Dynamic Programming

Hengfeng Wei

hfwei@nju.edu.cn

June 22, 2017

我走过最长的路就是你的套路

Steps for applying DP:

Steps for applying DP:

(1) Define subproblems
  - # of subproblems

(2) Set the goal

(3) Define the recurrence
  - larger subproblem $\leftarrow$ # smaller subproblems
  - init. conditions

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 some order

(5) Analyze the time complexity

(6) Extract the optimal solution (optionally)

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: Maximum-sum subarray, Longest increasing subsequence, Text justification (LaTeX)

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: $x_i, \ldots, x_j$
   #: $\Theta(n^2)$
   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

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 ...

1D DP

$f^{(S(n))} = 1$ (Problem 14.3)

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

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

$f^{(S(n))} = 1$ (Problem 14.3)

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

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

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

$f^{(S(n))} = 1$ (Problem 14.3)

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

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

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

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

$$S(1) = 0$$

Longest Increasing Subsequence (Problem 14.4)

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

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

Subproblem: $L(i)$: the length of the LIS ending with $A[i]$

Goal: $\max_i L(i)$

Subproblem: $L(i)$: the length of the LIS ending with $A[i]$

Goal: $\max_i L(i)$

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

Recurrence:

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

Subproblem: $L(i)$: the length of the LIS ending with $A[i]$

Goal: $\max_i L(i)$

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:

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

2D DP

LCS: Longest Common Subsequence (Problem 14.6)

$$X = X_1 \cdots X_m \quad Y = Y_1 \cdots Y_n$$

(1) Find (the length of) an LCS of X and Y

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

Subproblem: $L[i, j]$: the length of an LCS of $X[1 \cdots i]$ and $Y[1 \cdots j]$

Goal: $L[m, n]$

Subproblem: $L[i,j]$: the length of an LCS of $X[1 \cdots i]$ and $Y[1 \cdots j]$

Goal: $L[m,n]$

Make choice: Is $X_i = Y_j$?

Recurrence: (Proof!)

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

Subproblem: $L[i, j]$: the length of an LCS of $X[1 \cdots i]$ and $Y[1 \cdots j]$

Goal: $L[m, n]$

Make choice: Is $X_i = Y_j$?

Recurrence: (Proof!)

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

Init:

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

Time: $\Theta(mn)$

Longest Common Subsequence (Problem 14.6)

$$X = X_1 \cdots X_m \quad Y = Y_1 \cdots Y_n$$

(2) Allowing repetition of X

(3) Allowing repetition $\leq k$ of X

Longest Common Subsequence (Problem 14.6)

$$X = X_1 \cdots X_m \quad Y = Y_1 \cdots Y_n$$

(2) Allowing repetition of X

(3) Allowing repetition $\leq k$ of X

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

Longest Common Subsequence (Problem 14.6)

$$X = X_1 \cdots X_m \quad Y = Y_1 \cdots Y_n$$

(2) Allowing repetition of X

(3) Allowing repetition $\leq k$ of X

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

$$X \implies X^{(k)} \triangleq X_1^{(k)} \cdots X_m^{(k)}$$

Longest Contiguous Substring Both Forward and Backward (Problem 14.7)

- String $T[1 \cdots n]$
- Find a longest contiguous substring (LCS) both forward and backward

dynamicprogrammingmanytimes

- Subproblem $L[i]$: the length of an LCS in $T[1 \cdots i]$
- Subproblem $L[i, j]$: the length of an LCS in $T[i \cdots j]$

Subproblem: $L[i, j]$: the length of an LCS starting with T_i and ending with T_j

Goal: $\max_{1 \leq i \leq j \leq n} L[i, j]$

Subproblem: $L[i, j]$: the length of an LCS starting with T_i and ending with T_j

Goal: $\max_{1 \le i \le j \le n} L[i, j]$

Make choice: Is $T_i = T_j$?

Recurrence:

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

Subproblem: $L[i, j]$: the length of an LCS starting with T_i and ending with T_j

Goal: $\max\limits_{1 \le i \le j \le n} L[i, j]$

Make choice: Is $T_i = T_j$?

