Week 11 – S01
Dynamic Programming contd.

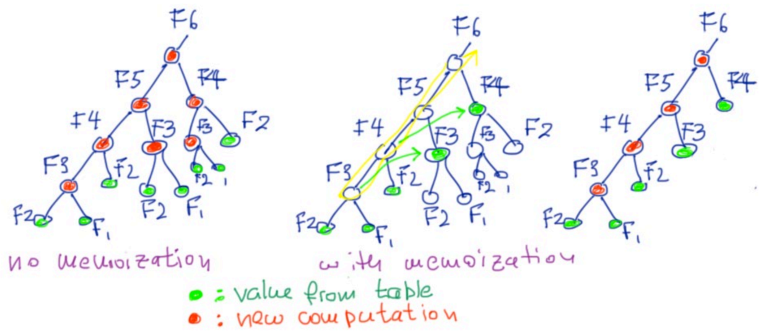
50.004 Introduction to Algorithms

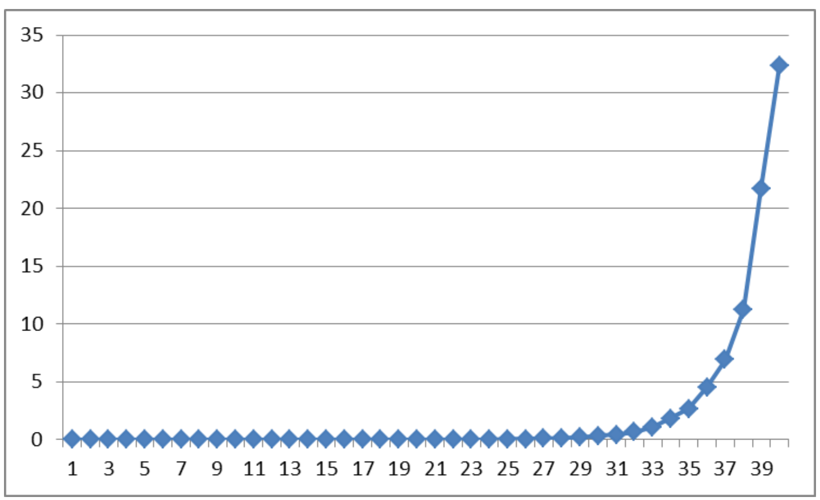
Dr. Subhajit Datta

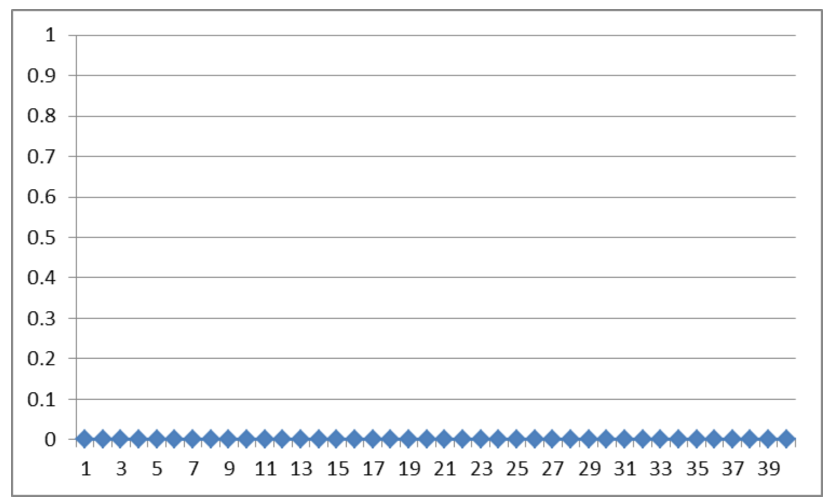
ISTD, SUTD

Memoization

- Store values of sub-problems in table, to avoid computing them twice



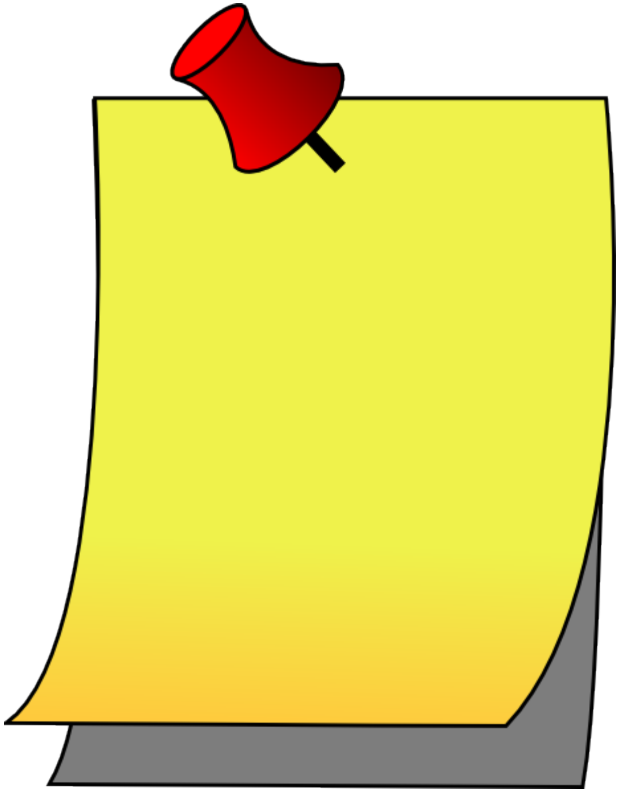


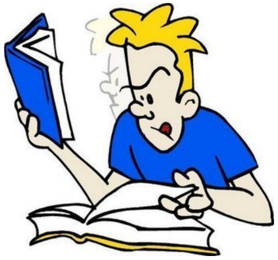


Complexity: $\Theta(2^n)$

Complexity: $\Theta(n)$

DP: Memoization *not* memorization





Fibonacci numbers

$$F_1 = F_2 = 1,\ F_n = F_{n-1} + F_{n-2}$$

Naïve algorithm

fib(n):

if $n \le 2 : f = 1$

else $f = \text{fib}(n-1) + \text{fib}(n-2)$

return f

<u>Memoized</u> DP algorithm

memo = {}

fib(n):

if n in memo: return memo[n]

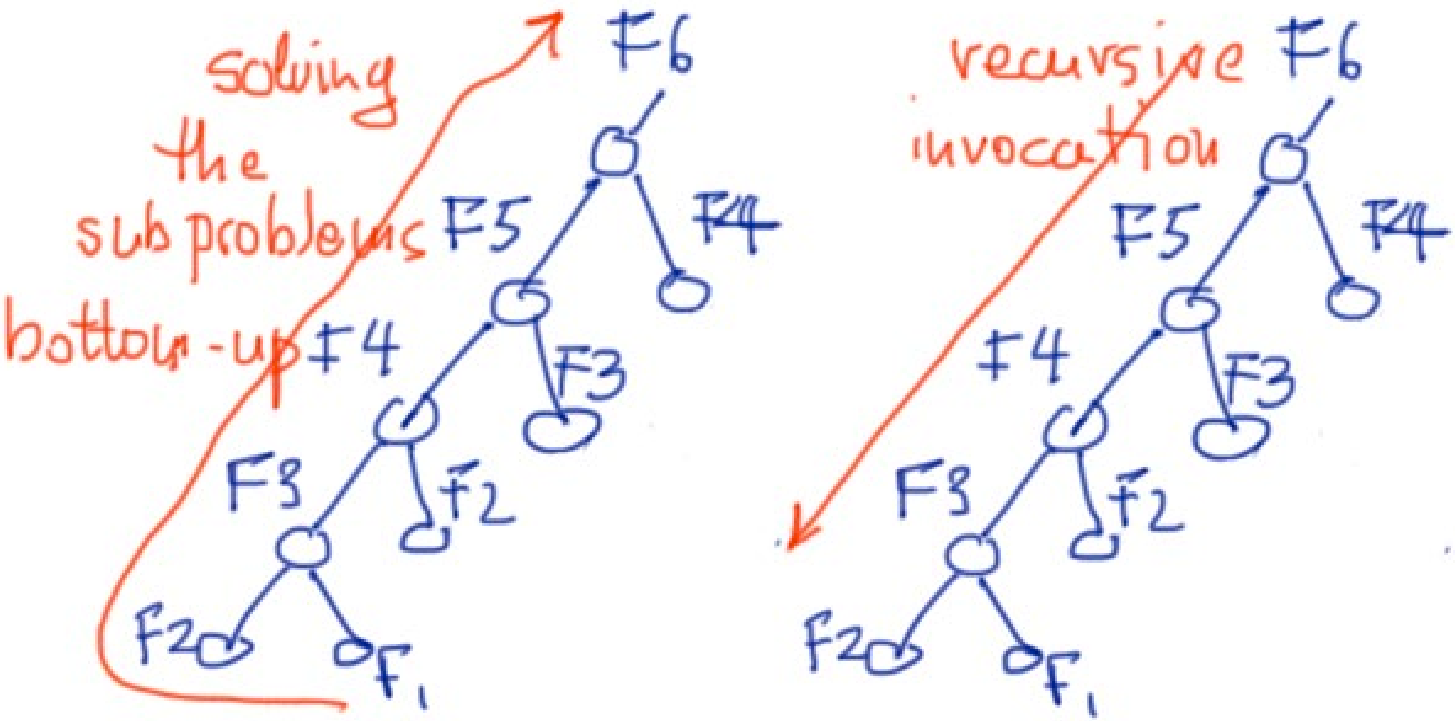
if $n \le 2 : f = 1$

else $f = \text{fib}(n-1) + \text{fib}(n-2)$

memo$[n] = f$

return f

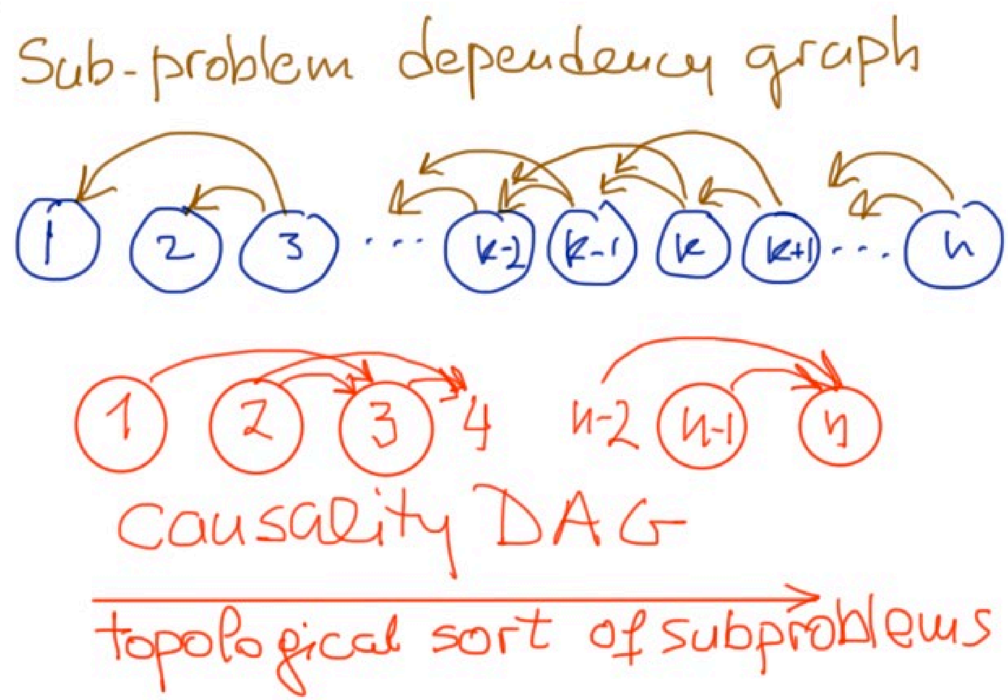
Top-down versus bottom-up



Bottom-up method:
Solve sub-problems in a "causal" sequence from smaller to larger

