Dynamic Programming

Hengfeng Wei

hengxin0912@gmail.com

June 13, 2016

Dynamic Programming

Dynamic Programming

Q: What is DP?

- A: Smart scheduling of subproblems.

Q: What does DP look like?

1. Define subproblems (types)
2. Set the goal: what is the solution to the original problem
3. Define recurrence: (ask the right questions $\Rightarrow$ reduce to subproblems)
   - larger problem $\Leftarrow$ a number of "smaller" subproblems
4. Write pseudo-code (fill the array/table/matrix in order)
5. Analyze time complexity
6. Extract optimal solutions

Common subproblems

1. 1-D subproblems
   - input: $x_1, x_2, \cdots, x_n$ (array, sequence, string)
   - subproblems: $x_1, x_2, \cdots, x_i$ (prefix/postfix)
   - $\# = O(n)$
   - examples: max-subarray sum, highway restaurants, breaking into lines
2. 2-D subproblems

   2.1 input: $x_1, x_2, \cdots, x_m; y_1, y_2, \cdots, y_n$
   - subproblems: $x_1, x_2, \cdots, x_i; y_1, y_2, \cdots, y_j$
   - $\# = O(mn)$
   - examples: edit distance

   2.2 input: $x_1, x_2, \cdots, x_n$
   - subproblems: $x_i, \cdots, x_j$
   - $\# = (n^2)$
   - examples: multiplying a sequence of matrices, optimal binary search tree

Common subproblems

3. 3-D subproblems:
   - example: Floyd-Warshall algorithm, Bellman-Ford algorithm
4. tree structure subproblems:
   - input: rooted tree
   - subproblems: rooted subtree
   - vertex cover in tree
5. knapsack problem
   - example: changing coins

Dynamic Programming

1-D DP

Maximal sum subarray [Problem: 2.2.3, 2.2.13, Google Interview Problem]

- array $A[1 \cdots n], a_i >=< 0$
- to find (the sum of) an MSS in A (case: sum $= 0$)

Remark.

- try MSUM$[i]$: the sum of the MSS in $A[1 \cdots i]$
- goal: is $a_i \in$ MSUM$[i]$?
- MSUM$[i] = \max\{$MSUM$[i-1], ?\}$

1-D DP

Solution.

- subproblem MSUM$[i]$: the sum of the MSS *ending with* a_i
- goal: $\max_{1 \le i \le n}$ MSUM$[i]$
- question: what is the relation between MSUM$[i-1]$ and MSUM$[i]$?
- recurrence:

$$\text{MSUM}[i] = \begin{cases} \text{MSUM}[i-1]\text{(why?)} + a_i & \text{if } a_i > 0 \\ \max\{\text{MSUM}[i-1] + a_i, 0\} & \text{if } a_i < 0 \end{cases}$$

$$\text{MSUM}[i] = \max\{\text{MSUM}[i-1] + a_i, 0\}$$

- initialization: MSUM$[0] = 0$
- code

1-D DP

Weighted interval/class scheduling [Problem: 2.2.20]

- $\mathcal{C} = \{c_1, c_2, \cdots, c_n\}$
- c_i: grade g_i
- c_i: s_i, f_i; conflict
- choosing pairwise non-conflicting classes to maximize your grades

Solution.

- subproblem $G[i]$: the maximal grades obtained from $\{c_1, c_2, \cdots, c_i\}$
- goal: $G[n]$
- question: choose c_i or not in $G[i]$? (binary chioce)
- recurrence: $G[i] = \max\{G_{i-1}, G[j] + g_i\}$
  c_j: the last class which does not conflict with c_i

1-D DP

Reconstructing document [Problem: 2.2.14]

- string $S[1 \cdots n]$
- dict for *lookup*: dict(w)
- is $S[1 \cdots n]$ valid (reconstructed as a sequence of valid words)

Solution.

- subproblem $V[i]$: is $S[1 \cdots i]$ valid?
- goal: $V[n]$
- question: where does the last word start if S is valid? (multi-way choices)
- recurrence: $V[i] = \bigvee_{j=1 \ldots i} (\text{dict}(S[j \cdots i]) \wedge V[j])$
- initialization: $V[0] = \text{true}$

1-D DP

A trip through hotels [Problem: 2.2.21]

- hotel sequence (distance): $a_0 = 0, a_1, \cdots, a_n$
- stop at only hotels
- cost: $(200 - x)^2$
- to minimize overall cost

Solution.

- $C[i]$: minimum cost when the destination is a_i
- goal: $C[n]$
- question: what is the last but one hotel a_j to stop in the optimal solution?
- recurrence: $C[i] = \min_{0 \le j < i} \{C[j] + (200 - (a_i - a_j))^2\}$
- initialization: $C[0] = 0$

Dynamic Programming

2-D DP (part 1)

LCS: longest common subsequence [Problem: 2.2.7]

- $X = \{x_1 \cdots x_m\}; Y = \{y_1 \cdots y_n\}$
- find (the length of) a LCS of X and Y

Solution.

- subproblem: $L[i, j]$: the length of a LCS of $X[1 \cdots i]$ and $Y[1 \cdots j]$
- goal: $L[m, n]$
- question: is $X_i = Y_j$?
- recurrence:

$$L[i,j] = \begin{cases} L[i-1, j-1] + 1 & \text{if } X_i = Y_j \\ \max\{L[i-1, j], L[i, j-1]\} & \text{if } X_i \neq Y_j \end{cases}$$

- initialization: $L[0, j] = 0, L[i, 0] = 0 (0 \leq i \leq m, 0 \leq j \leq m)$

2-D DP (part 1)

Remarks.

1. why is $L[i-1, j-1] + 1$ if $X_i = Y_j$?

