From Application to Algorithm Design

• Problem: Find minefields from airborne IR data
  – E.g., anti-tank landmines laid out in patterns (why?)
  – IR imaging shows thermal contrast vs. background, but lots of clutter

• Find all maximal regularly-spaced, collinear subsets of given pointset

• Lower bound? \( \Omega(n^2) \)

• Naïve algorithm?

• Simpler variants?
Number Placement

- Given a list of n distinct integers and a sequence of n boxes with preset inequality signs inserted between them, design an algorithm that places the numbers into the boxes to satisfy those inequalities.

- For example, given four boxes and three inequality signs between them: __ < __ > __ < __, the numbers 2, 5, 1, 0 can be placed as 0 < 5 > 1 < 2.

- Algorithm?
DP, Flow, LP Consolidation...
Linear Programming (LP)

- Tool for optimal allocation of scarce resources
  - Optimizations subject to “compatibility constraints”

- Powerful and general problem-solving method
  - Shortest paths, maximum flows, min-cost flows, MST, matching, 2-person games, …

- Significance and Practice
  - Among most important scientific advances of 20th century
  - Dominates industrial practice
    - Delta Airlines: $100M/year benefit from use of LP
  - Commercial solvers (CPLEX, COIN, OSL), modeling languages (AMPL)
  - General tool for attacking intractable (NP-hard) optimization problems
**LP Example: Production of Bowls vs. Mugs**

\[ 4x_1 + 3x_2 \leq 120 \]

**Constraints:** There are 40 hours of labor and 120 pounds of clay available each day.

**Decision variables:**
- \( x_1 \) = number of bowls to produce
- \( x_2 \) = number of mugs to produce

**Note 1:** \( x_1, x_2 \) must be non-negative

**Note 2:** \( x_1, x_2 \) can be fractional (non-integer)

<table>
<thead>
<tr>
<th>PRODUCT</th>
<th>Labor (hr/unit)</th>
<th>Clay (lb/unit)</th>
<th>Revenue ($/unit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bowl</td>
<td>1</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>Mug</td>
<td>2</td>
<td>3</td>
<td>50</td>
</tr>
</tbody>
</table>
LP Example: Production of Bowls vs. Mugs

<table>
<thead>
<tr>
<th>PRODUCT</th>
<th>Labor (hr/unit)</th>
<th>Clay (lb/unit)</th>
<th>Revenue ($/unit)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bowl</td>
<td>1</td>
<td>4</td>
<td>40</td>
</tr>
<tr>
<td>Mug</td>
<td>2</td>
<td>3</td>
<td>50</td>
</tr>
</tbody>
</table>

Constraints: There are 40 hours of labor and 120 pounds of clay available each day

Decision variables: \( x_1 = \text{number of bowls to produce} \)
\( x_2 = \text{number of mugs to produce} \)

Maximize \( Z = 40x_1 + 50x_2 \)

Subject to
\[
\begin{align*}
x_1 + 2x_2 & \leq 40 \text{ hr (labor constraint)} \\
4x_1 + 3x_2 & \leq 120 \text{ lb (clay constraint)} \\
x_1, x_2 & \geq 0
\end{align*}
\]

Example solution: \( x_1 = 24 \text{ bowls} \) \( x_2 = 8 \text{ mugs} \)

Revenue = $1,360
Geometric Interpretation

Feasible Region
= Area common to both constraints

\[ 4x_1 + 3x_2 \leq 120 \text{ lb} \]

\[ x_1 + 2x_2 \leq 40 \text{ hr} \]

\[ z = 40x_1 + 50x_2 \]

Which vertex of the feasible region has highest \( z \) value?

