## CS 435: Algorithms

Dynamic Programming
Prem Nair

#### Dynamic Programming - Basics

Dynamic Programming is a general algorithm design technique for solving problems defined by or formulated as recurrences with overlapping sub-problems.

- Invented by American mathematician Richard Bellman in the 1950s to solve optimization problems and later assimilated by CS
- "Programming" here means "planning"
- Main idea:
  - set up a recurrence relating a solution to a larger instance to solutions of some smaller instances.
  - solve smaller instances once.
  - record solutions in a table thus avoid recomputing that subproblem.
  - extract solution to the "computed instance" from that table.

#### Dynamic Programming - Basics

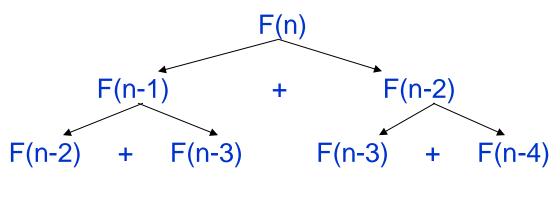
- Similar to Divide and Conquer
- However, Divide and Conquer uses non-overlapped sub-problems whereas DP uses overlapped sub-problem.
- Use of Divide and Conquer will result in many unnecessary computations as sub-problems are overlapped.
- DP saves computations in each iteration which can be used in the next iteration. thus saves computation time BUT at the cost of more memory!

#### Example: Fibonacci numbers

Recall definition of Fibonacci numbers:

$$F(n) = F(n-1) + F(n-2)$$
  
 $F(0) = 0$   
 $F(1) = 1$ 

Computing the nth Fibonacci number recursively (top-down):



#### Example: Fibonacci numbers

$$F(n) = F(n-1) + F(n-2)$$
  
 $F(0) = 0$   
 $F(1) = 1$ 

If we have values for F(n-1) and F(n-2), there is no need to recompute those value.

In the dynamic programming approach, we order the computation as follows:

This ordering eliminates need for re-computation.

#### Example: Fibonacci numbers

```
algorithm DynamicFibonacci(n)
Input: n a non-negative integer
Output: The n-th Fibonacci number
if (n = 0 \text{ or } n = 1) \text{ return } n
previous <- 0
current <- 1
for (i <- 2 to n)
   temp <- previous + current
   previous <- current</pre>
   current <- temp
return current
```

## Computing a binomial coefficient by DP

## Binomial coefficients are coefficients of the binomial formula:

```
(a + b)^n = C(n,0)a^nb^0 + ... + C(n,k)a^{n-k}b^k + ... + C(n,n)a^0b^n

Recurrence: C(n,k) = C(n-1,k) + C(n-1,k-1) for n > k > 0

C(n,0) = 1, C(n,n) = 1 for n \ge 0
```

#### Value of C(n,k) can be computed by filling a table:

```
0 1 2 3 . . k-1 k

0 1
1 1 1
2 1 2 1
3 1 3 3 1

C(n-1,k-1) C(n-1,k)
C(n,k)
```

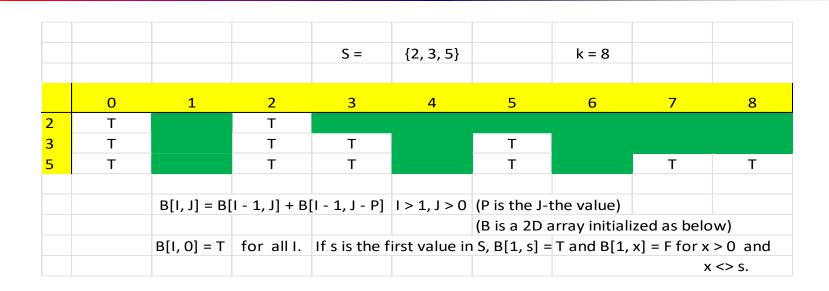
# Computing C(n,k): pseudocode and analysis

```
ALGORITHM
                Binomial(n, k)
    //Computes C(n, k) by the dynamic programming algorithm
    //Input: A pair of nonnegative 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
                  C[i, j] \leftarrow 1
             else C[i, j] \leftarrow C[i-1, j-1] + C[i-1, j]
    return C[n, k]
```

Time complexity: 9(nk)

Space complexity: 9(nk)

#### Subset Sum (True or False)



Time Complexity : O(n(k+1))

Space Complexity : O(n(k+1))

### Subset Sum (one subset)

				S =	{2, 3, 5}		k = 8		
	0	1	2	3	4	5	6	7	8
2	{}		{2}						
3	{}		{2}	{3}		{2, 3}			
5	{}		{2}	{3}		{2, 3}		{2,5}	{3, 5}

