Dynamic Programming

How to Design DP Algos

1. Define "Subproblems"

2. Relate the subproblems using a recurrence relation.

3. Design an algorithm: 1) Order of solving subproblems; 0) Base cases.
EXAMPLE: COMPUTING OPTIMAL STRATEGIES

→ GAME with two different MOVES:

**Move A** with prob. $\frac{1}{2}$ + 1 point

**Move B** with prob $\frac{1}{2}$ - 1 point

→ NEED to PLAY K Moves

→ WIN if end up with > 100 points.

Find Strategy that maximizes the Probability of WINNING.
What is a strategy?

Strategic:

\[ S : \begin{cases} \text{points} & \text{you have} \\ \text{moves are} & \text{left} \end{cases} \]

\[ \rightarrow \text{Move A} \]

or

\[ \text{Move B} \]
1) $T[a \text{ points, } b \text{ moves left}] = \text{Probability of the optimal strategy winning the game.}$

2) $T[a, b] = \max$

\[
\begin{align*}
&\text{Move } A \\
&\frac{1}{2} T[a+1, b-1] + \frac{1}{2} T[a-1, b-1]
\end{align*}
\]

\[
\begin{align*}
&\text{Move } B \\
&\frac{1}{2} T[a+10, b-1] + \frac{1}{2} T[a-10, b-1]
\end{align*}
\]

Base case:

$T[a, 0] = 1$ if $a > 100$
**Base case**

\[ T[a,0] = 1 \text{ if } a > 100 \]

for \( b = 1 \) to \( K \) moves left

for \( a = -10K \) to \( 10K \)

\[
T[a,b] = \max \left\{ \begin{array}{l}
\text{Move A} \quad \frac{1}{2} T[a+1, b-1] + \frac{1}{2} T[a-1, b-1] \\
\text{Move B} \quad \frac{1}{2} T[a+10, b-1] + \frac{1}{2} T[a-10, b-1]
\end{array} \right\}
\]
\begin{align*}
\Pr \left[ \text{win } \mid \text{a points left} \right] \\
&= \Pr \left[ \text{MoveA} \right] \cdot \Pr \left[ \text{win } \mid \text{a points } +1 \right] \\
&\quad + \Pr \left[ \text{Move} \right] \cdot \Pr \left[ \text{win } \mid \text{a points } -1 \right] \\
&\quad + \Pr \left[ \text{Move} \right] \cdot \Pr \left[ \text{win } \mid \text{b moves } -1 \right]
\end{align*}
**TRAVELLING SALESMAN PROBLEM:**

**Input:** n cities with distances \( d_{ij} \).

**Goal:** Find the shortest tour:
1) starting at A
2) visiting every city exactly once
3) ending at A.

**Naive Alg:** \( \Theta(n!) \) time.

**Goal:** \( \Theta(n^2 - 2^n) \) algo

---

**Example: A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow A**

- \( 1 + 5 + 7 + 3 + 10 = 25 \)
- \( 6 + 7 + 3 + 2 + 1 = 19 \)
Help Toody and Muldoon find the shortest round trip route to visit all 33 locations shown on the map.

All you do is draw connecting straight lines from location to location to show the shortest round trip route.

HERE’S THE CORRECT START...

Begin at Chicago, Illinois. From there, lines show correct route as far as Erie, Pennsylvania. Next, do you go to Carlisle, Pennsylvania or Wana, West Virginia? Check the easy instructions on back of this entry blank for details.
**Subproblem:** For $S$ such that $\{i \in S\}$ and $\{i \in S\}$

$$T[S, i] = \text{length of shortest path/tour that}$$

1) starts at 1
2) visits every node in subset $S$
3) ends at $i$

Every vertex in $S$. 
Let $j$ := penultimate vertex on the path $T[S, i]$.

1) Starts at $i$
2) Ends at $j$
3) Visits $S \setminus \{i\}$

$$T[S, i] = \min_{j \in S \setminus \{i\}} \left[ T[S \setminus \{i\}, j] + d_{ij} \right]$$
Base Case: \( T[d_1, 1] = 0 \)

\[
\begin{align*}
\text{Alg:} \quad & \frac{2^n}{2^n} \rightarrow \{ \text{for set size } = 2 \text{ to } n \} \\
& \text{for subsets } S, |S| = \text{set size}, 1 \in S \\
& \text{for } i \in S \\
& \{ T[S, i] = \min_{j \neq i} \{ T[S \setminus \{i\}, j] + d_{ij} \} \} \\
\end{align*}
\]

\( \Theta(2^n \cdot n^2) \)
**INDEPENDENT SET:**

**Input:** Graph $G = (V,E)$

**Defn:** A set $S$ is an independent set if NO EDGES inside $S$, i.e. $\forall u,v \in S$, $(u,v) \notin E$

**Goal:** Find the largest independent set.

**Goal:** $\Theta(1 + |E|)$ for independent set on trees
\[ G = (V, E) \text{ is a tree,} \]
- Assign some vertex \( r = \text{root} \)

\[ \text{If vertices } v, \]
\[ T_v = \text{subtree hanging at } v \]

\text{SUBPROBLEM:}
\[ I(v) = \text{size of the largest independent set in subtree } T_v \{ \text{hanging at } v \} \]

\[ I(\text{root}) = ?? \]
Subproblem:

\[ I(v) = \text{size of the largest independent set in subtree } T_v \text{ (hanging at } v) \]

Recurrence:

\[
I(v) = \max_{u \text{ child of } v} \left\{ \begin{array}{ll}
\text{Case 1: largest ind set includes } v \\
1 + \sum_{w \text{ grandchild}} I(w)
\end{array} \right. \\
\text{Case 2: doesn't include } v \\
\sum_{\text{children } u} I(u)
\]

Order: Bottom up on the tree.
IMPLEMENTING via DFS:

explore \((v)\) if

\[ \text{visited}[v] = \text{TRUE} \]

for each edge \((v,w) \in E\)

if \(\text{visited}[w] = \text{FALSE} \)

explore \((w)\)

\[
I[v] = \begin{cases} 
\text{Case 1: largest indset include } v & 1 + \sum_{w \text{ grandchild}} I(w) \\
\text{Case 2: doesn't include } v & \sum_{\text{children } u} I(u) 
\end{cases}
\]

This Code is Incomplete

DOES NOT HANDLE BASE CASE.