Bottom-up DP algorithm

```
fib={}
for k in [1,2,...,n]:
    if k ≤ 2 : f = 1
    else:  f = fib[k-1]+fib[k-2]
    fib[k]=f
return fib[n]
```



- Exactly the same computation as memoized DP (recursion rolled back)
- In general: topological sort of sub-problem dependency graph => needs DAG structure!!
- **What are the benefits?**
  - Practically faster (no recursion)
  - Can save space (just remember last 2 values) => Θ(1) space

Bottom up: filling-in a table

- Fill-in the table of sub-problems

Obey the dependency ordering

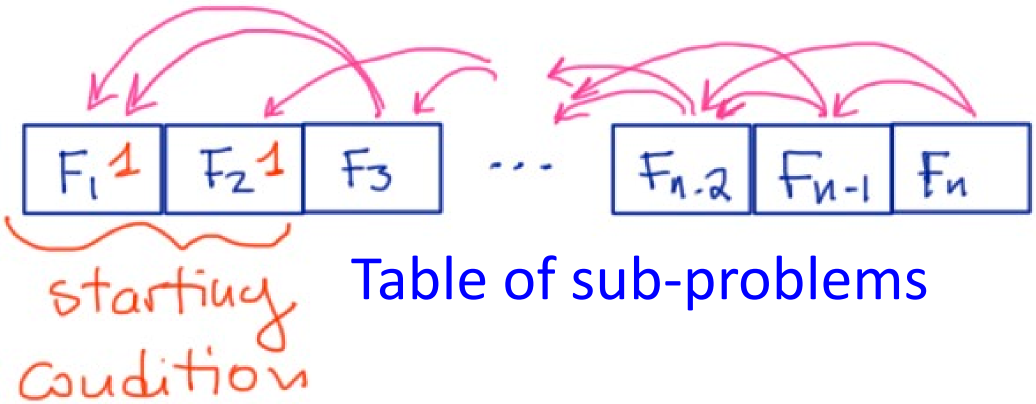


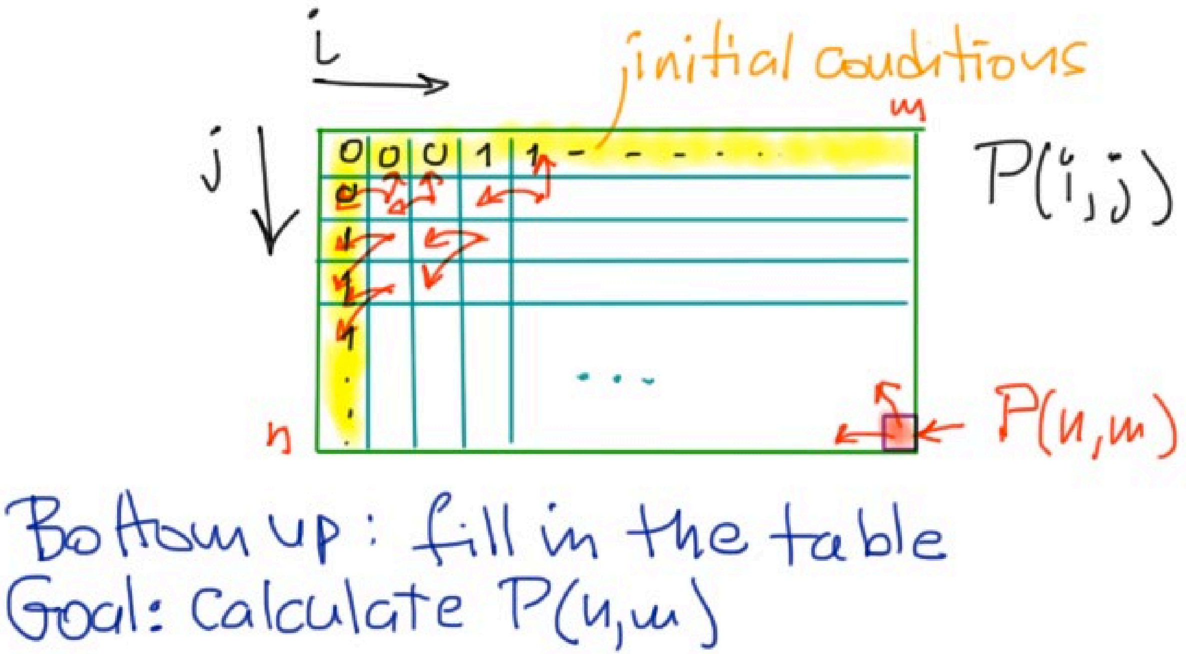
Table of sub-problems

Note: There should be no circular dependency between sub-problems

Bottom up: filling-in a table

Example with 2 dimensions

sub-problem is $P(i, j)$, hence 2-d table



Note: There should be no circular dependency between sub-problems

Fibonacci: Bottom-up (Iterative)

```
table = {}
def fiboBottomUp(n):
    for i in range(1,n+1):
        if i <= 2:
            fibo = 1
        else:
            fibo = table[i-1]+table[i-2]
        table[i] = fibo
    return table[n]
```

Dynamic programming: Steps

1. Define sub-problems
2. Guess (part of the solution)
3. Relate sub-problem solutions
4. Recurse plus memoize
   – Or, Build table bottom up (if there are no circular dependencies)
5. Solve original problem by combining sub-problem solutions

Elements of dynamic programming

Optimal substructure
The solution to a problem can be obtained by solutions to sub-problems with no circular dependency

Overlapping sub-problems
A recursive solution contains a "small" number of distinct sub-problems (repeated many times)

DP relies on sub-problems being independent and overlapping: A contradiction?

- Two sub-problems of the same problem are independent if they do not share resources
- Two sub-problems are overlapping if they are really the same sub-problem that occurs as a sub-problem of different problems
- So, this is not a contradiction!

Example

1

7 5

8 3 5

4 5 7 4

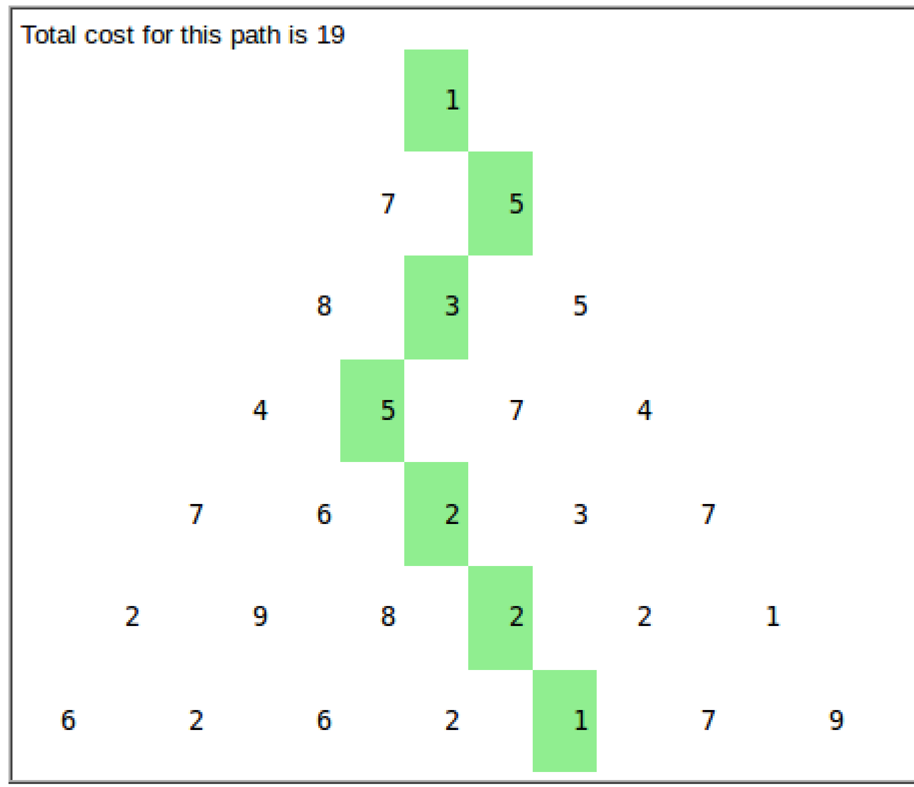
7 6 2 3 7

2 9 8 2 2 1

6 2 6 2 1 7 9

One path connecting top to bottom

1
7 5
8 3 5
4 5 7 4
7 6 2 3 7
2 9 8 2 2 1
6 2 6 2 1 7 9



What is this approach called?

