Dynamic Programming

Hengfeng Wei

hfwei@nju.edu.cn

June 11 – June 19, 2017

What is DP?

DP $\approx$ " brute force"

DP $\approx$ "smart scheduling of subproblems"

DP $\approx$ "shortest/longest paths in some DAG"

What is DP?

DP $\approx$ "smarter brute force"

DP $\approx$ "smart scheduling of subproblems"

DP $\approx$ "shortest/longest paths in some DAG"

What is not DP?

Programming $=$ Planning

What is not DP?

$$\text{Programming} = \text{Planning}$$

$$\text{Programming} \neq \text{Coding}$$

(Richard Bellman, 1940s)

Steps for applying DP

1. Define subproblems
   - # of subproblems
2. Set the goal
3. Define the recurrence
   - larger subproblem $\leftarrow$ # smaller subproblems
   - init. conditions
4. Write pseudo-code: fill "table" in topo. order
5. Analyze Time/Space complexity
6. Extract the optimal sulution

Common subproblems in DP

1D subproblems:

Input: $x_1, x_2, \ldots, x_n$ (array, sequence, string)

Subproblems: $x_1, x_2, \ldots, x_i$ (prefix/suffix)

#: $\Theta(n)$

Common subproblems in DP

1D subproblems:

Input: $x_1, x_2, \ldots, x_n$ (array, sequence, string)

Subproblems: $x_1, x_2, \ldots, x_i$ (prefix/suffix)

#: $\Theta(n)$

Examples: Fib, Maximum-sum subarray, Longest increasing subsequence, Highway restaurants, Text justification

Common subproblems in DP

2D subproblems:

1. Input: $x_1, x_2, \ldots, x_m; \quad y_1, y_2, \ldots, y_n$

   Subproblems: $x_1, x_2, \ldots, x_i; \quad y_1, y_2, \ldots, y_j$

   #: $\Theta(mn)$

Common subproblems in DP

2D subproblems:

1. Input: $x_1, x_2, \ldots, x_m; \quad y_1, y_2, \ldots, y_n$

   Subproblems: $x_1, x_2, \ldots, x_i; \quad y_1, y_2, \ldots, y_j$

   #: $\Theta(mn)$

   Examples: Edit distance, Longest common subsequence

Common subproblems in DP

2D subproblems:

1. Input: $x_1, x_2, \ldots, x_m; \quad y_1, y_2, \ldots, y_n$

   Subproblems: $x_1, x_2, \ldots, x_i; \quad y_1, y_2, \ldots, y_j$

   #: $\Theta(mn)$

   Examples: Edit distance, Longest common subsequence

2. Input: $x_1, x_2, \ldots, x_n$

   Subproblems: $x_i, \ldots, x_j$

   #: $\Theta(n^2)$

Common subproblems in DP

2D subproblems:

1. Input: $x_1, x_2, \dots, x_m; \quad y_1, y_2, \dots, y_n$

   Subproblems: $x_1, x_2, \dots, x_i; \quad y_1, y_2, \dots, y_j$

   #: $\Theta(mn)$

   Examples: Edit distance, Longest common subsequence

2. Input: $x_1, x_2, \dots, x_n$

   Subproblems: $x_i, \dots, x_j$

   #: $\Theta(n^2)$

   Examples: Matrix chain multiplication, Optimal BST

Common subproblems in DP

3D subproblems:

- Floyd-Warshall algorithm

$$\mathsf{d}(i, j, k) = \min\{\mathsf{d}(i, j, k-1), \mathsf{d}(i, k, k-1) + \mathsf{d}(k, j, k-1)\}$$

Common subproblems in DP

3D subproblems:

- Floyd-Warshall algorithm

$$\mathsf{d}(i,j,k) = \min\{\mathsf{d}(i,j,k-1), \mathsf{d}(i,k,k-1) + \mathsf{d}(k,j,k-1)\}$$

DP on graphs:

1. On rooted tree

   Subproblems: rooted subtrees

2. On DAG

   Subproblems: nodes after/before in the topo. order

Common subproblems in DP

3D subproblems:

- Floyd-Warshall algorithm

$$\mathsf{d}(i, j, k) = \min\{\mathsf{d}(i, j, k-1), \mathsf{d}(i, k, k-1) + \mathsf{d}(k, j, k-1)\}$$

DP on graphs:

1. On rooted tree

   Subproblems: rooted subtrees

2. On DAG

   Subproblems: nodes after/before in the topo. order

Knapsack problem:

- Subset sum problem, change-making problem

Common subproblems in DP

And Others . . .

