Chapter 8: Dynamic Programming

Those who do not remember the past are condemned to repeat it. (George Santayana)

Introduction

Dynamic programming is a technique for solving problems with the following properties:

- An instance is solved using the solutions for smaller instances.
- The solution for a smaller instance might be needed multiple times.
- The solutions to smaller instances are stored in a table, so that each smaller instance is solved only once.
- Additional space is used to save time.

Example: Computing Fibonacci Numbers

Compare recursive solution (exponential):

algorithm $F(n)$
  if $n = 0$ or $n = 1$ **then return** n
  else return $F(n-1) + F(n-2)$

To dynamic programming solution (linear):

algorithm $F(n)$
  $A[0] \leftarrow 0$; $A[1] \leftarrow 1$
  for $i \leftarrow 2$ **to** n **do**
    $A[i] \leftarrow A[i-1] + A[i-2]$
  return $A[n]$

Computing Binomial Coefficients

Definition of Binomial Coefficients

- The binomial coefficient, denoted $C(n,k)$ or $\binom{n}{k}$ is the number of subsets of k elements from an n-element set.
- The binomial formula is:

  $(a+b)^n = C(n,0)\,a^n + \cdots + C(n,k)\,a^{n-k}b^k + \cdots + C(n,n)\,b^n$
- One property of binomial coefficients is:

  $C(n,k) = C(n-1,k-1) + C(n-1,k)$
  $C(n,0) = C(n,n) = 1$
- A dynamic programming approach stores the values of $C(n,k)$ as they are computed.

Table for Computing Binomials

	0	1	2	...	$k-1$	k
0	1					
1	1	1				
2	1	2	1			
$\vdots$						
k	1					1
$\vdots$						
$n-1$	1				$C(n-1, k-1)$	$C(n-1, k)$
n	1					$C(n, k)$

FIGURE 8.1 Table for computing the binomial coefficient $C(n, k)$ by the dynamic programming algorithm

Binomial Algorithm

algorithm $\mathit{Binomial}(n,k)$
    // Computes $C(n,k)$ using dynamic programming
    // Input: Integers $n \ge k \ge 0$
    // Output: The value of $C(n,k)$
    for $i \leftarrow 0$ **to** n **do**
        for $j \leftarrow 0$ **to** $\min(i,k)$ **do**
            if $j = 0$ **or** $j = i$ **then**
                $A[i,j] \leftarrow 1$
            else $A[i,j] \leftarrow A[i-1,j-1] + A[i-1,j]$
    return $A[n,k]$

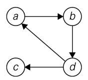
Comments on Binomial Algorithm

- The algorithm computes each value $C(i,j)$ once where $j \le i$, $j \le k$, and $i \le n$.
- $$\underbrace{1 + 2 + \cdots + k}_{\substack{k \text{ terms} \\ 0 \le j \le i < k}} + \underbrace{(k{+}1) + \cdots + (k{+}1)}_{\substack{n-k+1 \text{ terms} \\ 0 \le j \le k \le i \le n}}$$
- $$\frac{k(k+1)}{2} + (k+1)(n-k+1) = (k+1)(n+1-k/2) \in \Theta(nk)$$
- More efficient algorithms are suggested in the exercises.

Warshall's Algorithm

Transitive Closure

- The transitive closure of a directed graph is the set of pairs (i, j) where there is a path from vertex i to vertex j
- If (i, j) is an edge, it is in the transitive closure.
- If (i, j) and (j, k) are in the transitive closure, then so is (i, k).



$$A = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array} \qquad T = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

(a) (b) (c)

FIGURE 8.2 (a) Digraph. (b) Its adjacency matrix. (c) Its transitive closure.

Bad Algorithm for Transitive Closure

algorithm $BadTransitiveClosure(A[1..n, 1..n])$
  // Doesn't compute transitive closure
  // Input: Adjacency matrix A
  // Output: Transitive closure matrix R
  $R \leftarrow A$
  for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** n **do**
      for $k \leftarrow 1$ **to** n **do**
        if $R[i, j]$ **and** $R[j, k]$ **then**
          $R[i, k] \leftarrow$ **true**
  return R

Need to repeat triple loop until no changes, so $O(n^4)$.

