A Puzzle ("Post’s Correspondence Problem")

Given a set of cards:
- \( N \) card types (can use as many of each type as possible).
- Each card has a top string and bottom string.

**Example 1:**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>N = 4</td>
</tr>
</tbody>
</table>

**Puzzle:**
- Is it possible to arrange cards so that top and bottom strings are the same?

**Solution 1.**

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>0</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

PCP Puzzle Contest

**Contest:**
- Additional restriction: string must start with ‘S’.
- Be the first to solve this puzzle!
  - (no credit, just fame and acclamation)
- Check solution by putting STRING ONLY (blanks and line breaks OK) in a file `solution.txt`, then type
  
  
  ```
  /u/cs126/bin/pcp < solution.txt
  ```
- Extra credit for the bored:
  - Write a program that reads a set of Post cards, and determines whether or not there is a solution.
**Why Learn Theory**

In theory . . .
- Deeper understanding of what is a computer and computing.
- Foundation of all modern computers.
- Pure science.
- Philosophical implications.

In practice . . .
- Web search: theory of pattern matching.
- Sequential circuit: theory of finite state automata.
- Compilers: theory of context free grammar.
- Cryptography: theory of complexity.
- Data compression: theory of information.

---

**Finite State Automata**

Simple machine with N states.
- Start in state 0.
- Read an input bit.
- Move to new state
  - depends on input bit and current state
- Stop when last bit read.
  - 'yes' if end in accept state(s)
  - 'no' otherwise

'Yes' also called accepted or recognized inputs from a language.
C Code for FSA

```c
#include <stdio.h>

int main(void) {
    int c, state = 0;
    while ((c = getchar()) != EOF) {
        if (state == 0 && c == '0') state = 2;
        if (state == 0 && c == '1') state = 1;
        if (state == 1 && c == '0') state = 3;
        if (state == 1 && c == '1') state = 1;
        if (state == 2 && c == '0') state = 2;
        if (state == 2 && c == '1') state = 2;
        if (state == 3 && c == '0') state = 2;
        if (state == 3 && c == '1') state = 1;
    }

    if (state == 3)
        printf("Yes.\n");
    else
        printf("No.\n");
    return 0;
}
```

Better C Code for FSA

```c
#include <stdio.h>

#define STATES          4
#define ALPHABET_SIZE   2
#define START_STATE     0
#define ACCEPT_STATE    3

int main(void) {
    int c, state = START_STATE
    int transition[STATES][ALPHABET_SIZE] =
        { {2, 1}, {3, 2}, {2, 2}, {2, 1} };

    while ((c = getchar()) != EOF)
        if (c >= '0' && c < '0' + ALPHABET_SIZE)
            state = transition[state][c - '0'];

    if (state == ACCEPT_STATE) printf("Yes.\n");
    else printf("No.\n");
    return 0;
}
```

A Second Example

Consider the following two state FSA.

What bit strings does it accept?
- Yes: 0, 11110, 00000, 100100111011, all bit strings with an odd number of 0's.
- No: 1, 1111, 00, 1011100111011, all bit strings with an even number of 0's.

A Third Example

Build an FSA that accepts all strings that contain 'acat' as a substring.
- tgacatg
- acacatg

Start building:
A Third Example

Build an FSA that accepts all strings that contain 'acat' as a substring.

- tgacatg
- acacatg

Finish building:

![Finite State Automaton Diagram]

An Application: Bounce Filter

Bounce filter: remove isolated b's and g's in input.

- Input: b b g b b b g g g b b b b
- Output (one-bit delay): b b b b b b g g g g g g g b b b b

State interpretations.

- 0: start
- BB: at least two consecutive b's.
- G: sequence of b's followed by g.
- GG: at least two consecutive g's.
- B: sequence of g's followed by b.

egrep

General regular expressions pattern matching.

- Acts as filter.
- Sends lines from stdin to stdout that "match" argument string.

Elementary Examples

```
% egrep 'beth' classlist
03/Smythe/Elizabeth/6/esmythe
03/Bethke/Kristen/3/kbethke
% egrep '/3/' classlist
03/Marin/Anthony/3/amarin
03/Arellano/Belen/3/arellano
...03/Weiss/Jacob/3/weiss
% egrep 'zeuglodon' mobydick.txt
rechristened the monster zeuglodon and in his
% egrep 'acat' human.data
gcaacgcacacaacatgcatttt
```

Find all lines in file classlist with substring 'beth'

List all people in precept 3.
Crossword Puzzle or Scrabble Too Hard?

/usr/dict/words is a list of (25,143) words in dictionary.

