CS3000: Algorithms & Data
Jonathan Ullman

Lecture 9:

- Dynamic Programming: Edit Distance, RNA Folding

Oct 5, 2018

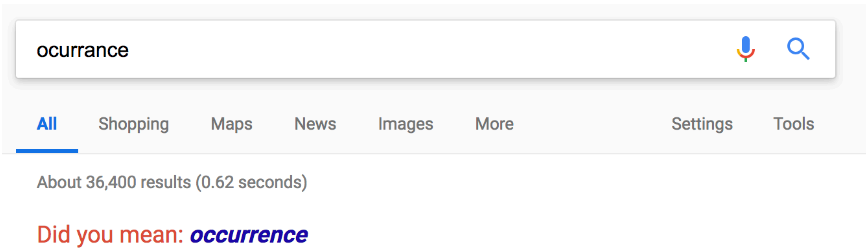
Office Hours Sched (Starting Oct 8)

Mon	Tue	Wed	Thu
4:30-6:30	5:00-7:00	3:00-5:00	12:00-2:00
			5:00-7:00

Edit Distance Alignments

Distance Between Strings

- Autocorrect works by finding similar strings



- `ocurrance` and `occurrence` seem similar, but only if we define similarity carefully

```
ocurrance
occurrence
```

7 mismatches

insert "c"

```
oc urrance
occurrence
```

2 mismatches

Edit Distance / Alignments

- Given two strings $x \in \Sigma^n, y \in \Sigma^m$, the **edit distance** is the number of insertions, deletions, and swaps required to turn x into y.

Edit Dist = Minimum Cost Alignment

- Given an alignment, the cost is the number of positions where the two strings don't agree

x	o	c		u	r	r	a	n	c	e
y	o	c	c	u	r	r	e	n	c	e

cost of the alignment is the # of columns where the two symbols disagree

Ask the Audience

- What is the minimum cost alignment of the strings **smitten** and **sitting**

s	m	i	t	t	e	n	
s	i	t	t	i	n	g	
	1	2		3	4	5	

An optimal alignment

s	m	i	t	t	e	n	-
s	-	i	t	t	i	n	g
	1				2		3

Edit Distance / Alignments

- **Input:** Two strings $x \in \Sigma^n, y \in \Sigma^m$
- **Output:** The minimum cost alignment of x and y
  - **Edit Distance** = cost of the minimum cost alignment

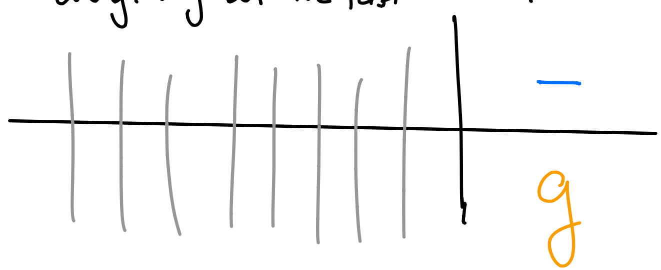
what is the optimal alignment

everything but the last

last column

smitten →

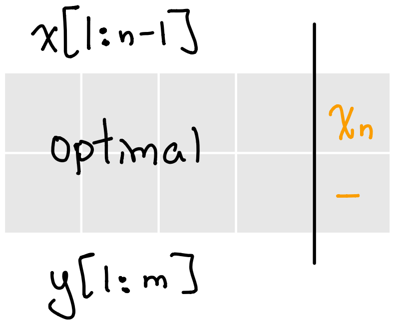
sittin →

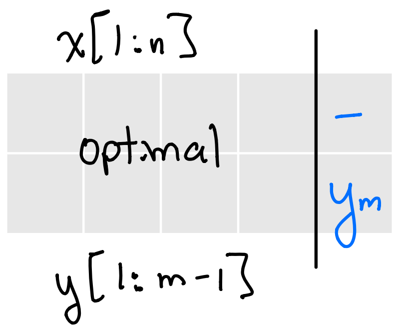


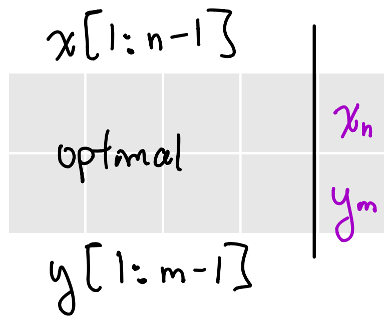
Dynamic Programming

$x[1:n]$ $y[1:m]$

- Consider the **optimal** alignment of x, y
- Three choices for the final column
  - **Case I:** only use x ($x_n, -$)
  - **Case II:** only use y ($-, y_m$)
  - **Case III:** use one symbol from each (x_n, y_m)