1

7 5

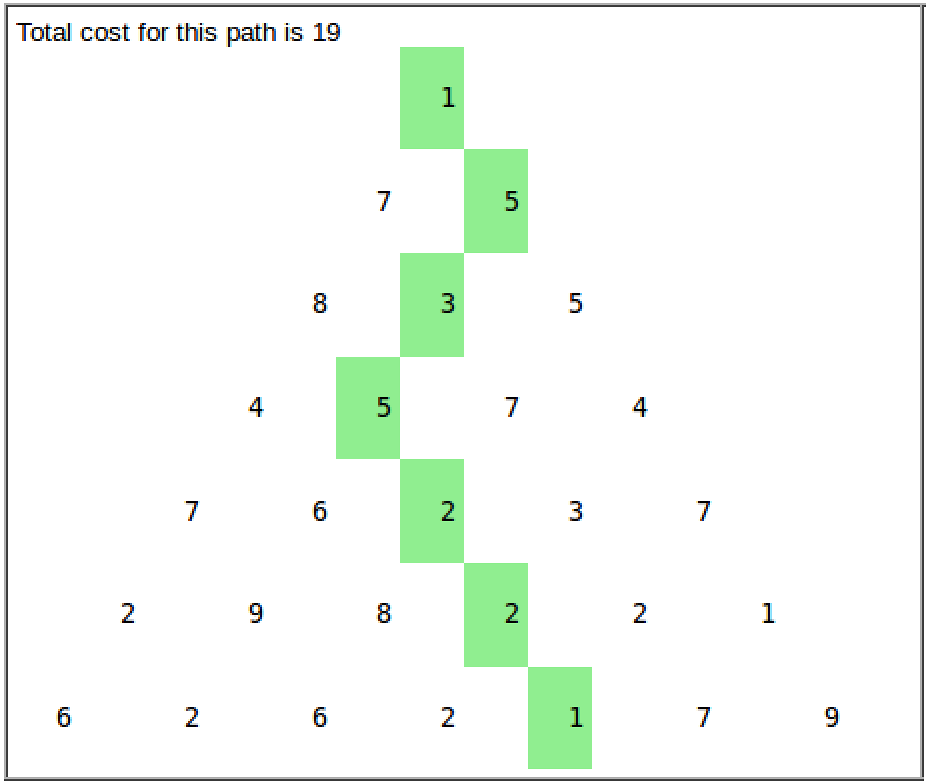
8 3 5

4 5 7 4

7 6 2 3 7

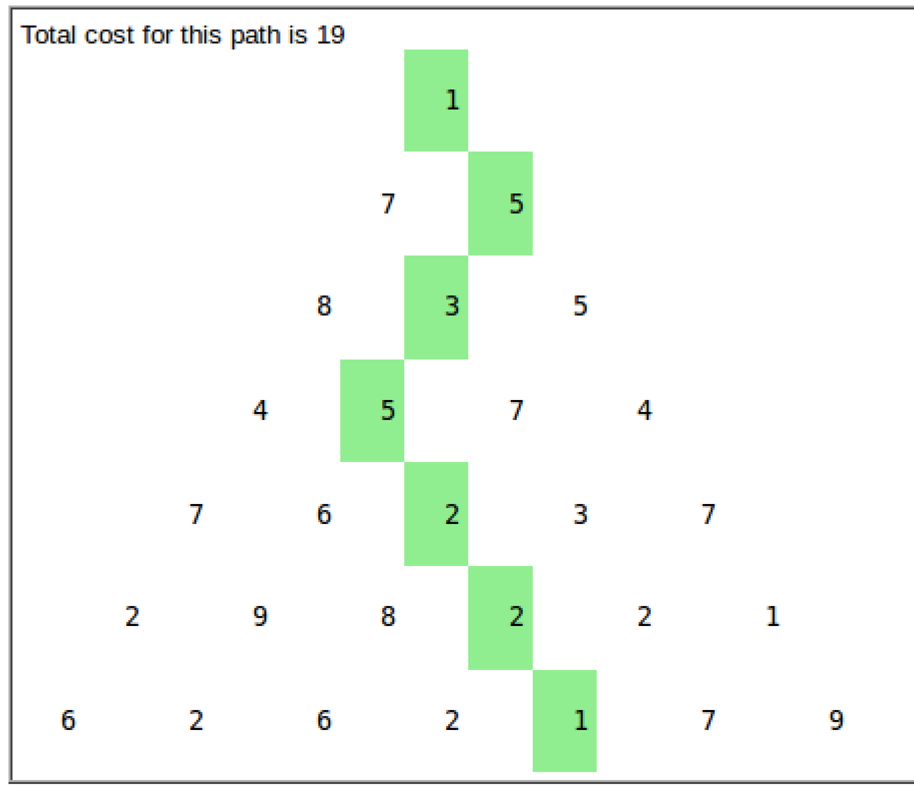
2 9 8 2 2 1

6 2 6 2 1 7 9



Greedy approach!

