Objectives

- Dynamic Programming
  - Wrapping up Knapsack
  - Sequence Alignment
  - Shortest Path

Review

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

Knapsack Problem: Bottom-Up

- Fill up an n-by-W array

```
Input: W, N, w_1,…,w_N, v_1,…,v_N

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 w_i > w :  # item's weight is more than available
         M[i, w] = M[i-1, w]
      else
         M[i, w] = max{ M[i-1, w], v_i + M[i-1, w-w_i] }

return M[n, W]
```


Knapsack Algorithm

i = 5

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

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

W = 11

OPT:
Solution =

What is the optimal solution?

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

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

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

W = 11

Analyzing Solution

How do we figure out the optimal *solution*?

```
Input: W, N, w_1,…,w_N, v_1,…,v_N

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 w_i > w :  # item's weight is more than available
         M[i, w] = M[i-1, w]
      else
         M[i, w] = max{ M[i-1, w], v_i + M[i-1, w-w_i] }

return M[n, W]
```

Costs?

Analyzing Solution

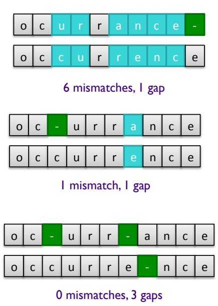
```
Input: W, N, w1,…,wN, v1,…,vN

for w = 0 to W                                        O(W)
   M[0, w] = 0

for i = 1 to N      # for all items
   for w = 1 to W  # for all possible weights        O(N W)
      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 Problem: Running Time

- Running time. $\Theta(n\ W)$
  - **Not** polynomial in input size!
  - "Pseudo-polynomial"
    - Reasonably efficient when W is reasonably small
  - Decision version of Knapsack is NP-complete [Chapter 8]
- Knapsack approximation algorithm. There exists a polynomial algorithm that produces a feasible solution that has value within 0.01% of optimum. [Section 11.8]

String Similarity

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

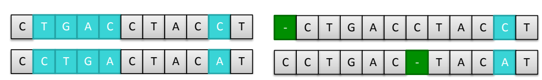


Which is the best alignment?

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

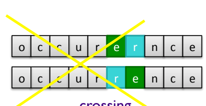
Parameters allow us to tweak cost

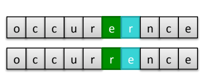


$\alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA}$ $2\delta + \alpha_{CA}$

Sequence Alignment

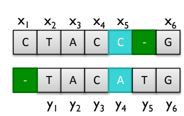
- Goal: Given two strings $X = x_1\ x_2 \ldots x_m$ and $Y = y_1\ y_2 \ldots y_n$ find alignment of minimum cost
- An *alignment* M is a set of ordered pairs x_i-y_j such that each item occurs in at most one pair and **no** crossings
- The pair x_i-y_j and $x_{i'}$-$y_{j'}$ *cross* if i < i', but j >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





Sequence Alignment Example

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

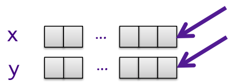


$$\text{cost}(M) = \underbrace{\sum_{(x_i, y_j) \in M} \alpha_{x_i y_j}}_{\text{mismatch}} + \underbrace{\sum_{i\,:\,x_i \text{ unmatched}} \delta + \sum_{j\,:\,y_j \text{ unmatched}} \delta}_{\text{gap}}$$

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

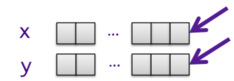
Sequence Alignment Case Analysis

- 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 $y_1 y_2 \ldots y_j$ (arrows: x → i, 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
    - $OPT(M, N) = \alpha_{x_M y_N} + OPT(M-1, N-1)$
  - Case 2: x_M is not matched
    - Pay gap for x_M + min cost of aligning rest of strings
    - $OPT(M, N) = \delta + OPT(M-1, N)$
  - Case 3: y_N is not matched
    - Pay gap for y_N + min cost of aligning rest of strings
    - $OPT(M, N) = \delta + OPT(M, N-1)$

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?

```
X = CTACCG
Y = TACTG
```


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				

Looking Ahead

- PS8 – Due Friday