Dynamic Programming II [MIT OpenCourseWare 6.006]

DP ≈ "careful brute force" (version 1)

DP ≈ guessing + recursion + memoization (version 2)

DP ≈ shortest path in some DAG (version 3)

time = # subproblems × $\underbrace{\text{time/subproblem}}$

treating recursive calls as $\Theta(1)$, since we only pay for the recursion the first time we calculate it

5 "easy" steps to DP:

1. define subproblems
   - # subproblems
2. guess (part of solution)
   - # choices
3. relate subproblems solution (through recurrence)
   - time / subproblem
4. recurse & memoize

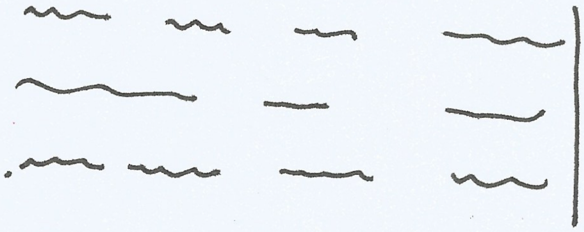
   Or build DP table bottom-up (usually the fastest in terms of constants)
   - check subproblem recurrence is acyclic
5. solve original problem

Examples:

	Fibonacci	Shortest Paths
① subproblems	F_k for $k = 1, \ldots, n$	$\delta_k(s, v)$ for $v \in V$, $0 \le k < \lvert V \rvert$
# subproblems:	n	V^2
② guess:	nothing	edge into $\underline{v}$ (if any)
# choices:	1	indegree(v) + 1
③ recurrence:	$F_k = F_{k-1} + F_{k-2}$	$\delta_k(s, v) = \min\{\delta_{k-1}(s, u) + w(u, v) \mid (u, v) \in E\}$
time/subproblem	$\Theta(1)$	$\Theta(\text{indegree}(v) + 1)$
④ topological order		

Text Justification Problem

- Split text into "good" lines



Always end on the same column whilst minimizing the number of empty spaces

- define badness(i,j) for line of words$[i:j]$:

  e.g. $\begin{cases} \infty, & \text{if the words do not fit on the line} \\ (\text{PageWidth} - \text{TotalWidth})^3, & \text{otherwise} \end{cases}$

  Why power 3? Who knows... This is the one used by LaTeX

  It might very well be that it worked with power 1, 2, ...

- **Goal**: split words into lines in order to minimize the sum of badn

① **Subproblem**: minimize badness for suffix words$[i:]$

#subproblems $= \Theta(n)$ where n = # words

② **Guessing**: where to end first line, ~~[illegible]~~

Do we end it on the $i+1$ word? / " " " " " " $i+2$ word? } # choices $\leq n-i = O(n)$

③ **Recurrence**:

$$DP(i) = \min(\text{badness}(i,j) + DP(j) \text{ for } j \text{ in range}(i\ ,n\ ))$$

$$DP(n) = 0 \text{ (base case)}$$

time/subproblem?

- Memoization $\Rightarrow$ Repeated Recursive calls cost $\Theta(1)$
- We have # choices $= O(n)$ for j
- Therefore: time/subproblem $= O(n) \times \Theta(1) = \Theta(n)$

④ Order: We have to do this from the end to the beginning

Topological order: $n, n-1, \dots, 0$

Recursion goes first to process the n-th terms, then goes to n-1-th term ...

Total time: = # subproblems x time/subproblem

$$= n \times O(n)$$

$$= \Theta(n^2)$$

⑤ Original problem: $DP(0)$

" "

Parent Pointers ~~Problems~~:

- Idea: Remember which guess was best. Applies to all dynamic programs. Allows one to find the actual solution and not just the cost of the solution.
- In the expression:

$$\min(\text{badness}(i,j) + DP(j) \text{ for } j \text{ in range}(i+1, n+1))$$

When we compute this minimum we are trying all choices of j, one or more of them resulted in the minimum (in mathematics: argmin)

Argmin: What is the argument that gave the minimum value.

Lets call the argmin the parent pointer, i.e.:

$$\text{parent}[i] = \underbrace{\text{argmin}(\dots)}_{\text{best } j \text{ value}}.$$

- We store the parent pointer for each <u>i</u> and once we compute DP(0) we can follow the parent pointers to determine where the best choices are.

$$\underbrace{0}_{\text{The first line starts here in this index}} \rightarrow \underbrace{\text{parent}[0]}_{\text{The second line starts here in this index}} \rightarrow \underbrace{\text{parent}[\text{parent}[0]]}_{\text{The third line starts here in this index}} \rightarrow \cdots \rightarrow \underbrace{p[p[\ldots[p[0]]]}_{O(n)\ \text{times}}$$

- This way we can also know the "solution path" besides knowing the minimum value.

