Overview

logistics:
- pset2 out, due W10 — can submit in groups of ≤ 3

last time:
- shortest paths
  - with negative lengths
  - all-pairs

today:
- dynamic programming *optimized*
  - edit distance
  - longest increasing subsequence
dynamic programming:
- develop recursive algorithm
- understand structure of subproblems
  - names of subproblems
  - number of subproblems
  - dependency graph amongst subproblems
- memoize (implicitly, or explicitly)
- analysis (time, space)
- further optimization

remarks:
- memoizing a recursive algorithm does not necessarily lead to an efficient algorithm (e.g., knapsack problem) — you need the right recursion
- recognizing that dynamic programming applies to a problem can be non-obvious
Definition

Let $x, y \in \Sigma^*$ be two strings over the alphabet $\Sigma$. The **edit distance** between $x$ and $y$ is the minimum number of insertions, deletions and substitutions required to transform $x$ into $y$.

Example

\[
\text{money} \rightarrow \text{boney} \rightarrow \text{bone} \rightarrow \text{bona} \rightarrow \text{bo_a} \rightarrow \text{boba} \implies \text{edit distance} \leq 5
\]

**remarks:**

- edit distance $\leq 4$
- intermediate strings can be arbitrary in $\Sigma^*$
**Definition**

Let $x, y \in \Sigma^*$ be two strings over the alphabet $\Sigma$. An **alignment** is a sequence $M$ of pairs of indices $(i,j)$ such that:

- an index could be empty, such as $(,4)$ or $(5,)$
- each index appears exactly once per coordinate
- no crossings: for $(i,j), (i',j') \in M$ either $i < i'$ and $j < j'$, or $i > i'$ and $j > j'$

The **cost** of an alignment is the number of pairs $(i,j)$ where $x_i \neq y_j$.

**Example**

```
money
bo ba
```

$M = \{(1,1), (2,2), (3,), (,3), (4,4), (5,)\}$, cost 5
question: given two strings $x, y \in \Sigma^*$, compute their edit distance

Lemma

*The edit distance between two strings $x, y \in \Sigma^*$ is the minimum cost of an alignment.*

Proof.

Exercise.

question: given two strings $x, y \in \Sigma^*$, compute the minimum cost of an alignment

remarks:

- can also ask to compute the alignment itself
- widely solved in practice, e.g., the BLAST heuristic for DNA edit distance
Lemma

Let $x, y \in \Sigma^*$ be strings, and $a, b \in \Sigma$ be symbols. Then

$$\text{dist}(x \circ a, y \circ b) = \min \begin{cases} 
\text{dist}(x, y) + 1[a \neq b] \\
\text{dist}(x, y \circ b) + 1 \\
\text{dist}(x \circ a, y) + 1 
\end{cases}.$$ 

Proof.

In an optimal alignment from $x \circ a$ to $y \circ b$, either:

- $a$ aligns to $b$, with cost $1[a \neq b]$
- $a$ is deleted, with cost $1$
- $b$ is deleted, with cost $1$
Edit Distance (V)

**iterative algorithm:**

\[
\text{dist}(x_1x_2\cdots x_n, y_1y_2\cdots y_m)
\]

for \(0 \leq i \leq n\)

\[
d[i][0] = i
\]

for \(0 \leq j \leq m\)

\[
d[0][j] = j
\]

for \(0 \leq i \leq n\)

for \(0 \leq j \leq m\)

\[
\begin{cases}
\quad d[i][j] = \min \left\{ 
\quad d[i-1][j-1] + 1[x_i \neq y_j] \\
\quad d[i-1][j] + 1 \\
\quad d[i][j-1] + 1
\end{cases}
\]

return \(d[n][m]\)

**correctness:** clear

**complexity:**

- \(O(nm)\) time
- space
  - clearly \(O(nm)\)
  - better: only store \(d[\text{cur}][\cdot]\) and \(d[\text{prev}][\cdot] \Rightarrow O(m)\)

**question:** are we done?
Corollary

Given two strings $x, y \in \Sigma^*$ can compute the minimum cost alignment in $O(nm)$-time and $O(nm)$-space.

Proof.

