BLG 336E – Analysis of Algorithms II Recitation IV

Dynamic Programming

Tuğba PAMAY

Istanbul Technical University – Department of Computer Engineering

18.04.2017

What is Dynamic Programming ?

Dynamic programming is efficient in finding **optimal solutions** for cases with lots of overlapping sub-problems.

It solves problems by:

recombining solutions of **sub-problems**, Some examples:

Fibonacci Numbers
Maximum Value Contiguous Subsequence
Balanced Partition
Minimum Edit Distance
Knapsack Problem

For more examples, click here.

Minimum Edit Distance

Given two strings: source and target

Set of editing operations that can **performed on source**

Find **minimum number of edits** (operations) required **to convert** source **into** target

Minimum Edit Distance

Pseudocode:

```
EDITDISTANCE(s1, s2)
int m[i, j] = 0
for i ← 1 to |s1|
do m[i, 0] = i
for j ← 1 to |s2|
do m[0, j] = j
for i ← 1 to |s1|
do for j ← 1 to |s2|
    do m[i, j] = min{m[i − 1, j − 1] + if (s1[i] = s2[j]) then 0 else 1fi,
                     m[i − 1, j] + 1,
                     m[i, j − 1] + 1}
return m[|s1|, |s2|]
```

String distance metrics: **Levenshtein**

Simple set of operations:

- Delete a character in **s**: cost 1
- Insert a character in **t**: cost 1
- Substitute one character for another: cost 1 (or cost 2)

Let’s start with an easy example…

Source word : AT
Target word : TO

Find the **minimum number of operations** to convert source word to target word.

Example

Target

Source

	null	T	O
null		insert	
A	delete	substitute	
T			

Example

- Initialize the array
- Insert "null" character at the beginning of the words

Target

Source

	null	T	O
null	0	1	2
A	1		
T	2		

Example (row #1)

Target

Source

	null	T	O
null	0	1	2
A	1	1	2
T	2		

Example (row #2)

Target

Source

	null	T	O
null	0	1	2
A	1	1	2
T	2	1	2

Example

Target

Source

	null	T	O
null	0	1	2
A	1	1	2
T	2	1	**2**

So, Now **backtrack** to find necessary operations for conversion.

Target

Source

	null	T	O
null	0	1	2
A	1	1	2
T	2	1	2

Scenario #1

So, Now **backtrack** to find necessary operations for conversion.

Target

Source

	null	T	O
null	0	1	2
A	1	1	2
T	2	1	**2**

Scenario #2

Another Example

Source word : SPARE
Target word : PAIR

Find the **minimum number of operations** to convert source word to target word.

D(i,j) = score of best alignment from **s1..si** to **t1..tj**

$$
\text{Min} = \begin{cases} D(i-1,j-1)+2\text{, if } si!=tj\text{, else } +0 & \text{// } \textbf{substitute} \\ D(i-1,j)+1 & \text{// } \textbf{insert} \\ D(i,j-1)+1 & \text{// } \textbf{delete} \end{cases}
$$

Example

- Initialize the array
- Insert "null" character at the beginning of the words

	null	P	A	I	R
null					
S					
P					
A					
R					
E					

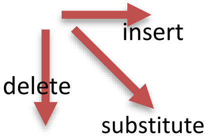
insert

delete

substitute

Example

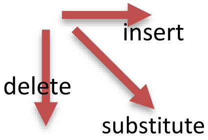
- Initialize the first row to 0...n
- Initialize the first column to 0...m



	null	P	A	I	R
null	0	1	2	3	4
S	1				
P	2				
A	3				
R	4				
E	5				

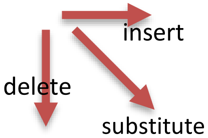
Example (row #2)

- Fill the second row
- For each cell, think that
  by which operation the minimum cost is achieved ???



	null	P	A	I	R
null	0	1	2	3	4
S	1	2	3	4	5
P	2				
A	3				
R	4				
E	5				

Example (row #3)

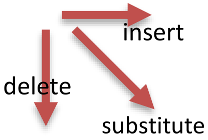


- Fill the third row
- For each cell, think that
  by which operation the minimum cost is achieved ???

	null	P	A	I	R
null	0	1	2	3	4
S	1	2	3	4	5
P	2	1	2	3	4
A	3				
R	4				
E	5				

Example (row #4)