Proof.

- $Z_k = X_i = Y_j$
- X_i or a previous one; Y_j or a previous one

□

2. why is $\max\{\cdot\}$ if $X_i \neq Y_j$?

Theorem

If $X_i \neq Y_j$, then either $X_i \notin LCS[i,j]$ or $Y_j \notin LCS[i,j]$.

Proof.

By contradiction.

□

2-D DP (part 1)

LCS with repetition of X_i [Problem: 2.2.8]

1. repetition of X_i
2. k-bounded repetition of X_i

Solution.

1. repetition of x_i

$$L[i,j] = \begin{cases} L[i, j-1] + 1 & \text{if } X_i = Y_j \\ \max\{L[i-1,j], L[i,j-1]\} & \text{if } X_i \neq Y_j \end{cases}$$

2. k-bounded repetition of X_i
   $X^{(k)} = X_1^{(k)} \cdots X_m^{(k)}$

2-D DP (part 1)

Shortest common supersequence [Problem: 2.2.10]

- $X = \{x_1 \cdots x_m\}; Y = \{y_1 \cdots y_n\}$
- to find (the length of) a SCS of X and Y

Solution.

- subproblem $L[i, j]$: the length of an SCS of $X[1 \cdots i]$ and $Y[1 \cdots j]$
- goal: $L[m, n]$
- question: is $X_i = Y_j$
- recurrence:

$$L[i, j] = \begin{cases} L[i-1, j-1] + 1 & \text{if } X_i = Y_j \\ \max\{L[i-1, j] + 1, L[i, j-1] + 1\} & \text{if } X_i \neq Y_j \end{cases}$$

Remark

2-D DP (part 1)

Edit distance revisited

2-D DP (part 1)

Shuffle of strings [Problem: 2.2.12]

- $X[1 \cdots m]; Y[1 \cdots n], Z[1 \cdots m+n]$
- is Z a shuffle of X and Y?

Solution.

- $S[i,j]$: Is $Z[1 \cdots i+j]$ a shuffle of $X[1 \cdots i]$ and $Y[1 \cdots j]$?
- goal: $S[m,n]$
- question: what is the relationship among $x_i, y_j,$ and z_{i+j}?
- recurrence:

$$S[i,j] = \begin{cases} \text{false} & \text{if } Z_{i+j} \neq X_i \wedge Z_{i+j} \neq Y_j \\ S[i-1,j] & \text{if } Z_{i+j} = X_i \wedge Z_{i+j} \neq Y_j \\ S[i,j-1] & \text{if } Z_{i+j} \neq X_i \wedge Z_{i+j} = Y_j \\ S[i-1,j] \vee S[i,j-1] & \text{if } Z_{i+j} = X_i = Y_j \end{cases}$$

2-D DP (part 1)

Solution.

- initialization:
  $S[0,0] =$ true
  $S[0,j] = \text{true if } Y = Z, S[i,0] = \text{true if } X = Z$

2-D DP (part 2)

2-D DP

Longest contiguous substring both forward and backward [Problem: 2.2.9]

- string $T[1 \cdots n]$
- to find ...

Solution.

- try $L[i]$; $L[i, j]$: the length of LCS in $T[i \cdots j]$
- subproblem $L[i, j]$: the length of LCS starting with T_i and ending with T_j
- goal: $\max_{1 \le i \le j \le n} L[i, j]$
- $O(n^3 \Rightarrow n^2)$
- question: is $T_i = T_j$?

2-D DP (part 2)

Solution.

- recurrence:

$$L[i,j] = \begin{cases} \text{false} & \text{if } Z_{i+j} \neq X_i \wedge Z_{i+j} \neq Y_j \\ L[i+1, j-1] + 1 & \text{if } T_i = T_j \\ 0 & \text{if } T_i \neq T_j \end{cases}$$

- initialization:

$$L[i,i] = 0, 0 \leq i \leq n$$

$$L[i, i+1] = \begin{cases} 1 & \text{if } T_i = T_{i+1} \\ 0 & \text{if } T_i \neq T_{i+1} \end{cases}$$

- three ways of filling the table

2-D DP (part 2)

Longest subsequence palindrome [Problem: 2.2.15 (a)]

- string $S[1 \cdots n]$
- to find (the length of) a longest subsequence palindrome

Solution.

- subproblem $L[i, j]$: the length of the LSP in $S[i \cdots j]$
- goal: $L[1, n]$
- question: is $S[i] = S[j]$?
- recurrence

2-D DP (part 2)

Longest subsequence palindrome [Problem: 2.2.15 (b)]

- string $S[1 \cdots n]$
- decompose into a sequence of palindromes

Solution.

- Num$[i, j]$: the minimum number of palindromes obtained from $S[i \cdots j]$
- subproblem MinPals$[i]$: the minimum number of palindromes obtained from $S[1 \cdots i]$
- goal: MinPals$[n]$
- question: what is the start index of the last palindrome?
- recurrence:

2-D DP (part 2)

String split problem [Problem: 2.2.16]

- split a string S into many pieces
- cost $|S| = n \Rightarrow O(n)$
- given locations of m cuts: $C_0, C_1, \cdots, C_m, C_{m+1}$
- to find the minimum cost of splitting the string into $m+1$ pieces $S_0 \cdots S_m$

Solution.

- subproblem: $\text{MinCost}[i, j]$: the minimum cost of breaking the string $S_i \cdots S_{j-1}$
- goal: $\text{MinCost}[0, m+1]$
- question: what is the "first"/"root" cut between $Ci \cdots C_{j-1}$
- recurrence

Dynamic Programming

Dynamic Programming

Dynamic Programming


THANK YOU