Time Complexity : O(n(k+1))

Space Complexity : O(n(k+1))

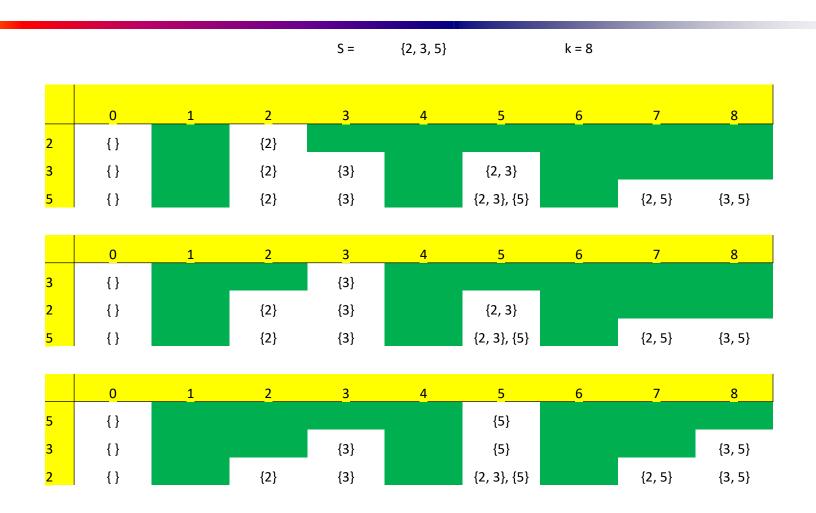
#### Subset Sum

				S =	{2, 3, 5}		k = 8		
	0	1	2	3	4	5	6	7	8
2	{}		{2}						
3	{}		{2}	{3}		{2, 3}			
5	{}		{2}	{3}		{2, 3}, {5}		{2,5}	{3, 5}

Time Complexity : O(n(k+1))

Space Complexity : O(n(k+1))

#### Subset Sum



# Properties of a problem that can be solved with dynamic programming

#### □ Simple Subproblems

- We should be able to break the original problem to smaller subproblems that have the same structure

#### □ Optimal Substructure of the problems

- The solution to the problem must be a composition of subproblem solutions

#### □Subproblem Overlap

- Optimal subproblems to unrelated problems can contain subproblems in common

#### 0-1 Knapsack Problem

Given a knapsack with maximum capacity W, and a set S consisting of n items.

Each item i has some weight  $w_i$  and value  $v_i$  (all  $w_i$ ,  $v_i$  and W are positive integer values)

<u>Problem</u>: How to pack the knapsack such that value of packed items is the maximum possible?

#### Knapsack Problem by DP

Given *n* items of

positive integer weights:  $w_1$   $w_2$  ...  $w_n$ 

positive values:  $v_1 \quad v_2 \quad ... \quad v_n$ 

a knapsack of positive integer capacity W

find most valuable subset of the items that fit into the knapsack

Consider instance defined by first i items and capacity j ( $j \le W$ ).

Let V[i,j] be optimal value of such an instance. Then

#### Knapsack Problem by DP

$$\max\{V[i\text{-}1,j],\ v_i+V[i\text{-}1,j\text{-}w_i]\} \quad \text{if } j\text{-}w_i \geq 1$$
 
$$V[i,j] = V[i\text{-}1,j] \qquad \qquad \text{if } j\text{-}w_i < 1$$

Initial conditions: V[0,j] = 0 and V[i,0] = 0.

### Knapsack Problem by DP

		k - w		k
		y		X
Item with weight w and value v				max(x, y + v)

Item	a	b	c	d	e	
value	15	12	9	16	17	
Weight	2	5	3	4	6	

The maximum allowable total weight in the knapsack is Wmax = 12. Column heading shows maximum weight considered.

Note we start with 1 and it goes up to Wmax = 12.

	1	2	3	4	5	6	7	8	9	10	11	12
a												
b												
С												
d												
e												

Item	a	b	c	d	e
value	15	12	9	16	17
Weight	2	5	3	4	6

*row 1.* 

	1	2	3	4	5	6	7	8	9	10	11	12
a	0	15	15	15	15	15	15	15	15	15	15	15
b												
С												
d												
e												

Item	a	b	С	d	e
value	15	12	9	16	17
Weight	2	5	3	4	6

row 2.

$$wt ext{ of } b = 5. \ j - wt(b) = 12 - 5 = 7. \ V[2, 12] = max\{V[1, 12], \ V[1, 7] + v(b)\}$$
  
=  $max\{15, 15 + 12\} = 27$ 

Since weight of b is 5, until allowed weight is >= 5, we have only one choice. That is a. Once allowed weight is 5, we have two choices: a and b. Now value (a) > value (b). Hence no change.