Recurrences in DP

Make choices by asking yourself the right question:

1. Binary choice
   - whether . . .
2. Multi-way choices
   - where to . . .
   - which one . . .

$f^{(S(n))} = 1$

$f^{(S(n))} = 1$ (Problem 7.2)

$$f(n) = \begin{cases} n - 1 & \text{if } n \in \mathbb{Z}^+ \\ n/2 & \text{if } n\%2 = 0 \\ n/3 & \text{if } n\%3 = 0 \end{cases}$$

$S(n)$: minimum number of steps taking n to 1.

$f^{(S(n))} = 1$

$f^{(S(n))} = 1$ (Problem 7.2)

$$f(n) = \begin{cases} n-1 & \text{if } n \in \mathbb{Z}^+ \\ n/2 & \text{if } n\%2 = 0 \\ n/3 & \text{if } n\%3 = 0 \end{cases}$$

$S(n)$: minimum number of steps taking n to 1.

$S(i)$: minimum number of steps taking i to 1

$f^{(S(n))} = 1$

$f^{(S(n))} = 1$ (Problem 7.2)

$$f(n) = \begin{cases} n-1 & \text{if } n \in \mathbb{Z}^+ \\ n/2 & \text{if } n\%2 = 0 \\ n/3 & \text{if } n\%3 = 0 \end{cases}$$

$S(n)$: minimum number of steps taking n to 1.

$S(i)$: minimum number of steps taking i to 1

$$S(i) = 1 + \min\{N(i-1), N(i/2)(\text{if } n\%2 = 0), N(i/3)(\text{if } n\%3 = 0)\}$$

$$S(1) = 0$$

$f^{(S(n))} = 1$

Collatz $(3n+1)$ conjecture:

$$f(n) = \begin{cases} n/2 & \text{if } n\%2 = 0 \\ 3n+1 & \text{if } n\%2 = 1 \end{cases}$$

$$f^*(n) = 1?$$

$f^{(S(n))} = 1$

Collatz $(3n+1)$ conjecture:

$$f(n) = \begin{cases} n/2 & \text{if } n\%2 = 0 \\ 3n+1 & \text{if } n\%2 = 1 \end{cases}$$

$$f^*(n) = 1?$$

"Mathematics may not be ready for such problems."

— Paul Erdős

Longest increasing subsequence

Longest increasing subsequence (Problem 7.3)

- Given an integer array $A[1 \dots n]$
- To find (the length of) a longest increasing subseqence.

$$5, 2, 8, 6, 3, 6, 9, 7 \implies 2, 3, 6, 9$$

Longest increasing subsequence

Subproblem: $L(i)$: the length of the LIS of $A[1 \ldots i]$

Goal: $L(n)$

Longest increasing subsequence

Subproblem: $L(i)$: the length of the LIS of $A[1 \dots i]$

Goal: $L(n)$

Make choice: whether $A[i] \in LIS[1 \dots i]$?

Recurrence:

$$L(i) = \max\{L(i-1), 1 + \max_{j<i \wedge A[j] \leq A[i]} L(j)\}$$

Longest increasing subsequence

Subproblem: $L(i)$: the length of the LIS of $A[1 \ldots i]$

Goal: $L(n)$

Make choice: whether $A[i] \in LIS[1 \ldots i]$?

Recurrence:

$$L(i) = \max\{L(i-1), 1 + \max_{j < i \wedge A[j] \leq A[i]} L(j)\}$$

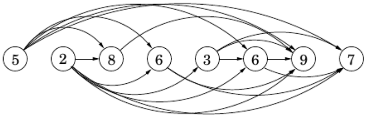
Init:

$$L(0) = 0$$

Time:

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

Longest increasing subsequence



Longest path distance in the DAG!

