**CS216: Program and Data Representation**
University of Virginia Computer Science
Spring 2006 David Evans

# Lecture 4: Dynamic Programming, Trees

http://www.cs.virginia.edu/cs216

## Schedule This Week

- Sections today and Tuesday
  - Thornton D rooms
- Wednesday, Feb 1:
  - Go to Ron Rivest's talk
  - 10:30am-noon, Newcomb Hall South Meeting Room
- Thursday, Feb 2:
  - Portman Wills, "Saving the World with a Computer Science Degree"
  - 3:30pm, MEC 205 (CS290/CS390 +)



## RSA
(Rivest, Shamir, Adelman 1978)

- Public-Key Encryption
  - Allows parties who have not established a shared secret to communicate securely
  - Your web browser used it when you see
- Most important algorithm invented in past 100 years
- Link from notes




## Wednesday's Class

- Ron Rivest, "Security of Voting Systems"
- Newcomb Hall South Meeting Room, 10:30am-noon
- This replaces CS216 class (so you don't have to walk out at 11am)



## Sequence Alignment

- Brute force algorithm in PS1

```
bestAlignment (U, V) =
  base case if |U| == 0 or |V| == 0
  otherwise f(bestAlignment (U[1:], V),
              bestAlignment (U, V[1:]),
              bestAlignment (U[1:], V[1:])
```

- Compare to Fibonacci:

```
fibonacci (n) =
  base case if n == 0 or n == 1
  g(fibonacci (n-1),
    fibonacci (n-2))
```

Running time $\in \Theta(\phi^n)$
$\phi = 1.618\ldots$



## Sequence Alignment

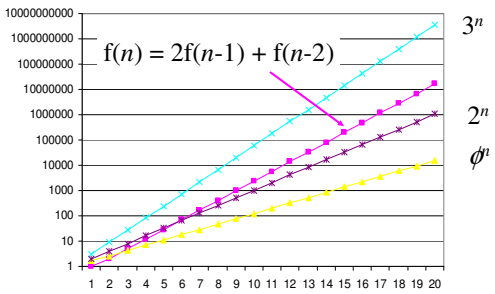
- Input size = $n = |U| + |V|$

```
bestAlignment (U, V) =
  base case if |U| == 0 or |V| == 0
  otherwise
    f(bestAlignment (U[1:], V),          size = n-1
      bestAlignment (U, V[1:]),          size = n-1
      bestAlignment (U[1:], V[1:])       size = n-2
```

$$a(n) = a(n\text{-}1) + a(n\text{-}1) + a(n\text{-}2)$$
$$> a(n\text{-}1) + a(n\text{-}2) \in \Theta(\phi^n)$$

