Discrete Mathematics MA 210 — Lent 2009

Notes for lectures 17 and 18

3.10 Chromatic number of graphs

Definition 3.16. *A k-**colouring** of a graph $G = (V, E)$ is a labelling $f : V \to \{1, 2, \ldots, k\}$. The labels are **colours**; the vertices of one colour form a **colour class**.*

*A k-colouring is **proper** if adjacent vertices have different labels. A graph is k-**colourable** if it has a proper k-colouring.*

*The **chromatic number** $\chi(G)$ is the least k such that G is k-colourable.*

*A graph G is k-**chromatic** if $\chi(G) = k$.*

Each colour class is an *independent set* (i.e., it contains no edge). Hence, a graph is 2-colourable if and only if it is bipartite.

Definition 3.17. *The **independence number** of a graph G, written $\alpha(G)$, is the maximumm size of an independent set in G. The **clique number** of a graph G, written $\omega(G)$, is the maximumm size of a pairwise adjacent vertices (clique) in G.*

Lemma 3.18. *For every n-vertex graph G, $\chi(G) \geq \omega(G)$ and $\chi(G) \geq \frac{n}{\alpha(G)}$.*

There exist graphs with $\chi(G) > \omega(G)$ or with $\chi(G) > \frac{n}{\alpha(G)}$.

The **greedy algorithmm** for colouring an n-vertex graph G proceeds as follows:

1. Choose some ordering $x_1, x_2, \ldots, x_n$ of the vertices of G.
2. Colour x_1 with colour 1.
3. Colour remaining vertices in $n - 1$ steps: at the step j, where $j = 2, 3, \ldots, n$, list the colours of all the neighbours of x_j in the set $\{x_1, x_2, \ldots, x_{j-1}\}$. Give x_j the smallest colour not used on this list.

The actual number of colours used by the greedy algorithmm depends on the graph G and on the ordering of the vertices.

By analysing the greedy algorithm, we obtain the following result:

Theorem 3.19. *For every graph G, we have $\chi(G) \leq \Delta(G) + 1$, where $\Delta(G) = \max_{v \in V(G)} \deg(v)$ is the maximum degree of G.*

4 Coding Theory

Coding theory is not the area of mathematics dealing with the theory of secret codes for MI5 etc. That are is called cryptography

Coding theory deals with the mathematical theory behind the designs of codes for storing and transmitting information in such a way that there is a built-in capacity to recognize and perhaps even correct errors.

4.1 Introduction and Notation

Instead of usual 26 letters of the alphabet, we shall work with two symbols only: "0" and "1". These are, not by accident, also the symbols with which computers work and in which most digital communication is performed.

We almost always assume that our code consists of a number of sequences of length n, for some natural number n. The set of all possible $0,1$-sequences of length n will be denoted by $\{0,1\}^n$. Each symbol in such a sequence is called a *bit*. An element of $\{0,1\}^n$ (a $0,1$-sequence of length n) is called a *word*.

Definition 4.1. *A binary code C (of length n) is a subset of $\{0,1\}^n$. An element of C is called a codeword.*

For example, when $n = 3$, we have

$$\{0,1\}^3 = \{000, 001, 010, 100, 011, 101, 110, 111\}$$

and a particular code might be $C = \{000, 011, 101, 110\}$. So, 011 is a codeword in C, whereas 111 is not.

The code C can send only four messages. So, why should we use this code instead of a shorter code $\{00, 10, 01, 11\}$ that can also send four messages?

One reason is as follows: We want to send one of four different messages and we encode them by $00, 01, 10, 11$. Suppose that we send 01. If something goes wrong during the transmission, say the first bit gets changed from 0 to 1, then the receiver will receive 11. But since this is one of the codewords, the receiver will think that we have sent 11 and will act accordingly, with all the consequences.

Assume now that instead of 01 we send a codeword 011 from C. And again, suppose that the first bit gets changed from 0 to 1. So, the receiver will receive 111. But this is not a codeword! So, the receiver will know something is wrong, and most likely will ask to repeat the message.