\( x_1 = \# \text{bowls} \)
\( x_2 = \# \text{mugs} \)
The feasible region is an intersection of half-spaces that arise from the constraints:
- Vertices of the feasible region = where 2 constraints are tight
- Vertices are like “corners”
Extreme Point = Solution of Simultaneous Equations

\[
\begin{align*}
4x_1 + 3x_2 &\leq 120 \text{ lb} \\
x_1 + 2x_2 &\leq 40 \text{ hr}
\end{align*}
\]

\[
\begin{align*}
x_1 + 2x_2 &= 40 \\
4x_1 + 3x_2 &= 120 \\
4x_1 + 8x_2 &= 160 \\
-4x_1 - 3x_2 &= -120 \\
5x_2 &= 40
\end{align*}
\]

\[
x_2 = 8
\]

\[
x_1 = 24
\]

\[
Z = 50(24) + 50(8) = 1360
\]
The closed line segment with endpoints \( \mathbf{a} \) and \( \mathbf{b} \) consists of all points \( \lambda \cdot \mathbf{a} + (1 - \lambda) \cdot \mathbf{b}, \quad 0 \leq \lambda \leq 1 \). These points are \textit{convex combinations} of \( \mathbf{a} \) and \( \mathbf{b} \).
Geometry: Convex, Concave, Extreme

• Inequalities induce halfspaces with respect to hyperplanes.

• Bounded feasible region: convex polygon or polytope.

• Feasible region = convex set: If a, b feasible, so is \((a+b)/2\).

• Extreme point: Feasible solution x that cannot be written as \((a+b)/2\) for two distinct feasible solutions a and b.

(Regions, points)
Functions: Convex, Concave, and Linear

- Function $f$ is convex if $f[\lambda x_1 + (1-\lambda)x_2] \leq \lambda f(x_1) + (1-\lambda)f(x_2)$
  - $f$ defined over a convex domain
  - If $f$ is convex, then a local minimum is a global minimum

- Function $f$ is concave if $-f$ is convex
  - If $f$ is concave, then a local minimum can occur only at an extreme point of the domain of $f$

- **KEY OBSERVATION:** A linear function is both convex and concave (!!!)
  - Local minima occur only at extreme points (from concavity)
  - Any local minimum is a global minimum (from convexity)
  - $\rightarrow$ To find a global minimum, only need to look at extreme points
Geometry: Convex Sets, Convex Functions

- Inequalities induce *halfspaces* with respect to *hyperplanes*.
- Bounded feasible region: convex *polygon* or *polytope*.
- Feasible region = convex set: If a, b feasible, so is (a+b)/2.
- Extreme point: Feasible solution x that cannot be written as (a+b)/2 for two distinct feasible solutions a and b.

Function f is *convex* if f[λx₁ + (1-λ)x₂] ≤ λf(x₁) + (1-λ)f(x₂)
  - f defined over a convex domain
  - If f is convex, then a local minimum is a global minimum.
Geometric Interpretation of Objective Function

New $Z = 70x_1 + 20x_2$

New Optimum:
\[
\begin{align*}
x_1 &= 30 \text{ bowls} \\
x_2 &= 0 \text{ mugs} \\
Z &= $2,100
\end{align*}
\]

- A global optimum has maximum projection on (or, maximum inner product with) the objective function
Minimization Formulation

**CHEMICAL CONTRIBUTION**

<table>
<thead>
<tr>
<th>Brand</th>
<th>Nitrogen (lb/bag)</th>
<th>Phosphate (lb/bag)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gro-plus</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>Crop-fast</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

Minimize \( Z = 6x_1 + 3x_2 \)

subject to

\[
2x_1 + 4x_2 \geq 16 \text{ lb of nitrogen}
\]

\[
4x_1 + 3x_2 \geq 24 \text{ lb of phosphate}
\]

\[x_1, x_2 \geq 0\]
Geometric Interpretation

Possible scenarios: (1) Optimal solution can be a vertex of the feasible region, or (2) optimal solution can be unbounded, or (3) there is no feasible region at all

\[ Z = 6x_1 + 3x_2 \]
Simplex Method

- Invented by George Dantzig in 1947
  - Developed shortly after WWII in response to logistics challenges
  - Used for 1948 Berlin airlift

- Objective function is linear, and feasible region is convex
  - can use a greedy strategy (called “hill-climbing” in textbook)
    - Start at an extreme point
    - “Pivot” from one extreme point to a neighboring one
      - NEVER WORSEN THE OBJECTIVE FUNCTION IN DOING SO !!!
    - Stop when the current extreme point is a local optimum

