15-853:Algorithms in the Real World

Computational Biology I
- Introduction
- Longest Common Subsequence and Edit Distance

DNA

DNA: suquence of **base-pairs** (bp):
{**A, C, T, G**}

Human Genome

about 3×10^9 bps divided into
46 chromosomes with between
5×10^7 and 25×10^7 bps each

Each chromosome is a **sequence** of base-pairs

DNA is used to generate proteins:

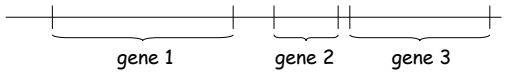
DNA $\xrightarrow{\text{transcription}}$ mRNA $\xrightarrow{\text{translation}}$ Protein

Proteins

Proteins: **sequence** of **Amino Acids**
{**gly, trp, cys,** …}

Each DNA bp triple (a "**codon**") forms 1 amino acid
Since there are 64 possible codons, this is a many to one mapping. Some triples have special meanings, e.g. EOF.

Chromosomes are partitioned into **genes** each of which codes a protein. Some regions of the chromosome do not code anything (**intergene DNA**).



Form and Function

The Amino Acid sequence determines the protein's 3d **structure** . The structure is also be affected by the environment.
- The **primary structure** refers to the amino acid sequence.
- The **secondary structure** refers to general configuration into **alpha helixs** and **beta sheets**
- The **tertiary structure** refers to the full 3d structure

Protein's 3d structure determines its **function**.

Some Goals in Molecular Biology

1. Extract genome sequence for various organisms.
2. Determine what proteins they code.
3. Determine structure and purpose of coded proteins.

Goals 2. and 3. can often be aided by **matching** genome or protein sequences to previously studied sequences

Use to:

- study and cure genetic diseases
- design drugs
- study evolution
- understand molecular processes

Example of MS

Multiple Sclerosis is a disease in which the immune system attacks the myelin sheaths of nerve cells

Conjecture: The immune system T-cells incorrectly identify the myelin sheaths as a virus or bacteria from an earlier infection.

This was tested by comparing the proteins sequences of myelin sheaths with a database of viral and bacterial proteins.

The ones that matched were tested in the laboratory to see if they were attacked by the T-cells. **Some were.**

Matching and Sequence Alignment

How similar are:

```
actagtctac
cgacgtcgata   ?
```

Allow for:

mutations: x to y
insertions: _ to y
deletions: x to _
} "indels"

e.g.

```
acta_gtc__tac
 |  | |||  ||
_cgacgtcgata_
```

1 mutation
3 insertions
2 deletions

Applications of matching and alignment

Used in many ways in computational biology

- Sequencing (finding the sequence of DNA or Proteins)
- Physical mapping (locating unique tags in DNA)
- Database searches (does this DNA match any other DNA?)
- Evolutionary trees (how similar are two species?)

Before talking about the general matching problem in computational Biology, we will talk about a closely related, but simpler problem: **longest common subsequence** (LCS)

Longest Common Subsequence

Subsequence (⊆): Any subset of the elements of a sequence that maintain relative order

e.g. $A = a_1a_2a_3a_4a_5a_6$

$A' = a_2a_4a_5$ (a subsequence of A)

$A' = a_2a_1a_5$ (not a subsequence of A)

Longest Common Subsequence (LCS):

LCS(A,B) = C, where $C \subseteq A$, $C \subseteq B$, $|C|$ is maximized

e.g. A = a b a c d a c

B = c a d c d d c

C = a c d c $|C| = 4$

Minimum Edit Distance

Minimum Edit Distance: D(A,B) = minimum # of insertions and deletions needed to change A to B.

e.g. A = a b a c d a c

B = c a d c d d c

delete

insert

Claim: D(A,B) = |A| + |B| - 2|LCS(A,B)|

Proof outline:

C = LCS(A,B)

A – C = deletions, B – C = insertions

#deletions = |A| - |C| #insertions = |B| - |C|

This reduction works both ways, hence equality.

Applications

Unix `diff`:

Find Minimum Edit Distance and print edits

GNU diff based on algorithm by Eugene Myers, who is VP of Informatics at Celera

Screen redisplay:

Find minimum number of changes that need to be sent to the display along a "skinny" wire.

Used, for example, by Emacs.

Computational biology:

This will use generalizations of LCS, but the algorithms will work with slight modifications.

Outline

Will work our way up to the GNU diff algorithm

- Recursive solution
- Memoized solution
- Dynamic programming
- Memory efficient solution
- Myers/Ukkonon algorithm

Recursive Solution

D(A, empty) = |A| **delete(A)**
D(empty, B) = |B| **insert(B)**
D(A:x, B:x) = D(A, B)
D(A:a, B:b) = min(1 + D(A:a, B), 1 + D(A, B:b))
insert(b) **delete(a)**

Note that this just returns the edit distance, but it is easy to include the edits.

Can work from start or end (here from end).

Why does it work?

What is the running time?

Recursive Solution

Convert to use indices

```
int ED(int i, int j) {
   if (i==0) r = j;
   if (j==0) r = i;
   if (A[i] == B[j])
      r = ED(i-1,j-1);
   else
      r = min(1 + ED(i-1,j), 1 + ED(i,j-1));
   return r;
}

ED(n,m);
```

What is the running time?