Dynamic Programming

- Consider the **optimal** alignment of x, y
- **Case I:** only use x ($x_n, -$)
  - deletion + optimal alignment of $x_{1:n-1}, y_{1:m}$
- **Case II:** only use y ($-, y_m$)
  - insertion + optimal alignment of $x_{1:n}, y_{1:m-1}$
- **Case III:** use one symbol from each (x_n, y_m)
  - If $x_n = y_m$: optimal alignment of $x_{1:n-1}, y_{1:m-1}$
  - If $x_n \neq y_m$: mismatch + opt. alignment of $x_{1:n-1}, y_{1:m-1}$

To decide which case is the best, I need to know the edit distance btw $x[1:i]$, $y[1:j]$

Dynamic Programming

$0 \le i \le n \quad 0 \le j \le m \Rightarrow O(nm)$ problems

- $\mathbf{OPT}(\boldsymbol{i},\boldsymbol{j})$ = cost of opt. alignment of $x_{1:i}$ and $y_{1:j}$
- **Case I:** only use x ($x_i, -$) $1 + \mathrm{OPT}(i-1, j)$
- **Case II:** only use y ($-, y_j$) $1 + \mathrm{OPT}(i, j-1)$
- **Case III:** use one symbol from each (x_i, y_j)

$$= \begin{cases} \mathrm{OPT}(i-1, j-1) + 1 & \text{if } x_i \neq y_i \\ \mathrm{OPT}(i-1, j-1) & \text{if } x_i = y_i \end{cases}$$

$$\mathrm{OPT}(i,j) = \begin{cases} \min\{1 + \mathrm{OPT}(i-1,j),\ 1 + \mathrm{OPT}(i,j-1),\ \mathrm{OPT}(i-1,j-1)\} & x_i = y_i \\ 1 + \min\{\mathrm{OPT}(i-1,j),\ \mathrm{OPT}(i,j-1),\ \mathrm{OPT}(i-1,j-1)\} & x_i \neq y_i \end{cases}$$

Dynamic Programming

- $\mathbf{OPT}(\boldsymbol{i}, \boldsymbol{j})$ = cost of opt. alignment of $x_{1:i}$ and $y_{1:j}$
- **Case I:** only use x ($x_i, -$)
- **Case II:** only use y ($-, y_j$)
- **Case III:** use one symbol from each (x_i, y_j)

which case is "the min" tells you the final column of the alignment

Recurrence:

$$\mathrm{OPT}(i,j) = \begin{cases} 1 + \min\{\mathrm{OPT}(i-1,j), \mathrm{OPT}(i,j-1), \mathrm{OPT}(i-1,j-1)\} \\ \min\{1 + OPT(i-1,j), 1 + \mathrm{OPT}(i,j-1), \mathrm{OPT}(i-1,j-1)\} \end{cases}$$

Base Cases:

$\mathrm{OPT}(i,0) = i, \mathrm{OPT}(0,j) = j$

Example

p	e	–	r	t
b	e	a	s	t

x = pert

y = beast

edit dist. of "beas" and "p"

$b \neq p \Rightarrow 1 + \min\{OPT(0,1), OPT(1,0), OPT(0,0)\}$

	-	b	e	a	s	t
-	0	1	2	3	4	5
p	1	1	2	3	4	5
e	2	2	1	2	3	4
r	3	3	2	2	3	4
t	4	4	3	3	3	$\boxed{3}$

Finding the Alignment

- $\mathbf{OPT}(\boldsymbol{i},\boldsymbol{j})$ = cost of opt. alignment of $x_{1:i}$ and $y_{1:j}$
- **Case I:** only use x ($x_i, -$)
- **Case II:** only use y ($-, y_j$)
- **Case III:** use one symbol from each (x_i, y_j)

Edit Distance ("Bottom-Up")

```
// All inputs are global vars
FindOPT(n,m):
  M[0,j] ← j, M[i,0] ← i

  for (i= 1,…,n):
    for (j = 1,…,m):
      if (x_i = y_j):
        M[i,j] = min{1+M[i-1,j],1+M[i,j-1],M[i-1,j-1]
      elseif (x_i != y_j):
        M[i,j] = 1+min{M[i-1,j],M[i,j-1],M[i-1,j-1]}

  return M[n,m]
```

