Search

- Motivations
  - Play tic-tac-toe
  - Play chess
  - Play with the Web
  - Play Darwin*

*Except in Kansas …

The Human Genome Project

- human DNA is a string of ~3 billion letters (A, T, G, C), making up about 20,000-25,000 genes

"Genetics 101"

Cells are the fundamental working units of every living system. All the instructions needed to direct their activities are contained within the chemical DNA (deoxyribonucleic acid).

DNA from all organisms is made up of the same chemical and physical components. The DNA sequence is the particular side-by-side arrangement of bases along the DNA strand (e.g., ATTCCGGA). This order spells out the exact instructions required to create a particular organism with its own unique traits.

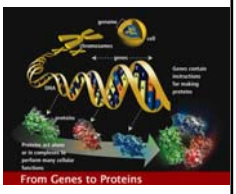
The **genome** is an organism's complete set of DNA. Genomes vary widely in size: the smallest known genome for a free-living organism (a bacterium) contains about 600,000 DNA base pairs, while human and mouse genomes have some 3 billion.

DNA in the human genome is arranged into 24 distinct **chromosomes**--physically separate molecules that range in length from about 50 million to 250 million base pairs. A few types of major chromosomal abnormalities, including missing or extra copies or gross breaks and rejoinings (translocations), can be detected by microscopic examination. Most changes in DNA, however, are more subtle and require a closer analysis of the DNA molecule to find perhaps single-base differences.

Each chromosome contains many **genes**, the basic physical and functional units of heredity. Genes are specific sequences of bases that encode instructions on how to make proteins. Genes comprise only about 2% of the human genome; the remainder consists of noncoding regions, whose functions may include providing chromosomal structural integrity and regulating where, when, and in what quantity proteins are made. The human genome is estimated to contain 20,000-25,000 genes.

"Genetics 101"

Although genes get a lot of attention, it's the **proteins** that perform most life functions and even make up the majority of cellular structures. Proteins are large, complex molecules made up of smaller subunits called amino acids. Chemical properties that distinguish the 20 different amino acids cause the protein chains to fold up into specific three-dimensional structures that define their particular functions in the cell.

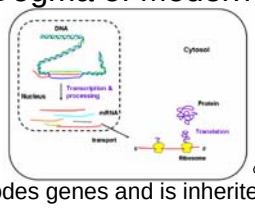


The constellation of all proteins in a cell is called its **proteome**. Unlike the relatively unchanging genome, the dynamic proteome changes from minute to minute in response to tens of thousands of intra- and extracellular environmental signals. A protein's chemistry and behavior are specified by the gene sequence and by the number and identities of other proteins made in the same cell at the same time and with which it associates and reacts. Studies to explore protein structure and activities, known as proteomics, will be the focus of much research for decades to come and will help elucidate the molecular basis of health and disease.

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

Central Dogma of Modern Biology



Kuo, JBI 37 (2004) 293–303

Recursion!

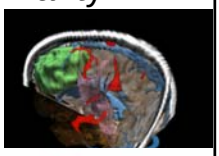
- DNA encodes genes and is inherited
- DNA is transcribed under control of proteins into RNA
- RNA is translated into proteins by ribosomes
- Proteins run the cell, and thus organisms

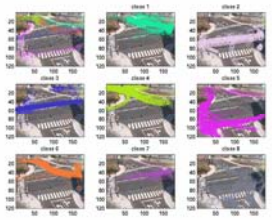
Genetics

- 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

Searching for similarity

- Same idea holds in other domains
  - Medical diagnosis and treatment
    - Find examples in database of similar cases and lookup treatment and prognosis
  - Marketing analytics
    - Look for patterns in custome usage and relate to custome behavior
  - Anti-terrorism analysis
    - Look for patterns in communication traffic or in actual physical movement patterns and relate to behaviors of groups





Matching in genetics case

```
unk:   a t c g c c t a t t g t c g a c c
       ↑ ↑ ↑ ↑ ↑
known: a t a g c a g c t c a t c g a c g
```


