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×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* ∪ {*word*}
            write *word*
        else if the user says replace
            r := the replacement word specified
            suggestions := *suggestions* ∪ {(*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 of the sets.

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

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

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 "octets" so that $n \in S$ iff 1 at position n
$$00010111,\ 01110010,\ 01100101$$

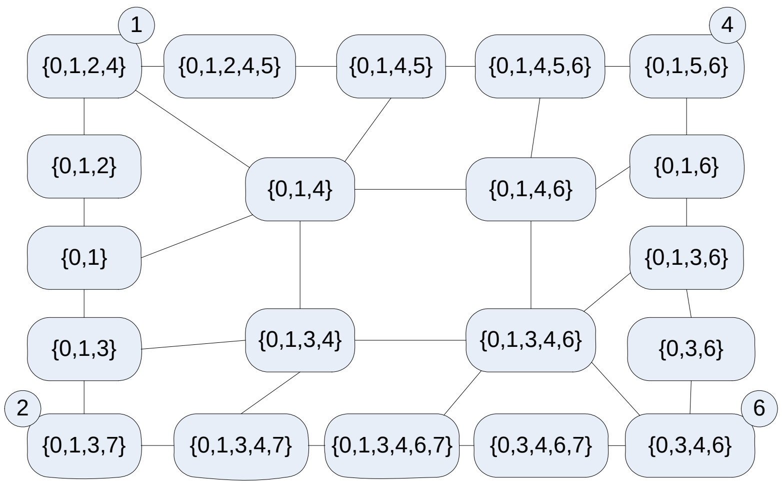
Decoding Method

- The bits may flip in transit.
$$00010111,\ 0111001\underline{1},\ \underline{1}110010\underline{0}$$
- Each 8 bit segment is mapped to a set so that $n \in S$ iff 1 at position n
$$\{3,5,6,7\},\ \{1,2,3,6,7\},\ \{0,1,2,5\}$$
- If the set is distance 0 or 1 from a set in the code, we use that set. Otherwise, an error is detected.
$$\{3,5,6,7\},\ \{1,2,3,6\},\ \text{error}$$
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 it works?

Consider a graph with points representing each of the 258 subsets of $\{0,1,\ldots,7\}$ and lines between two points if their Hamming distance is 1. Here is a small portion of the graph.



Suppose S and T are two points in the graph, the Hamming distance is the length of the shortest path between the points.

This means that $|S\Delta T| \leq |S\Delta U| + |U\Delta T|$, for any sets S, T, and U. Why? There is a path from S to U of length $|S\Delta U|$ and there is a path from U to T of length $|U\Delta T|$. Putting these paths together we get a path of length $|S\Delta U| + |U\Delta T|$ from S to T, so the shortest path can't be longer than that.

The Hamming distance of the code is 4.

Suppose S is in the code and U is any subset of $\{0,1,\ldots,7\}$. such that $|S\Delta U| = 1$.

- Let T be any set in the code other than S. Thus $|S\Delta T| \geq 4$.
- Then $|U\Delta T| \geq 3$.
- Why? Suppose $|U\Delta T| < 3$. Then we could construct a path from S to T of length less than 4 by going through U. In other words we'd have

$$|S\Delta T| \quad \leq \quad |S\Delta U| + |U\Delta T| \quad = \quad 1 + |U\Delta T| \quad < \quad 4$$

Now suppose S is in the code and U is any subset of $\{0,1,\ldots,7\}$. such that $|S\Delta U| = 2$.

- Let T be any set in the code other than S. Thus $|S\Delta T| \geq 4$.
- Then $|U\Delta T| \geq 2$.

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 errors.
  - 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 will not be correctable.
- More than 2 bits errors.
  - The error may go undetected.

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.

Why this is practical

Suppose that bit errors are randomly distributed and there is one error in 1,000,000 bits.

(On a 8 MBit/s channel, this is about 8 errors a second!)

Now suppose we use the above code.

- Chance that an octet has no errors
$$\left(\frac{999999}{1000000}\right)^8 \simeq 0.999\,99$$
- Chance that an octect has a one bit error
$$\frac{1}{1000000} \times \left(\frac{999999}{1000000}\right)^7 \times 8 \simeq 7.999\,9 \times 10^{-6}$$
- Chance that there is a 2 bit error
$$\left(\frac{1}{1000000}\right)^2 \times \left(\frac{999999}{1000000}\right)^6 \times \binom{8}{2} \simeq 2.800\,0 \times 10^{-11}$$
- Chance of a 3, 4, 5, 6, 7, or 8 bit error
$$\sum_{i=3}^{8} \left(\frac{1}{1000000}\right)^i \times \left(\frac{999999}{1000000}\right)^{8-i} \times \binom{8}{i} \simeq 5.600\,0 \times 10^{-17}$$
On an 8 MBit/s channel this is less than 1 undetected error every $18,000$ years.