Idea Behind Warshall's Algorithm

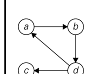
- The idea is to process each vertex as an intermediate vertex.
- Suppose a path (d, a, c, b, e).
- Consider a. Edges (d, a) and (a, c) imply (d, c) in the transitive closure.
- Consider b. Edges (c, b) and (b, e) imply (c, e) in the transitive closure.
- Consider c. Edges (d, c) and (c, e) in the transitive closure imply (d, e) in it, too.

Warshall's Algorithm

algorithm $Warshall(A[1..n, 1..n])$
  // Computes transitive closure matrix
  // Input: Adjacency matrix A
  // Output: Transitive closure matrix R
  $R \leftarrow A$
  for $j \leftarrow 1$ **to** n **do**
    for $i \leftarrow 1$ **to** n **do**
      for $k \leftarrow 1$ **to** n **do**
        if $R[i, j]$ **and** $R[j, k]$ **then**
          $R[i, k] \leftarrow$ **true**
  return R

This algorithm is $\Theta(n^3)$.

Example of Warshall's Algorithm



$$R^{(0)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}$$

Ones reflect the existence of paths with no intermediate vertices ($R^{(0)}$ is just the adjacency matrix); boxed row and column are used for getting $R^{(1)}$.

$$R^{(1)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & \mathbf{1} & 1 & 0 \end{bmatrix} \end{array}$$

Ones reflect the existence of paths with intermediate vertices numbered not higher than 1, i.e., just vertex a (note a new path from d to b); boxed row and column are used for getting $R^{(2)}$.

$$R^{(2)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & \mathbf{1} \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & \mathbf{1} \end{bmatrix} \end{array}$$

Ones reflect the existence of paths with intermediate vertices numbered not higher than 2, i.e., a and b (note two new paths); boxed row and column are used for getting $R^{(3)}$.

Example Continued

$$R^{(3)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

Ones reflect the existence of paths with intermediate vertices numbered not higher than 3, i.e., *a, b,* and *c* (no new paths); boxed row and column are used for getting $R^{(4)}$.

$$R^{(4)} = \begin{array}{c} \\ a \\ b \\ c \\ d \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} \mathbf{1} & 1 & \mathbf{1} & 1 \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & 1 \\ 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix} \end{array}$$

Ones reflect the existence of paths with intermediate vertices numbered not higher than 4, i.e., *a, b, c,* and *d* (note five new paths).

FIGURE 8.4 Application of Warshall's algorithm to the digraph shown. New ones are in bold.

Other Dynamic Programming Problems 14

Optimal Binary Search Tree

- Suppose for a set of sorted keys, we know the probability $P[i]$ each key will be searched.
- The problem is to find the binary search tree that minimizes the number of comparisons.
- Let $P[i, j] = \sum_{k=i}^{j} P[i]$.
  Let $P[i, i-1] = 0$ by definition.
- Let $C[i, j]$ be the average number of comparisons for the best BST for keys i to j.
  Let $C[i, i] = P[i]$ and $C[i, i-1] = 0$ by definition.
- $C[i, j] = P[i, j]+$
  the minimum $C[i, k-1] + C[k+1, j]$ where $i \leq k \leq j$.

The Knapsack Problem

- Suppose for a set of items, we know the value w_i and weight v_i of each item.
- The problem is to find the most valuable subset of items that weigh no more than W.
- Let $V[i, j]$ be the value for the knapsack problem for the first i items and weight j.
  Let $V[i, 0] = 0$ by definition.
- $V[i, j]$ is the maximum value of:
  - $V[i, j-1]$
  - $V[i-1, j]$
  - $v_i + V[i-1, j-w_i]$ (assuming $j \geq w_i$)

Example for the Knapsack Problem

		capacity *j*					
	i	0	1	2	3	4	5
	0	0	0	0	0	0	0
$w_1 = 2$, $v_1 = 12$	1	0	0	12	12	12	12
$w_2 = 1$, $v_2 = 10$	2	0	10	12	22	22	22
$w_3 = 3$, $v_3 = 20$	3	0	10	12	22	30	32
$w_4 = 2$, $v_4 = 15$	4	0	10	15	25	30	37