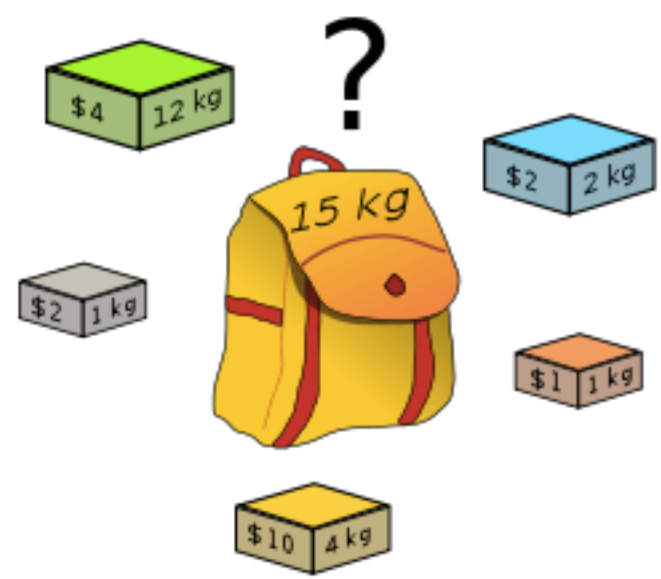
Lecture – 38-39

Dynamic Programming Approaches

- ✓ Knapsack problem

(A dynamic programming algorithm for 0/1 knapsack)

- ✓ LCS (HW)



Example: Fibonacci numbers

$F(n) = F(n\text{-}1) + F(n\text{-}2)$
$F(0) = 0$
$F(1) = 1$

Computing the n^{th} Fibonacci number recursively (top-down):

$F(n)$

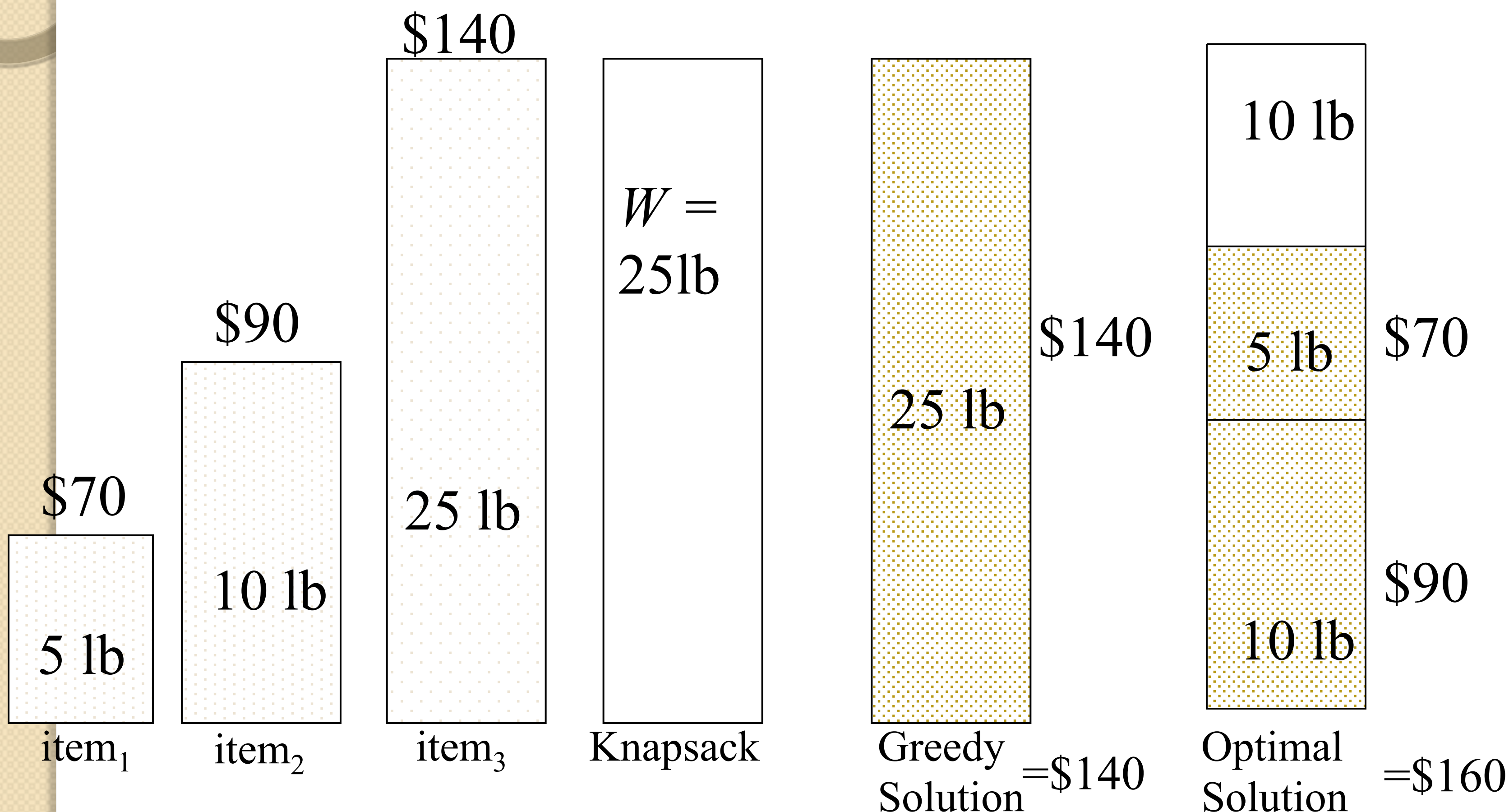
$F(n\text{-}1) \quad + \quad F(n\text{-}2)$