**Exercise.** *Hint:* follow how each subproblem was solved.
dependency graph:

computing the alignment:

- *how* update rule is computed yields a pointer for each \((i, j)\)
- one pointer per optimal choice — multiple pointers are possible
- *any* path from \((n, m)\) to boundary yields optimal alignment
- compute path via graph search

saving space:

- only keep most recent two columns

⇒ we lost the pointers!

**question:** compute the alignment in \(O(n + m)\) space?
Lemma

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$,

$$\text{dist}(x, y) = \min_{1 \leq j \leq m} \{\text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{>i}, y_{>j})\}.$$ 

Proof.

$\leq$: Fix $j$. Let $A_{\leq}$ and $A_{>}$ be alignments respectively between $x_{\leq i}, y_{\leq j}$ and $x_{>i}, y_{>j}$, with respective costs $\text{dist}(x_{\leq i}, y_{\leq j})$ and $\text{dist}(x_{>i}, y_{>j})$. Then $A_{\leq} \circ A_{>}$ is an alignment between $x$ and $y$ of cost $\text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{>i}, y_{>j})$.

$\geq$: Any alignment $A$ between $x$ and $y$ will align $x_{\leq i}$ to some prefix $y_{\leq j}$ of $y$ in an alignment $A_{\leq}$, and align $x_{>i}$ to the suffix $y_{>j}$ in an alignment $A_{>}$, and hence for this $j$ we have $\text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{>i}, y_{>j})$. $\square$
Definition

Let $x, y \in \Sigma^*$ be strings, with $n = |x|$ and $m = |y|$. Then for any $1 \leq i \leq n$, define $\text{meet}_i(x, y)$ to be the $j \in [m]$ where $x_{\leq i}$ aligns to $y_{\leq j}$ in an optimal alignment. That is,

$$\text{meet}_i(x, y) = \min \{j : \text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})\}.$$ 

**remark:** previous lemma asserts such a $j$ exists

**complexity:**

- $\text{dist}(x_{\leq i}, y)$ already computes $\text{dist}(x_{\leq i}, y_{\leq j})$ for all $j$
  - $O(nm)$ time, $O(m)$ space
- $\text{dist}(\text{reverse}(x_{> i}), \text{reverse}(y))$ already computes $\text{dist}(x_{> i}, y_{> j})$ for all $j$
  - $\implies O(nm)$ time, $O(m)$ space

**meet**

$$(i, x_1 x_2 \cdots x_n, y_1 y_2 \cdots y_m)$$

for $1 \leq j \leq m$

compute $\text{dist}(x_{\leq i}, y_{\leq j})$

for $1 \leq j \leq m$

compute $\text{dist}(x_{> i}, y_{> j})$

output $\min j$ st $\text{dist}(x, y) = \text{dist}(x_{\leq i}, y_{\leq j}) + \text{dist}(x_{> i}, y_{> j})$

**correctness:** clear
divide and conqueror:

\[
\text{dist-align}(x_1x_2 \cdots x_n, y_1y_2 \cdots y_m)
\]

if \( n = 1 \)

\[ \text{use dist}(x, y) \]

if \( m = 1 \)

\[ \text{use dist}(x, y) \]

\( j = \text{meet}(n - 1, x, y) \)

\( A_\leq = \text{dist-align}(x_{\leq n-1}, y_{\leq j}) \)
\( A_\geq = \text{dist-align}(x_{> n-1}, y_{> j}) \)

return \( A_\leq \circ A_\geq \)

**complexity:**

- **base cases**
  - \( O(m) \) time, \( O(1) \) space
  - \( O(n) \) time, \( O(1) \) space

- **meet\(_{n-1}\)(x, y)**
  - \( O(nm) \) time, \( O(n + m) \) space

- **space recurrence**
  - \( S(n, m) \leq \max\{O(n + m), S(n - 1, m), S(1, m)\} \)
  \[ \implies S(n, m) \leq O(n + m) \]

- **time recurrence**
  - \( T(n, m) \leq O(nm) + T(n - 1, m) + T(1, m) \)
  \[ \implies T(n, m) \leq O(n^2 m) \]