- Which (improving) neighbor?
- How to avoid degeneracy / stalling (= new set of “tight constraints” but same extreme point)
- Cycling
- Known implementations can take exponential in worst-case, but in practice see linear #pivots
This Week: One Big Picture

(various – e.g., Matching)

Network Flow

Reduces to (“boils down to”)

Linear Programming

Reduces to (“boils down to”)

More specialized, efficient

More general
Network Flow Can Be Solved as a LP

- Optimization variables: \( f_{SA}, f_{BA}, \) etc.
- Capacity constraints: \( 0 \leq f_{SA} \leq 3, 0 \leq f_{ET} \leq 2, \) etc.
- Conservation constraints: \( f_{SA} + f_{BA} = f_{AD}, \) etc.
- Non-negative flow constraints: \( f_{IJ} \geq 0 \) for all \( I, J \)
- LP: maximize \( F = f_{SA} + f_{SB} \) subject to constraints

\[ \text{Maximize} \quad f_{DT} + f_{ET} \quad \text{(same as } f_{SA} + f_{SB}) \]

What is the maximum flow?
Vertex Capacities?

- Optimization variables: $f_{SA}$, $f_{BA}$, etc.
- Capacity constraints: $0 \leq f_{SA} \leq 3$, $0 \leq f_{ET} \leq 2$, etc.
- Conservation constraints: $f_{SA} + f_{BA} = f_{AD}$, etc.
- Non-negative flow constraints: $f_{IJ} \geq 0$ for all $I,J$
- LP: maximize $F = f_{SA} + f_{SB}$ subject to constraints

What is the maximum flow?
Lower Bounds on Flows?

- Optimization variables: $f_{SA}$, $f_{BA}$, etc.
- Capacity constraints: $0 \leq f_{SA} \leq 3$, $0 \leq f_{ET} \leq 2$, etc.
- Conservation constraints: $f_{SA} + f_{BA} = f_{AD}$, etc.
- Non-negative flow constraints: $f_{ij} \geq 0$ for all $i,j$
- LP: maximize $F = f_{SA} + f_{SB}$ subject to constraints

What is the maximum flow?

Require $f_{SA} \geq 1$
Maximum Circulation

- Optimization variables: $f_{SA}$, $f_{BA}$, etc.
- Capacity constraints: $0 \leq f_{SA} \leq 3$, $0 \leq f_{ET} \leq 2$, etc.
- Conservation constraints: $f_{SA} + f_{BA} = f_{AD}$, etc.
- Non-negative flow constraints: $f_{IJ} \geq 0$ for all $I,J$
- LP: maximize $F = f_{SA} + f_{SB}$ subject to constraints

What is the maximum circulation?
Matching Reduces to Max Flow

What is the size of the maximum matching between Workers and Jobs?

```
W_1  J_1
W_2  J_2
W_3  J_3
W_4  J_4
```

"boils down to", "can be solved using"
Matching Reduces to Max Flow

What is the size of the maximum flow from S to T? (All edge capacities = 1; red-colored edges used by max flow.)

What is the size of the maximum matching between Workers and Jobs?
LP for System of Distinct Representatives

Many Problems Reduce to Max-Flow (1)
- The UCSD Algorithms Club has $N$ members.
- There are $M$ committees of ASUCSD to which the club can send a representative.
- Each club member is suited to some subset of committees.
- Can we assign club members so that each committee has a distinct representative? (No one serves on $>1$ committee.)

$$\text{maximize } f_{s,1} + \ldots + f_{s,M}$$

subject to

$$f_{s,1} = f_{1,2} + f_{1,8}$$

$\vdots$

$$f_{2,9} + f_{3,9} = f_{9,T} \text{ at } 9$$

$$0 \leq f_{i,j} \leq 1 \quad \forall (i,j) \in E$$

Diagram of a network flow problem with sources $S$ and $T$, nodes labeled with numbers 1 to 10, and edges connecting them.
LP: Fitting a Line

Problem: Given points \((x_1, y_1), \ldots, (x_m, y_m)\), find the best-fit line.

