Week 9 - Friday

COMP 4500

Last time

- What did we talk about last time?
- Subset sum
- Knapsack

Questions?

Assignment 5

Logical warmup

- Consider the following sequence, which should be read from left to right, starting at the top row

```
    1
   1 1
   2 1
  1 2 1 1
1 1 1 2 2 1
```

- What are the next two rows in the sequence?

Back to Knapsack

Knapsack example

- Items ($\boldsymbol{w_i}$, $\boldsymbol{v_i}$):
  - (7, 9)
  - (3, 4)
  - (2, 3)
  - (6, 2)
  - (4, 5)
  - (5, 7)
- Maximum weight: 10
- Create the table to find all of the optimal values that include items 1, 2,…, ***i*** for every possible weight ***w*** up to 10

Fill in the table

i	w_i	v_i	0	1	2	3	4	5	6	7	8	9	10
0	0	0											
1	7	9											
2	3	4											
3	2	3											
4	6	2											
5	4	5											
6	5	7											

Three-sentence Summary of Sequence Alignment

Sequence Alignment

Edit distance between strings

- "long jeverdy" → "longevity"
- What is the distance?
  - `LONG JEV`**`E`**`R`**`D`**`Y`
  - `LONG--EV`**`I`**`-`**`T`**`Y`
- Or what if we want no mismatches?
  - `LONG JEV-ERD-Y`
  - `LONG--EVI---TY`

Edit distance is important

- It can be used in a spell-checker (or auto-correct) to suggest similar words
- There are applications in DNA analysis:
  - How different is this sequence from that sequence?
- We want a general metric for handling both gaps and mismatches

Alignment

- An alignment is a list of matches between characters in strings *X* and *Y* that doesn't cross
- Consider:
  - **`stop-`**
  - **`-tops`**
- This alignment is (2,1), (3,2), (4,3)

Alignment cost

- Some optimal alignment will have the lowest cost
- Cost:
  - Gap penalty $\boldsymbol{\delta} > 0$, for every gap
  - Mismatch cost $\boldsymbol{\alpha_{pq}}$ for aligning $\boldsymbol{p}$ with $\boldsymbol{q}$
    - $\boldsymbol{\alpha_{pp}}$ is presumably 0 but does not have to be
  - Total cost is the sum of the gap penalties and mismatch costs

Designing the algorithm

- We always try to think backwards when doing dynamic programming
- Let strings ***X*** and ***Y*** have length ***m*** and ***n***, respectively
- In the optimal alignment ***M***, either characters ***m*** and ***n*** are matched, or they're not
- In other words, at least one of the following is true:

1. (***m***,***n***) is in ***M***
2. The ***m***th position of ***X*** is not matched
3. The ***n***th position of ***Y*** is not matched

Formulating the recurrence

- Let OPT(***i***, ***j***) be the minimum cost of an alignment of the first ***i*** characters in ***X*** to the first ***j*** characters in ***Y***
- In case 1, we would have to pay a matching cost of matching the character at ***i*** to ***j***
- In cases 2 and 3, you will pay a gap penalty