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

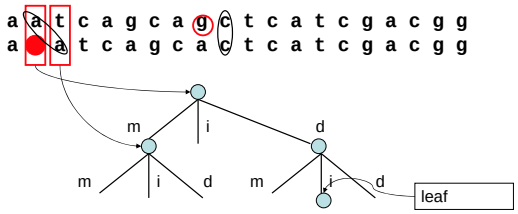
More Complex Example

```
a a t c t g c c t a t t g t c g a c g c
        m     m m m   m m           m
a a t c a g c a g c t c a t c g a c g g

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


a a t c a g c a g c t c a t c g a c g g
              ^
a g a t c a g c a c t c a t c g a c g g
```


Matching

- Every differing position has 3 possible explanations:
  - mutuation
  - insertion
  - deletion

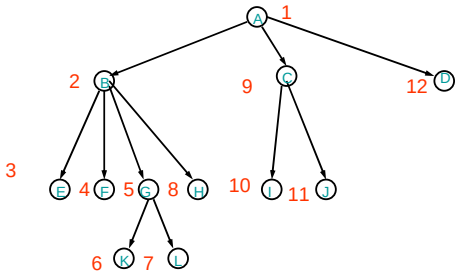
Matching As Tree Search



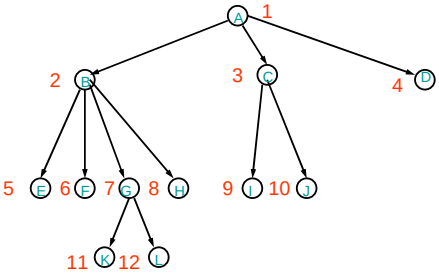
Every path through the tree is an hypothesis about matching sequences

Want to evaluate likelihood of path to a leaf of the tree, and compare to other paths to leaves – this lets us decide how similar two sequences are, and causes of differences in sequences

Depth first search



Breadth first search



If it's 6.001

- It's gotta have code:
  - Represent a tree by:
    - a root node (the start of the tree)
    - Plus a set of "children" nodes for each node, unless the node is a leaf (has no children)
  - Represent search by:
    - a queue of nodes to visit

If it's 6.001

- It's gotta have code:

```
(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)))
```

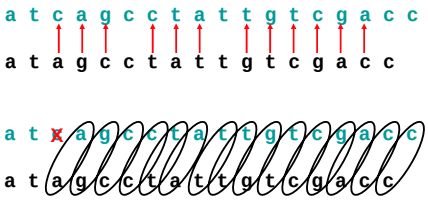

If it's 6.001

- It's gotta have code:

```
(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

Now that we can search a tree, how do we decide which paths are more interesting?



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

	A	C	G	T
A	0	.3	.4	.3
C	.4	0	.2	.3
G	.1	.3	0	.2
T	.3	.4	.1	0

2-D Table

```
(define point-mutations (make-table2))
(table2-set! point-mutations 'A 'A 0)
(table2-set! point-mutations 'A 'C .3)
(table2-set! point-mutations 'A 'G .4)
...

(table2-get point-mutations 'A 'C)
>> .3
(table2-get point-mutations 'A 'X)
>> #f
```

- But how to implement a 2-D table?

A Table Abstraction using alists

```
(define (find-assoc-binding key alist)
  (cond ((null? alist) #f)
        ((equal? key (caar alist)) (car alist))
        (else (find-assoc-binding key (cdr alist)))))

(define (find-assoc key alist)
  (let ((binding (find-assoc-binding key alist)))
    (if binding
        (cadr binding)
        #f)))

(define (add-assoc key val alist)
  (cons (list key val) alist))
```

Note Scheme's
assoc
assv
assq

Non-Abstract but Compact!

```
(define mutation-penalties
  '((a       (c .3) (g .4) (t .3))
    (c (a .4)       (g .2) (t .3))
    (g (a .1) (c .3)       (t .2))
    (t (a .3) (c .4) (g .1)       )))

(define (mutation to from)
  (if (eq? from to)
      0
      (let ((row (find-assoc-binding ; == assoc
                  to mutation-penalties)))
        (if row
            (find-assoc from (cdr row)); == cadr of assoc
            #f))))
```

A Table ADT

```
(define table1-tag 'table1)
(define (make-table1)
  (cons table1-tag '()))
(define (table1-get tbl key)
  (find-assoc key (cdr tbl)))
(define (table1-set! tbl key val)
  (set-cdr! tbl
            (add-assoc! key val (cdr tbl))))
```

- Note: we mutate structure, unlike before.

Mutating Version of add-assoc