1
7 5
8 3 5
4 5 7 4
7 6 2 3 7
2 9 8 2 2 1
6 2 6 2 1 7 9



Find the path from top to the bottom with the least cost

Individual costs at each step

1

8 7

1 5 3

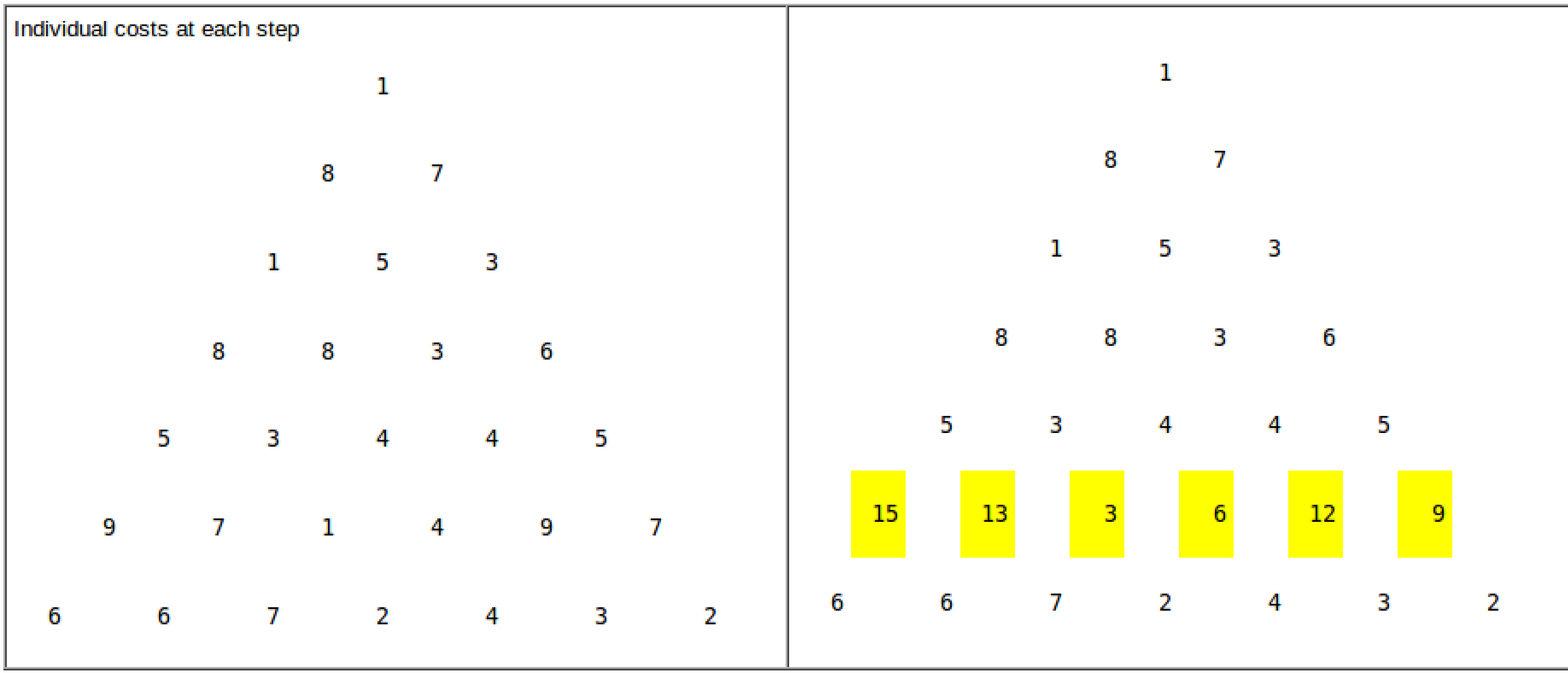
8 8 3 6

5 3 4 4 5

9 7 1 4 9 7

6 6 7 2 4 3 2

Solution steps



Solution steps

Individual costs at each step

1

8 7

1 5 3

8 8 3 6

5 3 4 4 5

9 7 1 4 9 7

6 6 7 2 4 3 2

1

8 7

1 5 3

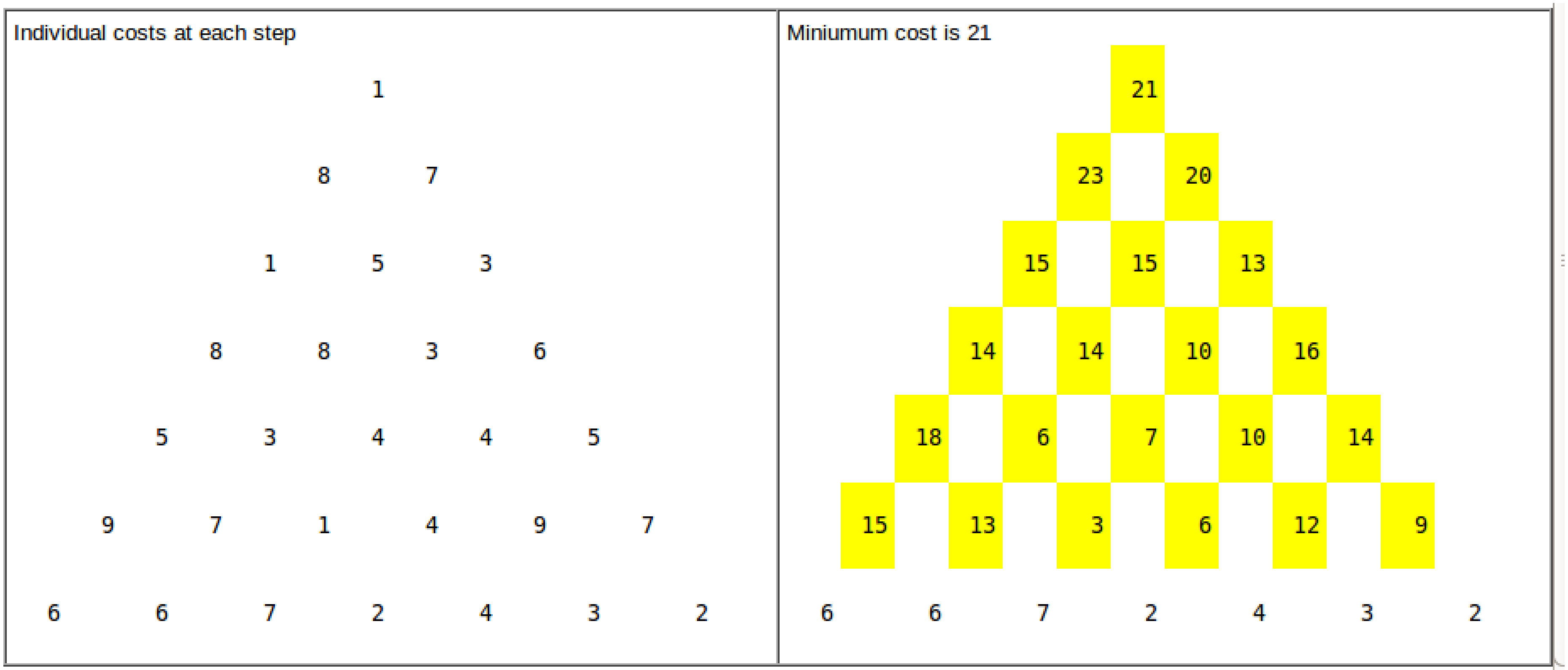
8 8 3 6

18 6 7 10 14

15 13 3 6 12 9

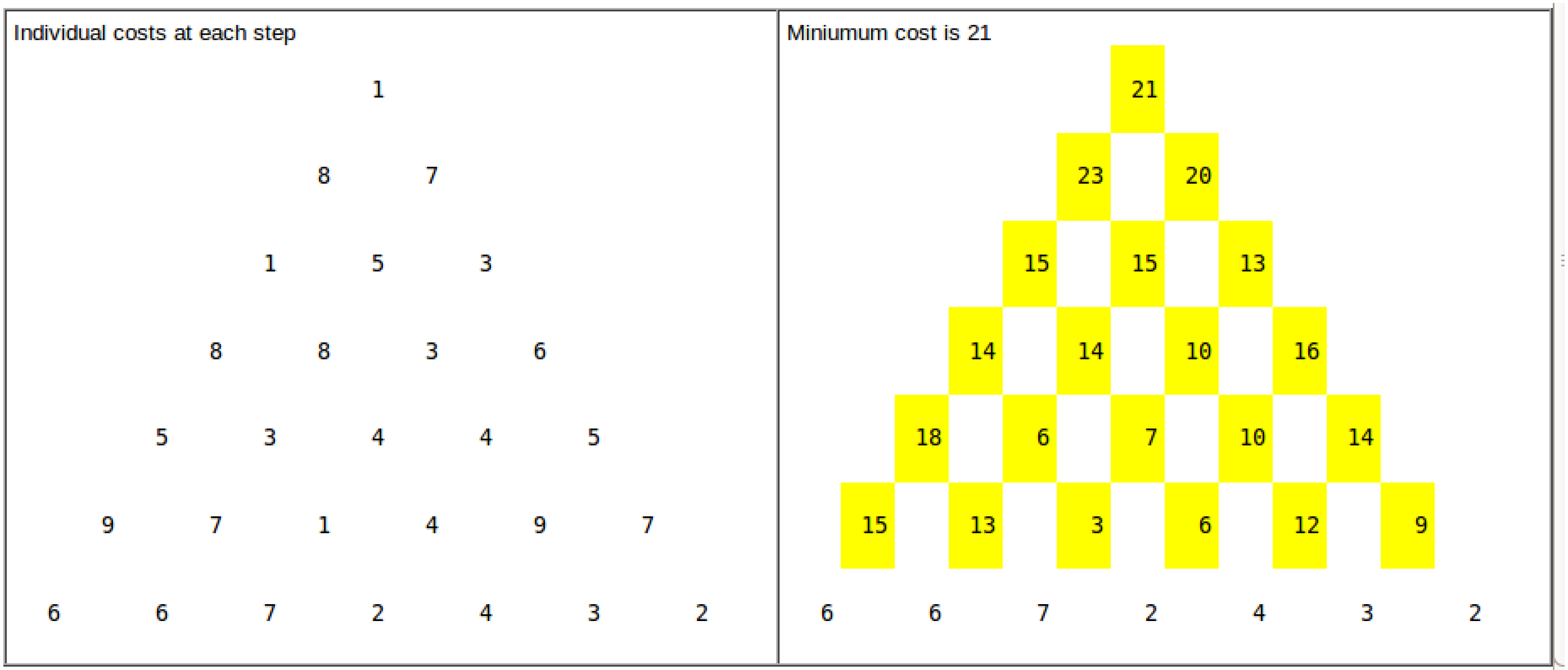
6 6 7 2 4 3 2

Solution

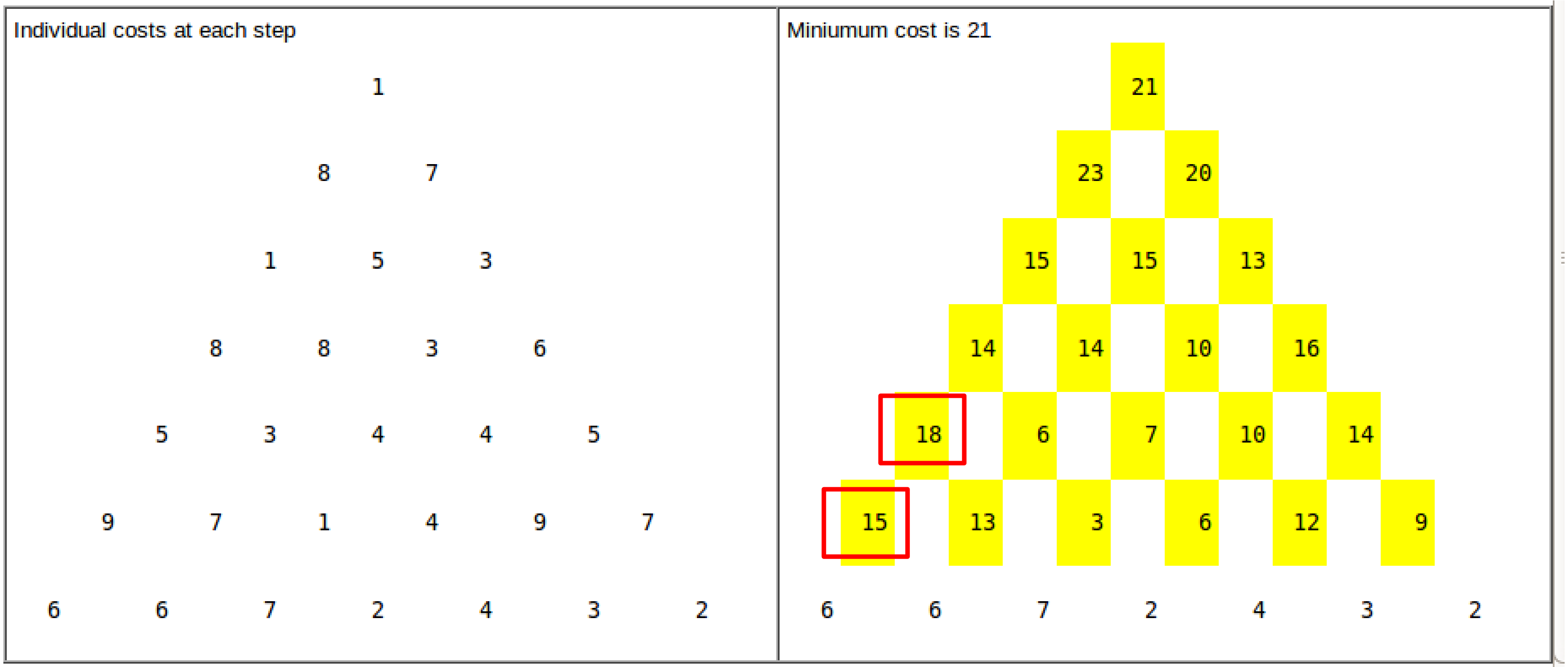


Individual costs at each step
1
8 7
1 5 3
8 8 3 6
5 3 4 4 5
9 7 1 4 9 7
6 6 7 2 4 3 2
Miniumum cost is 21
21
23 20
15 15 13
14 14 10 16
18 6 7 10 14
15 13 3 6 12 9
6 6 7 2 4 3 2

Where is dynamic programming here?

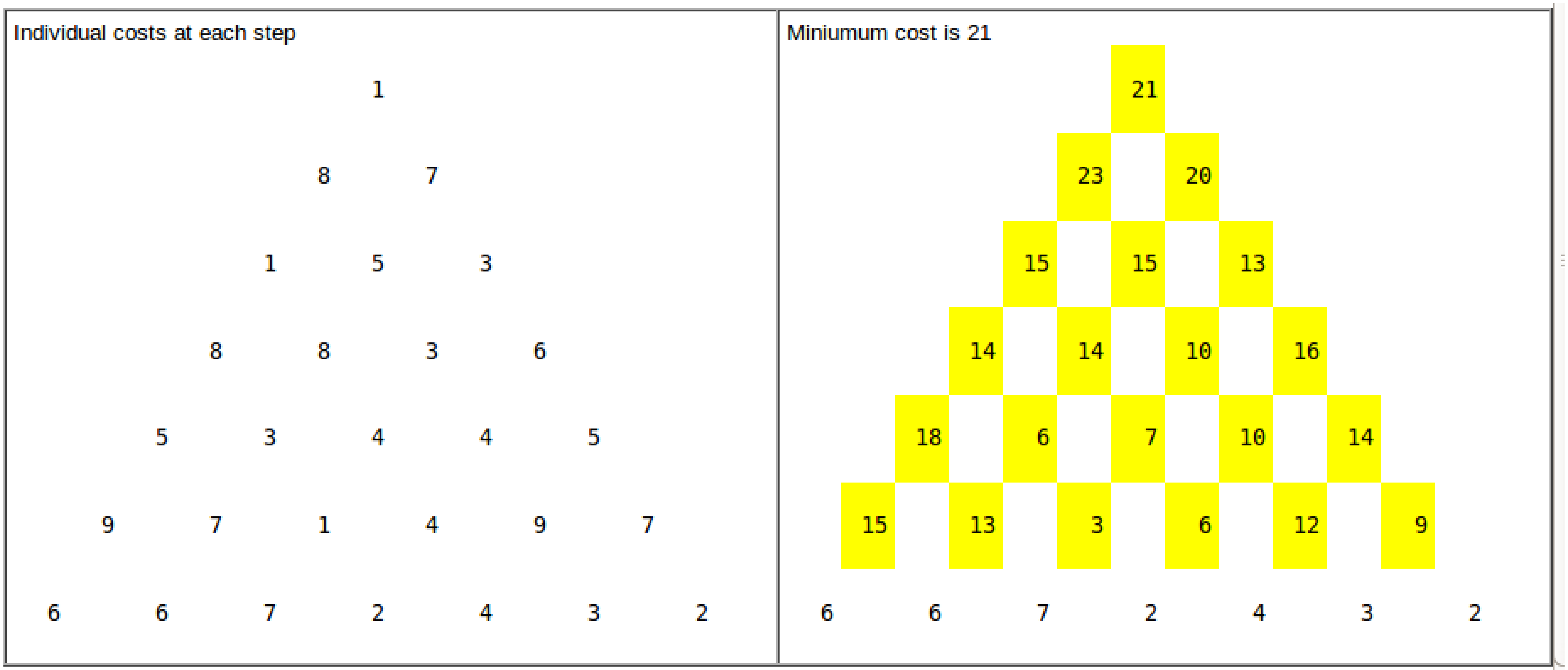


Where is dynamic programming here?