**question:** can we do better?
**divide and conquer: dist-align'**

\[
\begin{align*}
\text{if } n &= 1 \\
& \quad \text{use } \text{dist}(x, y)
\end{align*}
\]

\[
\begin{align*}
\text{if } m &= 1 \\
& \quad \text{use } \text{dist}(x, y)
\end{align*}
\]

\[
j = \text{meet}(\lfloor \frac{n}{2} \rfloor, x, y)
\]

\[
A_\leq = \text{dist-align}'(x_{\leq \lfloor \frac{n}{2} \rfloor}, y_{\leq j})
\]

\[
A_\geq = \text{dist-align}'(x_{> \lfloor \frac{n}{2} \rfloor}, y_{> j})
\]

return \(A_\leq \circ A_\geq\)

---

**complexity:**

- **base cases:** \(O(n + m)\) time, \(O(1)\) space
- **meet_{\lfloor \frac{n}{2} \rfloor}(x, y):** \(O(nm)\) time, \(O(n + m)\) space
- **space recurrence**
  - \(S(n, m) \leq \max\{O(n + m), S(\lfloor \frac{n}{2} \rfloor, m), S(n - \lfloor \frac{n}{2} \rfloor, m)\}\)
  \[\implies S(n, m) \leq O(n + m)\]
- **time recurrence**
  - \(T(n,m) \leq O(nm) + T(\lfloor \frac{n}{2} \rfloor,j) + T(n - \lfloor \frac{n}{2} \rfloor, m - j)\)
  - guess \(T(n, m) \leq \alpha \cdot nm\)
  - \(T(n,m) \leq \beta \cdot nm + \alpha \cdot \frac{n}{2} \cdot j + \alpha \cdot \frac{n}{2} \cdot (m-j) = (\beta + \frac{\alpha}{2})nm\)
  \[\implies \text{valid as long as } \alpha \geq 2\beta\]
  \[\implies T(n, m) \leq O(nm)\]

\[\implies\] computing actual alignment in \(O(nm)\)-time and \(O(n + m)\)-space.
Longest Increasing Subsequence

Definition

A sequence of integers, of length $n$, is an ordered list $a_1, a_2, \ldots, a_n \in \mathbb{Z}$. The sequence is increasing if $a_1 < a_2 < \cdots < a_n$.

A subsequence of $a_1, a_2, \ldots, a_n$ is any sequence of the form $a_{i_1}, a_{i_2}, \ldots, a_{i_m}$, where $1 \leq i_1 < \cdots < i_m \leq n$. The subsequence is increasing (IS) if $a_{i_1} < \cdots < a_{i_m}$.

Example

- 02139947200854008540943059472061801 — sequence
- 02139947200854008540943059472061801 — subsequence
- 02139947200854008540943059472061801 — increasing subsequence
- 02139947200854008540943059472061801 — longer increasing subsequence
The **longest increasing subsequence problem (LIS)** is to, given a sequence \(a_1, a_2, \ldots, a_n \in \mathbb{Z}\), compute the (length of) the longest increasing subsequence.

**Goal:** solve with dynamic programming
- identify subproblems
- develop recursion
- memoize
- analyze
- optimize time

**Remark:** without loss of generality the \(a_i\) are distinct, up to a cost of \(\Theta(n \log n)\) in runtime (**exercise**).
Lemma

For a sequence $\bar{a} = a_1, a_2, \ldots, a_n$, define $\text{LIS}(\bar{a})$ to be the length of the longest increasing subsequence. Define $\text{LIS}^*(\bar{a})$ to be the length of the longest increasing subsequence that contains the last element $a_n$. Then

1. $\text{LIS}(a_1, a_2, \ldots, a_n) = \max_{1 \leq i \leq n} \text{LIS}^*(a_1, a_2, \ldots, a_i)$.
2. $\text{LIS}^*(a_1, a_2, \ldots, a_n) = \max_{i : a_i < a_n} \{1 + \text{LIS}^*(a_1, a_2, \ldots, a_i), 1\}$.

Proof.