```
(define (add-assoc! key val alist)
  (let ((binding (find-assoc-binding key alist)))
    (cond (binding
           (set-car! (cdr binding) val)
           alist)
          (else
           (add-assoc key val alist)))))
```


Table2 is a table of Table1's

```
(define table2-tag 'table2)
(define (make-table2)
  (cons table2-tag (make-table1)))

(define (table2-get tbl key-row key-col)
  (let ((row (table1-get (cdr tbl) key-row)))
    ;; row is itself a table1!
    (if row (table1-get row key-col) #f)))

(define (table2-set! tbl key-row key-col val)
  (let ((row (table1-get (cdr tbl) key-row)))
    (if row
        (table1-set! row key-col val)
        (let ((new-row (make-table1)))
          (table1-set! new-row key-col val)
          (table1-set! (cdr tbl) key-row new-row)))))
```


Defining Mutations More Abstractly

```
(table2-set! point-mutations 'a 'a 0)
(table2-set! point-mutations 'a 'c 0.3) ;; e.g., from c to a
(table2-set! point-mutations 'a 'g 0.4)
(table2-set! point-mutations 'a 't 0.3)
(table2-set! point-mutations 'c 'a 0.4)
(table2-set! point-mutations 'c 'c 0)
(table2-set! point-mutations 'c 'g 0.2)
(table2-set! point-mutations 'c 't 0.3)
(table2-set! point-mutations 'g 'a 0.1)
(table2-set! point-mutations 'g 'c 0.3)
(table2-set! point-mutations 'g 'g 0)
(table2-set! point-mutations 'g 't 0.2)
(table2-set! point-mutations 't 'a 0.3)
(table2-set! point-mutations 't 'c 0.4)
(table2-set! point-mutations 't 'g 0.1)
(table2-set! point-mutations 't 't 0)
```


We have the Penalties

```
point-mutations
>>
(table2
  table1
  (t (table1 (t 0) (g 0.1) (c 0.4) (a 0.3)))
  (g (table1 (t 0.2) (g 0) (c 0.3) (a 0.1)))
  (c (table1 (t 0.3) (g 0.2) (c 0) (a 0.4)))
  (a (table1 (t 0.3) (g 0.4) (c 0.3) (a 0))))

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

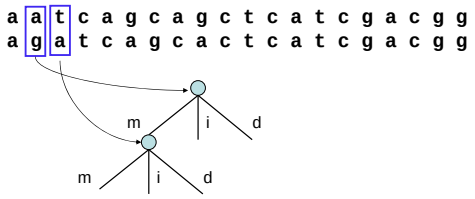

Simplest Matcher

… slooooooooooow!!!

$\approx 3^n$

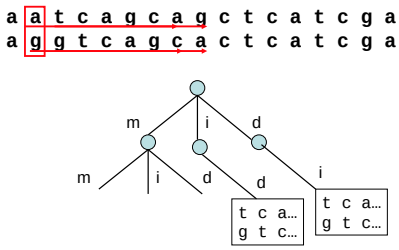
```
(define (match0 one two)
  (define (helper x y score)
    (cond ((and (null? x) (null? y)) score)
          ((null? x)
           (helper x (cdr y) (+ score omit-penalty)))
          ((null? y)
           (helper (cdr x) y (+ score insert-penalty)))
          ((eq? (car x) (car y))
           (helper (cdr x) (cdr y) score))
          (else
           (let ((mutated
                  (helper (cdr x) (cdr y)
                          (+ score (mutation (car x) (car y)))))
                 (omitted
                  (helper x (cdr y) (+ score omit-penalty)))
                 (inserted
                  (helper (cdr x) y (+ score insert-penalty))))
             (min mutated omitted inserted)))))
  (helper one two 0.0))
```

Matching As Tree Search



Time complexity? $T(n) = \Theta(3^n)$

Observation



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)

