Dynamic Programming

Hengfeng Wei

hengxin0912@gmail.com

June 16, 2016

1 Overview

2 1-D DP

3 2-D DP

4 3-D DP

5 DP on Graphs

6 The Knapsack Problem

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. DP on graphs (tree, DAG ...)
   - input: rooted tree
   - subproblems: rooted subtree
5. knapsack problem
   - example: changing coins
6. others ...

Dynamic Programming

1. Overview
2. 1-D DP
3. 2-D DP
4. 3-D DP
5. DP on Graphs
6. The Knapsack Problem

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
  - special case: mss $= 0$ if all negative

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

Trial and error.

- try subproblem MSS$[i]$: the sum of the MS (MS[i]) in $A[1 \cdots i]$
- goal: mss $=$ MSS$[n]$
- question: Is $a_i \in$ MS$[i]$?
- recurrence:

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

1-D DP

Solution.

- subproblem $\mathsf{MSS}[i]$: the sum of the MSS *ending with* a_i or 0
- goal: mss $= \max_{1 \le i \le n} \mathsf{MSS}[i]$
- question: where does the $\mathsf{MS}[i]$ start?
- recurrence:

$$\mathsf{MSS}[i] = \max\{\mathsf{MSS}[i-1] + a_i, 0\}$$

- initialization: $\mathsf{MSS}[0] = 0$

1-D DP

Code.

```
MSS[0] = 0
For i = 1 to n
  MSS[i] = max{MSS[i-1] + A[i], 0}
return max_{i = 1 to n} MSS[i]
```

Simpler code.

```
mss = 0
MSS = 0
For i = 1 to n
  MSS = max{MSS + A[i], 0}
  mss = max{mss, MSS}
return mss
```

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*:

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

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

Solution.

- subproblem $V[i]$: is $S[1 \cdots i]$ valid?
- goal: $V[n]$

1-D DP

Solution.

- question: where does the last word start? (multi-way choices)
- recurrence:

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

- initialization: $V[0] = \text{true}$

1-D DP

$$V[i] = \begin{cases} 0 & i = 0 \\ \bigvee_{j=1\ldots i}(V[j-1] \wedge \mathsf{dict}(S[j\cdots i])) & 1 \leq i \leq n \end{cases}$$

Code.

```
V[0] = true
For i = 1 to n
  For j = 1 to i
    V[i] = V[j-1] and Dict(S[j...i])
return V[n]
```

1 Overview

2 1-D DP

3 2-D DP
- 2-D DP (part 1)
- 2-D DP (part 2)

4 3-D DP

5 DP on Graphs

6 The Knapsack Problem

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

$$X = \langle A, B, C, B, D, A, B \rangle$$
$$Y = \langle B, D, C, A, B, A \rangle$$
$$Z = \langle B, C, B, A \rangle$$

Solution.

- subproblem: $L[i, j]$: the length of a LCS of $X[1 \cdots i]$ and $Y[1 \cdots j]$
- goal: $L[m, n]$

2-D DP (part 1)

Solution.

- 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:

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

2-D DP (part 1)

Counterexample?

$$L[i,j] = L[i-1,j-1] + 1 \text{ if } X_i = Y_j$$

$$X = a, b, c, c, c \qquad X = a, b, c, c, c$$

$$Y = a, b, c \qquad Y = a, b, c$$

$$Z = a, b, c \qquad Z = a, b, c$$

2-D DP (part 1)

Correctness proof (I).

Theorem

$L[i,j] = L[i-1, j-1] + 1$ *if* $X_i = Y_j$.

Theorem

$\exists$ *a LCS* $Z[1 \cdots k]$ *of* $X[1 \cdots i]$ *and* $Y[1 \cdots j]$: $Z_k \equiv X_i \wedge Z_k \equiv Y_j$.

Proof.