Once allowed weight is 7, we have extra 5 space after a. We can put b also. Thus total value is 27.

	1	2	3	4	5	6	7	8	9	10	11	12
a	0	15	15	15	15	15	15	15	15	15	15	15
b	0	15	15	15	15	15	27	27	27	27	27	27
С												
d												
e												

Item	a	b	c	d	e
value	15	12	9	16	17
Weight	2	5	3	4	6

*row 3*.

$$wt \ of \ c = 3. \ j - wt(c) = 12 - 3 = 9. \ V[3, \ 12] = max\{V[2, \ 12], \ V[2, \ 9] + v(c)\}$$

$$= max\{27, \ 27 + 9\} = 36$$
 $wt \ of \ c = 3. \ j - wt(c) = 9 - 3 = 6. \ V[3, \ 9] = max\{V[2, \ 9], \ V[2, \ 6] + v(c)\}$ 

$$= max\{27, \ 15 + 9\} = 27$$
 $Wt \ of \ c = 3. \ j - wt(c) = 6 - 3 = 3. \ V[3, \ 6] = max\{V[2, \ 6], \ V[2, \ 3] + v(c)\}$ 

$$= max\{15, \ 15 + 9\} = 24$$

Since wt(c) is 3, after placing a, still there is space for c. Hence when allowed weight is 5 and 6, we can place a and c.

Once allowed weight is 7, we can place a and b, since value of b = 12 is than value of c = 9. When total allowed weight is 10, we have extra 3 space to place c. Thus we place a, b and c. Thus total value = 36.

	1	2	3	4	5	6	7	8	9	10	11	12
a	0	15	15	15	15	15	15	15	15	15	15	15
b	0	15	15	15	15	15	27	27	27	27	27	27
c	0	15	15	15	24	24	27	27	27	36	36	36
d												
e												

Item	a	b	c	d	e	_
value	15	12	9	16	17	
Weight	2	5	3	4	6	

row 4.

$$wt ext{ of } d = 4. \ j - wt(d) = 12 - 4 = 8. \ V[4, 12] = max\{V[3, 12], \ V[3, 8] + v(d)\}$$

$$= max\{36, 27 + 16\} = 43$$
 $wt ext{ of } d = 4. \ j - wt(d) = 10 - 4 = 6. \ V[4, 10] = max\{V[3, 10], \ V[3, 6] + v(d)\}$ 

$$= max\{36, 24 + 16\} = 40$$
 $Wt ext{ of } d = 4. \ j - wt(d) = 6 - 4 = 2. \ V[4, 6] = max\{V[3, 6], \ V[3, 2] + v(c)\}$ 

$$= max\{24, 15 + 16\} = 31$$

	1	2	3	4	5	6	7	8	9	10	11	12
a	0	15	15	15	15	15	15	15	15	15	15	15
b	0	15	15	15	15	15	27	27	27	27	27	27
c	0	15	15	15	24	24	27	27	27	36	36	36
d	0	15	15	16	24	31	31	31	40	40	43	43
e												

Item	a	b	c	d	e
value	15	12	9	16	17
Weight	2	5	3	4	6

$$wt \ of \ e = 6. \ j - wt(e) = 12 - 6 = 6. \ V[5, 12] = max\{V[4, 12], \ V[4, 6] + v(e)\}$$

$$= max\{43, 31 + 17\} = 48$$
 $wt \ of \ e = 6. \ j - wt(e) = 11 - 6 = 5. \ V[5, 11] = max\{V[4, 11], \ V[4, 5] + v(e)\}$ 

$$= max\{43, 24 + 17\} = 43$$
 $Wt \ of \ e = 6. \ j - wt(e) = 8 - 6 = 2. \ V[5, 8] = max\{V[4, 8], \ V[4, 2] + v(e)\}$ 

$$= max\{31, 15 + 17\} = 32$$

	1	2	3	4	5	6	7	8	9	10	11	12
a	0	15	15	15	15	15	15	15	15	15	15	15
b	0	15	15	15	15	15	27	27	27	27	27	27
c	0	15	15	15	24	24	27	27	27	36	36	36
d	0	15	15	16	24	31	31	31	40	40	43	43
e	0	15	15	16	24	31	31	32	40	40	43	48

#### 0-1 Knapsack Problem

Note that all we need is just one row. You can keep updating starting with J = W and all the way up to j = 1.