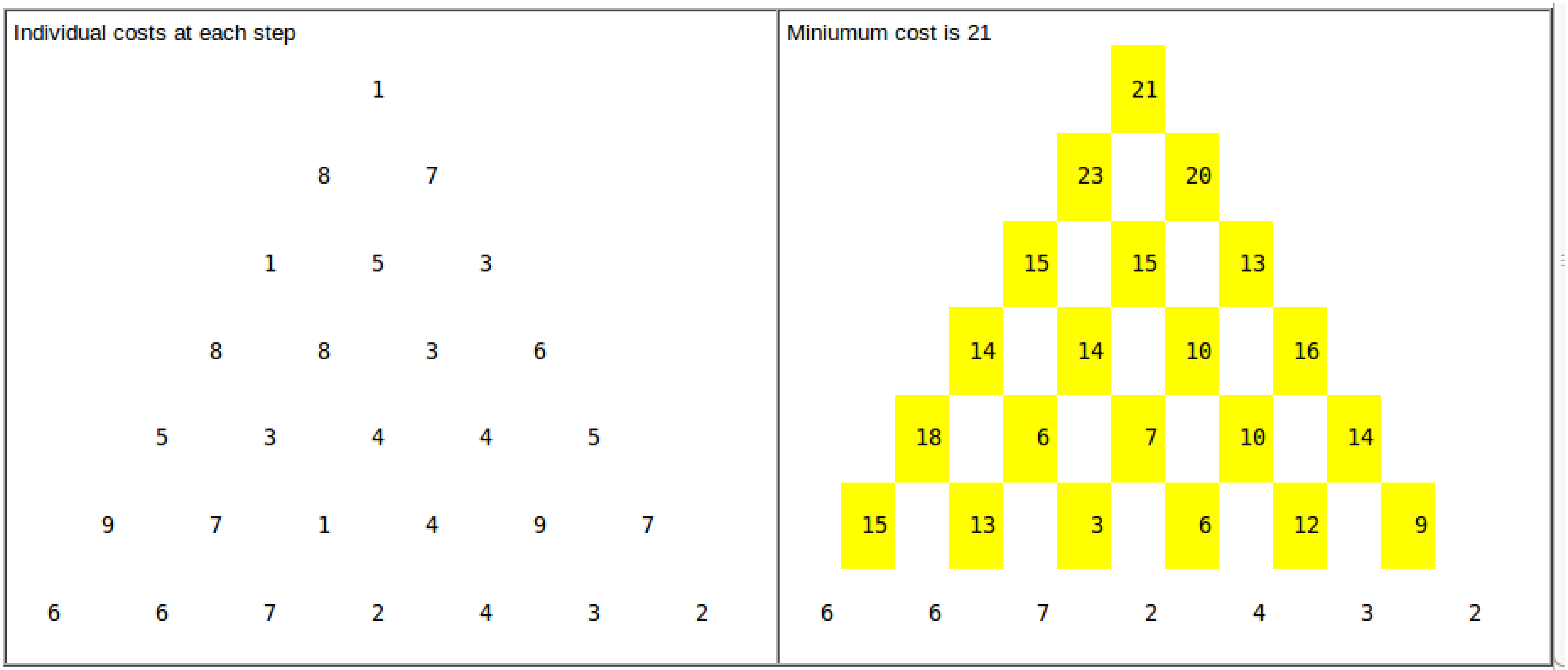
The minimum cost at each level is computed and stored for later use.

What value would a greedy approach give?



1->7->3->3->4->1->2 => 21, or

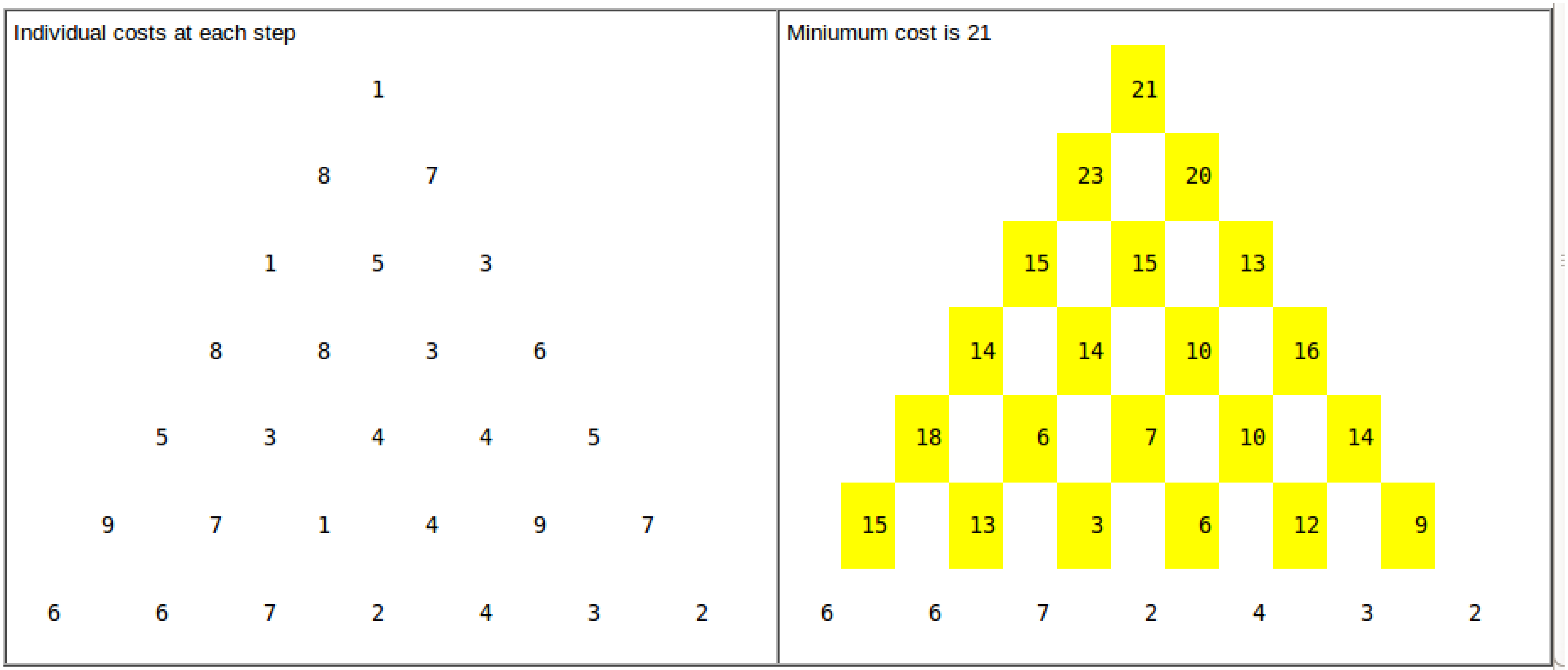
1->7->3->3->4->4->2 => 24



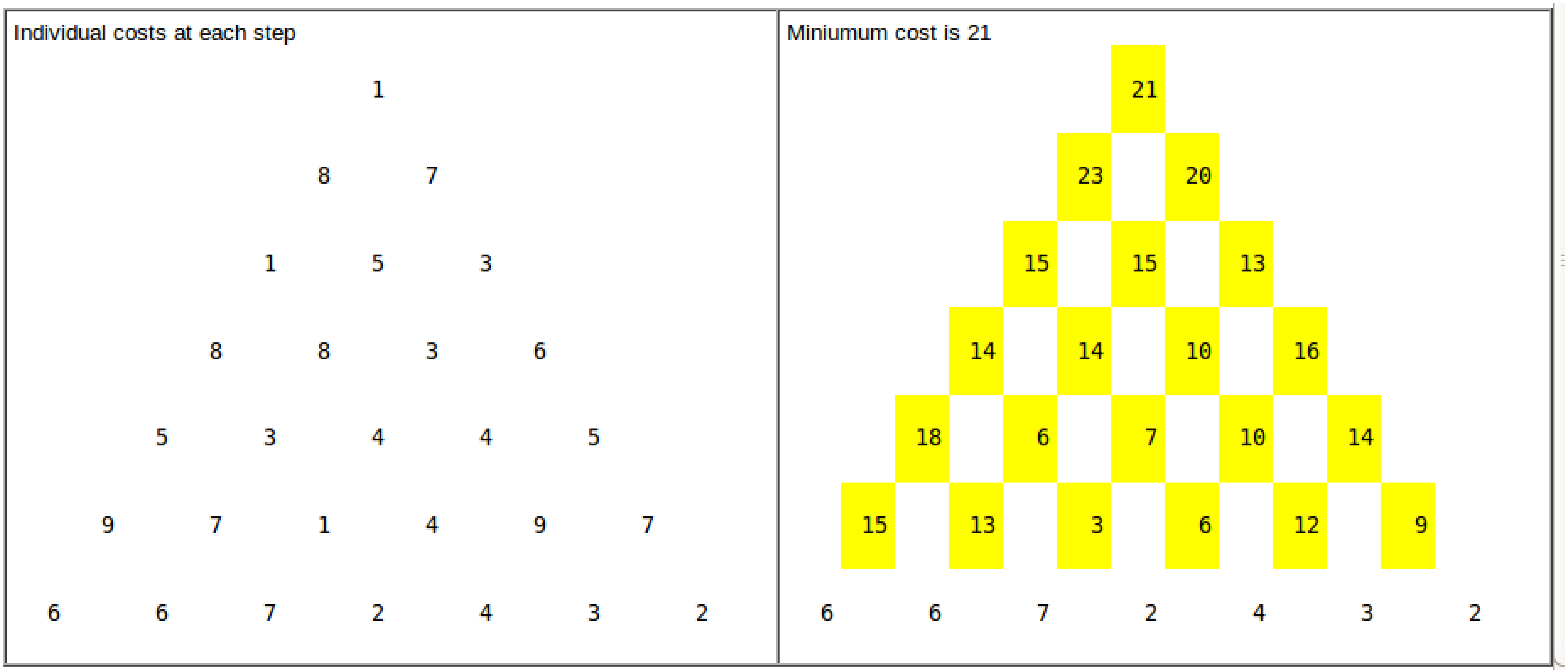
1->7->3->3->4->1->2 => 21, or

1->7->3->3->4->4->2 => 24

Moral of the story?