“Linear Regression Modeling”

- General form of a line: \(ax + by = c\)
- \(\rightarrow\) Find \(a, b, c\) such that \(\max_i |ax_i + by_i - c|\) is minimized

How do we write this down as an LP?
LP: Fitting a Line

Problem: Given points \((x_1, y_1), \ldots, (x_m, y_m)\), find the best-fit line.

- General form of a line: \(ax + by = c\)
- \(\rightarrow\) Find \(a, b, c\) such that \(\max_i |ax_i + by_i - c|\) is minimized
- Trick: Introduce a new variable \(e = \max\) deviation from line
- LP: minimize \(e\)
  \[
  \begin{align*}
  \text{s.t.} \quad & ax_i + by_i - c \leq e \quad \text{for all } i = 1, \ldots, m \\
  & ax_i + by_i - c \geq -e \quad \text{for all } i = 1, \ldots, m
  \end{align*}
  \]
- E.g., with \(m = 7\) points, we get an LP with 14 constraints
Manufacturing of metal wires in a chip requires *planarization* of each layer before the next layer is fabricated.

Varying pattern density of wires causes unevenness in planarization result.

Chip layouts have “density uniformity rules”, e.g., “In any 200um x 200um window of the layout of each wiring layer, the metal density must be between 40% and 60%.”

Designers satisfy these rules by adding “dummy fill” into the chip layout.
$0 \leq \text{Amount of Dummy Fill} \leq \text{"Slack"}$
LP: “Dummy Fill” in Chip Design

- Manufacturing of metal wires in a chip requires *planarization* of each layer before the next layer is fabricated.
- Varying pattern density of wires causes unevenness in planarization result.
- Chip layouts have “density uniformity rules”, e.g., “In any 200um x 200um window of the layout of each wiring layer, the metal density must be between 40% and 60%.”
- Designers satisfy these rules by adding “dummy fill” into the chip layout.
- Problem: Given a layout of wires and a set of “windows” to verify, insert the dummy fill to minimize the maximum difference of window densities.
LP for Dummy Fill

• Same trick of introducing a new variable, \( M \)
• \( M \) is lower-bounded by the difference of densities between any two windows \( \rightarrow \) pushing down \( M \) means more uniform density
• Minimize \( M \) s.t.

\[
M \geq |\text{dens}(W_i) - \text{dens}(W_j)| \quad \text{for all windows } i,j
\]

// force minimum variation of post-fill window densities
\[
\text{dens}(W) = \text{density(orig layout + fill)} \text{ in all tiles of } W
\]

// density of a window = original + added fill
\[
0 \leq \text{fill}_{ij} \leq \text{slack}_{ij} \text{ for all tiles } T_{ij}
\]

// cannot put more fill than will fit (i.e., more than “slack”)
// or negative amount of fill into any window
Fibonacci DP: Tabulation

\[ f[0] = 0; \quad f[1] = 1; \]

\[
\text{for (int } i = 2; i <= n; i++) \{
\quad f[i] = f[i - 1] + f[i - 2];
\}
\]

\textit{Fibonacci: } \[ f[0] = 0; \quad f[1] = 1; \quad f[n] = f[n-1] + f[n-2], \quad n > 1 \]

- Iterative, bottom-up
- Use values for subproblems \( f[i - 1] \) and \( f[i - 2] \) to compute \( f[i] \)
- Start with smaller subproblems and advance to larger ones
Memoization: General Case

• If we want to compute some function \( f(x) \) using memoization, then our implementation can look something like this:

```python
compute(x):
    if f(x) == undefined:
        BASE CASE
        f(y) = compute(y) (for all y, such that \( f(x) \) depends on \( f(y) \))
        f(x) = g({f(y)}) (for all y, such that \( f(x) \) depends on \( f(y) \), and where \( g(...) \) is some function over values \( f(y) \))
    return f(x)
```

• This implementation allows each \( f(x) \) to be computed only once.
• Complexity: \( O(\text{#possible values for } x) \times \text{time to compute } g(...) \)
Fibonacci: Memoization

```c
int compute(int n) {
    if(f[n] == -1) {
        if(n <= 1)
            f[n] = n;
        else {
            f[n - 1] = compute(n - 1);
            f[n - 2] = compute(n - 2);
            f[n] = f[n - 1] + f[n - 2];
        }
    return f[n];
}
```

- Recursive, top-down
- Check whether the value f[n] has been computed already
- If not, then compute f[n] using the DP recurrence
- Otherwise, just return the value of f[n]
- Each value of f[n] will be computed only once

USB example: Slides 56-57
• **Problem**: You are given a number n. In how many ways can n be represented as the sum of a **non-decreasing** sequence of positive integers?