Memoized Solution

```
int ED(int i, int j) {
   if (M[i,j] != -1) return M[i,j];
   if (i==0) r = j;
   if (j==0) r = i;
   if (A[i] == B[j])
      r = ED(i-1,j-1);
   else
      r = min(1 + ED(i-1,j), 1 + ED(i,j-1));
   return M[i,j] = r;
}

M[1..n,1..m] = -1;
ED(n,m);
```


Dynamic programming

```
for i = 1 to n
   M[i,1] = i;
for j = 1 to m
   M[1,j] = j;

for i = 1 to n
  for j = 1 to m
    if (A[i] == B[j])
      M[i,j] = M[i-1,j-1];
    else
      M[i,j] = 1 + min(M[i-1,j],M[i,j-1]);
```

Example

		B						
		a	t	c	a	c	a	c
	0	1	2	3	4	5	6	7
	t	1	2	1	2	3	4	5	6
A	c	2	3	2	1	2	3	4	5
	a	3	2	3	2	1	2	3	4
	t	4	3	2	3	2	3	4	5

→ insert
↓ delete
↘ common

Note: can be filled in any order as long as the cells to the left and above are filled.

Can follow path back through matrix to construct edits.

Space Efficient Solution

```
for i = 1 to n
  for j = 1 to m
    if (A[i] == B[j])
      R[j] = Rprev[j-1];
    else
      R[j] = 1 + min(Rprev[j],R[j-1]);
  Rprev[0..m] = R[0..m]
```

Requires only O(m) space.

What is the problem?

Space Efficient Solution

For each entry in a row past n/2 keep track of which column it comes from in row n/2.

		0	1	2	3	4	5	6	7
			a	t	c	a	c	a	c
0		0	1	2	3	4	5	6	7
1	t	1	2	1	2	3	4	5	6
2	c	2	3	2	1	2	3	4	5
3	a	3,0	2,0	3,0	2,3	1,3	2,3	3,3	4,3
4	t	4,0	3,0	2,0	3,0	2,3	3,3	4,3	5,3

The function ED'(A,B) returns column: ☐

Space efficient solution

Now solve recursively:

0	1	2	3	4	5	6	7
1	2	1	2	3	4	5	6
2	3	2	1	2	3	4	5
3,0	2,0	3,0	2,3	1,3	2,3	3,3	4,3
4,0	3,0	2,0	3,0	2,3	3,3	4,3	5,3

k

function RecED(A,B) =
  k = ED'(A,B)
  concatenate RecED(A[1..n/2],B[1..k])
    then RecED(A[n/2..n],B[k..m])

Space Efficient Solution

Time:

T(n,m) = T(n/2,k) + T(n/2,m-k) + O(nm)
= O(nm)

Space:

S(n,m) = O(m) for ED' and O(m + n) for result
= O(n+m)

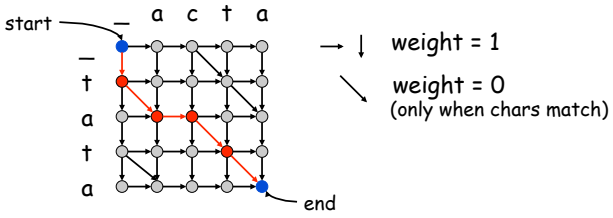
Improving Time Bounds

For many applications (e.g. diff), the difference between strings tends to be small. Can we do something better for this case?

This idea was exploited by Myers and Ukkonen (independenty) in 1985. Now the basis for GNU Diff.

Viewing as a Graph

Edit distance can be expressed as find the shortest path in the following graph:

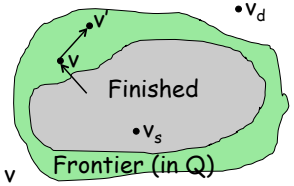


How many vertices will Dijkstra's algorithm visit as a function of n, m and distance d?

Dijkstra's algorithm (review)

Single source (v_s) single destination (v_d) shortest path

$d(v) = \infty$ for all $v \neq v_s$
$d(v_s) = 0$
Q = new empty queue
insert(Q, v_s)
While min(Q) $\neq v_d$
  v = deleteMin(Q)
  for each neighbor v' of v
    d(v') = min(d(v'), d(v) + weight(v,v'))
    insert(Q,v') or decreaseKey(Q,v') if there



Takes |E| inner iterations, each taking O(log |V|) time. Can be improved to O(|E| + |V|log|V|) time.

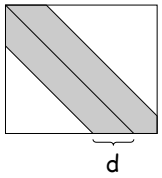
Bounding Visited Vertices

Theorem: edit distance to (i,j) ≥ |j-i|

Proof: every step away from the diagonal is along a horizontal or vertical edge. Each such edge contributes 1, so the total distance, must be at least the distance from the diagonal

Corollary: Dijkstra's algorithm visits at most min(n,m)*2d vertices.

searched vertices



Bounding time

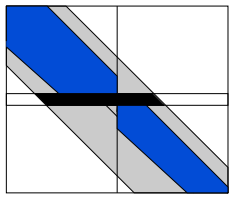
Priority Queue can be kept in constant time per operation (basically a modified breadth-first search), so **total time** is O(min(n,m) d).

In practice this can be much less than O(nm).

What about space?

Optimizing Space

Use recursive trick from before

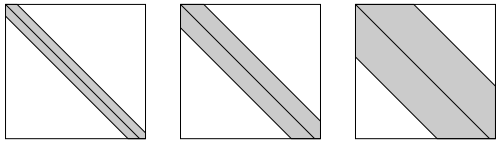


Problem: We do not know d ahead of time. Scanning row by row will not work without d.

Need to bound size of frontier.

Increasing Band-Widths

Start with band of |m-n| and double on each step until large enough



Visits at most twice as many vertices as it should.