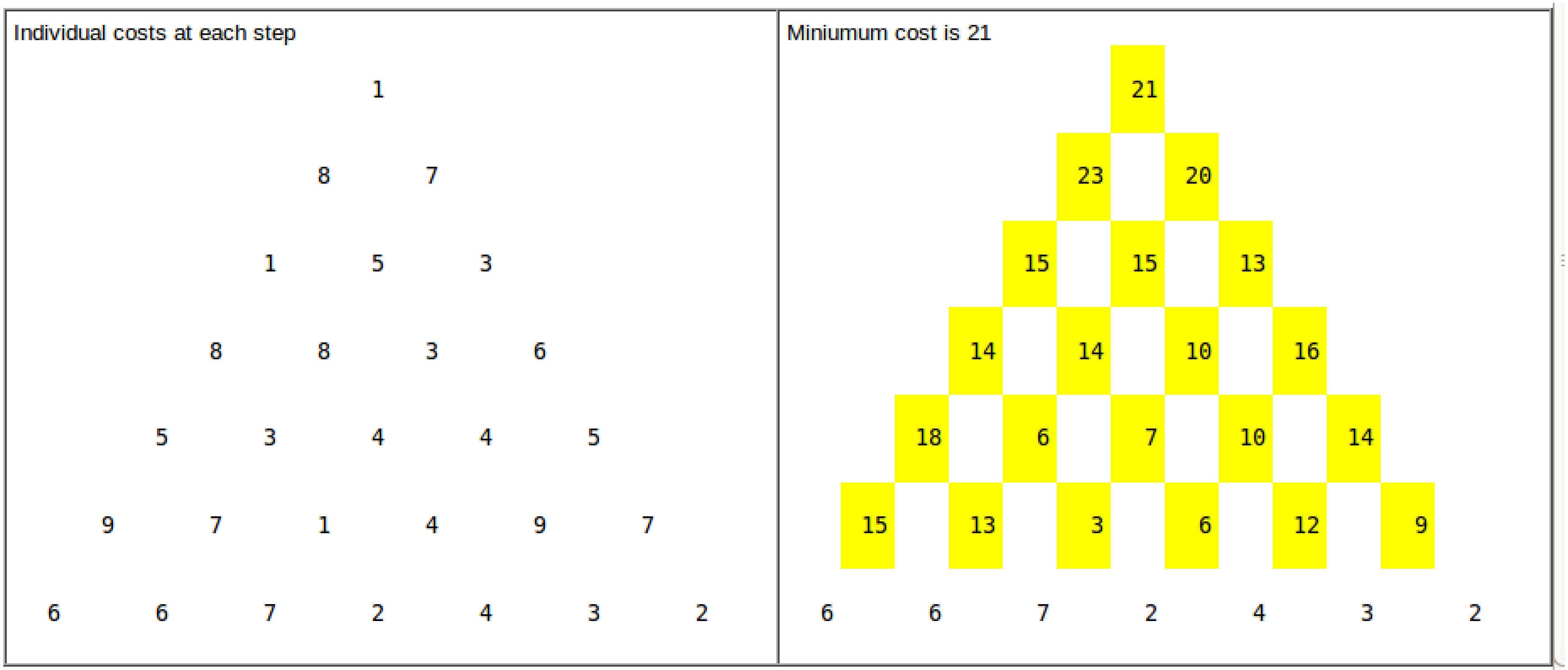
1->7->3->3->4->1->2 => 21, or
1->7->3->3->4->4->2 => 24
Greed doesn't *always* work!



