Chapter 6
Dynamic Programming

Subset Sum
Subset Sum

Setup.
- Have $n$ items with size $s_1$, $s_2$, ..., $s_n$.
- Decision version: is there a subset of size $S$?
- Optimization version: find the largest a subset of size $\leq S$.

Question.
- Can we solve this using an algorithm we already have?

Rocks Problem
Rocks Game

Setup.
- Two piles of rocks: one with $m$ rocks, and one with $n$ rocks.
- You take turns with an opponent. On a turn, you may remove one rock from pile A, one rock from pile B, or one rock from each pile.

Question.
- If you go first, do you have a winning strategy?

Solution.
- 2D version of "binary choice".
- Can be extended to arbitrary dimensions (i.e. $k$ piles of rocks).
- Can be extended to 2D "multiple choice" (i.e. can remove up to $d$ rocks from each pile).
6.6 Sequence Alignment

String Similarity

How similar are two strings?

- occurrence
- occurrence

6 mismatches, 1 gap

1 mismatch, 1 gap

0 mismatches, 3 gaps
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.

**Def.** An alignment \( A \) is a set of ordered pairs \( x_i - y_j \) such that each item occurs in at most one pair and no crossings.

**Def.** The pair \( x_i - y_j \) and \( x_{i'} - y_{j'} \) cross if \( i < i' \), but \( j > j' \).

\[
\text{cost}(A) = \sum_{(x_i, y_j) \in A} \alpha_{x_i y_j} + \sum_{i \text{ or } j \text{ mismatch}} \delta + \sum_{i \text{ or } j \text{ gap}} \delta
\]

**Ex:** CTACCG vs. TACATG,

\begin{align*}
\begin{array}{cccccccc}
\text{CTACCG} & \text{v} & \text{s} & \text{TACATG} \\
\text{CTACCG} & \text{v} & \text{s} & \text{TACATG} \\
\text{CTACCG} & \text{v} & \text{s} & \text{TACATG} \\
\text{CTACCG} & \text{v} & \text{s} & \text{TACATG} \\
\text{CTACCG} & \text{v} & \text{s} & \text{TACATG} \\
\end{array}
\end{align*}

String Distance

**Applications.**
- Basis for Unix diff.
- Speech recognition.
- Computational biology.

**String Distance.**
- Gap penalty \( \delta \); mismatch penalty \( c_{pq} \) for all \( p, q \) in language \( \Delta \).
- Cost = sum of gap and mismatch penalties.

\[
\begin{align*}
\alpha_{TC} + \alpha_{GT} + \alpha_{AG} + 2\alpha_{CA} & \quad & 2\delta + \alpha_{CA}
\end{align*}
\]
String Distance

String Distance.
- Gap penalty $\delta$, mismatch penalty $\alpha_{pq}$.
- Cost = sum of gap and mismatch penalties.

Specific Applications.
- Often $\alpha_{yx} = \alpha_{xy}$.
- Often $\alpha_{xx} = 0$, e.g. in DNA sequence alignment.
- For words in English $\alpha_{AE} < \alpha_{AX}$.
- Could also have $\delta_X$ or $\delta_{top}, \delta_{bot}$.
- Image similarity might have $\delta = \infty$.

Sequence Alignment: Problem Structure

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

- Case 1: $OPT$ matches $x_i, y_j$.
  - pay mismatch (if any) for $x_i, y_j$ + $OPT(i-1, j-1)$

- Case 2a: $OPT$ leaves $x_i$ unmatched.
  - pay gap for $x_i$ and $OPT(i-1, j)$

- Case 2b: $OPT$ leaves $y_j$ unmatched.
  - pay gap for $y_j$ and $OPT(i, j-1)$

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

Sequence-Alignment($x_1, x_2, \ldots, x_m, y_1, y_2, \ldots, y_n, \delta, \alpha$) {

for $i = 0$ to $m$
  $M[0, i] = i\delta$

for $j = 0$ to $n$
  $M[j, 0] = j\delta$

for $i = 1$ to $m$
  for $j = 1$ to $n$
    $M[i, j] = \min(\alpha[x_i, y_j] + M[i-1, j-1],$
    $\delta + M[i-1, j],$
    $\delta + M[i, j-1])$

return $M[m, n]$
}

In what order do we fill in this table? Is this our only option?