Recurrence:

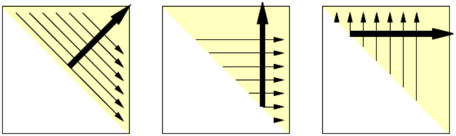
$$L[i, j] = \begin{cases} 0 & \text{if } T_i \neq T_j \\ L[i+1, j-1] + 1 & \text{if } T_i = T_j \end{cases}$$

Init:

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

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

Code: three ways of filling the table



for all $d \leftarrow 2 \ldots n-1$ **do**
    for all $i \leftarrow 1 \ldots n-d$ **do**
        $j \leftarrow i + d$
        $\ldots$
return $\max_{1 \le i \le j \le n} L[i,j]$

Longest Palindrome Subsequence (Problem 14.11 (1))

(1) Find (the length of) a longest palindrome subsequence of $S[1 \cdots n]$

Subproblem: $L[i, j]$: the length of an LSP of $S[i \cdots j]$

Goal: $L[1, n]$

Longest Palindrome Subsequence (Problem 14.11 (1))

(1) Find (the length of) a longest palindrome subsequence of $S[1 \cdots n]$

Subproblem: $L[i, j]$: the length of an LSP of $S[i \cdots j]$

Goal: $L[1, n]$

Make choice: Is $S[i] = S[j]$?

Recurrence:

$$L[i, j] = \begin{cases} L[i+1, j-1] + 2 & \text{if } S[i] = S[j] \\ \max\{L[i+1, j], L[i, j-1]\} & \text{if } S[i] \neq S[j] \end{cases}$$

Longest Palindrome Subsequence (Problem 14.11 (1))

(1) Find (the length of) a longest palindrome subsequence of $S[1 \cdots n]$

Subproblem: $L[i, j]$: the length of an LSP of $S[i \cdots j]$

Goal: $L[1, n]$

Make choice: Is $S[i] = S[j]$?

Recurrence:

$$L[i, j] = \begin{cases} L[i+1, j-1] + 2 & \text{if } S[i] = S[j] \\ \max\{L[i+1, j], L[i, j-1]\} & \text{if } S[i] \neq S[j] \end{cases}$$

Init:

$$L[i, i] = 1, \ \forall 1 \le i \le n$$

$$L[i, i+1] = 2, \ \text{if } S[i] = S[i+1], \ \forall 1 \le i \le n-1$$

Palindrome Splitting (Problem 14.11 (2))

(2) Split a string $S[1 \ldots n]$ into minimum number of palindromes ($\#$ cuts)

Subproblem: $C[i, j]$: minimum number of cuts for string $S[i \ldots j]$

Goal: $C[1, n] + 1$

Palindrome Splitting (Problem 14.11 (2))

(2) Split a string $S[1 \ldots n]$ into minimum number of palindromes ($\#$ cuts)

Subproblem: $C[i,j]$: minimum number of cuts for string $S[i \ldots j]$

Goal: $C[1,n] + 1$

Make choice: Where is the first cut?

Recurrence:

$$C[i,j] = \begin{cases} 0 & \text{if } S[i \ldots j] \text{ is a palindrome} \\ \displaystyle\min_{i+1 \le k \le j-1} C[i,k-1] + 1 + C[k,j] & \text{o.w.} \end{cases}$$

Palindrome Splitting (Problem 14.11 (2))

(2) Split a string $S[1 \dots n]$ into minimum number of palindromes (# cuts)

Subproblem: $C[i,j]$: minimum number of cuts for string $S[i \dots j]$

Goal: $C[1,n] + 1$

Make choice: Where is the first cut?

Recurrence:

$$C[i,j] = \begin{cases} 0 & \text{if } S[i \dots j] \text{ is a palindrome} \\ \min\limits_{i+1 \le k \le j-1} C[i,k-1] + 1 + C[k,j] & \text{o.w.} \end{cases}$$

Init: $C[i,i] = 0$

Time: $O(n^3)$

Palindrome Splitting (Problem 14.11 (2))

(2) Split a string $S[1 \dots n]$ into minimum number of palindromes

Subproblem: $P[i]$: minimum number of palindromes for $S[1 \cdots i]$

Goal: $P[n]$

Palindrome Splitting (Problem 14.11 (2))

(2) Split a string $S[1 \ldots n]$ into minimum number of palindromes

Subproblem: $P[i]$: minimum number of palindromes for $S[1 \cdots i]$

Goal: $P[n]$

Make choice: Where does the last palindrome start from?

