Lecture 3: Applications of Induction & Recursion

1. Error detecting and correcting codes

   (ask students what they know about this topic)

   Examples of error detecting codes:
   a) parity
   b) checksums

   Examples of error correcting codes:
   a) Hamming codes
   b) Reed-Solomon codes

Parity codes:
- can use either even parity code or odd parity code
  - total # of 1's has to be even
  - total # of 1's has to be odd

Information space - all n-bit words

Code space - only particular combinations of n bits, which allows us to detect (sometimes correct) when errors occur (result is not in code space, we say it is not a code word)

Codes are primarily used in communication and storage:
Parity codes are single-bit-error detecting codes, e.g., w/ an even parity code, suppose a 1-bit error occurs: what happens to the parity? either a '0' changes to a '1' (one extra '1') or a '1' changes to a '0' (one fewer '1') in both cases, the # of 1s will become odd, the result is not a code word, and we detect the error however, if 2 bits are in error, the result will still have even parity and so we cannot detect 2-bit errors w/ a parity code How do we design a parity code? Just add a parity bit to end of word If the data we want to store/transmit has n-1 bits, then for an even parity code:
  1) if # of 1s in data bits is odd, add a parity bit of 1
  2) if # of 1s in data bits is even, add a parity bit of 0

Note that w/ a parity code, exactly half of the words in the information space are code words (1/2 words have even parity, 1/2 have odd parity)
It turns out, this is a requirement for a single-bit-error correcting code, i.e. the code words can not be more than $\frac{1}{2}$ of the total words in the info. space (there are lots of SEC codes, parity is just one example).

Theorem: If $C$ is a single-bit-error detecting code of length $n \geq 1$, then the number of code words is at most $2^{n-1}$. (At most $\frac{1}{2}$ of the info. space can be used for code words.)

Proof by induction:

Base case: $n = 1$

There are 2 words in the info. space, 0 and 1 to detect an error, only one of these can be a code word, if both are code words, every possible info. word is a code word, so we could never detect an error, no matter what we receive, it's a valid code word so we can never detect an error.

Inductive step: Assume theorem true for codes of length $n$,

Show it's true for codes of length $n+1$

$C$ is a single-bit-error detecting code of length $n+1$

Split the code words of $C$ into 2 parts: those that have a leading '0' and those that have a leading '1'.
Do is all \( n \)-bit words that are the last \( n \) bits of words in \( C_0 \).

\( D_1 \) is all \( n \)-bit words that are the last \( n \) bits of words in \( C_1 \).

What can we say about \( D_0 \) and \( D_1 \)?

There cannot be any 2 words in \( D_0 \) (or in \( D_1 \)) that differ in exactly one bit.

Why not?

Because if so, then \( C_0 \) (or \( C_1 \)) would have two words differing in exactly one bit (since their 1st bits are the same) and this cannot happen if \( C \) is a single-bit-error detecting code.

What does this tell us about the number of code words in \( C \)?

\[
\#\text{code words in } C = \#\text{of words in } C_0 + \#\text{words in } C_1 = \#\text{of words in } D_0 + \#\text{words in } D_1
\]

By inductive hypothesis: \( D_0 \) and \( D_1 \) are single-bit-error detecting codes and can have at most \( 2^{n-1} \) words.

\[\therefore \# \text{code words in } C \leq 2^{n-1} + 2^{n-1} = 2^n \quad \text{QED}\]

Generalization of single-bit-error detection:

if \( w \) and \( x \) are \( n \)-bit words, we say that the distance between \( w \) and \( x \), \( \text{dist}(w, x) \) is the number of bits that are different between the two words.

\[\text{e.g. } \quad \text{dist}(0011, 1001) = 2\]
Hamming distance of a code

For some n-bit code $C$, the Hamming distance of $C$ is $\min_{u, v \in C} \text{dist}(u, v)$ (the minimum distance between any 2 code words).

**Theorem:** If $C$ is a code with Hamming distance $d \geq 2$, then $C$ is a $(d-1)$-bit-error detecting code.

**Proof:** Since $C$ has Hamming dist. $d$, any $d-1$ bit errors cannot change one code word into another code word. \therefore we can always detect the errors.