$F(n\text{-}2) \quad + \quad F(n\text{-}3) \qquad F(n\text{-}3) \quad + \quad F(n\text{-}4)$

…

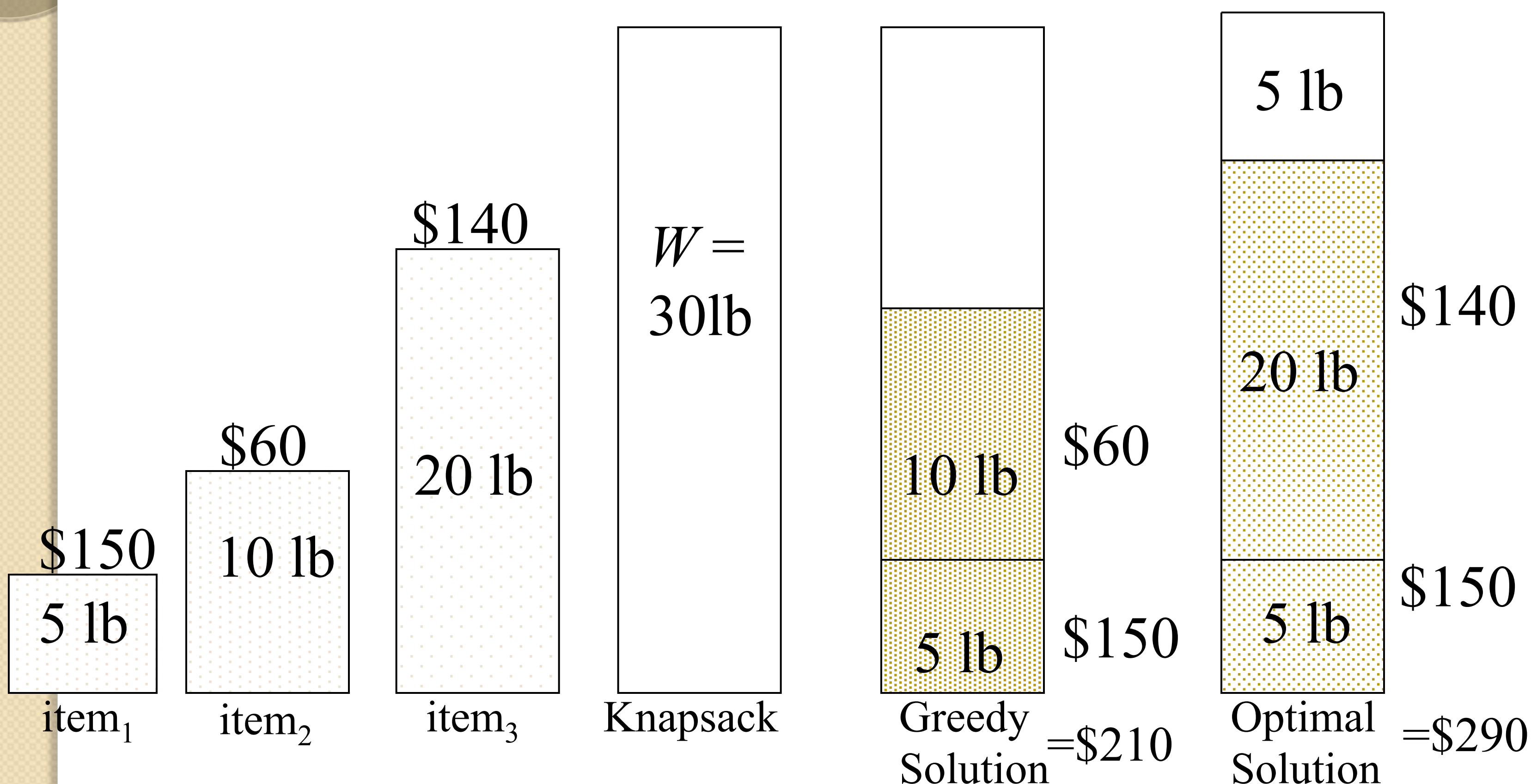
Selection criteria: *Maximum beneficial* item.
Counter Example:

$S = \{ (item_1 , 5, \$70), (item_2 ,10, \$90), (item_3, 25, \$140) \}$



Selection criteria: *Minimum weight* item Counter Example:

$S = \{ (item_1 , 5, \$150), (item_2 ,10, \$60), (item_3, 20, \$140) \}$



The 0/1 Knapsack problem

- **Given a knapsack with weight $W > 0$.**
- **A set S of n items with weights $w_i > 0$ and benefits $b_i > 0$ for $i = 1, \ldots, n$.**
- **$S = \{ (item_1, w_1, b_1), (item_2, w_2, b_2), \ldots, (item_n, w_n, b_n) \}$**
- **Find a subset of the items which does not exceed the weight W of the knapsack and maximizes the benefit.**

0/1 Knapsack problem

Determine a subset *A* of { 1, 2, …, *n* } that satisfies the following:

$$\max \sum_{i \in A} b_i \text{ where } \sum_{i \in A} w_i \leq W$$

In 0/1 knapsack a specific item is either selected or not

Knapsack Problem by DP

- Given n items of

  integer weights: $w_1 \quad w_2 \quad \ldots \quad w_n$

  values/benefits/profit: $v_1 \quad v_2 \quad \ldots \quad v_n$

  A knapsack of 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 ($j \leq W$).Let $V[i,j]$ be optimal value of such an instance. Then

$$V[i,j] = \begin{cases} \max \{V[i-1, j], v_i + V[i-1, j - w_i]\} & \text{if } j - w_i \geq 0 \\ V[i-1, j] & \text{if } j - w_i < 0 \end{cases}$$

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

//If any cell value is not define leave it

Knapsack Problem by DP (pseudocode)

```
Algorithm DPKnapsack(w[1..n], v[1..n], W)
  var V[0..n,0..W], P[1..n,1..W]: int
  for j := 0 to W do
        V[0,j] := 0
    for i := 0 to n do
          V[i,0] := 0
    for i := 1 to n do
        for j := 1 to W do
               if  w[i] ≤ j and v[i] + V[i-1,j-w[i]] > V[i-1,j] then
                      V[i,j] := v[i] + V[i-1,j-w[i]]; P[i,j] := j-w[i]
               else
                      V[i,j] := V[i-1,j]; P[i,j] := j
  return V[n,W] and the optimal subset by backtracing
```

Running time and space: O(nW).

Given *4* items Bag weight = 8

Weights: *2 3 4 5*

Values: *1 2 5 6*

			0	1	2	3	4	5	6	7	8
V	W	0	0	0	0	0	0	0	0	0	0
1	2	1	0								
2	3	2	0								
5	4	3	0								
6	5	4	0								

Given *4* items Bag weight = 8

Integer Weights: *2 3 4 5*

Values: *1 2 5 6*

V	W		0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	**1**	1	1	1	1	1	1
2	3	2	0								
5	4	3	0								
6	5	4	0								

Given *4* items Bag weight = 8

Integer Weights: *2 3 4 5*

Values: *1 2 5 6*

			0	1	2	3	4	5	6	7	8
V	W	0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	**1**	1	1	1	1	1	1
2	3	2	0	0	1	**2**					
5	4	3	0								
6	5	4	0								

Given *4* items Bag weight = 8

Integer Weights: *2 3 4 5*

Values: *1 2 5 6*

			0	1	2	3	4	5	6	7	8
V	W	0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	**1**	1	1	1	1	1	1
2	3	2	0	0	1	**2**		**3**			
5	4	3	0								
6	5	4	0								