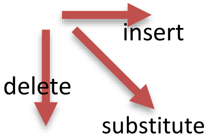
- Fill the fourth row
- For each cell, think that
  by which operation the minimum cost is achieved ???



	null	P	A	I	R
null	0	1	2	3	4
S	1	2	3	4	5
P	2	1	2	3	4
A	3	2	1	2	3
R	4				
E	5				

Example (row #5)

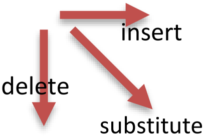
- Fill the fifth row
- For each cell, think that by which operation the minimum cost is achieved ???



	null	P	A	I	R
null	0	1	2	3	4
S	1	2	3	4	5
P	2	1	2	3	4
A	3	2	1	2	3
R	4	3	2	3	2
E	5				

Example (row #6)

- Fill the last row
- For each cell, think that
  by which operation the minimum cost is achieved ???



	null	P	A	I	R
null	0	1	2	3	4
S	1	2	3	4	5
P	2	1	2	3	4
A	3	2	1	2	3
R	4	3	2	3	2
E	5	4	3	4	3

Knapsack problem

- Given some items, pack the knapsack to get the maximum total value .
- Each item has some weight and some (benefit) value.
- Total weight that we can carry is no more than some fixed number W.

Item #	Weight	Value
1	1	8
2	3	6
3	5	5

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 benefit value b_i (all w_i and W are integer values)
- <u>Problem</u>: How to pack the knapsack to achieve maximum total value of packed items?

Knapsack Algorithm

```
for w = 0 to W
  V[0,w] = 0
for i = 1 to n
  V[i,0] = 0

for i = 1 to n
  for w = 0 to W
        if w_i <= w // item i can be part of the solution
        if b_i + V[i-1,w-w_i] > V[i-1,w]
                V[i,w] = b_i + V[i-1,w- w_i]
            else
                V[i,w] = V[i-1,w]
        else V[i,w] = V[i-1,w]  // w_i > w
```

Running time

```
for w = 0 to W
  V[0,w] = 0
for i = 1 to n
  V[i,0] = 0
for i = 1 to n
  for w = 0 to W
      < the rest of the code >
```

O(W)

Repeat *n* times

O(W)

What is the running time of this algorithm?

O(n*W)

Remember that the brute-force algorithm takes $O(2^n)$

Example

Let’s run our algorithm on the following data:

n = 4 (# of elements)
W = 5 (max weight)
Elements (weight, benefit):
(2,3), (3,4), (4,5), (5,6)

Example (2)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1						
2						
3						
4						

for w = 0 to W

    V[0,w] = 0

Example (3)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0					
2	0					
3	0					
4	0					

for i = 1 to n

    V[i,0] = 0

Example (4)

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0				
2	0					
3	0					
4	0					

$i=1$

$b_i=3$

$w_i=2$

$w=1$

$w-w_i=-1$

if $w_i <= w$ // item i can be part of the solution

if $b_i + V[i-1,w-w_i] > V[i-1,w]$
$V[i,w] = b_i + V[i-1,w- w_i]$ else
$V[i,w] = V[i-1,w]$

else **V[i,w] = V[i-1,w]** // $w_i > w$

Example (5)

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	**3**			
2	0					
3	0					
4	0					

i=1

$b_i=3$

$w_i=2$

$w=2$

$w-w_i=0$

if $w_i <= w$ // item i can be part of the solution

  if $b_i + V[i-1,w-w_i] > V[i-1,w]$

    $V[i,w] = b_i + V[i-1,w-w_i]$

  else

    $V[i,w] = V[i-1,w]$

else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (6)

Items:
1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	**3**		
2	0					
3	0					
4	0					

$i=1$
$b_i=3$
$w_i=2$
$w=3$
$w-w_i=1$

if $w_i <= w$ // item i can be part of the solution
    if $b_i + V[i-1,w-w_i] > V[i-1,w]$
        $V[i,w] = b_i + V[i-1,w-w_i]$
    else
        $V[i,w] = V[i-1,w]$
else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (7)

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	**3**	
2	0					
3	0					
4	0					

$i=1$

$b_i=3$

$w_i=2$

$w=4$

$w-w_i=2$

if $w_i <= w$ // item i can be part of the solution

    if $b_i + V[i-1,w-w_i] > V[i-1,w]$

        $\mathbf{V[i,w] = b_i + V[i-1,w-w_i]}$

    else

        $V[i,w] = V[i-1,w]$

else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (8)

