Objectives

- Dynamic Programming
  - Review Knapsack
  - Sequence Alignment

Review

- What is the knapsack problem?
- What is our solution?

Dynamic Programming: Adding a New Variable

- Def. OPT(i, **w**) = max profit subset of items 1, …, i with weight limit **w**
  - Case 1: OPT does not select item *i*
    - OPT selects best of { 1, 2, …, i-1 } using weight limit ***w***
  - Case 2: OPT selects item *i*
    - new weight limit = $w - w_i$
    - OPT selects best of { 1, 2, …, i–1 } using new weight limit, $\boldsymbol{w - w_i}$

$$OPT(i,w) = \begin{cases} 0 & \text{if } i = 0 \\ OPT(i-1,w) & \text{if } w_i > w \\ \max\{ OPT(i-1,w),\ \ v_i + OPT(i-1,w-w_i)\} & \text{otherwise} \end{cases}$$

Knapsack Problem: Bottom-Up

- Fill up an n-by-W array

```
Input: W, N, w1,…,wN, v1,…,vN

for w = 0 to W
   M[0, w] = 0

for i = 1 to N     # for all items
   for w = 1 to W  # for all possible weights
      if wi > w :  # item's weight is more than available
         M[i, w] = M[i-1, w]
      else
         M[i, w] = max{ M[i-1, w], vi + M[i-1, w-wi] }

return M[N, W]
```

Knapsack Input

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Knapsack Algorithm

i = 4

W + 1

n + 1

	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
{ 1, 2, 3, 4, 5 }	0											

OPT:
Solution =

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Knapsack Algorithm

W + 1 →

i = 5

n + 1 ↓

	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
{ 1, 2, 3, 4, 5 }	0	1	6	7	7	18	22	28	29	34	35	40

Observations?
Questions from last time?

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Knapsack Algorithm

W + 1 →

i = 5

n + 1 ↓

	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
{ 1, 2, 3, 4, 5 }	0	1	6	7	7	18	22	28	29	34	35	40

OPT:
Solution =

What is the optimal solution?

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

Knapsack Algorithm

W + 1 →

n + 1 ↓

	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
{ 1 }	0	1	1	1	1	1	1	1	1	1	1	1
{ 1, 2 }	0	1	6	7	7	7	7	7	7	7	7	7
{ 1, 2, 3 }	0	1	6	7	7	18	19	24	25	25	25	25
{ 1, 2, 3, 4 }	0	1	6	7	7	18	22	24	28	29	29	40
{ 1, 2, 3, 4, 5 }	0	1	6	7	7	18	22	28	29	34	35	40

OPT: 40 = 22 + 18
Solution={4, 3}

W = 11

Item	Value	Weight
1	1	1
2	6	2
3	18	5
4	22	6
5	28	7

SEQUENCE ALIGNMENT

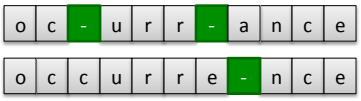
String Similarity

- How similar are two strings?
  - ocurrance
  - occurrence
- Measurements
  - Gap (-): add a letter
  - Mismatch

Which is the best alignment?

6 mismatches, 1 gap

1 mismatch, 1 gap

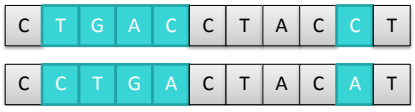


0 mismatches, 3 gaps

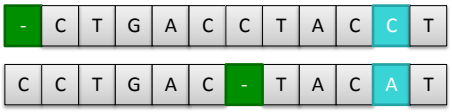
Edit Distance

- [Levenshtein 1966, Needleman-Wunsch 1970]
  - Gap penalty: δ
  - Mismatch penalty: α_{pq}
    - If p and q are the same, then mismatch penalty is 0
  - **Cost** = sum of gap and mismatch penalties

Parameters allow us to tweak cost



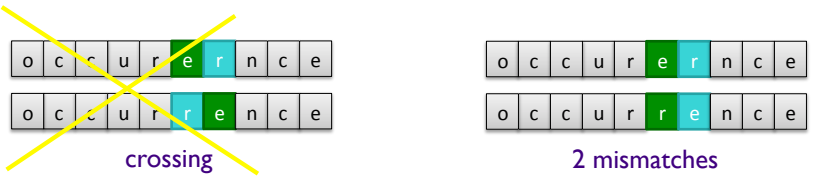
$\alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA}$



$2\delta + \alpha_{CA}$

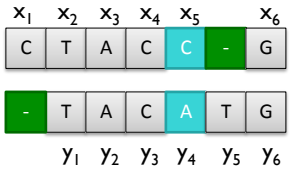
Sequence Alignment

- Goal: Given two strings $X = x_1\ x_2\ \ldots\ x_m$ and $Y = y_1\ y_2\ \ldots\ y_n$ find alignment of minimum cost
- An *alignment* M is a set of ordered pairs x_i-y_j such that each item occurs in at most one pair and **no** crossings
- The pair x_i-y_j and $x_{i'}$-$y_{j'}$ *cross* if i < i', but j > j'.