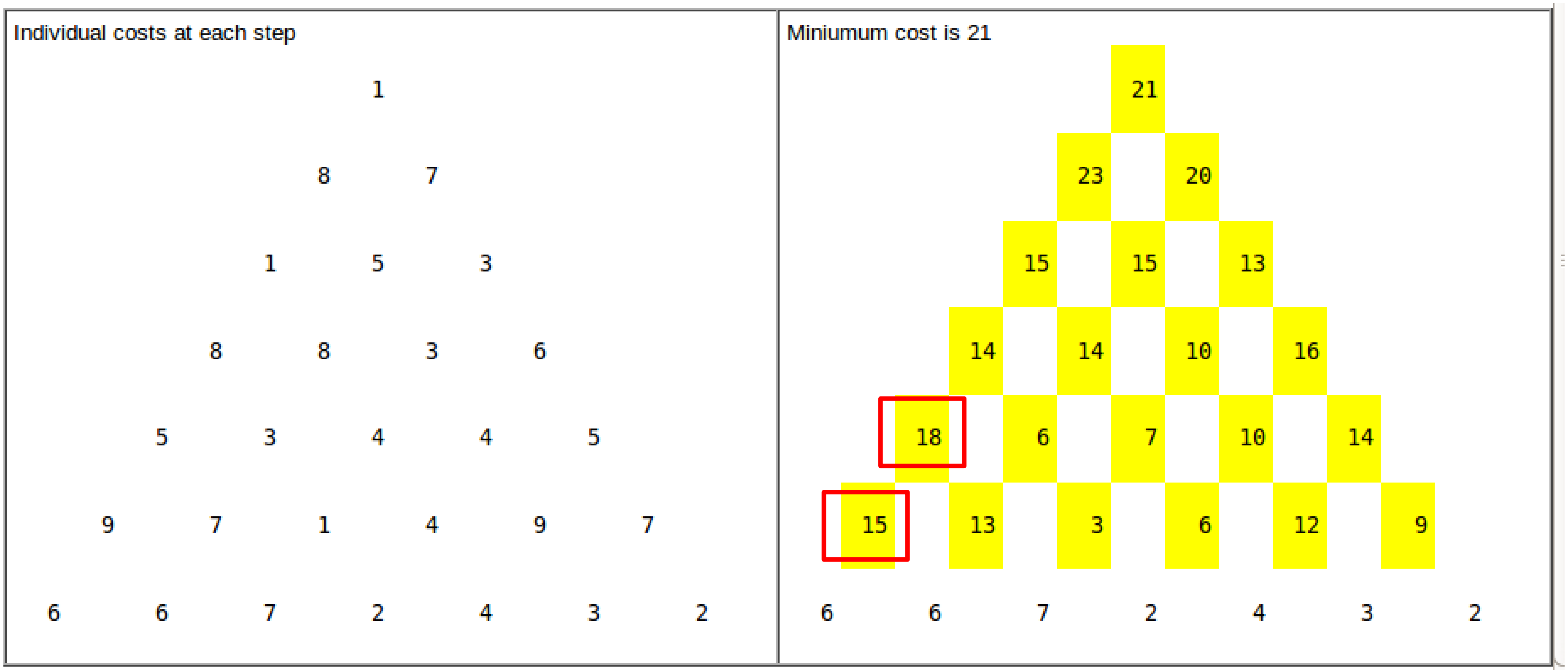
1->7->3->3->4->1->2 => 21, or

1->7->3->3->4->4->2 => 24

So, better be DP than greedy ☺

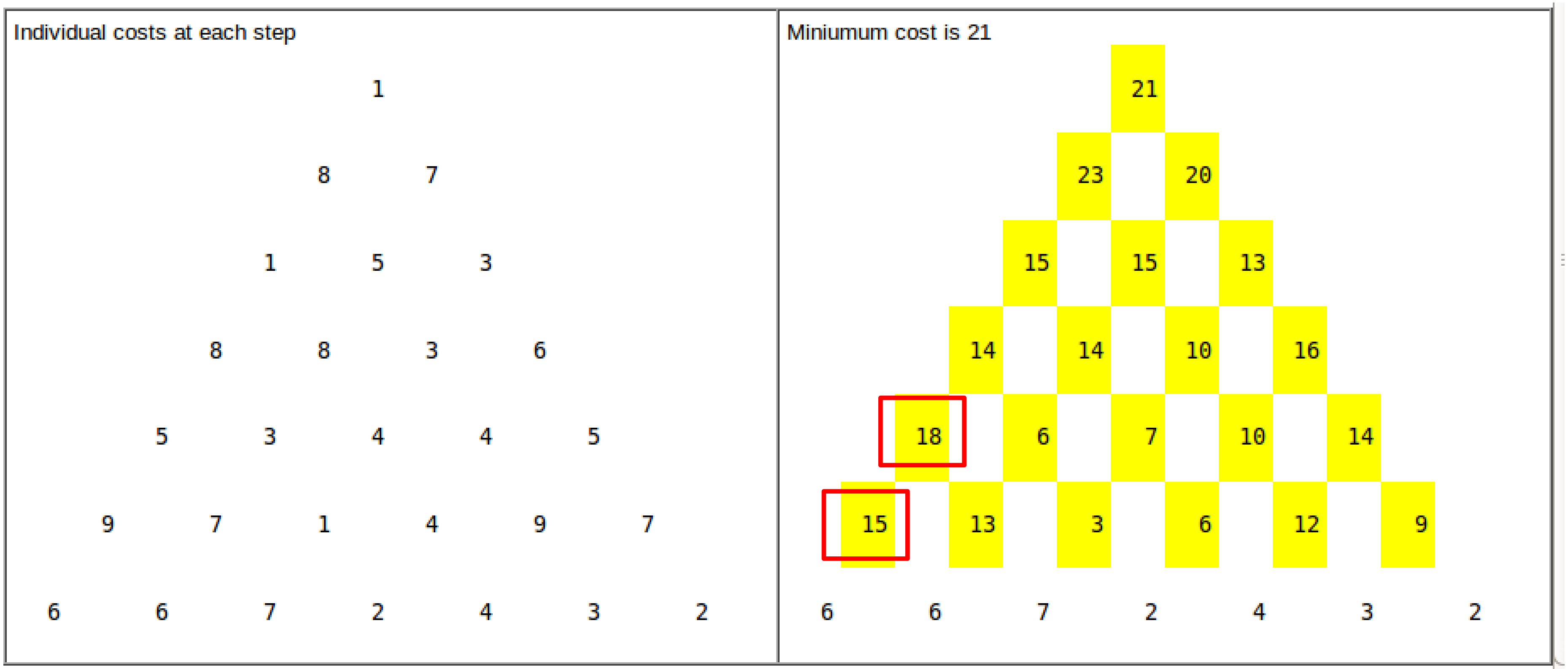


Write down the DP steps you followed in the form of an algorithm



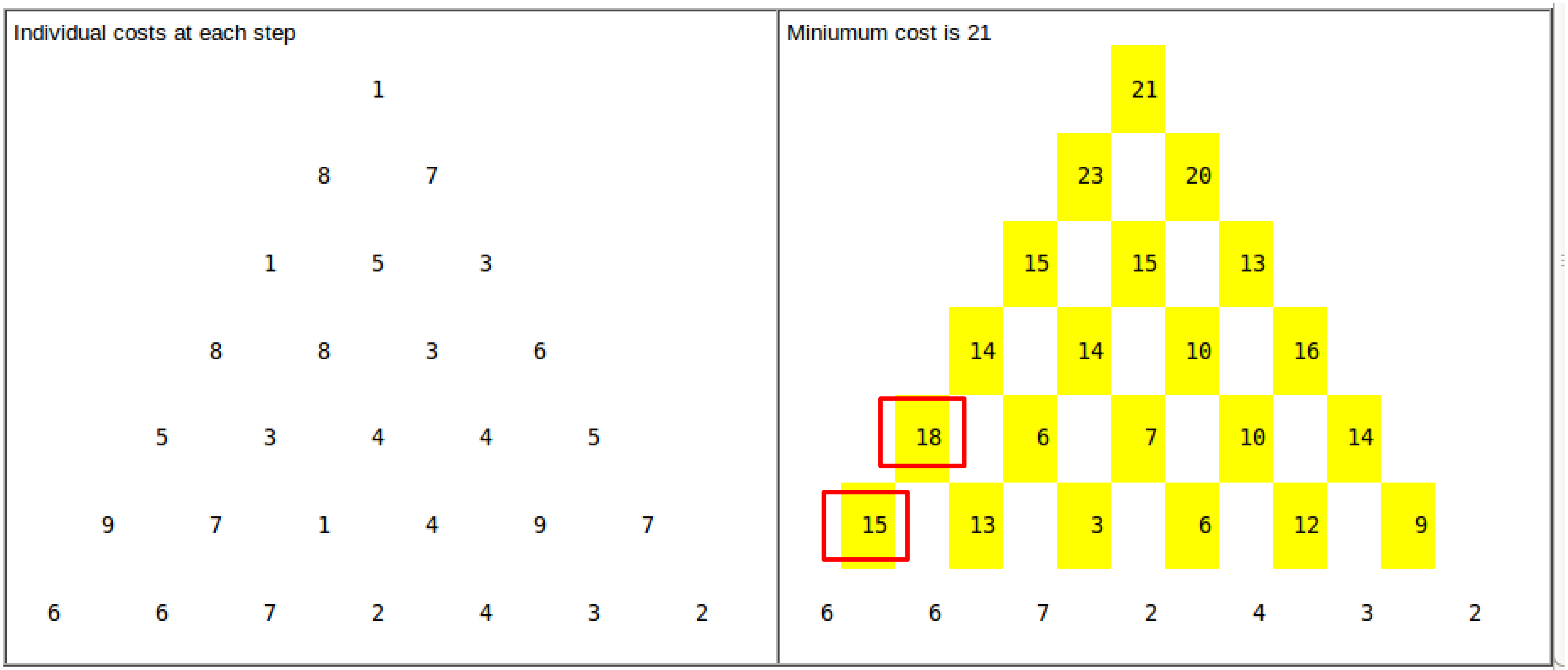
The minimum cost at each level is computed and stored for later use.

Think about how the DP algorithm can be coded into a computer program



The minimum cost at each level is computed and stored for later use.

Check whether your program returns the correct result



The minimum cost at each level is computed and stored for later use.

How do we use dynamic programming to solve some problem P?

- Break-down P using divide-and-conquer into sub-problems (SPs)
  - By writing the solution of P in terms of the solutions of sub-problems of P
  - Recurse and memoize OR build SP table bottom up
  - Complexity: we assume that due to memoization, each sub-problem is solved only once

Alternative way to think about it

- How do we use dynamic programming to solve some problem P?
- Break-down P using divide-and-conquer into sub-problems
  - Define potential sub-problems
  - Guess: a possible solution in terms of solutions of sub-problems
  - Optimization: optimize over all possible guesses, k= # of guesses
  - Recurse and memoize OR build table bottom up
  - Complexity: # of potential sub-problems that need to be solved x k guesses/sub-problem (due to memoization each sub-problem is solved only once; each guess takes $\Theta(1)$ time)

Rod cutting problem

Rod cutting problem

- A rod of size n
- A table of prices p_i = price in the market of a rod of size $i = 1,2,\ldots,n$
- Determine the maximum revenue $DP[n]$ obtained by cutting the rode into pieces and selling these to the market

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Rod cutting problem

- A rod of size n
- A table of prices p_i = price in the market of a rod of size $i = 1,2,\ldots,n$
- Determine the maximum revenue $DP[n]$ obtained by cutting the rode into pieces and selling these to the market

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Do we need to consider all possible combinations?

Rod cutting problem

- A rod of size n
- A table of prices p_i = price in the market of a rod of size $i = 1,2,\ldots,n$
- Determine the maximum revenue $DP[n]$ obtained by cutting the rode into pieces and selling these to the market

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Do we need to consider all possible combinations? No! use DP!

Example

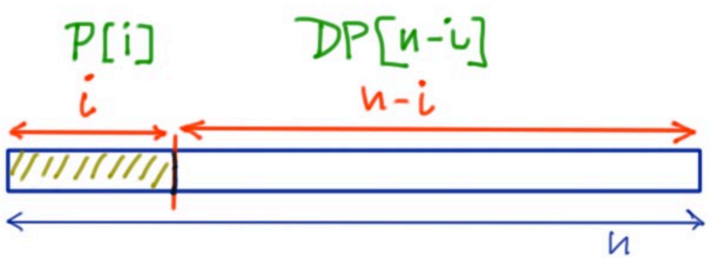
size of piece	1	2	3	4
market price	2	5	7	8

- Rod of size n=4
- Write down the DP equations (recurrence)
- Which are my sub-problems?
- How are these related?

Example

size of piece	1	2	3	4
market price	2	5	7	8

- Rod of size n=4
- Write down the DP equations (recurrence)
- Which are my sub-problems?
- How are these related?



$$DP[n] = \max_{i=0,\ldots,n} \{ P[i] + DP[n-i] \}$$

$$DP[j] = \text{max revenue obtained from a rod of size } j$$

a guess

$$= p_k + DP[j-k] \text{ for some } k \Rightarrow$$

$$DP[j] = \max_{i=0,1,\ldots,j} \{ p_i + DP[j-i] \}$$

Example

size of piece	1	2	3	4
market price	2	5	7	8

- Rod of size n=4
- Write down the DP equations (recurrence)
- Which are my sub-problems?
- How are these related?

$$DP[k] = \text{maximum value i can get from a rod of size } k$$

$$DP[k] = \max_i \{p_i + DP[k-i]\}, 1 \le i \le k$$

$$DP[0] = 0$$

$DP[0]$	$DP[1]$	$DP[2]$	$DP[3]$	$DP[4]$
0				

Example

size of piece	1	2	3	4
market price	2	5	7	8

- Rod of size n=4
- Write down the DP equations (recurrence)
- Which are my sub-problems?
- How are these related?

$$DP[k] = \text{maximum value i can get from a rod of size } k$$

$$DP[k] = \max_i \{p_i + DP[k-i]\}, 1 \leq i \leq k$$

$$DP[0] = 0$$

$$DP[1] = p_1 + DP[0] = 2$$

+2

$DP[0]$	$DP[1]$	$DP[2]$	$DP[3]$	$DP[4]$
0	2			

Example

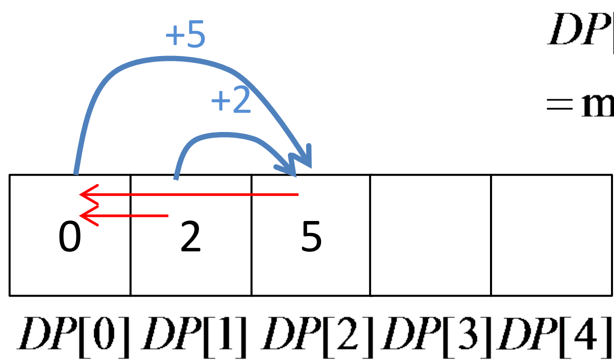
size of piece	1	2	3	4
market price	2	5	7	8

- Rod of size n=4
- Write down the DP equations (recurrence)
- Which are my sub-problems?
- How are these related?

$$DP[k] = \text{maximum value i can get from a rod of size } k$$

$$DP[k] = \max_i \{p_i + DP[k-i]\}, 1 \le i \le k$$

$$DP[0] = 0$$



$$DP[2] = \max\{p_2 + DP[0], p_1 + DP[1]\}$$
$$= \max\{5+0, 2+2\}$$

Example

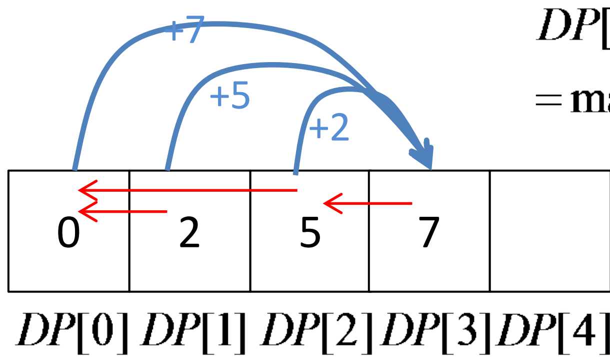
size of piece	1	2	3	4
market price	2	5	7	8

- Rod of size n=4
- Write down the DP equations (recurrence)
- Which are my sub-problems?
- How are these related?

$$DP[k] = \text{maximum value i can get from a rod of size } k$$

$$DP[k] = \max_i \{p_i + DP[k-i]\}, 1 \le i \le k$$

$$DP[0] = 0$$



$$DP[3] = \max\{p_3 + DP[0], p_2 + DP[1], p_1 + DP[2]\}$$
$$= \max\{7+0, 5+2, 2+5\}$$

Example

size of piece	1	2	3	4
market price	2	5	7	8

- Rod of size n=4
- Write down the DP equations (recurrence)
- Which are my sub-problems?
- How are these related?