- $Z_k = X_i = Y_j$ (by contradiction)
- But, $Z_k = X_i \not\Rightarrow Z_k \equiv X_i; Z_k = Y_i \not\Rightarrow Z_k \equiv Y_i$
- $Z_k = X_i = Y_j \Rightarrow$ either $Z_k \equiv X_i$ or $Z_k \equiv Y_j$ (by contradiction)
  1. $Z_k \equiv X_i \wedge Z_k \equiv Y_j$
  2. $Z_k \not\equiv X_i \wedge Z_k \equiv Y_j$
  3. $Z_k \equiv X_i \wedge Z_k \not\equiv Y_j$

2-D DP (part 1)

Correctness proof (II).

Theorem

$L[i,j] = \max\{L[i-1,j], L[i,j-1]\}$ *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)

Edit distance revisited

$$\text{ED}[i,j] = \min \begin{cases} \text{ED}[i-1,j]+1 \\ \text{ED}[i,j-1]+1 \\ \text{ED}[i-1,j-1]+\mathbb{I}\{X_i = Y_j\} \end{cases}$$

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

Theorem

If $X_i = Y_j$, then $ED[i-1,j-1] \leq \min\{ED[i-1,j]+1, ED[i,j-1]+1\}$.

2-D DP (part 2)

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

- string $T[1 \cdots n]$
- to find LCS both forward and backward

dynamicprogrammingmanytimes

Trial.

- try subproblem $L[i]$: the length of a LCS in $T[1 \cdots i]$
- try subproblem $L[i, j]$: the length of a LCS in $T[i \cdots j]$

2-D DP (part 2)

Solution.

- $L[i,j]$: the length of a 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 $T_i = T_j$?
- recurrence:

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

- initialization:

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

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

2-D DP (part 2)

Code: three ways of filling the table.

```
for d = 2 to n-1
  for i = 1 to n-d
    j = i + d
    ...
return max_{1 <= i <= j <= n} L[i,j]
```

```
for i = n-2 to 1
  for j = i+2 to n
    ...
return ...
```

```
for j = 3 to n
  for i = j-2 to 1
    ...
return ...
```

2-D DP (part 2)

String split problem [Problem: 2.2.16]

- split a string S into many pieces
- cost $|S| = n \Rightarrow n$
- 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 $S_i \cdots S_{j-1}$ using cuts $C_{i+1} \cdots C_{j-1}$
- goal: $\text{MinCost}[0, m+1]$

2-D DP (part 2)

Solution.

- question: what is the first cut in $C_{i+1} \cdots C_{j-1}$?
- recurrence:

$$\text{MinCost}[i, j] = \min_{i<k<j} \left(\text{MinCost}[i, k] + \text{MinCost}[k, j] + l(S_i \cdots S_{j-1}) \right)$$

- initialization:

$$\text{MinCost}[i, i+1] = 0$$

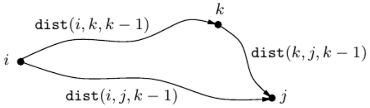
Dynamic Programming

1. Overview
2. 1-D DP
3. 2-D DP
4. 3-D DP
5. DP on Graphs
6. The Knapsack Problem

3-D DP

Floyd-Warshall algorithm

- subproblem dist$[i, j, k]$: the length of the shortest path from i to j via only nodes $v_1 \cdots v_k$
- goal: dist$[i, j, n], \forall i, j$
- question: Is v_k in ShortestPath$[i, j, k]$?
- recurrence:



$$\text{dist}[i, j, k] = \min\{\text{dist}[i, j, k-1], \text{dist}[i, k, k-1] + \text{dist}[k, j, k-1]\}$$

Dynamic Programming

1. Overview
2. 1-D DP
3. 2-D DP
4. 3-D DP
5. DP on Graphs
6. The Knapsack Problem

DP on graphs

Minimum vertex cover [Problem: 2.2.18]

- tree T
- compute (the size of) a minimum vertex cover of T

Solution.