Sequence Alignment Example

- X = CTACCG
- Y = TACATG
- Solution: M = x_2-y_1 , x_3-y_2, x_4-y_3, x_5-y_4 , x_6-y_6

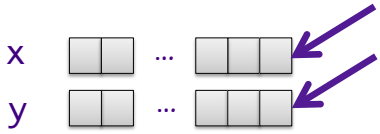


$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i : x_i \text{ unmatched}} \delta + \sum_{j : y_j \text{ unmatched}} \delta}_{\text{gap}}$$

Recall: mismatch penalty is 0 if x_i and y_j are the same

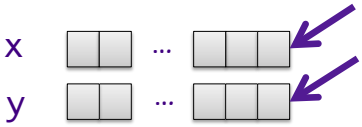
Sequence Alignment Case Analysis

- Consider last character of the strings X and Y: x_M and y_N
  - M and N are not necessarily equal
    - i.e., strings are not necessarily the same length
- What are the possibilities for x_M and y_N in terms of the alignment?



Sequence Alignment Case Analysis

- Consider last character of strings X and Y: x_M and y_N
  - Case 1: x_M and y_N are aligned
  - Case 2: x_M is not matched
  - Case 3: y_N is not matched



Formulate the optimal solution's value

Sequence Alignment Case Analysis

- Consider last character of strings X and Y: x_M and y_N
  - Case 1: x_M and y_N are aligned
  - Case 2: x_M is not matched
  - Case 3: y_N is not matched

What are the costs for these cases?

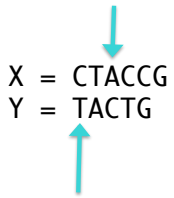
- OPT(i, j) = min cost of aligning strings $x_1\ x_2\ \ldots\ x_i$ and $y_1\ y_2\ \ldots\ y_j$

Sequence Alignment Cost Analysis

- Consider last character of strings X and Y: x_M and y_N
  - Case 1: x_M and y_N are aligned
    - Pay mismatch for x_M-y_N + min cost of aligning rest of strings
    - OPT(M, N) = α_{XmYn} + OPT(M-1, N-1)
  - Case 2: x_M is not matched
    - Pay gap for x_M + min cost of aligning rest of strings
    - OPT(M, N) = δ + OPT(M-1, N)
  - Case 3: y_N is not matched
    - Pay gap for y_N + min cost of aligning rest of strings
    - OPT(M, N) = δ + OPT(M, N-1)

Sequence Alignment Cost Analysis

- Base costs? → i or j is 0
  - What happens when we run out of letters in one string before the other?



Sequence Alignment: Problem Structure

Gaps for remainder of Y

$$OPT(i,j) = \begin{cases} j\delta & \text{if } i = 0 \\ \min \begin{cases} \alpha_{x_i y_j} + OPT(i-1, j-1) \\ \delta + OPT(i-1, j) \\ \delta + OPT(i, j-1) \end{cases} & \text{otherwise} \\ i\delta & \text{if } j = 0 \end{cases}$$

Ran out of 1st string

Ran out of 2nd string

Gaps for remainder of X

Sequence Alignment: Algorithm

Cost parameters

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α)
   for i = 0 to m
      M[i, 0] = iδ
   for j = 0 to n
      M[0, j] = jδ