Error correcting codes:

Suppose a code has a Hamming distance of 3; e.g.,
(big dots are code words, small dots are not code words)

[Diagram of code words and distances]

Pick any code word, e.g.

Suppose a single bit error occurs, we will wind up in the circle shown.

Suppose same thing happens w/ a neighboring code word, then we get the 2nd code word.

Can we correct single-bit-errors? How?

Draw circles around all code words as shown, if received signal falls in the circle of a particular code word, say $C$, then
Theorem - If a code $C$ has Hamming distance $d \geq 2$, then $C$ can be used as either a $(d-1)$-error detecting code or a $\lceil \frac{d-1}{2} \rceil$-error correcting code. (Hamming designed the most efficient possible single-error-correcting codes - they are called Hamming codes.)

II) Fast Fourier Transform

(ask what students know about FT, DFT, FFT)

Discrete Fourier Transform is defined as: $X(n) = \sum_{k=0}^{N-1} x(k) e^{-j \frac{2\pi k n}{N}}$, $n=0,1,\ldots,N-1$

where $x(0), x(1), x(2), \ldots, x(N-1)$ are samples of a signal $x$.

Let $W_N = e^{-j \frac{2\pi}{N}}$, then $X(n) = \sum_{k=0}^{N-1} x(k) W_n^{kn}$ (1)

(just a notation simplification)

What is the time complexity of directly computing DFT according to (1)? $O(n^2)$ - why?

Let us rewrite (1) as:

$X(n) = \sum_{r=0}^{\frac{N}{2}-1} x(2r) W_n^{2rn} + \sum_{r=0}^{\frac{N}{2}-1} x(2r+1) W_n^{(2r+1)n}$

even terms of (1) odd terms of (1)
\[ X(n) = \sum_{r=0}^{N/2-1} X(2r) W_n^{2rn} + W_n^n \sum_{r=0}^{N/2-1} X(2r+1) W_n^{2rn} \]

now \( W_n^{2rn} = e^{j2\pi n/2} = W_n^m \)

\[ X(n) = \sum_{r=0}^{N/2-1} X(2r) W_n^{rn} + W_n^n \sum_{r=0}^{N/2-1} X(2r+1) W_n^{rn} \]

\[ \text{N/2-point DFT, G(n)} \]

\[ \text{N/2-point DFT, H(n)} \]

thus, we have just found a divide-and-conquer solution to calculating DFT

\[ G(n), H(n) \text{ are periodic} \Rightarrow G(0) = G(N/2), G(1) = G(N/2 + 1) \]

\[ \text{same is true for H(n)} \]
I won't go into details but we could also split each $\frac{N}{2}$-point DFT into 2 $\frac{N}{4}$-point DFTs, etc. doing this yields a set of butterfly computations at each recursive stage over $\log_2 N$ stages (why?) (this is Fast Fourier Transform or FFT Alg.)

in terms of running time, we get the following recurrence equation:

$$T(N) = 2T\left(\frac{N}{2}\right) + bN, \quad T(2) = b$$

time to compute an $N$-point DFT

this is of the form $T(N) = cT\left(\frac{N}{d}\right) + bN^k$

with $c = d = 2$ and $k = 1$

$\Rightarrow$ $T(N)$ is $O(N \log N)$

8-point FFT example:

<table>
<thead>
<tr>
<th>Input</th>
<th>Direct DFT</th>
<th>FFT</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$X(0)$</td>
<td>4096</td>
<td>4096</td>
<td>683</td>
</tr>
<tr>
<td>$X(1)$</td>
<td>1024</td>
<td>1024</td>
<td>205</td>
</tr>
<tr>
<td>$X(2)$</td>
<td>256</td>
<td>256</td>
<td>64</td>
</tr>
<tr>
<td>$X(3)$</td>
<td>67,009,000</td>
<td>48,960,000</td>
<td>1</td>
</tr>
</tbody>
</table>