$[(n+1)(m+1) \text{ entries}] \times [O(1) \text{ operations per entry}] = O(nm)$ time

Space is also $O(nm)$

Ask the Audience

- Suppose **inserting/deleting costs $\delta > 0$** and **swapping $a \leftrightarrow b$ costs $c_{a,b} > 0$**
- Write a recurrence for the min-cost alignment

Edit Distance Summary

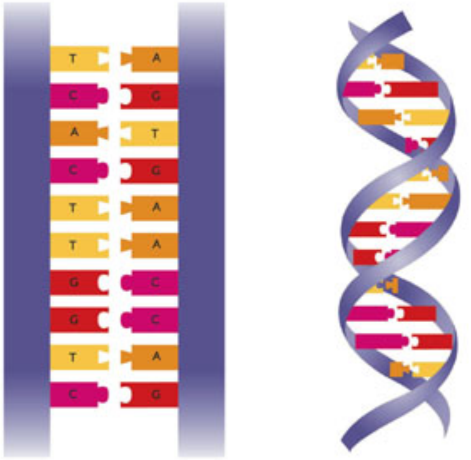
- Compute the **edit distance**, or **min-cost alignment** between two strings in time/space $O(nm)$

- Dynamic Programming:
  - Decide the final pair of symbols in the alignment
- Space can be prohibitive in practice
  - Compute edit distance in space $O(\min\{n, m\})$
  - Can also find alignment in space $O(n + m)$ using a clever divide-and-conquer algorithm!

RNA Folding

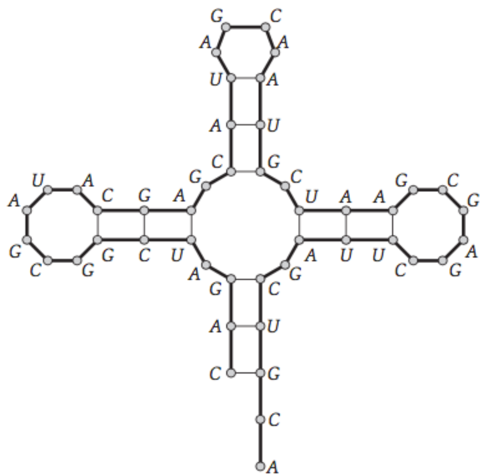
DNA

- DNA is a string of four bases **{A,C,G,T}**
- Two complementary strands of DNA stick together and form a **double helix**
    - **A—T** and **C—G** are complementary pairs