Running time of bestAlignment $\in \Omega(\phi^n)$

# Growth of Best Alignment






# BLAST

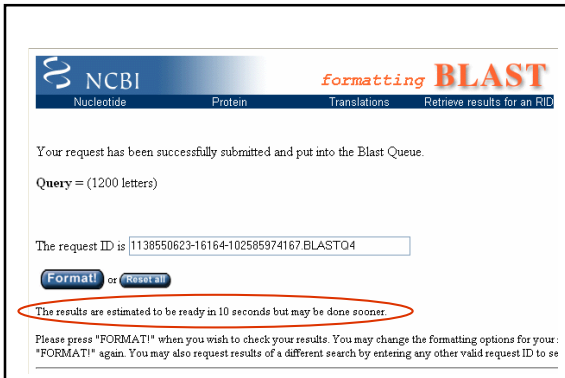
Basic Local Alignment Search Tool

http://www.ncbi.nlm.nih.gov/BLAST/

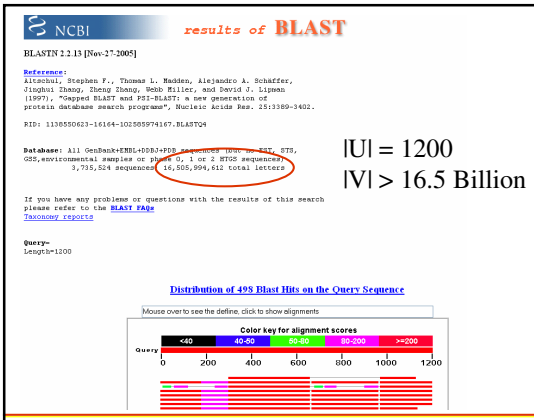
GCGTTGCTGGCGTTTTTCCATAGGCTCCGCCCCCCTGACGAGCATCACAAAAATCGACGCGGTGGCGAAACCCGAC
AGGACTATAAAGATACCAGGCGTTTCCCCCTGGAAGCTCCCTCGTGTTCCGACCCTGCCGCTTACCGGATACCTGTC
CGCCTTTCTCCCTTCGGGAAGCGTGGCTGCTCACGCTGTACCTATCTCAGTTCGGTGTAGGTCGTTCGCTCCAAGC
TGGGCTGTGTGCCGTTCAGCCCGACCGCTGCGCCTTATCCGGTAACTATCGTCTTGAGTCCAACCCGGTAAAGTAG
GACAGGTGCCGGCAGCGCTCTGGGTCATTTTCGGCGAGGACCGCTTTCGCTGGAGATCGGCCTGTCGCTTGCGGT
ATTCGGAATCTTGCACGCCCTCGCTCAAGCCTTCGTCACTCCAAACGTTTCGGCGAGAAGCAGGCCATTATCGCCG
GCATGGCGGCCGACGCGCTGGGCTGGCGTTCGCGACGCGAGGCTGGATGGCCTTCCCCATTATGATTCTTCTCGC
TTCCGGCGGCCCGCGTTGCAGGCCATGCTGTCCAGGCAGGTAGATGACGACCATCAGGGACAGCTTCAACGGCTCT
TACCAGCCTAACTTCGATCACTGGACCGCTGATCGTCACGGCGATTTATGCCGCACATGGACGCGTTGCTGGCGTT
TTTCCATAGGCTCCGCCCCCCTGACGAGCATCACAAACAAGTCAGAGGTGGCGAAACCCGACAGGACTATAAAGAT
ACCAGGCGTTTCCCCCTGGAAGCGCTCTCCTGTTCCGACCCTGCCGCTTACCGGATACCTGTCCGCCTTTCTCCCTT
CGGGCTTTCTCAATGCTCACGCTGTAGGTATCTCAGTTCGGTGTAGGTCGTTCGCTCCAAGCTGACGAACCCCCCG
TTCAGCCCGACCGCTGCGCCTTATCCGGTAACTATCGTCTTGAGTCCAACACGACTTAACGGGTTGGCATGGATTG
TAGGGCGCCGCCCTATACCTTGTCTGCCTCCCCGCGGTGCATGGAGCCGGGCCACCTCGACCTGAATGGAAGCCGGC
GGCACCTCGCTAACGGCCAAGAATTGGAGCCAATCAATTCTTGCGGAGAACTGTGAATGCGCAAACCAACCCTTGG
CCATCGCGTCCGCCATCTCCAGCAGCCGCACGCGGCGCATCTCGGGCAGCGTTGGGTCCT

Dinosaur DNA from Jurassic Park (p. 103)








|U| = 1200

|V| > 16.5 Billion



# Needleman-Wunsch Algorithm

- Avoid effort duplication by storing intermediate results
- Computing a matrix showing the best possible score up to each pair of positions
- Find the alignment by following a path in that matrix



N-W Initialization

Sequence $V$

Sequence $U$

| | - | C | A | T | G |
|---|---|---|---|---|---|
| - | 0 | | | | |
| A | | | | | |
| T | | | | | |
| G | | | | | |
| G | | | | | |

## Filling Matrix

Sequence $V$

Sequence $U$

| | - | C | A | T | G |
|---|---|---|---|---|---|
| - | 0 | | | | |
| A | | | | | |
| T | | | | | |
| G | | | | | |
| G | | | | | |

For each square consider:

← Gap in $U$
score = score[←] - $g$

↑ Gap in $V$
score = score[↑] - $g$

↖ No gap
match: score = score[↖] + $c$
no match: score = score[↖]



## Filling Matrix

| | - | C | A | T | G |
|---|---|---|---|---|---|
| - | 0 | -2 | -4 | -6 | -8 |
| A | -2 | 0 | 8 | 6 | 4 |
| T | -4 | | | | |
| G | -6 | | | | |
| G | -8 | | | | |