• **Notation and Subproblems**:
  - Let $f[i][k]$ denote the number of ways to represent a number $i$ as a sum of a non-decreasing sequence of positive integers of length $k$.
  - That is, $f[i][k]$ is the number of distinct sequences $\{a[1], a[2], \ldots, a[k]\}$, such that:
    - $a[1] + a[2] + \ldots + a[k] = i$
    - $1 \leq a[1] \leq a[2] \leq \ldots \leq a[k]$
    - $a[j]$ is an integer for all $1 \leq j \leq k$

• Desired answer: $f[n][1] + f[n][2] + \ldots + f[n][n]$
[DP: Ways to Sum – Recurrence?]

- Let’s consider two cases:  \( a[1] = 1 \) and \( a[1] \neq 1 \)
  - Let \( a[1] = 1 \). Then we have \( 1 + a[2] + \ldots + a[k] = i \)
    - \( \Rightarrow a[2] + \ldots + a[k] = i - 1 \)
    - \( \Rightarrow \) the number of all such sequences (\( a[1] = 1 \)) is \( f[i - 1][k - 1] \)
  - Let \( a[1] \neq 1 \). Then we have \( a[1] + a[2] + \ldots + a[k] = i \)
    - \( \Rightarrow (a'[1] + 1) + (a'[2] + 1) + \ldots + (a[k] + 1) = i \), \( a'[1] + a'[2] + \ldots a'[k] = i - k \)
    - \( \Rightarrow \) the number of all such sequences (\( a[1] \neq 1 \)) is \( f[i - k][k] \)

- Recurrence:  \( f[i][k] = f[i - 1][k - 1] + f[i - k][k] \)

- Base case:  \( f[1][1] = 1 \)
DP: Ways to Sum

• Recall: desired answer = f[n][1] + f[n][2] + … + f[n][n]

• Iterative and recursive (memoization) implementations both have $O(n^2)$ time complexity
DP: Ways to Sum – Iterative Implementation

\[ f[1][1] = 1 \]
for (int \(i = 1; i \leq n; i++\))
  for (int \(k = 1; k \leq n; k++\))
    if (\(i \neq 1 \) || \(k \neq 1\))
      \[f[i][k] = f[i - 1][k - 1] + f[i - k][k]\]

Complexity: \(O(n^2)\)

Memoized Implementation: Slide 58
Algorithm Design Examples
From Application to Algorithm Design

• Find all maximal **regularly-spaced, collinear** subsets of given pointset

• Lower bound?

\[ \Omega(n^2) \]

• Naïve algorithm?

\[ n^2 \log n \]

• Simpler variants?
Number Placement

- Given a list of $n$ distinct integers and a sequence of $n$ boxes with preset inequality signs inserted between them, design an algorithm that places the numbers into the boxes to satisfy those inequalities.

- For example, given four boxes and three inequality signs between them: $\_ < \_ > \_ < \_$, the numbers 2, 5, 1, 0 can be placed as $0 < 5 > 1 < 2$.