How do we find the best alignment? (i.e. not just its cost)

Sequence Alignment: Algorithm

for $i = 1$ to $m$
  for $j = 1$ to $n$
    $M[i, j] = \alpha[x_i, y_j] + M[i-1, j-1]$
    Trace[$i, j$] = (i-1, j-1)

if $(\delta + M[i-1, j] < M[i, j])$
  $M[i, j] = \delta + M[i-1, j]$
  Trace[$i, j$] = (i-1, j)

if $(\delta + M[i, j-1] < M[i, j])$
  $M[i, j] = \delta + M[i, j-1]$
  Trace[$i, j$] = (i, j-1)

Post-process to print out the Trace.
Sequence Alignment: Algorithm

```
Sequence-Alignment(x_1, x_2, ..., x_m, y_1, y_2, ..., y_n, 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[x_i, y_j] + M[i-1, j-1],
                           δ + M[i-1, j],
                           δ + M[i, j-1])
    return M[m, n]
}
```

**Analysis.** $Θ(mn)$ time and space.

**English words or sentences:** $m, n ≤ 10$.

**Computational biology:** $m = n = 100,000$. 10 billions ops OK, but 10GB array?

---

**Sequence Alignment: In-Class Exercise**

Use the sequence alignment algorithm to solve the following problems:

**Hamming Distance.** Given two strings of equal length, find the number of positions on which they differ (e.g. roses and tone have hamming distance 3).

**Edit Distance.** Given two strings $X = x_1, x_2, ..., x_m$ and $Y = y_1, y_2, ..., y_n$, what is the cost of "editing" $X$ into $Y$ where the goal is to minimize the number of insert and deletes (e.g. hop and help have edit distance 3).

**Longest Increasing Subsequence.** Consider a permutation of $\{1, 2, ..., n\}$. Find the longest increasing subsequence in the permutation (e.g. 3 4 1 5 2 has a longest increasing subsequence of length 3).
Local Sequence Alignment
Local String Distance

Primary Application.
- Conserved Sequences in DNA

Local String Distance.
- Gap penalty $\delta$; mismatch penalty $\alpha_{pq}$
- Now trying to minimize the alignment score over all contiguous substrings $(x_1 x_2 \ldots x_i$ and $y_1 y_2 \ldots y_j)$.

- Assume $\alpha_{XX} = -1$ and $\alpha_{XY}, \delta > 0$.

$$\begin{array}{c}
\text{G} & \text{T} & \text{A} & \text{A} & \text{T} \\
\text{C} & \text{T} & \text{A} & \text{C} & \text{T}
\end{array}$$

$$\begin{array}{c}
\text{C} & \text{T} & \text{A} & \text{A} & \text{T} & \text{C} & \text{G} & \text{G}
\end{array}$$

Original cost: $5\delta + \delta + \alpha_{CA} + 4\delta$
New cost: $\delta + \alpha_{CA}$

Local Sequence Alignment: Problem Structure

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

- Case 1: $OPT$ matches $x_i$-$y_j$.
  - pay $\alpha_{xy}$ for matching $x_i$-$y_j$ and $OPT(i-1, j-1)$
- Case 2a: $OPT$ leaves $x_i$ unmatched.
  - pay $\delta$ for $x_i$ and $OPT(i-1, j)$
- Case 2b: $OPT$ leaves $y_j$ unmatched.
  - pay $\delta$ or $y_j$ and $OPT(i, j-1)$
- Case 3: End the substring here.
  - pay 0

$$OPT(i, j) = \begin{cases} 
0 & \text{if } i = 0 \\
\min \left\{ \begin{array}{ll}
\alpha_{xy} + OPT(i-1, j-1) & \text{if } i > 0, j > 0 \\
\delta + OPT(i-1, j) & \text{if } i > 0 \\
\delta + OPT(i, j-1) & \text{otherwise} \\
0 & \text{if } j = 0 
\end{array} \right. 
\end{cases}$$
Common Mistakes in DP

The buck stops here
- Longest path problem (from HW 3)
- Gas station problem (from HW 3)

Principle of optimality
- Traveling Salesman Problem
Asymptotic Order of Functions

1. Know log rules
2. Compare logs vs polynomial
3. Compare polynomials vs exponentials
4. Know exponential rules

In-Class Exercise

Answer T/F for the following questions:

1. $2^{n^1} \in O(2^n)$
2. $2^{2n} \in O(2^n)$
3. $\log_2(8n^{10}) \in O(\log_{10} n)$
4. $2^{2\log_2 n} \in O(n^2)$
5. $n^{1/5} + \log_2 n \in O(\log_2 n)$
Chapter 5
Divide and Conquer

DOOOOOOM!!!!! (aka the midterm)
**EXACTLY what is on the midterm**

1) *Order a list of functions by asymptotic runtime.*
- know how to simplify logs and exponentials

2) Stable marriage question
- know the definition of a stable matching.
- know the GS algorithm.

3) Given pseudocode, find runtime upper and lower bound.

4) *Write a simple dynamic program.*
- pseudocode only, no proofs

5) *Given a problem, find greedy orders and counterexamples.*

6) *Given an existing algorithm, use it to solve a new problem.*

7) True / False questions (7).
- includes Matroids, Graphs, etc

* look at in-class examples