Maximum-sum subarray

Maximum-sum subarray (Google Interview)

- Array $A[1 \cdots n], a_i >=< 0$
- To find (the sum of) a maximum-sum subarray of A
  - mss $= 0$ if all negative

$$A[-2, 1, -3, 4, -1, 2, 1, -5, 4] \implies [4, -1, 2, 1]$$

Maximum-sum subarray

Maximum-sum subarray (Google Interview)

- Array $A[1 \cdots n], a_i >=< 0$
- To find (the sum of) a maximum-sum subarray of A
  - mss $= 0$ if all negative

$$A[-2, 1, -3, 4, -1, 2, 1, -5, 4] \implies [4, -1, 2, 1]$$

Subproblem: MSS$[i]$: the sum of the MS$[i]$ of $A[1 \cdots i]$

Goal: mss $=$ MSS$[n]$

Maximum-sum subarray

Maximum-sum subarray (Google Interview)

- Array $A[1 \cdots n], a_i >=< 0$
- To find (the sum of) a maximum-sum subarray of A
  - mss $= 0$ if all negative

$$A[-2, 1, -3, 4, -1, 2, 1, -5, 4] \implies [4, -1, 2, 1]$$

Subproblem: MSS$[i]$: the sum of the MS$[i]$ of $A[1 \cdots i]$

Goal: mss $=$ MSS$[n]$

Make choice: Is $a_i \in$ MS$[i]$?

Recurrence:

$$\text{MSS}[i] = \max\{\text{MSS}[i-1], ???\}$$

Maximum-sum subarray

Subproblem: MSS$[i]$: the sum of the MS$[i]$ *ending with* a_i or 0

Goal: mss $= \max_{1 \le i \le n}$ MSS$[i]$

Maximum-sum subarray

Subproblem: MSS$[i]$: the sum of the MS$[i]$ *ending with* a_i or 0

Goal: mss $= \max_{1 \le i \le n}$ MSS$[i]$

Make choice: where does the MS$[i]$ start?

Recurrence:

$$\text{MSS}[i] = \max\{\text{MSS}[i-1] + a_i, 0\} \text{ (prove it!)}$$

Maximum-sum subarray

Subproblem: MSS$[i]$: the sum of the MS$[i]$ *ending with* a_i or 0

Goal: mss $= \max_{1 \le i \le n}$ MSS$[i]$

Make choice: where does the MS$[i]$ start?

Recurrence:

$$\text{MSS}[i] = \max\{\text{MSS}[i-1] + a_i, 0\}$$ (prove it!)

Init:

$$\text{MSS}[0] = 0$$

Maximum-sum subarray

Subproblem: MSS$[i]$: the sum of the MS$[i]$ *ending with* a_i or 0

Goal: mss $= \max_{1 \le i \le n}$ MSS$[i]$

Make choice: where does the MS$[i]$ start?

Recurrence:

$$\text{MSS}[i] = \max\{\text{MSS}[i-1] + a_i, 0\} \text{ (prove it!)}$$

Init:

$$\text{MSS}[0] = 0$$

Time: $\Theta(n)$

Maximum-sum subarray

Maximum-product subarray

Maximum-product subarray (Problem 7.4)

Reconstructing string

Reconstructing string (Problem 7.9)

- String $S[1 \cdots n]$
- Dict for *lookup*:

$$\text{dict}(w) = \begin{cases} \text{true} & \text{if } w \text{ is a valid word} \\ \text{false} & \text{o.w.} \end{cases}$$

- Is $S[1 \cdots n]$ valid (reconstructed as a sequence of valid words)?

Reconstructing string

Subproblem: $V[i]$: is $S[1 \cdots i]$ valid?

Goal: $V[n]$

Reconstructing string

Subproblem: $V[i]$: is $S[1 \cdots i]$ valid?

Goal: $V[n]$

Make choice: where does the last word start?

Recurrence:

$$V[i] = \bigvee_{j=1\dots i} (V[j-1] \wedge \mathsf{dict}(S[j \cdots i]))$$

Reconstructing string

Subproblem: $V[i]$: is $S[1 \cdots i]$ valid?

Goal: $V[n]$

Make choice: where does the last word start?

Recurrence:

$$V[i] = \bigvee_{j=1 \dots i} \left(V[j-1] \wedge \text{dict}(S[j \cdots i]) \right)$$

Init:

$$V[0] = \text{true}$$

Time: $O(n^2)$

Hotel along a trip

Hotel along a trip (Problem 7.15)

- Hotel sequence (distance): $a_0 = 0, a_1, \cdots, a_n$
- $a_0 \rightsquigarrow a_n$
- Stop at only hotels
- Cost: $(200 - x)^2$
- To minimize overall cost

Hotel along a trip

Subproblem: $C[i]$: minimum cost when the destination is a_i

Goal: $C[n]$

Hotel along a trip

Subproblem: $C[i]$: minimum cost when the destination is a_i

Goal: $C[n]$

Make choice: what is the last but one hotel a_j to stop?