Remember Fibonacci

```
(define (fib n)
  (cond ((= n 0) 0)
        ((= n 1) 1)
        (else (+ (fib (- n 1)) (fib (- n 2))))))
(define old-vals (make-table1))

(define (fibmemo n)
  (let ((old-val (table1-get old-vals n)))
    (cond (old-val old-val)
          (else
           (let ((new-val
                  (cond ((= n 0) 0)
                        ((= n 1) 1)
                        (else (+ (fibmemo (- n 1))
                                 (fibmemo (- n 2)))))))
             (table1-set! old-vals n new-val)
             new-val)))))
```

$T(n) = \Theta(\phi^n)$

$T(n) = \Theta(n)$

Better Memoized Matching

```
(define (match1 one two)
  (let ((past (make-table2)))
    (define (helper x y score)
      (let ((old (table2-get past x y)))
        (if old (+ old score)
            (let ((new
                   <<guts of match0's helper>>))
              (table2-set! past x y (- new score))
              new))))
    (helper one two 0.0)))
```

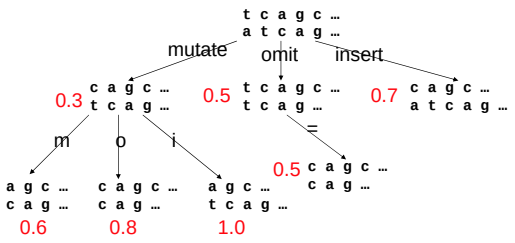
$T(n) = \Theta(n^2)$

- We store best score from *here* (x,y) to end.
- Still too slow for long sequences!
- Can we *not* consider some of the worst partial matches?

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

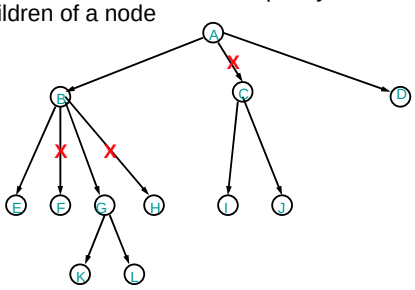
- Extend only the best sequence

```
(define (bestfsearch start-state)
 (define (search1 queue)
  (cond ((done? (car queue))
         (display "done")
         (car queue))
        (else
          (display "visiting ")
          (display (car queue))
          (search1 (merge (sort (children (car queue)))
                          (cdr queue))))))
  (search1 (list start-state)))
```

sort: take a list of states and reorder based on score of each state….
merge: take two sorted state lists and return sorted combined state list…

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))
  ```


General Search Framework

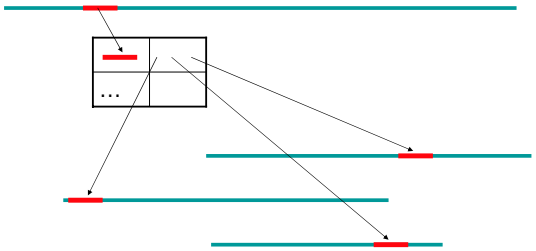
```
(define (search start-state done? succ-fn merge-fn)
  (define (search1 queue)
    (if (null? queue)
        #f
        (let ((current (car queue)))
          (if (done? current)
              current
              (search1
                (merge-fn (succ-fn current)
                          (cdr queue)))))))
  (search1 (list start-state)))
```

- Have we reached “goal”?
- Order in which to explore moves
- What “moves” can we make from current state?

Return of the Biologists

- Short queries, large databases…
- 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☐

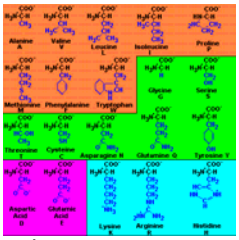


4/2/2007 44

Basic Local Alignment Search Tool (BLAST)

http://www.people.virginia.edu/~rjh9u/aminacid.html

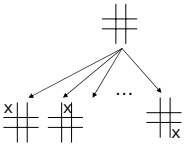
Generalize

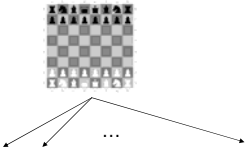


- DNA
  - Nucleotides: A, C, T, G
  - Mutation rates
  - Insertion/omission penalties
- Proteins
  - Amino Acids: val, leu, ile, met, phe, asn, glu, gln, …
  - Mutation rates
  - Insertion/omission penalties

4/2/2007 45

Let's Play Games…





4/2/2007 46