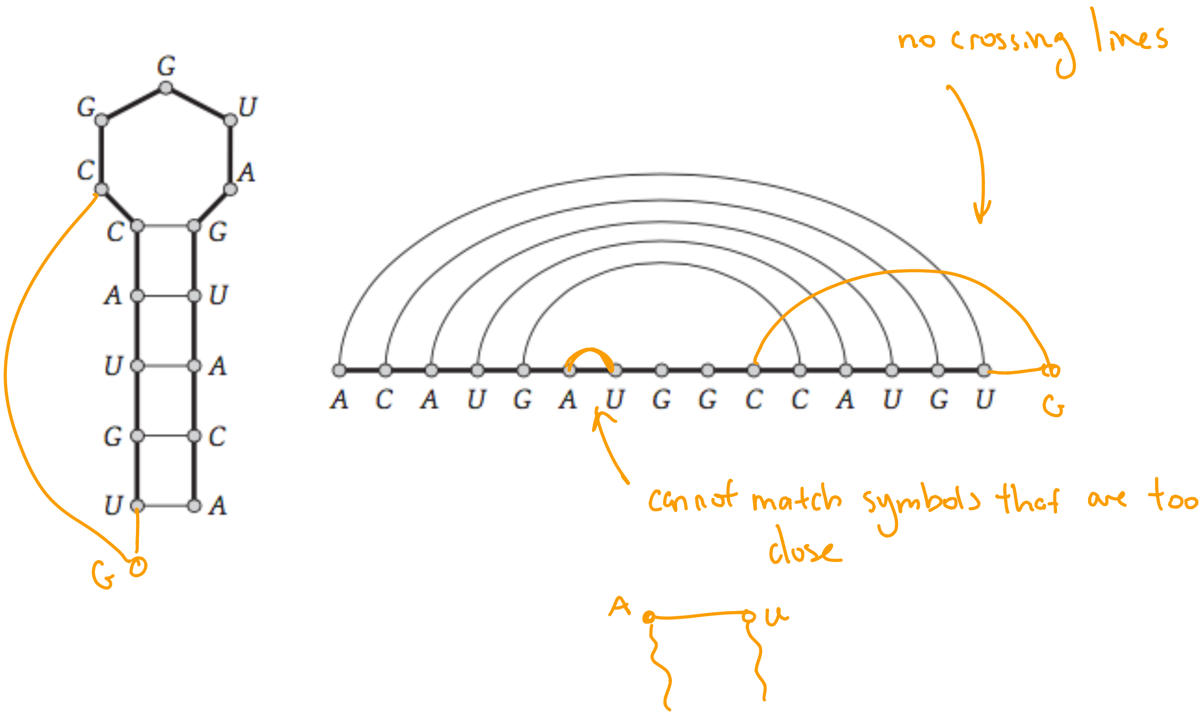
RNA Folding

- RNA is a string of four bases **{A,C,G,U}**
- A single RNA strand sticks to itself and folds into complex structures
  - **A—U** and **C—G** are complementary pairs



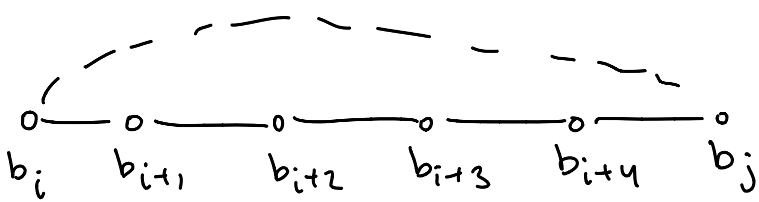
RNA Folding

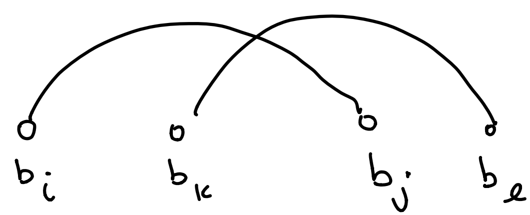
- RNA strand will try to **minimize energy** (form the most bonds) subject to **constraints**



RNA Folding

- RNA is a string of bases $\boldsymbol{b_1, \dots, b_n} \in \{\boldsymbol{A, C, G, U}\}$
- The structure is given by a set of **bonds** S consisting of pairs (i, j) with $i < j$
  - **(Complements)** Only $A - U$ or $C - G$ can be paired
  - **(Matching)** No base b_i is in two pairs in S
  - **(No Sharp Turns)** If $(i, j) \in S$, then $i < j - 4$
  - **(Non-Crossing)** If $(i, j), (k, \ell) \in S$ then it cannot be the case that $i < k < j < \ell$





RNA Folding

- **Input:** RNA sequence $b_1, \dots, b_n \in \{A, C, G, U\}$
- **Output:** A set of pairs $S \subseteq \{1, \dots, n\} \times \{1, \dots, n\}$
  - **Goal:** maximize the size of S
  - **(Complements)** Only $A - U$ or $C - G$ can be paired
  - **(Matching)** No base b_i is in two pairs in S
  - **(No Sharp Turns)** If $(i, j) \in S$, then $i < j - 4$
  - **(Non-Crossing)** If $(i, j), (k, \ell) \in S$ then it cannot be the case that $i < k < j < \ell$

Dynamic Programming

- Let O be the optimal set of pairs for $b_1 \cdots b_n$
- **Case 1:** n pairs with nothing in O

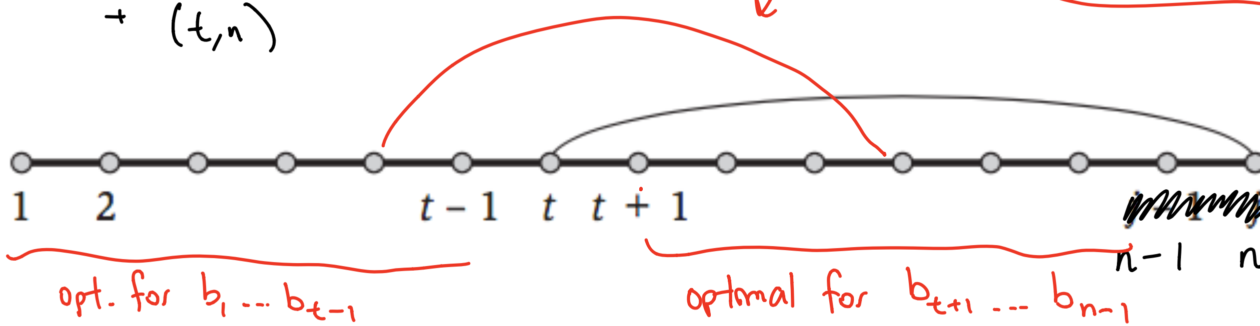
  O is the optimal set of pairs for $b_1 \cdots b_{n-1}$

  no sharp turns

- **Case 2:** n pairs with some $t < n - 4$ in O

  O is opt for $b_1 \cdots b_{t-1}$
  $+$ opt for $b_{t+1} \cdots b_{n-1}$
  $+ (t, n)$

