Application: Designing algorithms

Sets and n-tuples can be represented in computer memories in a number of ways.

We can design algorithms in terms of variables that represent sets.

Often using set notation is clearer than designing in a programming language.

Clearer algorithms means errors are easier to spot.

Example: A design for a spell checker

Notation $\mathbf{var}\ v : T$ introduces a variable called v whose values are restricted to elements of set T.

Notation $v := E$ assignment to v.

String the set of all strings.

Spell check algorithm

---

var *good* : $\mathcal{P}$(String)
var *suggestions* : $\mathcal{P}$(String$\times$String)
read *good* and *suggestions* from a file
while there are more input words {
    var *word, r* : String
    read *word*
    if *word* $\notin$ *good*
        prompt the user for a replacement suggesting
            $\{ w \mid (word,w) \in suggestions \}$ as possibilities
        if the user says "add" then
            good := *good* $\cup$ {*word*}
            write *word*
        else if the user says replace
            r := the replacement word specified
            suggestions := *suggestions* $\cup$ {(*word*,*r*)}
            write *r*
        else the user says ignore
            write *word*
write *good* and *suggestions* to file

Application: Error correcting codes.

In communication channels sometimes a 1 bit is changed to a 0 or the other way.

To combat this we use error correcting codes and error detecting codes.

Define $A \Delta B$ to mean $(A - B) \cup (B - A)$ i.e. the set of things in exactly one of the two sets.

The "Hamming distance"of two sets A and B is $|A \Delta B|$, the number of things in exactly one set.

In our example we'll use a mapping from hexadecimal digits to sets in $\mathcal{P}(\{0, 1, ..., 7\})$

$$
\begin{aligned}
0 &\longmapsto \{0,1,2,3,4,5,6,7\} \\
1 &\longmapsto \{0,1,2,4\} \\
2 &\longmapsto \{0,1,3,7\} \\
3 &\longmapsto \{0,2,6,7\} \\
4 &\longmapsto \{0,1,5,6\} \\
5 &\longmapsto \{0,4,5,7\} \\
6 &\longmapsto \{0,3,4,6\} \\
7 &\longmapsto \{0,2,3,5\} \\
8 &\longmapsto \emptyset \\
9 &\longmapsto \{3,5,6,7\} \\
A &\longmapsto \{2,4,5,6\} \\
B &\longmapsto \{1,3,4,5\} \\
C &\longmapsto \{2,3,4,7\} \\
D &\longmapsto \{1,2,3,6\} \\
E &\longmapsto \{1,2,5,7\} \\
F &\longmapsto \{1,4,6,7\}
\end{aligned}
$$

The sets in the mapping are chosen so that for any two sets $A \neq B$, we have $|A \Delta B| \geq d$ where d is called the Hamming distance of the code.

In the example $|A \Delta B| \geq 4$ for all $A \neq B$.

Encoding Method

- Any message to be sent will be encoded as a binary string. E.g.

$$100111011110$$

- The string is broken into blocks of say 4 bits

$$1001,\ 1101,\ 1110 \text{ or in hex } 9,\ D,\ E$$

- Each block is mapped to a subset of small natural numbers. E.g. from $P(\{0, 1, ..., 7\})$

$$\{3, 5, 6, 7\},\ \{1, 2, 3, 6\},\ \{1, 2, 5, 7\}$$

- The sets are encoded in binary so that $n \in S$ iff 1 at position n

$$00010111,\ 01110010,\ 01100101$$

Decoding Method

- The bits may flip in transit.

$$00010111,\ 01110011,\ 11100100$$

- Each 8 bit segment is mapped to a set so that $n \in S$ iff 1 at position n

$$\{3,5,6,7\},\ \{1,2,3,6,7\},\ \{0,1,2,5\}$$

- Each set received is mapped to the closest (by Hamming distance) set in the code, if any:

$$\{3,5,6,7\},\ \{1,2,3,6\},\ \text{none}$$

  We can then map back to binary:

$$1001,\ 1101,\ \text{error}$$

The example code corrects single-bit errors and detects double-bit errors (1C2D) within each block.

Why? When we receive a block, then following cases can hold.

- 0 bit difference. The set received is equal to a set in the code.
- 1 bit error.
  - The set received has a Hamming distance of 1 from the correct set.

  - The closest it could be to some other set in the code is 3,
  - since otherwise two sets in the code would have a distance of less than 4.
- 2 bit difference.
  - The set received has a Hamming distance of 2 from the correct set.
  - The closest it could be to some other set in the code is 2.
  - So the error will be detected — but may not be correctable.

By choosing a Hamming distance of 5, all double bit errors can be corrected (2C).

With Hamming distance 6, all triple bit errors can be detected (2C3D).

Sometimes it is useful to think of the code words as bit-vectors and sometimes as sets.