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min(α[xi, yj] + M[i-1, j-1],
                       δ + M[i-1, j],
                       δ + M[i, j-1])
   return M[m, n]
```


Example

X = bait **Y = boot**

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

j →

i ↓

		b	a	i	t
b					
o					
o					
t					

Example

X = bait **Y = boot**

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

j →

i ↓

		b	a	i	t
	0	2	4	6	8
b	2				
o	4				
o	6				
t	8				

Example

X = bait **Y = boot**

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

j →

i ↓

		b	a	i	t
	0	2	4	6	8
b	2	0	2	4	6
o	4				
o	6				
t	8				

Example

X = bait **Y = boot**

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

j →

i ↓

		b	a	i	t
	0	2	4	6	8
b	2	0	2	4	6
o	4	2	1	3	5
o	6				
t	8				

Example

X = bait **Y = boot**

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

j →

i ↓

		b	a	i	t
	0	2	4	6	8
b	2	0	2	4	6
o	4	2	1	3	5
o	6	4	3	2	4
t	8				

Example

What is the value for the problem?
What is the solution?

X = bait **Y = boot**

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

j →

i ↓

		b	a	i	t
	0	2	4	6	8
b	2	0	2	4	6
o	4	2	1	3	5
o	6	4	3	2	4
t	8	6	5	4	2

Example

X = bait **Y = boot**

α = 1, for vowel mismatch
α = 2, for other mismatches
δ = 2

j →

i ↓

		b	a	i	t
	0	2	4	6	8
b	2	0	2	4	6
o	4	2	1	3	5
o	6	4	3	2	4
t	8	6	5	4	2

Sequence Alignment: Analysis

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α)
   for i = 0 to m
      M[0, i] = iδ
   for j = 0 to n
      M[j, 0] = jδ

   for i = 1 to m                                  O(mn)
      for j = 1 to n
         M[i, j] = min(α[xi, yj] + M[i-1, j-1],
                       δ + M[i-1, j],
                       δ + M[i, j-1])
   return M[m, n]
```

Costs?

Sequence Alignment: Algorithm

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α)
   for i = 0 to m
      M[0, i] = iδ
   for j = 0 to n
      M[j, 0] = jδ

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min(α[xi, yj] + M[i-1, j-1],
                       δ + M[i-1, j],
                       δ + M[i, j-1])
   return M[m, n]
```

What are the space costs?

When computing M[i,j], which entries in M are used?

Sequence Alignment: Analysis

```
Sequence-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α)
   for i = 0 to m
      M[0, i] = iδ
   for j = 0 to n
      M[j, 0] = jδ

   for i = 1 to m
      for j = 1 to n
         M[i, j] = min(α[xi, yj] + M[i-1, j-1],
                       δ + M[i-1, j],
                       δ + M[i, j-1])
   return M[m, n]
```

Space Cost: O(mn)

Observation: to calculate the current value, we only need the row above us and the entry to the left

SEQUENCE ALIGNMENT IN LINEAR SPACE

Sequence Alignment: O(m) Space

- Collapse into an m x 2 array
  - M[i,0] represents previous row; M[i,1] -- current

```
Space-Efficient-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α)
   for i = 0 to m          # initialize first row
      M[i, 0] = iδ
   for j = 1 to n
      M[0, 1] = jδ         # first gap

      for i = 1 to m
            M[i, 1] = min(α[xi, yj] + M[i-1, 0],
                          δ + M[i, 0],
                          δ + M[i-1, 1])
      for i = 1 to m       # copy current row into previous
            M[i, 0] = M[i, 1]
   return M[m, 1]
```

Any drawbacks?

Sequence Alignment: O(m) Space

- Collapse into an m x 2 array
  - M[i,0] represents previous row; M[i,1] -- current

```
Space-Efficient-Alignment(m, n, x1x2...xm, y1y2...yn, δ, α)
   for i = 0 to m          # initialize first row
      M[i, 0] = iδ
   for j = 1 to n
      M[0, 1] = jδ         # first gap

      for i = 1 to m
            M[i, 1] = min(α[xi, yj] + M[i-1, 0],
                          δ + M[i, 0],
                          δ + M[i-1, 1])
      for i = 1 to m       # copy current row into previous
            M[i, 0] = M[i, 1]
   return M[m, 1]
```

Finds optimal value but will not be able to find alignment

Why Do We Care About Space?

- For English words or sentences, probably doesn't matter
- Matters for Biological sequence alignment
  - Consider: 2 strings with 100,000 symbols each
    - Processor can do 10 billion primitive operations
    - BUT dealing with a 10 GB array

Sequence Alignment: Linear Space

- Can we avoid using quadratic space?
  - Optimal value in O(m) space and O(mn) time.
    - Compute OPT(i, •) from OPT(i-1, •)
    - BUT, no simple way to recover alignment itself
- Theorem. [Hirschberg 1975] Optimal alignment in O(m + n) space and O(mn) time.
  - Clever combination of *divide-and-conquer* and *dynamic programming*
  - Section 6.7

Looking Ahead

- PS8