Recurrence:

$$C[i] = \min_{0 \le j < i} \{C[j] + (200 - (a_i - a_j))^2\}$$

Hotel along a trip

Subproblem: $C[i]$: minimum cost when the destination is a_i

Goal: $C[n]$

Make choice: what is the last but one hotel a_j to stop?

Recurrence:

$$C[i] = \min_{0 \le j < i} \{C[j] + (200 - (a_i - a_j))^2\}$$

Init:

$$C[0] = 0$$

Time:

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

Highway restaurants

Highway restaurants (Problem 7.16)

- Locations: $L[1 \dots n]$
- Profits: $P[1 \dots n]$
- Any two hotels should be $\geq k$ miles apart
- To maximize the total profit

Subproblem:

Goal:

Make choice:

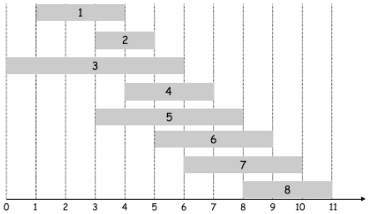
Recurrence:

Time:

Weighted interval/class scheduling

Weighted interval/class scheduling (Problem 7.14)

- Classes: $\mathcal{C} = \{c_1, c_2, \cdots, c_n\}$ $\quad c_i \triangleq \langle g_i, s_i, f_i \rangle$
- Choosing non-conflicting classes to maximize your grades



sort $\mathcal{C}$ by finishing time.

Weighted interval/class scheduling

Greedy algorithms by finishing time or weights fail.

Weighted interval/class scheduling

Subproblem: $G[i]$: the maximal grades obtained from $\{c_1, c_2, \cdots, c_i\}$

Goal: $G[n]$

Weighted interval/class scheduling

Subproblem: $G[i]$: the maximal grades obtained from $\{c_1, c_2, \cdots, c_i\}$

Goal: $G[n]$

Make choice: choose c_i or not in $G[i]$?

Recurrence:

$$G[i] = \max\{G[i-1], G[p(i)] + g_i\}$$

$p(i)$: the largest index $j < i$ *s.t.* c_i and c_j are disjoint

Weighted interval/class scheduling

Subproblem: $G[i]$: the maximal grades obtained from $\{c_1, c_2, \cdots, c_i\}$

Goal: $G[n]$

Make choice: choose c_i or not in $G[i]$?

Recurrence:

$$G[i] = \max\{G[i-1], G[p(i)] + g_i\}$$

$p(i)$: the largest index $j < i$ *s.t.* c_i and c_j are disjoint

Init:

$$G[0] = 0$$

Weighted interval/class scheduling

Subproblem: $G[i]$: the maximal grades obtained from $\{c_1, c_2, \cdots, c_i\}$

Goal: $G[n]$

Make choice: choose c_i or not in $G[i]$?

Recurrence:

$$G[i] = \max\{G[i-1], G[p(i)] + g_i\}$$

$p(i)$: the largest index $j < i$ *s.t.* c_i and c_j are disjoint

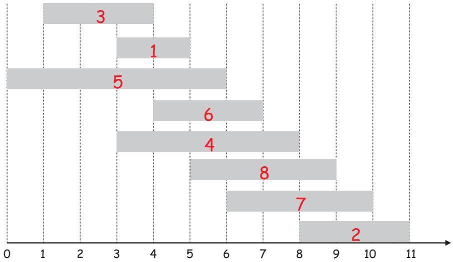
Init:

$$G[0] = 0$$

Time: $O(n \log n) + T(p(i)) + O(n) \cdot O(1)$

Weighted interval/class scheduling

Why is ordering necessary?

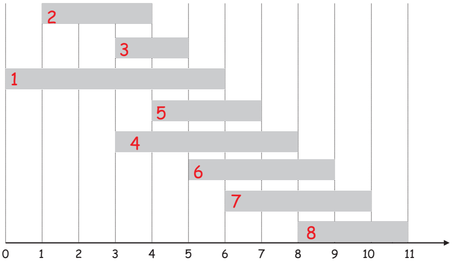


$$G[7] = \max\{G[6], G[\{1,3,5\}] + g_7\}$$

subproblems changed: all $O(2^n)$ subsets

Weighted interval/class scheduling

What about sorting by starting time?



$$G[6] = \max\{G[5], G[\{2,3\}] + g_6\}$$

subproblems changed: all $O(2^n)$ subsets

Subproblem:

Goal:

Make choice:

Recurrence:

Initialization:

Time:

Dynamic Programming

Dynamic Programming

Dynamic Programming


THANK YOU