Recurrence:

$$P[i] = \min_{\substack{1 \le k \le i \\ S[k \ldots i] \text{ is a palindrome}}} P[k-1] + 1$$

Palindrome Splitting (Problem 14.11 (2))

(2) Split a string $S[1 \ldots n]$ into minimum number of palindromes

Subproblem: $P[i]$: minimum number of palindromes for $S[1 \cdots i]$

Goal: $P[n]$

Make choice: Where does the last palindrome start from?

Recurrence:

$$P[i] = \min_{\substack{1 \le k \le i \\ S[k \ldots i] \text{ is a palindrome}}} P[k-1] + 1$$

Init: $P[0] = 1$

Time: $O(n^3)$ *vs.* $O(n^2)$

Dynamic Programming

3-D DP

Floyd-Warshall algorithm

(1) DP for Floyd-Warshall algorithm for APSP on directed graphs

Subproblem: dist$[i, j, k]$: the length of the shortest path from i to j via only nodes in $v_1 \cdots v_k$

Goal: dist$[i, j, n], \forall i, j$

3-D DP

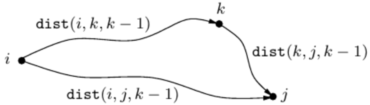
Floyd-Warshall algorithm

(1) DP for Floyd-Warshall algorithm for APSP on directed graphs

Make choice: Is v_k on the ShortestPath$[i, j, k]$?

Recurrence:

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



3-D DP

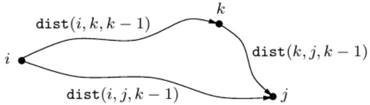
Floyd-Warshall algorithm

(1) DP for Floyd-Warshall algorithm for APSP on directed graphs

Make choice: Is v_k on the ShortestPath$[i, j, k]$?

Recurrence:

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



Init:

$$\text{dist}[i, j, 0] = \begin{cases} 0 & i = j \\ w(i, j) & (i, j) \in E \\ \infty & \text{o.w.} \end{cases}$$

3-D DP

Floyd-Warshall algorithm (Problem 6.25)

(2) Routing table for Floyd-Warshall algorithm

for all $k \leftarrow 1 \ldots n$ **do**
  for all $i \leftarrow 1 \ldots n$ **do**
    for all $j \leftarrow 1 \ldots n$ **do**
      if $\text{dist}[i,j] > \text{dist}[i,k] + \text{dist}[k,j]$ **then**
        $\text{dist}[i,j] \leftarrow \text{dist}[i,k] + \text{dist}[k,j]$

3-D DP

Floyd-Warshall algorithm (Problem 6.25)

(2) Routing table for Floyd-Warshall algorithm

for all $k \leftarrow 1 \ldots n$ **do**
  for all $i \leftarrow 1 \ldots n$ **do**
    for all $j \leftarrow 1 \ldots n$ **do**
      if $\text{dist}[i,j] > \text{dist}[i,k] + \text{dist}[k,j]$ **then**
        $\text{dist}[i,j] \leftarrow \text{dist}[i,k] + \text{dist}[k,j]$

Time: $\Theta(n^3)$ Space: $\Theta(n^2)$

3-D DP

Floyd-Warshall algorithm (Problem 6.25)

(2) Routing table for Floyd-Warshall algorithm

for all $k \leftarrow 1 \ldots n$ **do**
  for all $i \leftarrow 1 \ldots n$ **do**
    for all $j \leftarrow 1 \ldots n$ **do**
      if $\text{dist}[i, j] > \text{dist}[i, k] + \text{dist}[k, j]$ **then**
        $\text{dist}[i, j] \leftarrow \text{dist}[i, k] + \text{dist}[k, j]$
        $\text{Go}[i, j] \leftarrow \text{Go}[i, k]$

Time: $\Theta(n^3)$ Space: $\Theta(n^2)$

3-D DP

Floyd-Warshall algorithm (Problem 6.25)

(2) Routing table for Floyd-Warshall algorithm

for all $i \leftarrow 1 \dots n$ **do**
  for all $j \leftarrow 1 \dots n$ **do**
    dist$[i, j] \leftarrow \infty$
    Go$[i, j] \leftarrow$ Nil
for all $(i, j) \in E$ **do**
  dist$[i, j] \leftarrow w(i, j)$
  Go$[i, j] \leftarrow j$
for all $i \leftarrow 1 \dots n$ **do**
  dist$[i, i] \leftarrow 0$
  Go$[i, i] \leftarrow$ Nil

3-D DP

Floyd-Warshall algorithm (Problem 6.25)

