/ 02*

[5] Wikipedia The free encyclopedia. Website. https://en.wikipedia.org/wiki/Graph$_{t}heory$, 2018.

[6] Learn you a haskell for great good. Website. http://learnyouahaskell.com/chapters, *2018.*

Acknowledgements

We would like to acknowledge Dr. Gihan Marasingha for setting us this problem and thank all of the staff at Exeter Maths school for all assistance that they have given.