CATG
----
CATG
-A--

← score = score[←] - $g$
↑ score = score[↑] - $g$
↖ match: score = score[↖] + $c$
no match: score = score[↖]



## Finished Matrix

| | - | C | A | T | G |
|---|---|---|---|---|---|
| - | 0 | -2 | -4 | -6 | -8 |
| A | -2 | 0 | 8 | 6 | 4 |
| T | -4 | -2 | 6 | 18 | 16 |
| G | -6 | -4 | 4 | 16 | 28 |
| G | -8 | -6 | 2 | 14 | 26 |



## nding Alignment

Start in bottom right corner; follow arrows:
←gap V
↑gap U
↑no gap

| | | C | A | T | G |
|---|---|---|---|---|---|
| - | 0 | -2 | -4 | -6 | -8 |
| A | -2 | 0 | 8 | 6 | 4 |
| T | -4 | -2 | 6 | 18 | 16 |
| G | -6 | -4 | 4 | 16 | 28 |
| G | -8 | -6 | 2 | 14 | 26 |

| | | | | |
|---|---|---|---|---|
| C | A | T | – | G |
| – | A | T | G | G |



## N-W Correctness

- Informal argument:
  - Fills each cell by picking the best possible choice
  - Finds the best possible path using the filled in matrix
- Guaranteed to find the best possible alignment since all possibilities are considered



## N-W Analysis

- What is the space usage?

  Need to store the matrix:
  $= (|U| + 1) * (|V| + 1)$

# N-W Running Time

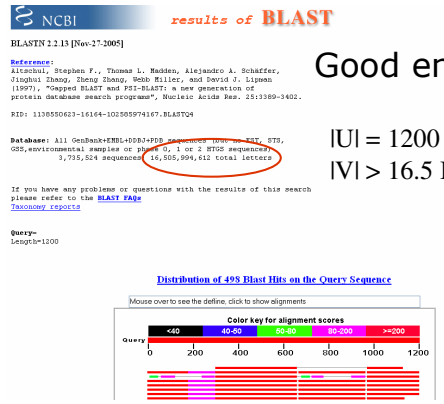
- Time to fill matrix
  - Each square $\in O(1)$
  - Assumes:
    - Lookups are $O(1)$
  - Time scales with number of cells:

    $\Theta(|U| * |V|)$
- Time to find alignment
  - One decision for each entry in answer

    $\Theta(|U| + |V|)$
- Total running time $\in \Theta(|U| * |V|)$

← score = score[←] - $g$
↑ score = score[↑] - $g$
↖ match: score = score[↖] + $c$
no match: score = score[↖]






Good enough?

|U| = 1200
|V| > 16.5 Billion



# Heuristic Alignment

- BLAST needs to be faster
- No asymptotically faster algorithm is guaranteed to find best alignment
  - Only way to do better is reject some alignments without considering them
- BLAST uses heuristics:
  - Looks for short sequences (~3 proteins = 9 nucleotides) that match well without gaps
  - Extend those sequences into longer sequences



# PS2

- Part 1 (1-3): List representations
- Part 2 (4-5): Dynamic programming implementation of sequence alignment
- Part 3 (6-10): Brute force implementation of phylogeny
  - Like alignment, brute force doesn't scale
  - This will be part of PS3

PS2 is longer and harder than PS1. You have 10 days to complete it - Get started early!

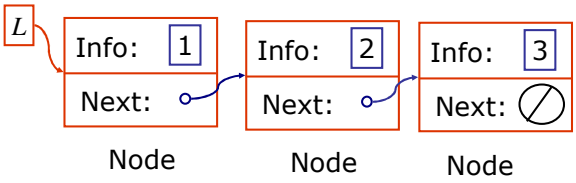


# Data Structures

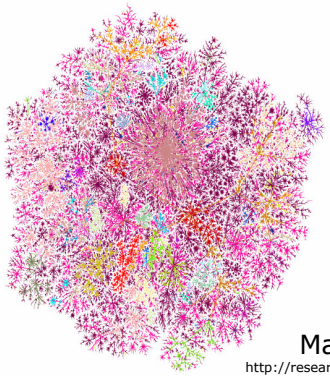
- If we have a good list implementation, do we need any other data structures?
- For computing: **no**
  - We can compute everything with just lists (actually even less). The underlying machine memory can be thought of as a list.
- For *thinking*: **yes**
  - Lists are a very limited way of thinking about problems.