(2) Routing table for Floyd-Warshall algorithm

```
for all i ← 1 ... n do
    for all j ← 1 ... n do
        dist[i, j] ← ∞
        Go[i, j] ← Nil
for all (i, j) ∈ E do
    dist[i, j] ← w(i, j)
    Go[i, j] ← j
for all i ← 1 ... n do
    dist[i, i] ← 0
    Go[i, i] ← Nil
```

```
procedure PATH(i, j)
    if Go[i, j] = Nil then
        Output "No Path."

    Output "i"
    while i ≠ j do
        i ← Go[i, j]
    Output "i"
```

3-D DP

Floyd-Warshall algorithm (Problem 6.29)

(3) Find minimum-weighted cycle of directed graph $(w(e) > 0)$

$$\text{dist}[i,i] \leftarrow 0 \implies \text{dist}[i,i] \leftarrow \infty$$

$$\forall i : \text{dist}[i,i] = \infty$$

3-D DP

Floyd-Warshall algorithm (Problem 6.29)

(3) Find minimum-weighted cycle of directed graph $(w(e) > 0)$

$$\text{dist}[i,i] \leftarrow 0 \implies \text{dist}[i,i] \leftarrow \infty$$

$$\forall i : \text{dist}[i,i] = \infty$$

Q: $\exists e : w(e) < 0$

3-D DP

Floyd-Warshall algorithm (Problem 6.29)

(3) Find minimum-weighted cycle of directed graph $(w(e) > 0)$

$$\textsf{dist}[i,i] \leftarrow 0 \implies \textsf{dist}[i,i] \leftarrow \infty$$

$$\forall i : \textsf{dist}[i,i] = \infty$$

Q: $\exists e : w(e) < 0$

$\exists i : \textsf{dist}[i,i] < 0$

$\forall i : \textsf{dist}[i,i] \geq 0 \; (= \infty)$

Shortest paths on undirected graphs

Finding shortest paths in undirected graphs with possibly negative edge weights

2

2

The book "Algorithms" by Robert Sedgewick and Kevin Wayne hinted that (*see the quote below*) there are efficient algorithms for finding shortest paths in undirected graphs with possibly negative edge weights (***not*** by treating an undirected edge as two directed one which means that a single negative edge implies a negative cycle). However, no references are given in the book. Are you aware of any such algorithms?

> Q. How can we find shortest paths in undirected (edge-weighted) graphs?
>
> A. For positive edge weights, Dijkstra's algorithm does the job. We just build an EdgeWeightedDigraph corresponding to the given EdgeWeightedGraph (by adding two directed edges corresponding to each undirected edge, one in each direction) and then run Dijkstra's algorithm. ***If edge weights can be negative*** (***emphasis added***), efficient algorithms are available, but they are more complicated than the Bellman-Ford algorithm.

algorithms graph-theory shortest-path weighted-graphs reference-question

share cite edit close delete flag

edited Jun 9 at 14:15

asked Jun 9 at 13:58

hengxin

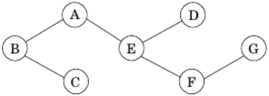
6,056 11 38

https://cs.stackexchange.com/q/76578/4911

DP on Graphs

Minimum Vertex Cover on Trees (Problem 14.14)

- Undirected tree $T = (V, E)$; No designated root!
- Compute (the size of) a minimum vertex cover of T



Rooted T at any node r.

Rooted T at any node r.

Subproblem: $I(u)$: the size of an MVC of subtree T_u rooted at u

Goal: $I(r)$

Rooted T at any node r.

Subproblem: $I(u)$: the size of an MVC of subtree T_u rooted at u

Goal: $I(r)$

Make choice: Is u in MVC$[u]$?

Recurrence:

$$I(u) = \min\{\#\text{ children of } u + \sum_{v:\text{ grandchildren of } u} I(v), \quad 1 + \sum_{v:\text{ children of } u} I(v)\}$$

Rooted T at any node r.

Subproblem: $I(u)$: the size of an MVC of subtree T_u rooted at u

Goal: $I(r)$

Make choice: Is u in MVC$[u]$?

Recurrence:

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

Init: $I(u) = 0$, if u is a leave

DFS on T from root r:

when u is "finished":
if u is a leave **then**
$I(u) \leftarrow 0$
else
$I(u) \leftarrow \dots$

DFS on T from root r:

when u is "finished":
if u is a leave **then**
$I(u) \leftarrow 0$
else
$I(u) \leftarrow \ldots$

Greedy algorithm (Rough Proof!):

Theorem

There is an MVC which contains no leaves.