More Examples

% egrep 'hh' /usr/dict/words
beachhead
highhanded
withheld
withhold

% egrep 'u.u.u' /usr/dict/words
cumulus

% egrep '...oo...o' /usr/dict/words
bloodroot
schoolbook
schoolroom

Egrep Pattern Conventions

Conventions for egrep:

c  any non-special character matches itself
.  any single character
r*  zero or more occurrence of r
(r) grouping
r1|r2 logical OR
[aeiou]  any vowel
[^aeiou] any non-vowel
^  beginning of line
$  end of line

Flags for egrep:

egrep -v match all lines except those specified by pattern

Still More Examples

Unix

% egrep 'n(ie|ei)ther' /usr/dict/words
neither

% egrep 'actg(atac)*gcta' human.data
ggtactggctaggac

% egrep 'actg(atac)*gcta' student.data
tatactgatacatacatacgtattac

% egrep '^y.(..)*y$' /usr/dict/words
yesterday

% egrep -v '[aeiou]' /usr/dict/words
rhythm
syzygy

Fundamental Questions: Theoretical Minimum

Which aspects are essential?

- Unix egrep regular expressions are useful.
- But more complex than theoretical minimum.

egrep theoretical minimum:

c  any non-special character matches itself
.  any single character
r*  zero or more occurrence of r
(r) grouping
r1|r2 logical OR
[aeiou]  any vowel
[^aeiou] any non-vowel
^  beginning of line
$  end of line

not needed
**Fundamental Questions: What Kinds of Patterns**

What kinds of patterns can be specified? (all but one of following)

<table>
<thead>
<tr>
<th>All bit strings that:</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin with 0 and end with 1.</td>
<td>0010110111</td>
</tr>
<tr>
<td>Have more 1's than 0's.</td>
<td>01111001100</td>
</tr>
<tr>
<td>Have no consecutive 1's.</td>
<td>01001010010</td>
</tr>
<tr>
<td>Has and odd number of 0's.</td>
<td>01001010010</td>
</tr>
<tr>
<td>Has 011010 as a substring.</td>
<td>00011101000</td>
</tr>
</tbody>
</table>

**Fundamental Questions: What Kinds of Patterns**

What kinds of patterns can be specified? (all but one of following)

<table>
<thead>
<tr>
<th>All bit strings that:</th>
<th>Regular Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>Begin with 0 and end with 1.</td>
<td>0(0</td>
</tr>
<tr>
<td>Have more 1's than 0's.</td>
<td>not possible</td>
</tr>
<tr>
<td>Have no consecutive 1's.</td>
<td>(0</td>
</tr>
<tr>
<td>Has and odd number of 0's.</td>
<td>(1<em>01</em>01*)<em>(1</em>01*)</td>
</tr>
<tr>
<td>Has 011010 as a substring.</td>
<td>(0</td>
</tr>
</tbody>
</table>

**Formal Languages**

An alphabet is a finite set of symbols.

- Binary alphabet = \{0, 1\}
- Lower-case alphabet = \{a, b, c, d, ..., y, z\}
- Genetic alphabet = \{a, c, t, g\}

A string is a finite sequence of symbols in the alphabet.
- ’0111011011’ is a string in the binary alphabet.
- ‘tigers’ is a string in the lower-case alphabet.
- ‘acctgaacta’ is a string in the genetic alphabet.

A formal language is an (unordered) set of strings in an alphabet.

- Can have infinitely many strings.
- Examples:
  \{0, 010, 0110, 01110, 011110, 0111110, \ldots\}
  \{11, 1111, 111111, 11111111, 111111111, \ldots\}

**Formal Languages**

Can cast any computation as a language recognition problem.

- Is x = 23,536,481,273 a prime number?
  - FSA.
  - Machine determines whether a string is in language.
- Regular expression.
  - Shorthand method for specifying a language.
Duality Between FSA’s and RE’s

Observation: for each FSA we create, we can find a regular expression that matches the same strings that the FSA accepts.

Is this always the case?

What about the OTHER way around?

I don’t see why?

Stay tuned: see Lecture T2.

Limitations of FSA

No FSA can recognize the language of all bit strings with an equal number of 0’s and 1’s.

Suppose an N-state FSA can recognize this language.

Consider following input: 000000011111111

FSA must accept this string.

Some state x is revisited during first N+1 0’s since only N states.

Machine would accept same string without intervening 0’s.

This string doesn’t have an equal number of 0’s and 1’s.

Looking Ahead

Today.

Defined a simple abstract machine = FSA.

Capable of pattern matching.

Incapable of “counting.”

Need to consider more powerful machines.

Future lectures.

Define an abstract machine.

Understand how it works and what it can do.

Find things it can’t do.

Define a more powerful machine.

Repeat until we run out of problems or machines.
A Fourth Example

FSA to decide if integer (represented in binary) is divisible by 3?

What bit strings does it accept?
- Yes: $11 (3_{10}), 110 (6_{10}), 1001 (9_{10}), 1100 (12_{10}), 1111 (15_{10}), 10011 (18_{10})$, integers divisible by 3.
- No: $1 (1_{10}), 10 (2_{10}), 100 (4_{10}), 101 (5_{10}), 111 (7_{10})$, integers not divisible by 3.

How does it work?
- State 0: input so far is divisible by 3.
- State 1: input has remainder 1 upon division by 3.
- State 2: input has remainder 2 upon division by 3.
- Transition example.
  - Input $1100 (12_{10})$ ends in state 0.
  - If next bit is 0 then stay in state 0: $11000 (24_{10})$.
  - Adding 0 to last bit is same as multiplying number by 2. Remains divisible by 3.