Thus the space complexity is O(W). The time complexity is O(nW) where n is number of items and W is the capacity of the knapsack.

If you first sort by weight, we can further simplify the calculations.

Item	a	b	c	d	e
value	15	9	16	12	17
Weight	2	3	4	5	6

	1	2	3	4	5	6	7	8	9	10	11	12
a												
b												
С												
d												
е												

Item	a	b	c	d	e
value	15	9	16	12	17
Weight	2	3	4	5	6

	1	2	3	4	5	6	7	8	9	10	11	12
a	0	15	15	15	15	15	15	15	15	15	15	15
b												
С												
d												
e												

Item	a	b	c	d	e
value	15	9	16	12	17
Weight	2	3	4	5	6

	1	2	3	4	5	6	7	8	9	10	11	12
a	0	15	15	15	15	15	15	15	15	15	15	15
b	0	15	15	15	24	24	24	24	24	24	24	24
С												
d												
e												

Item	a	b	c	d	e
value	15	9	16	12	17
Weight	2	3	4	5	6

	1	2	3	4	5	6	7	8	9	10	11	12
a	0	15	15	15	15	15	15	15	15	15	15	15
b	0	15	15	15	24	24	24	24	24	24	24	24
С	0	15	15	15	24	31	31	31	40	40	40	40
d												
е												

Item	a	b	c	d	e	
value	15	9	16	12	17	
Weight	2	3	4	5	6	

	1	2	3	4	5	6	7	8	9	10	11	12
a	0	15	15	15	15	15	15	15	15	15	15	15
b	0	15	15	15	24	24	24	24	24	24	24	24
С	0	15	15	15	24	31	31	31	40	40	40	40
d	0	15	15	15	24	31	31	31	40	40	43	43
е												

Item	a	b	c	d	e	
value	15	9	16	12	17	
Weight	2	3	4	5	6	

*row* 5.

$$wt \ of \ e = 6. \ j - wt(e) = 12 - 6 = 6. \ V[5, \ 12] = max\{V[4, \ 12], \ V[4, \ 6] + v(e)\}$$

$$= max\{43, \ 31 + 17\} = 48$$
 $wt \ of \ e = 6. \ j - wt(e) = 11 - 6 = 5. \ V[5, \ 11] = max\{V[4, \ 11], \ V[4, \ 5] + v(e)\}$ 

$$= max\{43, \ 24 + 17\} = 43$$
 $Wt \ of \ e = 6. \ j - wt(e) = 8 - 6 = 2. \ V[5, \ 8] = max\{V[4, \ 8], \ V[4, \ 2] + v(e)\}$ 

$$= max\{31, \ 15 + 17\} = 32$$

	1	2	3	4	5	6	7	8	9	10	11	12
a	0	15	15	15	15	15	15	15	15	15	15	15
b	0				24	24	24	24	24	24	24	24
С	0					31	31	31	40	40	40	40
d	0										43	43
е	0							32				48

Item	a	b	c	d	e
value	15	9	16	12	17
Weight	2	3	4	5	6

Since A(e, 12) = 48 and A(d, 12) = 43, we conclude e is in the subset. Weight of e is 6. Hence we look in the column 12 - 6 = 6. Since A(c, 6) = 31 and A(b, 6) = 24, we conclude c is in the subset. Weight of c = 4. Hence we look in the column 6 - 4 = 2. Since A(a, 2) = 15. We conclude a is in the subset. Thus subset =  $\{a, c, e\}$ . Total weight = 12. Total Value = 48.

	1	2	3	4	5	6	7	8	9	10	11	12
a	0	15	15	15	15	15	15	15	15	15	15	15
b	0				24	24	24	24	24	24	24	24
С	0					31	31	31	40	40	40	40
d	0										43	43
е	0							32				48

#### Solution

Item	a	b	c	d	e
value	15	9	16	12	17
Weight	2	3	4	5	6

# Fractional Knapsack Problem (Greedy Algorithm)

Item	a	b	С	d	e
value	5	10	15	8	4
Weight	10	5	3	4	1

Total weight = 11.

Max Value approach: c, b, 0.75d.

$$15 + 10 + 6 = 31$$
.

Min Weight approach: e, c, d, 0.6b

$$4+15+8+6=29$$

**Value per weight**: a : 0.5, b : 2, c : 5, d : 2, e : 4

Select: c, e, b, 0.5d 15 + 4 + 10 + 4 = 33