- Algorithm?
Number Placement

- Given a list of \( n \) distinct integers and a sequence of \( n \) boxes with preset inequality signs inserted between them, design an algorithm that places the numbers into the boxes to satisfy those inequalities.

\[
\_ \ < \ _\_ > \ _\_ \ < \ _\_ \quad 2, 5, 1, 0
\]

- **Greedy algorithm**
  - Sort the list in increasing order
  - Repeat \( n-1 \) times:
    - If the first inequality sign is "<", place the first (smallest) number in the first box; otherwise, place the last (largest) number in that box
    - Delete the number from the list along with the box it was put in
  - When one number remains, place it in the remaining box
Reduction
Recall From Lecture 4 … “Reduction”

- **Given:** Sorting LB is $\Omega(n \log n)$
- **Reduction from Sorting to Convex Hull**
  - **Input:** an arbitrary instance $I_{\text{SORT}}$ of SORT
  - **Transform** $I_{\text{SORT}} = \{x_1, x_2, \ldots, x_n\}$ into an instance $I_{\text{C-HULL}} = \{(x_1, x_1^2), (x_2, x_2^2), \ldots, (x_n, x_n^2)\}$ of C-HULL
  - **Solve** the C-HULL problem for instance $I_{\text{C-HULL}}$
  - **Transform** solution of $I_{\text{C-HULL}}$ to solution of $I_{\text{SORT}}$

- **Note:** Each **Transform** has $O(n)$ complexity
  $\Rightarrow$ C-HULL solver cannot be faster than $\Omega(n \log n)$
Reduction From A to B

- **A:** Problem with “known difficulty”
- **B:** Problem with “unknown difficulty”

**Reduction:** “Transfers the known difficulty of Problem A to Problem B”

\[ TX = \text{transformation} \]
Reduction From EU to SORT

- **EU**: “Element Uniqueness” – given a list of n elements, are they all distinct?  
  == a decision problem

- **SORT**: Given a list of n elements, arrange them in non-decreasing order

- Suppose that we are given that EU is $\Omega(n \log n)$. Can we prove an $\Omega(n \log n)$ LB on SORT?
Reduction From EU to SORT

- **EU**: “Element Uniqueness” – given a list of $n$ elements, are they all distinct?
- **SORT**: Given a list of $n$ elements, arrange them in non-decreasing order

Suppose that we are given that EU is $\Omega(n \log n)$. Can we prove an $\Omega(n \log n)$ LB on Sorting?
Matching in an Improved Euclidean TSP Approximation
Euclidean TSP: Recall the 2-Approximation

DFS traversal of MST

Shortcutting of the DFS tour (e.g., replacing $a-b-c-b-a-d$, by $a-b-c-d$)

$$\text{Tour}_{\text{Heur}} \leq 2 \times \text{MST} \leq 2 \times \text{Tour}_{\text{opt}}$$
Now: A 3/2-Approximation for Euclidean TSP

• Improving the construction of a heuristic TSP tour from a tree traversal: (Christofides, 1976)
  – New way to look at previous construction: we built an Eulerian circuit on top of the tree, by doubling each edge. Then we obtained a TSP tour by taking shortcuts in the Eulerian circuit.
  – Intuition: Tour_Heur has less cost than the cost of the Eulerian graph. If we can start with a lower-cost Eulerian graph, we will get a better bound !!!

⇒ What is a cheaper augmentation of the MST, such that the resulting graph is Eulerian?
3/2-Approximation for Euclidean TSP

- Key property of **Eulerian** graph: every node has even degree
- Basic property of the MST (or any graph): there is an even number of odd-degree nodes
  - Handshake Lemma: Sum of node degrees = 2 * #edges
  - Idea: add edges to MST to make it Eulerian: +1 edge for each odd-degree node in the MST
- Specifics
  - Find a minimum-cost matching among the odd-degree vertices of the MST
  - Add an edge between every matched pair
  - Result == an Eulerian graph, which we can traverse and shortcut exactly as we did with the doubled MST
3/2-Approximation for Euclidean TSP

MST plus matching: red line = Euler circuit

TSP tour obtained by shortcutting Euler circuit

Previous 2-approximation
3/2-Approximation for Euclidean TSP

- Consider the **optimal** TSP tour with cost $\text{Tour}_{opt}$
- Nodes marked by triangles are odd-degree nodes in the MST. The **solid green edges** correspond to the **optimal TSP tour**. The **red** dashed lines and **blue** dashed lines represent two possible matchings among those odd-degree nodes.