1. Clear.
2. For $i$ with $a_i < a_n$, an IS* $a_{i_1} < \cdots < a_{i_{m-1}} < a_{i_m = i}$ of $\bar{a}_{\leq i}$ can append $a_n$ to yield an IS* $a_{i_1} < \cdots < a_{i_{m-1}} < a_i < a_n$ of $\bar{a}$, and every IS* of $\bar{a}$ can be decomposed this way, or by taking the singleton sequence $a_n$. Now take maximums. \qed
Lemma

Define \( \text{LIS}^*(\bar{a}) \) to be the length of the longest increasing subsequence that contains the last element \( a_n \). Then \( \text{LIS}^*(a_1, a_2, \ldots, a_n) = \max_{i:a_i<a_n} \{ 1 + \text{LIS}^*(\bar{a}_{\leq i}), 1 \} \).

Example

\[
\begin{align*}
02139947200854008540943059472061801 & \quad \text{LIS}^*(a_1) = 1 \\
02139947200854008540943059472061801 & \quad \text{LIS}^*(a_1, a_2) = 2 \\
02139947200854008540943059472061801 & \quad \text{LIS}^*(a_1, \ldots, a_3) = 2 \\
02139947200854008540943059472061801 & \quad \text{LIS}^*(a_1, \ldots, a_4) = 3 \\
02139947200854008540943059472061801 & \quad \text{LIS}^*(a_1, \ldots, a_5) = 4 \\
02139947200854008540943059472061801 & \quad \text{LIS}^*(a_1, \ldots, a_6) = 4
\end{align*}
\]
**iterative algorithm:**

\[ \text{LIS}(a_1, a_2, \ldots, a_n): \]

\[
\begin{align*}
\text{for } 1 \leq i \leq n & \quad L^*[i] = 1 \\
L & = 0 \\
\text{for } 1 \leq i \leq n & \quad \text{for } 1 \leq j < i \\
& \quad \text{if } a_j < a_i \\
& \quad \quad L^*[i] = \max\{L^*[i], 1 + L^*[j]\}
L & = \max\{L, L^*[i]\}
\end{align*}
\]

return \( L \)

**correctness:** clear

**complexity:**

- \( O(n) \) space
- \( O(n^2) \) time — do better?
Longest Increasing Subsequence, Faster

\[ \text{LIS}^*(a_1, a_2, \ldots, a_i) = \max_{i: a_j < a_i} \{1 + \text{LIS}^*(a_1, a_2, \ldots, a_j), 1\}. \]

This recursive step does too much — all \((a_j, a_i)\) are compared! Use sorting?

**idea:** define subproblem based on *length* of increasing subsequences

**Definition**

For sequence \(a_1, a_2, \ldots, a_n\), define the **end of increasing subsequence** \(\text{EIS}(\ell, \overline{a})\) to be the minimum \(a_i\) such that there is an increasing sequence of length \(\ell\) that terminates at \(a_i\), that is,

\[ \text{EIS}(\ell, \overline{a}) := \min_{i: a_1 < a_2 < \cdots < a_\ell = i} a_i. \]

\(\text{EIS}(\ell, \overline{a}) = \infty\) if \(\ell > \text{LIS}(\overline{a})\).

**intuition:** prefer the ‘smallest’ IS of each size
Definition

For sequence $a_1, a_2, \ldots, a_n$, define $\text{EIS}(\ell, \overline{a})$ to be the minimum $a_i$ such that there is an increasing sequence of length $\ell$ that terminates at $a_i$. $\text{EIS}(\ell, \overline{a}) = \infty$ if $\ell > \text{LIS}(\overline{a})$.

Lemma

$\text{LIS}(\overline{a}) = \max_{\ell: \text{EIS}(\ell, \overline{a}) < \infty} \ell$.

Proof.

Clear. □
**Definition**

For sequence $a_1, a_2, \ldots, a_n$, define $EIS(\ell, \bar{a})$ to be the minimum $a_i$ such that there is an increasing sequence of length $\ell$ that terminates at $a_i$. $EIS(\ell, \bar{a}) = \infty$ if $\ell > \text{LIS}(\bar{a})$.