The Knapsack Problem

The Change-making Problem (Problem 14.13)

- Coins values: $x_1 \dots x_n$
- Amount: v
- Is it possible to make change for v?

The change-making problem

The change-making problem (Problem 14.13 (2), Problem 14.2 (Subset sum))

(2) Without repetition $(0/1)$

The change-making problem

The change-making problem (Problem 14.13 (2), Problem 14.2 (Subset sum))

(2) Without repetition $(0/1)$

Subproblem: $C[i, w]$: Make change for w using only values of $x_1 \ldots x_i$?

Goal: $C[n, v]$

The change-making problem

The change-making problem (Problem 14.13 (2), Problem 14.2 (Subset sum))

(2) Without repetition (0/1)

Subproblem: $C[i, w]$: Make change for w using only values of $x_1 \dots x_i$?

Goal: $C[n, v]$

Make choice: Using value x_i or not?

Recurrence:

$$C[i, w] = C[i-1, w] \lor (C[i-1, w - x_i] \land w \geq x_i)$$

The change-making problem

The change-making problem (Problem 14.13 (2), Problem 14.2 (Subset sum))

(2) Without repetition $(0/1)$

Subproblem: $C[i, w]$: Make change for w using only values of $x_1 \ldots x_i$?

Goal: $C[n, v]$

Make choice: Using value x_i or not?

Recurrence:

$$C[i, w] = C[i-1, w] \lor (C[i-1, w - x_i] \land w \geq x_i)$$

Init:

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

The Change-making Problem (Problem 14.13 (1))

(1) Unbounded repetition (∞)

The Change-making Problem (Problem 14.13 (1))

(1) Unbounded repetition (∞)

Subproblem: $C[i,w]$: Make change for w using only values of $x_1 \ldots x_i$?

Goal: $C[n,v]$

The Change-making Problem (Problem 14.13 (1))

(1) Unbounded repetition (∞)

Subproblem: $C[i, w]$: Make change for w using only values of $x_1 \ldots x_i$?

Goal: $C[n, v]$

Make choice: Using value x_i or not?

Recurrence:

$$C[i, w] = C[i - 1, w] \lor (C[i, w - x_i] \land w \geq x_i)$$

The Change-making Problem (Problem 14.13 (1))

(1) Unbounded repetition (∞)

Subproblem: $C[i, w]$: Make change for w using only values of $x_1 \ldots x_i$?

Goal: $C[n, v]$

Make choice: Using value x_i or not?

Recurrence:

$$C[i, w] = C[i-1, w] \lor (C[i, w - x_i] \land w \geq x_i)$$

Init:

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

Time: $O(nv)$

The Change-making Problem (Problem 14.13 (1))

(1) Unbounded repetition (∞)

The Change-making Problem (Problem 14.13 (1))

(1) Unbounded repetition (∞)

Subproblem: $C[w]$: Possible to make change for w?

Goal: $C[v]$

The Change-making Problem (Problem 14.13 (1))

(1) Unbounded repetition (∞)

Subproblem: $C[w]$: Possible to make change for w?

Goal: $C[v]$

Make choice: What is the first coin to use?

Recurrence:

$$C[w] = \bigvee_{i:\, x_i \le w} C[w - x_i]$$

The Change-making Problem (Problem 14.13 (1))

(1) Unbounded repetition (∞)

Subproblem: $C[w]$: Possible to make change for w?

Goal: $C[v]$

Make choice: What is the first coin to use?

Recurrence:

$$C[w] = \bigvee_{i:\, x_i \le w} C[w - x_i]$$

Init: $C[0] =$ true

Time: $O(nv)$

The Change-making Problem (Problem 14.13 (1))

(1) Unbounded repetition (∞)

$$C[i, w] \textit{ vs. } C[w]$$

$$C[i, w] = C[i-1, w] \lor (C[i, w - x_i] \land w \geq x_i)$$

$$C[w] = \bigvee_{i:\, x_i \leq w} C[w - x_i]$$

The Change-making Problem (Problem 14.13 (3))

(3) Unbounded repetition with $\leq k$ coins

The Change-making Problem (Problem 14.13 (3))

(3) Unbounded repetition with $\leq k$ coins

Subproblem: $C[i, w, l]$: Possible to make change for w with $\leq l$ coins of values of $x_1 \ldots x_i$?