Either the total length of **blue lines** or the total length of **red lines** is $\leq 0.5 \times \text{Tour}_{opt}$

The **minimum** matching cost is $\leq$ either the red or blue matching cost $\Rightarrow$ If we use Min Matching, total edge cost of the Euler circuit (before any shortcuts) will be $\leq 1.5 \times \text{Tour}_{opt}$
Supplementary: Viterbi, USB, etc.
USB: Memoization

- Memoization is **top-down**
- For the USB problem, the algorithm starts with the total size of the USB drive, and recursively calculates the optimal number of files for smaller and smaller USB drives all the way down to the base case, memoizing the results for any subproblems so we do not have to recurse multiple times on the same subproblem.

**Example: USB(5, [1,2,3])**
- We recurse top-down, solving $T(5)$, then $T(4)$, then $T(3)$, $T(2)$, and $T(1)$. When we encounter the problems $T(3)$ and $T(2)$ again in our first layer, we have saved these solutions and no longer have to recurse down their subtrees.
USB: Tabulation

- Tabulation is **bottom-up**
- Instead of starting with the total size of the USB drive, \( n \), we begin calculations by calculating the optimal number of files for a USB of size \( k = 1 \). Then, we calculate the optimal number of files for USB\([k+1]\) until we reach USB\([k = n]\), which is our solution.

- **Example: USB(5, [1,2,3])**
- We iterate from the bottom up, solving T(1), T(2), and T(3), then using these to solve T(4), and then T(5). We do not have to recurse since we know exactly what subproblems each new subproblem will depend on.
DP: Ways to Sum – Recursive Implementation

```cpp
int compute(int i, int k) {
    if(i == 1 && k == 1) f[i][k] = 1;
    if(f[i][k] == -1) {
        f[i][k] = 0;
        if(i - 1 > 0 && k - 1 > 0) {
            f[i - 1][k - 1] = compute(i - 1, k - 1); f[i][k] += f[i - 1][k - 1];
        }
        if(i - k > 0) {
            f[i - k][k] = compute(i - k, k); f[i][k] += f[i - k][k];
        }
    }
    return f[i][k];
}
```

Complexity: \(O(n^2)\)
DP: Viterbi Algorithm
DP: Speech Recognition (Viterbi Algorithm)

• Let us consider the problem of speech recognition in AI which involves converting speech into text.

• Suppose we are given a sequence of observations (could be the current/voltage values recorded by the microphone) $O = [o_1, o_2, o_3, \ldots, o_n]$.

• Next, we are given a set of alphabets $A = \{a_1, a_2, \ldots, a_k\}$, that each of the above observations can correspond to.

• Next, we are given an acoustic model, which gives the probability of an observation $o$ being generated by an alphabet $a$, represented by $p(o|a)$. 
DP: Speech Recognition

• For ex., let us say we have two alphabets $a_1$ and $a_2$. Whenever $a_1$ occurs in the speech, the current observed is high, whereas, whenever $a_2$ occurs in the speech, the current observed is low. Therefore, $p(o|a_1)$ will be high if current observed ($o$) is also high and $p(o|a_1)$ will be low if the current observed is low. It’ll be the opposite for $p(o|a_2)$. Since, the current recorded by the microphone depends on the acoustic property of the alphabet in the speech, this is called the “acoustic model”.

• Finally, we are also given a “language model”, which gives the probability of an alphabet $a_t$, following another alphabet $a_{t-1}$, represented by $p(a_t|a_{t-1})$. For ex., in English alphabets, the probability of $t$ following $a$ (as in “cat”, “bat”, “hat” etc”) is much higher than $t$ following $b$ (“...bt...” ??). Since, here we are trying to capture the dependence of observations on the language, this is called the language model.
DP: Speech Recognition

• Given all of the above information, our task is to find the sequence of alphabets \( S = [s_1, s_2, \ldots, s_n] \) which maximizes the probability of the observations, i.e.,
  \[ S = \arg\max_S \left( \prod_{i=1}^{n} (p(s_i|s_{i-1})p(o_i|s_i)) \right) \]