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

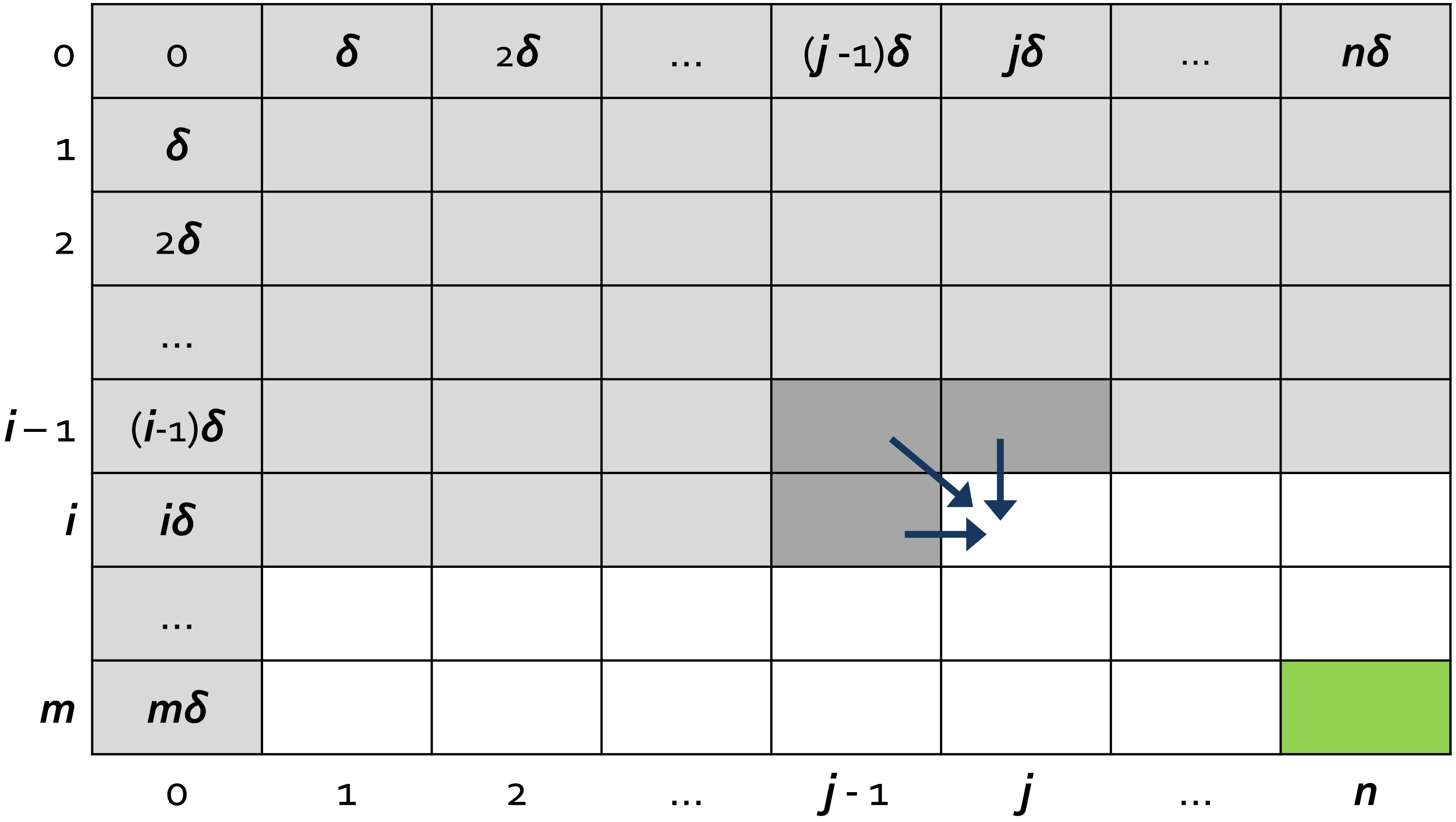
Now what?

- We do our usual thing
- Build up a table of values with $m + 1$ rows and $n + 1$ columns
- In row 0, column j has value $j\delta$ to build up strings from the empty string
- In column 0, row i has value $i\delta$ to build up strings from the empty string
- The other entries (i,j) can be computed from $(i - 1, j – 1)$, $(i – 1, j)$, $(i, j – 1)$

Alignment(*X*, *Y*)

- Create array ***A***[0…***m***][0…***n***]
- For ***i*** from 0 to ***m***
  - Set ***A***[***i***][0]= ***iδ***
- For ***j*** from 0 to ***n***
  - Set ***A***[0][***j***]= ***jδ***
- For ***i*** from 1 to ***m***
  - For ***j*** from 1 to ***n***
    - Set ***A***[***i***][***j***]= min($\alpha_{x_i y_j}$+***A***[***i***-1][***j***-1], ***δ*** + ***A***[***i***-1][***j***], ***δ*** + ***A***[***i***][***j***- 1])
- Return ***A***[***m***][***n***]

Table *A* of OPT values



Reconstructing and run-time

- As before, we can trace back through the table and find the changes, insertions, and deletes
- The running time is O(mn) because the table is O(mn) and we spend constant time on each entry
- Because we only need the previous (and current) row, we can reduce the space to O(n), but then reconstructing the solution becomes tricky
  - The book explains how such an algorithm can be done, but we won't focus on it

Sequence alignment example

- Find the minimum cost to align:
  - "anguished"
  - "language"
- The cost of an insertion (or deletion) δ is 1
- The cost of replacing any letter with a different letter is 1
- The cost of "replacing" any letter with itself is 0

Fill in the table

		a	n	g	u	i	s	h	e	d
	0	1	2	3	4	5	6	7	8	9
l	1									
a	2									
n	3									
g	4									
u	5									
a	6									
g	7									
e	8									

Quiz

Upcoming

Next time…

- Maximum-flow problem
- Minimum cuts

Reminders

- Work on Homework 5
- Read sections 7.1 and 7.2