- In algorithm form:

```
i = 0
while i is not none:
    print("start line before word %d, i)
    i = parent[i]
```

Example:

- Words = [Tushar Roy likes to code]
  0 1 2 3 4
- Number of characters:
  - Index 0 – Tushar = 6
  - Index 1 – Roy = 3
  - Index 2 – Likes = 5
  - Index 3 – to = 2
  - Index 4 – code = 4
- Page Width = 10 characters
- We can start from the beginning (page 4.2, my predilection) or the end (page 4.1)
- Pseudo code:

$$DP(i) = \begin{cases} 0 & i == n \\ \min(\text{badness}(i,j) + DP(j+1)) & \forall j \in [i, n-1] \end{cases}$$

If we start from the end:

- $DP(5) = 0$

$$DP(4) = \min(\text{badness}(4,4) + DP(5)) = (10-4)^2 + 0 = 36$$

$$DP(3) = \min(\text{badness}(3,3) + DP(4), \text{badness}(3,4) + DP(5))$$
$$= \min((10-2)^2 + 36, (10-7)^2 + 0)$$
$$= \min(102, 9) = 9$$

$$DP(2) = \min(\text{badness}(2,2) + DP(3), \text{badness}(2,3) + DP(4), \text{badness}(2,4) + DP(5))$$
$$= \min((10-5)^2 + 9, (10-8)^2 + 36, \infty) =$$
$$= \min(34, 40, \infty) = 34$$

$$DP(1) = \min(\text{badness}(1,1) + DP(2),\ \text{badness}(1,2) + DP(3),\ \text{badness}(1,3) + DP(4),\ \text{badness}(1,4) + DP(5))$$

$$= \min((10-3)^2 + 34,\ (10-9)^2 + 9,\ \infty,\ \infty)$$

$$= \min(83,\ 10,\ \infty,\ \infty) = 10$$

If we start from the beginning:

$$DP(0) = \min(\text{badness}(0,0) + DP(1),\ \text{badness}(0,1) + DP(2),\ \text{badness}(0,2) + DP(3),\ \text{badness}(0,3) + DP(4),\ \text{badness}(0,4) + DP(5))$$

$$= \min((10-6)^2 + 10,\ (10-10)^2 + 34,\ \infty,\ \infty,\ \infty)$$

$$= \min(26,\ 34,\ \infty,\ \infty,\ \infty) = 26 \ //$$

<u>If we store the parent pointers:</u>

Argmin(DP(0)) = 0 One line

Argmin(DP(1)) = 2

Argmin(DP(2)) = 2 Another line

Argmin(DP(3)) = 4

Argmin(DP(4)) = 4 Another line

" Tushar'in' Roy likes'in' to code "

Perfect-information Blackjack

- Rules of the game:
  - Two players: dealer and the client
  - If the sum of the cards of one player goes over 21 that player loses the game
  - Each player has the choice to take another card (hit) or not (stand) so that he/she get closer to the score of 21 without going over it
  - The dealer must hit until the cards total 17 or more points
  - Players win by not busting (not going over 21) and having a total higher than the dealer's
  - The dealer loses by busting or having a total less than the player's hand that has not busted.
  - If the player and the dealer have the same total, this is called a "push" and the player typically does not win or lose money on that hand.
  -

Card	Value
2	2
3	3
4	4
5	5
6	6
7	7
8	8
9	9
10	10

Card	Value
Joker	10
Queen	10
King	10
Ace	1 or 11

- Now lets focus on the algorithm details:

  - Perfect information blackjack requires you to know in advance the sequence of the deck, i.e:

$$\text{deck} = c_0, c_1, \dots, c_{n-1} \quad (\text{for } \underline{n} \text{ cards})$$

- That is we are cheating.... There goes your idea of going to the casino...

- We will only be dealing with the case where we have 1 player vs. dealer

- \$1 bet / hand

- $\boxed{\text{Guess:}}$ How many time should the player hit?

- $\boxed{\text{Subproblems:}}$ Where does a new hand start?
  - ↳ Suffix $C[i:]$
    - $\#$ subproblems $= n$
  - $\#$ choices:
    - The first four cards of each hand are fixed
    - The we neen ~~#~~ to gess how many hits $\underline{i}$ are left
    - That gives us: $\underbrace{n-4-i}_{\#\text{hits passibilities}} \leq n$
    - $\#$ choices $\leq n$

Recurrence:

- We are trying to maximize the winnings \$, i.e.:
- For a hand starting at index i we want to maximize the winnings

"do we win / tie / lose after j hits?"

$$BJ(i) = \max\Big(\text{outcome}(j) \in \{-1, 0, 1\}$$

Player loses \$1 → -1; Player draws → 0; Player wins \$1 → 1

/* $j = i + 4 + \#\text{PlayerHits} + \#\text{DealerHits}$ */

$$+ \; BJ(j) \quad \text{for } \#\text{PlayerHits in range}(0, n)$$

if valid play (i.e. ≤ 21

$$\Big)$$

- Question: What is the running time? Not obvious...
  - # subproblems $= n$
  - # choices $= n$
  - For each choice we need to run the dealer strategy. I.e. How long do we need to compute the outcome?
    - If we assume a general max value that we need to count of 21, then this is constant time.
    - If we assume a general max value that we need to count proportional to n, then this is linear time, i.e. O(

- The total time is then:
  - $\underline{n}$ subproblems x $\underline{n}$ choices x 2·1 = $\Theta(n^2)$
  - $\underline{n}$ subproblems x $\underline{n}$ choices x $\underline{n}$ "outcomes" = $\Theta(n^3)$