(define (filter! f lst)
  (let ((root (cons '() lst)))
    (define (iter l)
      (cond ((or (null? l) (null? (cdr l))) '())
            ((f (cadr l)) (iter (cdr l)))
            (else (set-cdr! l (cddr l)) (iter l))))
    (iter root) (cdr root)))

Search

• Motivations
  – Play tic-tac-toe
  – Play chess
  – Play Darwin*

* Except in Kansas

The Human Genome Project

• Good news: truckloads of data
• Bad news: what does it mean?
• Figure it out (in part) by matching
  – match unknown sequence against sequences of known functionality
  – the hope: similarity of structure suggests similarity of function

The Human Genome Project

• human DNA is a string of ~3 billion letters (A, T, G, C), making up about 20,000 genes

Central Dogma of Modern Biology

• DNA encodes genes and is inherited
• DNA is transcribed under control of proteins into RNA
• RNA is translated into proteins by ribosomes
• Proteins run the cell, and thus organisms

Genetics

• Proteins are made up of amino acids
• DNA represents each amino acid by a triple of letters in the “alphabet” of 4 nucleotides: adenine, thymine, guanine, cytosine.
• Hence
  – two similar sequences of DNA letters ➔
  – two similar sequences of amino acids ➔
  – two similar structures in proteins ➔
  – similar biochemical behavior of the proteins
Matching

unk: a t c g c c t a t t g t c g a c

known: a t a g c c g t c a t c g a c

The Biology Behind Matching

• Evolution happens.*
• Changes to the genome during replication:
  – Point mutations: change a letter, e.g., C → A
  – Omissions: drop a letter
  – Insertions: insert a letter
• Similarity of sequence useful to discover
  – Similarity of function
  – Evolutionary history

* Except in ...

More Complex Example

a a t c a g c a g c t c a t c g a c g
a g a t c a g c a c t c a t c g a c g

Matching

• Every differing position has 3 possible explanations:
  – mutation
  – insertion
  – deletion

Matching As Tree Search

Every path through the tree is an hypothesis about how one sequence matches another

Depth first search
Breadth first search

If it's 6.001

• It's gotta have code:

```scheme
(define (dfsearch start-state)
  (define (search1 queue)
    (cond
     ((null? queue)
      (display "done"))
     (else
      (display "visiting ")
      (display (car queue))
      (search1 (append (children (car queue))
                   (cdr queue))))
    (search1 (list start-state))))
```

If it's 6.001

• It's gotta have code:

```scheme
(define (bfsearch start-state)
  (define (search1 queue)
    (cond
     ((null? queue)
      (display "done"))
     (else
      (display "visiting ")
      (display (car queue))
      (search1 (append (cdr queue)
                       (children (car queue)))))))
  (search1 (list start-state)))
```

Matching

Matching a t c a g c t a t t g t c g a c c a t a g c c t a t t g t c g a c a t a g c c t a t t g t c g a c a t a g c c t a t t g t c g a c

Define a Distance Metric

• Given two sequences, s1 & s2,
  – Distance is 0 if they are identical
  – Penalty for each point mutation
    • Different for different mutations
  – Penalty for insertion/deletion of nucleotides
  – “Distance” is sum of penalties
• Now we can get the best explanation.

Representing Mutation Penalty

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>C</th>
<th>G</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0</td>
<td>.3</td>
<td>.4</td>
<td>.3</td>
</tr>
<tr>
<td>C</td>
<td>.4</td>
<td>0</td>
<td>.2</td>
<td>.3</td>
</tr>
<tr>
<td>G</td>
<td>.1</td>
<td>.3</td>
<td>0</td>
<td>.2</td>
</tr>
<tr>
<td>T</td>
<td>.3</td>
<td>.4</td>
<td>.1</td>
<td>0</td>
</tr>
</tbody>
</table>
We have the Penalties

```scheme
(define omit-penalty .5)
(define insert-penalty 0.7)
```

Matching As Tree Search

```
Matching As Tree Search

```

Observation

```
Observation

```

Memory to the Rescue

```
Memory to the Rescue

```

• "Memoization"
• Store the results of computing sub-paths and substitute lookup for computation
• How to store the results?
• (Still, ~n^2)
Can We Be Smarter Still?

- Cut off bad paths:
  - Estimate an upper bound on matches of interest
  - Declare any match worse than this to be infinitely bad (and stop pursuing it)
- Advantages?
- Disadvantage?

Idea: Pursue “Best” Matches

Best First Search

- Extend only the best sequence

```
(define (bestfsearch start-state)
  (define (search1 queue)
    (if (null? queue)
      #f
      (let ((current (car queue)))
        (if (done? current)
            current
            (search1 (merge (sort (children (car queue)))
                         (cdr queue)))))))

(search1 (list start-state)))
```

Beam Search

- Beam: like best-first, but keep only n best children of a node

```
Varieties of Search

- depth first
  (append (children (car queue)) (cdr queue))
- breadth first
  (append (cdr queue) (children (car queue)))
- best first
  (merge (sort (children (car queue))) (cdr queue))
- beam search
  (merge (list-head n (sort (children (car queue)))) (cdr queue))
```
Return of the Biologists

• Some large subsequences are common (clichés)
• Good matches will contain large identical subsequences
• Pre-compute table of all occurrences of specific patterns
• Extend match outward (both directions) from these exact matches

BLAST: Find common, extend

Generalize

• DNA
  – Nucleotides: A, C, T, G
  – Mutation rates
  – Insertion/omission penalties

• Proteins
  – Amino Acids: val, leu, ile, met, phe, asn, glu, gln, ...
  – Mutation rates
  – Insertion/omission penalties

Let's Play Games

Let's Play Games