$$DP[k] = \text{maximum value i can get from a rod of size } k$$

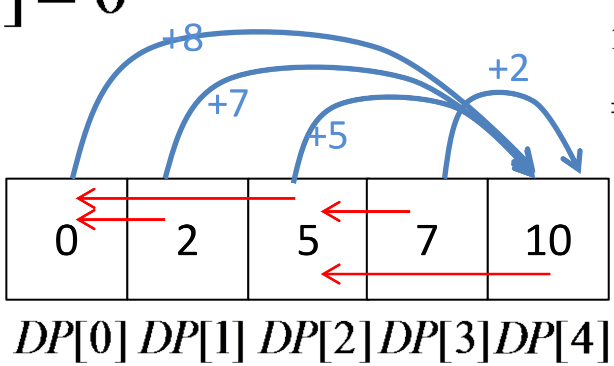
$$DP[k] = \max_i \{p_i + DP[k-i]\}, 1 \le i \le k$$

$$DP[0] = 0$$

$$DP[4] =$$

$$\max\{p_4 + DP[0], p_3 + DP[1], p_2 + DP[2], p_1 + DP[3]\}$$

$$= \max\{8+0, 7+2, 5+5, 2+7\}$$

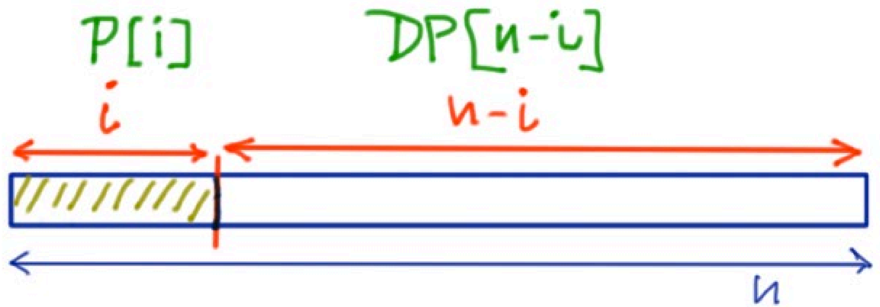


General DP formulation:

a guess

$$DP[j] = \text{max revenue obtained from a rod of size } j$$
$$= p[k] + DP[j-k] \text{ for some } k \quad \Rightarrow$$
$$DP[j] = \max_{i=0,1,\ldots,j} \{p[i] + DP[j-i]\}$$

j possible guesses



$$DP[n] = \max_{i=0,\ldots,n} \{ P[i] + DP[n-i] \}$$

$r[j] = DP[j]$, $s[j]$ = size of piece to be taken out from rod j

EXTENDED-BOTTOM-UP-CUT-ROD(p, n)

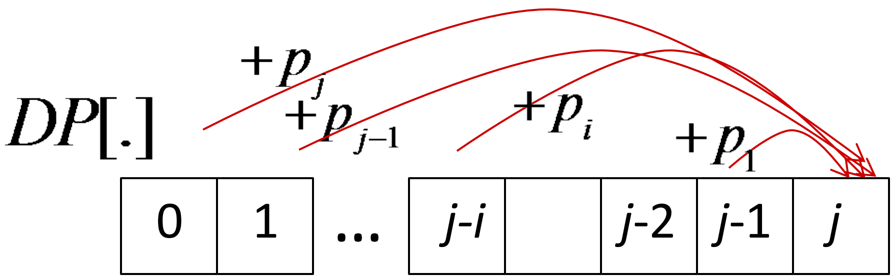
```
let r[0..n] and s[0..n] be new arrays
r[0] = 0
for j = 1 to n                      ← solve all sub-problems
    q = −∞
    for i = 1 to j
        if q < p[i] + r[j − i]
            q = p[i] + r[j − i]
            s[j] = i
    r[j] = q
return r and s
```

$q = \max_{1 \le i \le j} \{ p[i] + r[j-i] \}$

$s[j] = \operatorname{argmax}_{1 \le i \le j} \{ \quad \}$

Lets fill the table (bottom-up solution of DP)

$$DP[j] = \max_{i=0,1,\ldots,j} \{p_i + DP[j-i]\}$$



Topological sort:
compute *DP*[*j*] from left to right

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

	0	1	2	3	4	5	6	7	8	9	10
$DP[.]$	0										
$s[.]$	0										

$$s[j] = \arg\max_{i=0,1,\ldots,j} \{p_i + DP[j-i]\}$$

Exercise1: fill in the table with the solutions *DP*[*j*], *j*=0,…,10,
of the sub-problems, and the optimal decisions *s*[*j*] for each *j*
Exercise2: the complexity is a: n, b: n^2, c: n^3

Answer

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

len	0	1	2	3	4	5	6	7	8	9	10	
DP[]	0	1	5	8	10	13	17	18	22	25	30	
S[]	0	0	0	0	2	2	0	1	2	3	0	

$$
\begin{aligned}
r_1 &= 1 && \text{from solution } 1 = 1 \quad \text{(no cuts)}, \\
r_2 &= 5 && \text{from solution } 2 = 2 \quad \text{(no cuts)}, \\
r_3 &= 8 && \text{from solution } 3 = 3 \quad \text{(no cuts)}, \\
r_4 &= 10 && \text{from solution } 4 = 2 + 2, \\
r_5 &= 13 && \text{from solution } 5 = 2 + 3, \\
r_6 &= 17 && \text{from solution } 6 = 6 \quad \text{(no cuts)}, \\
r_7 &= 18 && \text{from solution } 7 = 1 + 6 \text{ or } 7 = 2 + 2 + 3, \\
r_8 &= 22 && \text{from solution } 8 = 2 + 6, \\
r_9 &= 25 && \text{from solution } 9 = 3 + 6, \\
r_{10} &= 30 && \text{from solution } 10 = 10 \quad \text{(no cuts)}.
\end{aligned}
$$

The text justification problem

- Split text into “good” lines so that a set of consecutive lines looks “nice”

bad!	blah blah blah	0b	blah blah	4b	good!
	blah	8b	blah blah	4b	
	averylongword	0b	averylongword	0b	

Question 1: Which algorithm is used in the “bad case”?

a. breaks randomly the text into lines that fit the given page length
b. works “myopically”: fills each line to the maximum possible without caring for the next lines

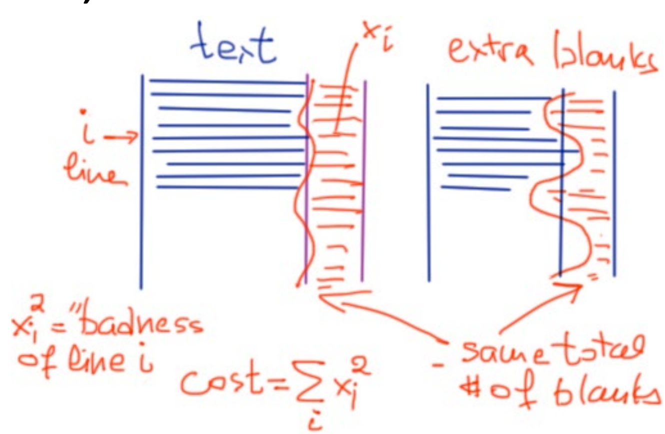
The text justification problem

- Split text into "good" lines so that a set of consecutive lines looks "nice"

bad!	blah blah blah	0b	blah blah	4b	good!
	blah	8b	blah blah	4b	
	averylongword	0b	averylongword	0b	

Question 2: Guess a good mathematical criterion for assessing "badness" of a solution

1. total # of blank spaces,
2. worse case of extra blank spaces in some line,
3. $\sum_{\text{all lines i}} (\text{\# of extra blank spaces in line } i)^2$



Why use the cost $\sum_{\text{all lines}} (\#\text{ of extra blank spaces in line } i)^a,\ a > 1$?

Consider the problem of allocating *K* extra blank spaces to *m* lines.

If we use the" badness" function $\sum_{i=1,\ldots,m} x_i^2,\ x_i = \text{extra blanks in line } i$

then solving

$$\min \sum_{i=1,\ldots,m} x_i^2, \quad \sum_{i=1,\ldots,m} x_i = K$$

gets us that $x_1 = x_2 = \ldots = x_m$ at the optimum. Hence this cost function is reduced not only when the total number of extra blank spaces is reduces, but also when our extra blanks are distributed more evenly among the lines. This property holds when the square is replaced by any value a>1.

- We are given a sequence of *n* words (different lengths)

$words[i..j-1]$

Array of input $words[0..n-1]$

0 1 2 ... i ... j-1 j ... n-1

length(i,j-1)

- *length*(*i*,*j*)=total useful space needed for words *i*,…,*j*-1

$$badness(i, j-1) = \begin{cases} \infty \ \text{ if } \ length(i, j-1) > \text{ page width} \\ (\text{page width - } length(i, j-1))^2 \ \text{ else} \end{cases}$$

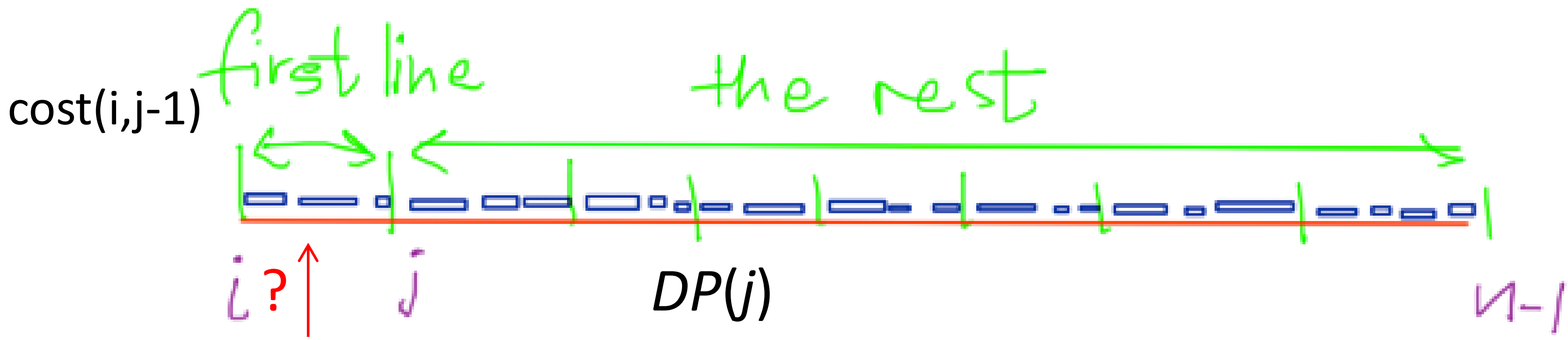
Goal: split words into lines to minimize $\sum_{k=1,2,\ldots} badness(\text{line } k)$

Which are the DP equations?

DP[*i*]: minimum possible cost for words[*i*..n-1]

$\Rightarrow$ sub-problem = suffix of [0..n-1], # of sub-problems = $\Theta(n)$

Guessing: *j* = where to end first line, # of choices = *n-i*



Recurrence:

$$DP[i] = \min\{badness(i, j-1) + DP[j], j \in i+1, \ldots, n\}, i = 0, \ldots, n$$

$$DP[n] = 0$$

this is known already.
someone should have calculated
all badness() and saved in a table

Recurrence:

$$DP[i] = \min\{badness(i, j-1) + DP[j], j \in i+1, \ldots, n\},$$

$$i = 0, \ldots, n$$

$$DP[n] = 0$$

Order of computation: *n*, *n*-1, …, 1, 0 (topological sort)

Total time = $\Theta(n^2)$

Solution = $DP[0]$