# List Limitations




In a list, every element has direct relationships with only two things: predecessor and successor



## Complex Relationships

Bill Cheswick's
Map of the Internet
http://research.lumeta.com/ches/map/gallery/



## List → Tree

- List: each element has relationships with up to **2** other elements:

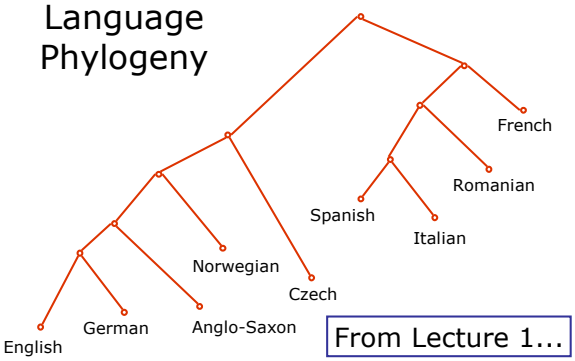
Predecessor → Element → Successor

- Binary Tree: each element has relationships with up to **3** other elements:

Parent
↓
Element
↙ ↘
Left Child | Right Child



## Language Phylogeny

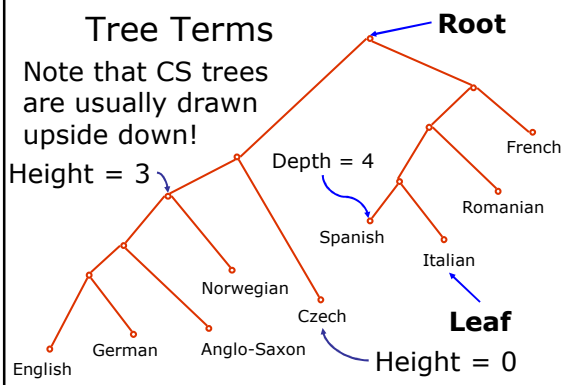



From Lecture 1…



## Tree Terms

Note that CS trees are usually drawn upside down!






## Tree Terms

- *Root*: a node with no parent
  - There can only be one root
- *Leaf*: a node with no children
- *Height* of a Node: length of the longest path from that node to a leaf
- *Depth* of a Node: length of the path from the Root to that node
- *Height* of a Tree: maximum depth of a node in that tree = height of the root



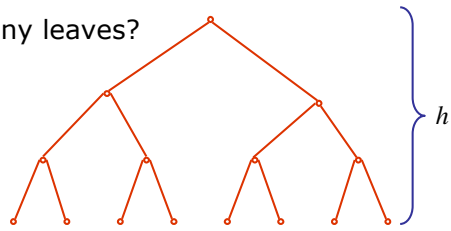
## Perfect Binary Tree

How many leaves?

$2^h$

How many nodes?

$2^{h+1}-1$




All leaves have the same depth

# Nodes = $1 + 2 + 2^2 + \ldots + 2^h$

## Tree Node Operations

- getLeftChild (Node)
- getRightChild (Node)
- getInfo (Node)

```
def isLeaf (self):
    return self.getLeftChild () == None
        and self.getRightChild () == None
```



## Calculating Height

*Height* of a Node: length of the longest path from that node to a leaf

```
def height (self):
  if self.isLeaf ():
    return 0
  else:
    return 1
      + max(self.getLeftChild().height(),
            self.getRightChild().height())
```



## Analyzing Height

- What is the asymptotic running time or our height procedure?

```
def height (self):
  if self.isLeaf ():
    return 0
  else:
    return 1
      + max(self.getLeftChild().height(),
            self.getRightChild().height())
```



## Charge

- Today and tomorrow:
  - Sections in Thorton D classrooms
- Wednesday:
  - Instead of CS216, go to Ron Rivest's talk
  - 10:30am, Newcomb Hall South Meeting Room
- Get started on PS2!