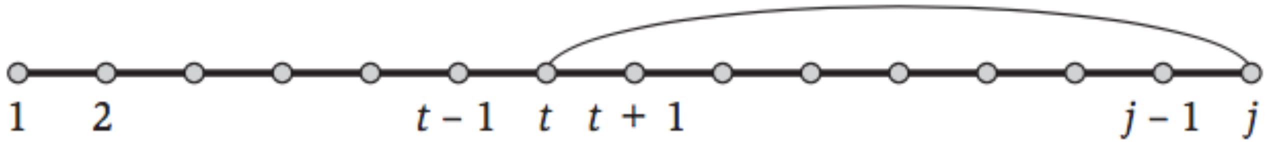
  can't happen by non-crossing



Dynamic Programming

- Let $O_{i,j}$ be the optimal set of pairs for $b_i \cdots b_j$
- **Case 1:** j pairs with nothing in $O_{i,j}$

- **Case 2:** j pairs with some $t < j - 4$ in $O_{i,j}$

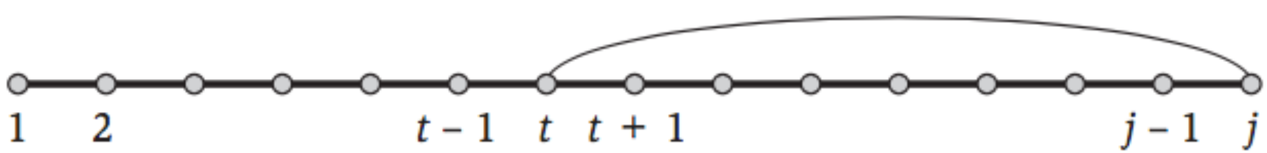


Dynamic Programming

- Let $\mathrm{OPT}(i, j)$ be the opt. **number** of pairs for $b_i \cdots b_j$
- **Case 1:** j pairs with nothing in $O_{i,j}$

  $\mathrm{OPT}(i,j) = \mathrm{OPT}(i, j-1)$

- **Case 2t:** j pairs with $t < j - 4$ in $O_{i,j}$
  - $\mathrm{OPT}(i,j) = 1 + \mathrm{OPT}(t+1, j-1) + \mathrm{OPT}(1, t-1)$
  - Consider all $i \le t < j-4$ s.t. b_t, b_j are complements



Dynamic Programming

- Let $\mathrm{OPT}(i,j)$ be the opt. **number** of pairs for $b_i \cdots b_j$
- **Case 1:** j pairs with nothing in $O_{i,j}$
- **Case 2t:** j pairs with $t < j - 4$ in $O_{i,j}$

Recurrence:

$i+1$

$$\mathrm{OPT}(i,j)$$
$$= \max\{\mathrm{OPT}(i,j-1), \max\{\mathrm{OPT}(i,t-1) + \mathrm{OPT}(t+1,j-1)\}\}$$

Maximum over all t such that
- $i \leq t < j - 4$
- b_t, b_j are compatible bases

Base Cases:

$$\mathrm{OPT}(i,j) = 0 \text{ if } i \geq j - 4$$

Because of no-sharp turns

Filling the Table

Sequence: $ACCGGUAGU$

Recurrence:

$$\mathrm{OPT}(i,j) = \max\left\{\mathrm{OPT}(i,j-1), \max_{\text{possible } t}\{\mathrm{OPT}(i,t-1) + \mathrm{OPT}(t+1,j-1)\}\right\}$$

	6	7	8	j = 9
4	0	0	0	
3	0	0		
2	0			
i = 1				

RNA Folding Summary

- Compute the **optimal RNA folding** in time $O(n^3)$ and space $O(n^2)$

- **Dynamic Programming:**
  - Decide on an optimal pair $b_t - b_n$
  - Remaining RNA is two non-overlapping pieces
  - **Adding variables:** one subproblem for each interval

- **Non-crossing** and **matching** are critical
  - Think about how the dynamic programming algorithm changes if we remove each of the conditions