- rooted T at r
- subproblem $I(u)$: the size of a MVC of T_u subtree
- goal: $I(r)$
- question: Is u in MVC$[u]$?
- recurrence:

$$I(u) = \max\{|\text{children of } u| + \sum_{v:\text{grandchildren of } u} I(v), 1 + \sum_{v:\text{children of } u} I(v)\}$$

DP on graphs

Solution.

- initialization:

$$I(u) = 0, \text{if } u \text{ is a leave}$$

Code.

```
DFS on T from root r:
  when u is ``finished'':
    I(u) = 0, if u is a leave
    I(u) = ..., otherwise
```

DP on graphs

Shortest paths in dags

- dag $G = (V, E, w)$
- $s \in V$
- compute shortest paths from s to all t

Solution.

- subproblem $\text{dist}[v]$: shortest distance from s to v
- goal: all $\text{dist}[v]$
- question: What is the relation between $\text{dist}[v]$ and $\text{dist}[u]$ of its predecessors u?
- recurrence:

$$\text{dist}[v] = \min_{u \to v} \left(\text{dist}[u] + w(u \to v) \right)$$

DP on graphs

Code.

```
dist[s] = 0
dist[v] = infty for others

for v != s in linearized order
  dist[v] = min_{u -> v} dist[u] + w(u \to v)
```

Remarks.

1. longest path
2. negative edges

1. Overview
2. 1-D DP
3. 2-D DP
4. 3-D DP
5. DP on Graphs
6. The Knapsack Problem

The knapsack problem

The changing-making problem [Problem: 2.2.17 (b), 2.2.4 (subset sum)]

- coins values: $x_1 \ldots x_n$
- amount: v
- possible to make change for v?
- without repetition

Trial and error.

- subproblem $C[i]$: is it possible to make change for v using only $x_1 \cdots x_n$
- goal: $C[n]$
- question: using x_i or not?
- recurrence:

$$C[i] = C[i-1] \vee ???$$

The knapsack problem

Solution.

- subproblem $C[i, w]$: is it possible to make change for w using only $x_1 \dots x_n$
- goal: $C[n, v]$
- question: using x_i or not?
- recurrence:

$$C[i, w] = C[i-1] \vee (C[i-1, v-w] \wedge v \geq w)$$

- initialization:

$$C[i, 0] = \text{true}$$
$$C[0, w] = \text{false, if } w > 0$$
$$C[0, 0] = \text{true}$$

The knapsack problem

The changing-making problem [Problem: 2.2.17 (a)]

- coins values: $x_1 \dots x_n$
- amount: v
- possible to make change for v?
- unbounded repetition

Solution.

- subproblem $C[i, w]$: is it possible to make change for w using only $x_1 \dots x_n$
- goal: $C[n, v]$
- question: using x_i or not?
- recurrence:

$$C[i, w] = C[i-1] \vee (C[i, w - x_i] \wedge w \geq x_i)$$

The knapsack problem

The changing-making problem [Problem: 2.2.17 (c)]

- coins values: $x_1 \ldots x_n$
- amount: v
- possible to make change for v?
- $\leq k$-coins

Solution.

- subproblem $C[i, w, l]$: is it possible to make change for w with $\leq l$ coins of $x_1 \ldots x_i$
- goal: $C[n, v, k]$

The knapsack problem

Solution.

- question: using x_i or not?
- recurrence:

$$C[i,w,l] = C[i-1,w,l] \vee (C[i,w-x_i,l-1] \wedge w \geq x_i)$$

- initialization:

$$\begin{aligned} C[0,0,l] &= \text{true} \\ C[0,w,l] &= \text{false, if } w > 0 \\ C[i,0,l] &= \text{true} \\ C[i,w,0] &= \text{false, if } w > 0 \end{aligned}$$

https://github.com/hengxin/algorithm-ta-tutorial.git