You should check for yourself that $C = \{000, 011, 101, 110\}$ has a property that whenever one bit of any codeword is changed, the result is not a codeword from C. In other words, as long as not more than one error occurs, the receiver will be able to realize that an error has occurred. We say that C is an *one-error-detecting code*.

4.2 Distance in codes

Definition 4.2. *Let $\bar{x}$, $\bar{y}$ be any two words in $\{0,1\}^n$. Then the* Hamming distance $d_H(\bar{x}, \bar{y})$ *of $\bar{x}$ and $\bar{y}$ is defined as the number of bits in which $\bar{x}$ and $\bar{y}$ are different. Hence, for $\bar{x} =$*

$x_1x_2\dots x_n$, $\bar{y} = y_1y_2\dots y_n$,

$$d_H(\bar{x},\bar{y}) = |\{i \,:\, x_i \neq y_i,\ i = 1,2,\dots,n\}|.$$

The *weight* $w(\bar{x})$ of a word $\bar{x}$ is the number of 1's in the word. Another way to define the weight is by $w(\bar{x}) = d_H(\bar{x},\bar{0})$, where $\bar{0}$ indicates the word $00\dots0$.

Theorem 4.3. *The Hamming distance has the following properties. For all $\bar{x}$, $\bar{y}$, $\bar{z} \in \{0,1\}^n$,*

1. $d_H(\bar{x},\bar{y}) = 0$ *if and only if* $\bar{x} = \bar{y}$;
2. $d_H(\bar{x},\bar{y}) = d_H(\bar{y},\bar{x})$;
3. $d_H(\bar{x},\bar{z}) + d_H(\bar{z},\bar{y}) \geq d_H(\bar{x},\bar{y})$.

Definition 4.4. *Let $C \subseteq \{0,1\}^n$ be a code of length n. Then the* minimum distance *of C is*

$$\delta(C) = \min\{d_H(\bar{x},\bar{y}) \,:\, \bar{x},\bar{y} \in C,\ \bar{x} \neq \bar{y}\}.$$

4.3 Error-detecting and error-correction

Let C be a code of length n. Suppose that a codeword $\bar{x} \in C$ is transmitted, but some of its bits get changed during the transmission. Hence, a word $\bar{y}$ of length n is received, with $\bar{y} \neq \bar{x}$. Then the receiver will only recognize that an error occurred if $\bar{y} \notin C$.

A code C which has a property that we can always recognize if up to d errors have occurred is called a *d-error-detecting code*.

Theorem 4.5. *A code C is d-error-detecting if and only if $\delta(C) \geq d + 1$.*

If we can only conclude that a received message contains errors, then we are still empty-handed and the best thing we can do is to ask for the message to be resent. This is rather inefficient, so we would like to have codes that can not only detect errors, but also tell us which bits are wrong and how to correct them.

Notice that in binary codes, each bit has only two possible values (0 or 1), so knowing which bits are wrong also tells us what are their correct values.

The way to get to error-correcting is by assuming that there is only a small probability that one errors occurs and, hence, if $p < q$, the it is more likely that p errors have occurred than that q have occurred. So, if a word $\bar{y}$ is received and it is not a codeword, then we look for a codeword $\bar{x} \in C$ that differs from $\bar{y}$ in the smallest possible numbers of bits. A problem can arise if this codeword is not unique, or it is not the original codeword (if too many errors occurred).

A more formal way to describe the process above is saying that if a word $\bar{y}$ is received, then we look for a codeword $\bar{x} \in C$ such that

$$d_H(\bar{y},\bar{x}) = \min\{d_H(\bar{y},\bar{z}) \,:\, \bar{z} \in C\}.$$

The principle above is called the *nearest-neighbour decoding*.

Definition 4.6. *A code C is d-*error-correcting *if the code can always uniquely correct up to d errors for every codeword.*

Theorem 4.7. *A code C is d-error-correcting if and only if $\delta(C) \geq 2d + 1$.*

For the proof see Biggs, Section 24.1.

Suppose we are given the code $C = \{000000, 111000, 000111, 111111\}$. Since $\delta(C) = 3$, this code is 1-error-correcting. For instance, if 011000 is received, then it is decoded to the codeword 111000.