Goal: $C[n, v, k]$

The Change-making Problem (Problem 14.13 (3))

(3) Unbounded repetition with $\leq k$ coins

Subproblem: $C[i, w, l]$: Possible to make change for w with $\leq l$ coins of values of $x_1 \ldots x_i$?

Goal: $C[n, v, k]$

Make choice: Using value x_i or not?

Recurrence:

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

The Change-making Problem (Problem 14.13 (3))

(3) Unbounded repetition with $\leq k$ coins

Subproblem: $C[i, w, l]$: Possible to make change for w with $\leq l$ coins of values of $x_1 \ldots x_i$?

Goal: $C[n, v, k]$

Make choice: Using value x_i or not?

Recurrence:

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

Init:

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

The Change-making Problem (Problem 14.13 (3))

(3) Unbounded repetition with $\leq k$ coins

The Change-making Problem (Problem 14.13 (3))

(3) Unbounded repetition with $\leq k$ coins

Subproblem: $C[w, l]$: Possible to make change for w with $\leq l$ coins?

Goal: $C[v, k]$

The Change-making Problem (Problem 14.13 (3))

(3) Unbounded repetition with $\leq k$ coins

Subproblem: $C[w, l]$: Possible to make change for w with $\leq l$ coins?

Goal: $C[v, k]$

Make choice: What is the first coin to use?

Recurrence:

$$C[w, l] = \bigvee_{i:\, x_i \leq w} C[w - x_i, l - 1]$$

The Change-making Problem (Problem 14.13 (3))

(3) Unbounded repetition with $\leq k$ coins

Subproblem: $C[w, l]$: Possible to make change for w with $\leq l$ coins?

Goal: $C[v, k]$

Make choice: What is the first coin to use?

Recurrence:

$$C[w, l] = \bigvee_{i:\ x_i \leq w} C[w - x_i, l - 1]$$

Init:

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

Dynamic Programming

More DPs . . .

Algorithms that use dynamic programming [edit | edit source]

This section **does not cite any sources**. Please help improve this section by adding citations to reliable sources. Unsourced material may be challenged ar
how and when to remove this template message)

- Recurrent solutions to lattice models for protein-DNA binding
- Backward induction as a solution method for finite-horizon discrete-time dynamic optimization problems
- Method of undetermined coefficients can be used to solve the Bellman equation in infinite-horizon, discrete-time, discounted, time-invariant dynamic optimization problems
- Many string algorithms including longest common subsequence, longest increasing subsequence, longest common substring, Levenshtein distance (edit distance)
- Many algorithmic problems on graphs can be solved efficiently for graphs of bounded treewidth or bounded clique-width by using dynamic programming on a tree decomposition of the graph.
- The Cocke–Younger–Kasami (CYK) algorithm which determines whether and how a given string can be generated by a given context-free grammar
- Knuth's word wrapping algorithm that minimizes raggedness when word wrapping text
- The use of transposition tables and refutation tables in computer chess
- The Viterbi algorithm (used for hidden Markov models)
- The Earley algorithm (a type of chart parser)
- The Needleman–Wunsch algorithm and other algorithms used in bioinformatics, including sequence alignment, structural alignment, RNA structure prediction
- Floyd's all-pairs shortest path algorithm
- Optimizing the order for chain matrix multiplication
- Pseudo-polynomial time algorithms for the subset sum, knapsack and partition problems
- The dynamic time warping algorithm for computing the global distance between two time series
- The Selinger (a.k.a. System R) algorithm for relational database query optimization
- De Boor algorithm for evaluating B-spline curves
- Duckworth–Lewis method for resolving the problem when games of cricket are interrupted
- The value iteration method for solving Markov decision processes
- Some graphic image edge following selection methods such as the "magnet" selection tool in Photoshop
- Some methods for solving interval scheduling problems
- Some methods for solving the travelling salesman problem, either exactly (in exponential time) or approximately (e.g. via the bitonic tour)
- Recursive least squares method
- Beat tracking in music information retrieval
- Adaptive-critic training strategy for artificial neural networks
- Stereo algorithms for solving the correspondence problem used in stereo vision
- Seam carving (content-aware image resizing)
- The Bellman–Ford algorithm for finding the shortest distance in a graph
- Some approximate solution methods for the linear search problem
- Kadane's algorithm for the maximum subarray problem


THANK YOU