• You may assume that \( s_0 = \text{null} \) and that \( p(s|\text{null}) \) is also given to you.
• Ex. $P(a_1a_2a_1a_3|o_1o_2o_3o_4) = \text{Probability of sequence } a_1a_2a_1a_3,$
given the observations $o_1o_2o_3o_4 =$
$p(a_1|\text{null})p(o_1|a_1)p(a_2|a_1)p(o_2|a_2)p(a_1|a_2)p(o_3|a_1)p(a_3|a_1)p(o_4|a_3)$
The previous expression should be interpreted as follows:

- The probability of going to $a_1$ from null = $p(a_1|\text{null})$.
- The probability of observing $o_1$, given alphabet is $a_1 = p(o_1|a_1)$.
- The probability of going from $a_1$ to $a_2 = p(a_2|a_1)$.
- The probability of observing $o_2$, given alphabet is $a_2 = p(o_2|a_2)$.
- The probability of going from $a_2$ to $a_1 = p(a_1|a_2)$.
- The probability of observing $o_3$, given alphabet is $a_1 = p(o_3|a_1)$.
- The probability of going from $a_1$ to $a_3 = p(a_3|a_1)$.
- The probability of observing $o_4$, given alphabet is $a_3 = p(o_4|a_3)$.
- The probability of sequence $a_1a_2a_1a_3$, given the observations $o_1o_2o_3o_4$ is the product of all of the above.
DP: Speech Recognition

• Brute force: Try each and every possible sequence. Complexity = \(O(k^n)\).

• Greedy Solution:
  – Greedy Heuristic 1: If currently at \(a_i\), go to \(a_j\) next such that \(p(a_j|a_i)\) is maximized.

  – Greedy Heuristic 2: If at \(a_i\) at time instance \(t\), go to \(a_j\) at time instance \(t+1\) such that \(p(a_j|a_i)p(o_{t+1}|a_j)\) is maximized.
DP: Speech Recognition

- Greedy Counterexample: $A = \{a_1, a_2\}$, $O = \{o_1, o_2\}$.

<table>
<thead>
<tr>
<th>alphabet \ observation</th>
<th>o1</th>
<th>o2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>$p(o_1</td>
<td>a_1) = 0.5$</td>
</tr>
<tr>
<td>a2</td>
<td>$p(o_1</td>
<td>a_2) = 0.5$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>prev. alphabet \ current alphabet</th>
<th>a1</th>
<th>a2</th>
</tr>
</thead>
<tbody>
<tr>
<td>null</td>
<td>$p(a_1</td>
<td>null) = 0.6$</td>
</tr>
<tr>
<td>a1</td>
<td>$p(a_1</td>
<td>a_1) = 0.99$</td>
</tr>
<tr>
<td>a2</td>
<td>$p(a_1</td>
<td>a_2) = 0.01$</td>
</tr>
</tbody>
</table>

- Optimal sequence: $a_2a_2 \Rightarrow 0.4*0.5*0.99*0.9 = 0.1782$
- Greedy Heuristic 1: $a_1a_1 \Rightarrow 0.6*0.5*0.99*0.1 = 0.0297$
- Greedy Heuristic 2: $a_1a_1 \Rightarrow 0.6*0.5*0.99*0.1 = 0.0297$
DP: Speech Recognition

- Subproblem: $T(i,j) = \text{Maximum probability of any sequence till timestamp } j, S = [s_1, s_2, \ldots, s_j] \text{ such that } s_j = a_i$.
- Recurrence relation:
  - $T(i,j) = \max_l(T(i-1,l) * p(a_i|a_l) * p(o_j|a_i))$
  - i.e., the maximum probability of any sequence till timestamp $j$ ending at $s_j$ is the maximum over $l$ of all the sequences ending at timestamp $j-1$ at alphabet $a_i$ times the probability of going from alphabet $a_i$ to $a_j$ times the probability of observing $o_j$ because of alphabet $a_j$.
- Base Case:
  - $T(i,1) = T(p(a_i|null)*p(o_1|a_i)$
- We are essentially building a $k \times n$ table.
- Final answer = $\max_i(T(i, n))$
- The sequence can be retrieved through backtracking.
- Time Complexity = $O(nk^2)$
- Space Complexity = $O(nk)$