Given *4* items Bag weight = 8

Weights: *2 3 4 5*

Values: *1 2 5 6*

V	W		0	1	2	3	4	5	6	7	8
		0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	**1**	1	1	1	1	1	1
2	3	2	0	0	1	**2**	2	**3**	3	3	3
5	4	3	0								
6	5	4	0								

Given *4* items **Bag weight = 8**

Weights: *2 3 4 5*

Values: *1 2 5 6*

			0	1	2	3	4	5	6	7	8
V	W	0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	**1**	1	1	1	1	1	1
2	3	2	0	0	1	**2**	2	**3**	3	3	3
5	4	3	0	0	1	2	**5**				
6	5	4	0								

Given *4* items Bag weight = 8

Weights: *2 3 4 5*

Values: *1 2 5 6*

			0	1	2	3	4	5	6	7	8
V	W	0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	**1**	1	1	1	1	1	1
2	3	2	0	0	1	**2**	2	**3**	3	3	3
5	4	3	0	0	1	2	**5**		**6**		
6	5	4	0								

Given *4* items **Bag weight = 8**

Weights: *2 3 4 5*

Values: *1 2 5 6*

V	W		0	1	2	3	4	5	6	7	8
V	W	0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	**1**	1	1	1	1	1	1
2	3	2	0	0	1	**2**	2	**3**	3	3	3
5	4	3	0	0	1	2	**5**		**6**	**7**	
6	5	4	0								

Given *4* items Bag weight = 8

Weights: *2 3 4 5*

Values: *1 2 5 6*

V	W		0	1	2	3	4	5	6	7	8
V	W	0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	**1**	1	1	1	1	1	1
2	3	2	0	0	1	**2**	2	**3**	3	3	3
5	4	3	0	0	1	2	**5**	5	**6**	**7**	7
6	5	4	0								

Given *4* items Bag weight = 8

Weights: *2 3 4 5*

Values: *1 2 5 6*

V	W		0	1	2	3	4	5	6	7	8
V	W	0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	**1**	1	1	1	1	1	1
2	3	2	0	0	1	**2**	2	**3**	3	3	3
5	4	3	0	0	1	2	**5**	5	**6**	**7**	7
6	5	4	0	0	1	2	5	**6**			

Given *4* items Bag weight = 8

Weights: *2 3 4 5*

Values: *1 2 5 6*

V	W		0	1	2	3	4	5	6	7	8
V	W	0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	**1**	1	1	1	1	1	1
2	3	2	0	0	1	**2**	2	**3**	3	3	3
5	4	3	0	0	1	2	**5**	6	**6**	**7**	7
6	5	4	0	0	1	2	5	**6**			**8**

Given *4* items Bag weight = 8

Weights: *2 3 4 5*

Values: *1 2 5 6*

V	W		0	1	2	3	4	5	6	7	8
V	W	0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	**1**	1	1	1	1	1	1
2	3	2	0	0	1	**2**	2	**3**	3	3	3
5	4	3	0	0	1	2	**5**	6	**6**	**7**	7
6	5	4	0	0	1	2	5	**6**	6	7	**8**

			0	1	2	3	4	5	6	7	8
V	W	0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	**1**	1	1	1	1	1	1
2	3	2	0	0	1	**2**	2	**3**	3	3	3
5	4	3	0	0	1	2	**5**	6	**6**	**7**	7
6	5	4	0	0	1	2	5	**6**	6	7	**8**

V	W		0	1	2	3	4	5	6	7	8
V	W	0	0	0	0	0	0	0	0	0	0
1	2	1	0	0	**1**	1	1	1	1	1	1
2	3	2	0	0	1	**2**	2	**3**	3	3	3
5	4	3	0	0	1	2	**5**	6	**6**	**7**	7
6	5	4	0	0	1	2	5	**6**	6	7	**8**

	x1	x2	x3	x4
Sol =	0	1	0	1

Assignment

Example: Knapsack of capacity W = 5 and 9

item	weight	value
1	2	$12
2	1	$10
3	3	$20
4	2	$15

Soln

Knapsack Problem by DP (example)

Example: Knapsack of capacity $W = 5$

item	weight	value
1	2	\$12
2	1	\$10
3	3	\$20
4	2	\$15

capacity j

		0	1	2	3	4	5
	0	0	0	0			
$w_1 = 2, v_1 = 12$	1	0	0	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

Backtracing finds the actual optimal subset, i.e. solution.