**Lemma**

For sequence $a_1, a_2, \ldots, a_n$, $EIS(\ell, \bar{a}) < EIS(\ell + 1, \bar{a})$, for all $\ell$. That is, $EIS(\cdot, \bar{a})$ is a strictly sorted sequence.

**Proof.**

Let $a_{i_1} < a_{i_2} < \cdots < a_{i_\ell}$ be a witness for $EIS(\ell, \bar{a}) = a_{i_\ell}$, and let $a_{i'_1} < a_{i'_2} < \cdots < a_{i'_\ell}$ be a witness for $EIS(\ell + 1, \bar{a}) = a_{i'_{\ell+1}}$. Then as $a_{i'_1} < a_{i'_2} < \cdots < a_{i'_{\ell+1}}$ is length-$\ell$ increasing sequence we have that $EIS(\ell, \bar{a}) \leq a_{i'_{\ell}} < a_{i'_{\ell+1}} = EIS(\ell + 1, \bar{a})$. 

[Proof box]


Lemma

\[ \text{EIS}(\ell, (a_1, \ldots, a_n, a_{n+1})) = \]

1. \( \text{EIS}(\ell, a) \), if \( \text{EIS}(\ell, a) < a_{n+1} \)

2. \( \text{EIS}(\ell, a) \), if \( \text{EIS}(\ell - 1, a) > a_{n+1} \)

3. \( a_{n+1} \), if \( \text{EIS}(\ell, a) > a_{n+1} \) and \( \text{EIS}(\ell - 1, a) < a_{n+1} \)

Proof.

1. Clear.

2. Clear.

3. Exists increasing sequence of length \( \ell \) terminating at \( a_{n+1} \)
   
   iff exists increasing sequence of length \( \ell - 1 \) terminating at \( a_i < a_{n+1} \), for some \( i \)
   
   iff exists increasing sequence of length \( \ell - 1 \) terminating at \( \text{EIS}(\ell - 1, a) < a_{n+1} \) \( \square \)
Lemma

For a fixed $\bar{a}$, $\text{EIS}(\ell, \bar{a})$ strictly increases with $\ell$.

Lemma

$\text{EIS}(\ell, (a_1, \ldots, a_n, a_{n+1})) =$

1. $\text{EIS}(\ell, \bar{a})$, if $\text{EIS}(\ell, \bar{a}) < a_{n+1}$ or $\text{EIS}(\ell - 1, \bar{a}) > a_{n+1}$
2. $a_{n+1}$, if $\text{EIS}(\ell, \bar{a}) > a_{n+1}$ and $\text{EIS}(\ell - 1, \bar{a}) < a_{n+1}$

Corollary

- $\text{EIS}(\ell, (\bar{a}, a_{n+1})) \neq \text{EIS}(\ell, \bar{a})$ for exactly one value of $\ell$
- This value of $\ell$ can be found by binary search.

remarks:

- uses distinctness of the $a_i$
- boundary cases need attention, e.g., $\text{EIS}(\ell, \bar{a}) = \infty$, or $\ell - 1 = 0$
**Longest Increasing Subsequence, Faster (VI)**

**LIS’**\((a_1, a_2, \ldots, a_n):\)

\[
\text{for } 1 \leq \ell \leq n \\
E[\ell] = \infty \\
\text{for } 1 \leq i \leq n \\
\ell = \min\{k : E[k] > a_i\} \\
E[\ell] = a_i \\
\text{for } 1 \leq i \leq n \\
\text{if } E[i] < \infty \\
L = i \\
\text{return } L
\]

**correctness:** clear

**complexity:**
- \(O(n)\) space
- time
  - \(E[\cdot]\) remains sorted throughout
  \(\implies O(\log n)\) time to compute \(\min\{k : E[k] > a_i\}\)
  \(\implies O(n \log n)\) total runtime

**remarks:**
- making \(a_i\) distinct costs \(\Theta(n \log n)\) extra time
- can compute actual subsequence in same time bound, using back pointers (exercise)
Overview (II)

logistics:
- pset2 out, due W10 — can submit in groups of $\leq 3$

today:
- dynamic programming \textit{optimized}
  - edit distance
  - longest increasing subsequence

next time:
- randomized algorithms