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	**3**
2	0					
3	0					
4	0					

i=1

b_i=3

w_i=2

w=5

w-w_i =3

if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w] // w_i > w

Example (9)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	**0**				
3	0					
4	0					

i=2

b_i=4

w_i=3

w=1

$w-w_i$ =-2

if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else **V[i,w] = V[i-1,w]** // w_i > w

Example (10)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	**3**			
3	0					
4	0					

$i=2$

$b_i=4$

$w_i=3$

$w=2$

$w-w_i=-1$

if w_i <= w // item i can be part of the solution
  if $b_i + V[i-1,w-w_i] > V[i-1,w]$
    $V[i,w] = b_i + V[i-1,w-w_i]$
  else
    $V[i,w] = V[i-1,w]$
else **V[i,w] = V[i-1,w]** // $w_i > w$

Example (11)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	**4**		
3	0					
4	0					

i=2

$b_i=4$

$w_i=3$

w=3

$w-w_i=0$

if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w] // w_i > w

Example (12)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	**4**	
3	0					
4	0					

$i=2$

$b_i=4$

$w_i=3$

$w=4$

$w-w_i=1$

if $w_i <= w$ // item i can be part of the solution
    if $b_i + V[i-1,w-w_i] > V[i-1,w]$
        $V[i,w] = b_i + V[i-1,w- w_i]$
    else
        $V[i,w] = V[i-1,w]$
else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (13)

Items:

1: (2,3)

2: (3,4)

3: (4,5)

4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	**7**
3	0					
4	0					

$i=2$

$b_i=4$

$w_i=3$

$w=5$

$w-w_i=2$

```
if wi <= w // item i can be part of the solution
    if bi + V[i-1,w-wi] > V[i-1,w]
        V[i,w] = bi + V[i-1,w- wi]
    else
        V[i,w] = V[i-1,w]
else V[i,w] = V[i-1,w]  // wi > w
```

Example (14)

Items:
1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	**0**	**3**	**4**		
4	0					

i=3
b_i=5
w_i=4
w= 1..3

if w_i <= w // item i can be part of the solution
    if b_i + V[i-1,w-w_i] > V[i-1,w]
        V[i,w] = b_i + V[i-1,w- w_i]
    else
        V[i,w] = V[i-1,w]
else **V[i,w] = V[i-1,w]** // w_i > w

Example (15)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	**5**	
4	0					

$i=3$
$b_i=5$
$w_i=4$
$w=4$
$w-w_i=0$

if $w_i <= w$ // item i can be part of the solution
    if $b_i + V[i-1,w-w_i] > V[i-1,w]$
        $\mathbf{V[i,w] = b_i + V[i-1,w-w_i]}$
    else
        $V[i,w] = V[i-1,w]$
else $V[i,w] = V[i-1,w]$ // $w_i > w$

Example (16)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	**7**
4	0					

$i=3$

$b_i=5$

$w_i=4$

$w= 5$

$w- w_i=1$

if w_i <= w // item i can be part of the solution

  if $b_i + V[i-1,w-w_i] > V[i-1,w]$

    $V[i,w] = b_i + V[i-1,w- w_i]$

  else

    V[i,w] = V[i-1,w]

else V[i,w] = V[i-1,w] // $w_i > w$

Example (17)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	**0**	**3**	**4**	**5**	

$i=4$

$b_i=6$

$w_i=5$

$w= 1..4$

if w_i <= w // item i can be part of the solution
  if $b_i + V[i-1,w-w_i] > V[i-1,w]$
    $V[i,w] = b_i + V[i-1,w- w_i]$
  else
    $V[i,w] = V[i-1,w]$
else **V[i,w] = V[i-1,w]** // $w_i > w$

Example (18)

Items:

1: (2,3)
2: (3,4)
3: (4,5)
4: (5,6)

i\W	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	3	3	3	3
2	0	0	3	4	4	7
3	0	0	3	4	5	7
4	0	0	3	4	5	**7**

$i=4$
$b_i=6$
$w_i=5$
$w= 5$

$w- w_i=0$

if $w_i <= w$ // item i can be part of the solution
    if $b_i + V[i-1,w-w_i] > V[i-1,w]$
      $V[i,w] = b_i + V[i-1,w- w_i]$
    else
      V[i,w] = V[i-1,w]
else $V[i,w] = V[i-1,w]$ // $w_i > w$

Result:

Chosen items: 1 & 2
Total Benefit : 7
Total Weight: 5