Longest Common Subsequence (LCS)

- A subsequence of a sequence/string *S* is obtained by deleting zero or more symbols from *S*. For example, the following are some subsequences of "president": pred, sdn, predent. In other words, the letters of a subsequence of S appear in order in *S*, but they are not required to be consecutive.

- The longest common subsequence problem is to find a maximum length common subsequence between two sequences.

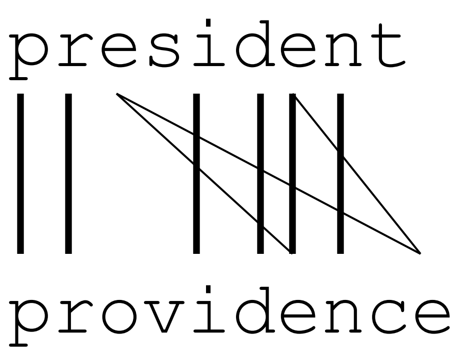
LCS

For instance,

Sequence 1: president

Sequence 2: providence

Its LCS is priden.



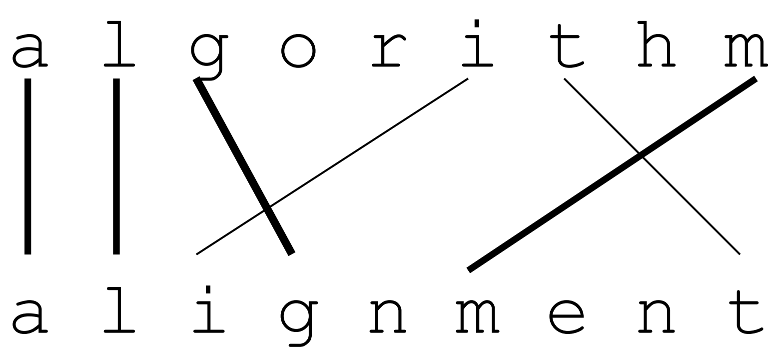
LCS

Another example:

Sequence 1: algorithm

Sequence 2: alignment

One of its LCS is algm.



How to compute LCS?

- Let A=$a_1a_2\ldots a_m$ and B=$b_1b_2\ldots b_n$.
- *len*(*i, j*): the length of an LCS between
  $a_1a_2\ldots a_i$ and $b_1b_2\ldots b_j$
- With proper initializations, *len*(*i, j*) can be computed as follows.

i	*j* *0*	*1* *p*	*2* *r*	*3* *o*	*4* *v*	*5* *i*	*6* *d*	*7* *e*	*8* *n*	*9* *c*	*10* *e*
0	0	0	0	0	0	0	0	0	0	0	0
1 *p*	0	↖ 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
2 *r*	0	↑ 1	↖ 2	← 2	← 2	← 2	← 2	← 2	← 2	← 2	← 2
3 *e*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2	↖ 3	← 3	← 3	↖ 3
4 *s*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2	↑ 3	↑ 3	↑ 3	↑ 3
5 *i*	0	↑ 1	↑ 2	↑ 2	↑ 2	↖ 3	← 3	↑ 3	↑ 3	↑ 3	↑ 3
6 *d*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↖ 4	← 4	← 4	← 4	← 4
7 *e*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↖ 5	← 5	← 5	↖ 5
8 *n*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↑ 5	↖ 6	← 6	← 6
9 *t*	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↑ 5	↑ 6	↑ 6	↑ 6

Running time and memory: O(mn) and O(mn).

i	j 0	1 p	2 r	3 o	4 v	5 i	6 d	7 e	8 n	9 c	10 e
0	0	0	0	0	0	0	0	0	0	0	0
1 p	0	↖ 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1	← 1
2 r	0	↑ 1	↖ 2	← 2	← 2	← 2	← 2	← 2	← 2	← 2	← 2
3 e	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2	↖ 3	← 3	← 3	↖ 3
4 s	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 2	↑ 2	↑ 3	↑ 3	↑ 3	↑ 3
5 i	0	↑ 1	↑ 2	↑ 2	↑ 2	↖ 3	← 3	↑ 3	↑ 3	↑ 3	↑ 3
6 d	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↖ 4	← 4	← 4	← 4	← 4
7 e	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↖ 5	← 5	← 5	↖ 5
8 n	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↑ 5	↖ 6	← 6	← 6
9 t	0	↑ 1	↑ 2	↑ 2	↑ 2	↑ 3	↑ 4	↑ 5	↑ 6	↑ 6	↑ 6

Output: *priden*