How to tackle the problem

- Step 1: Understand the problem. Come up with a couple of examples and try to solve them.
- Step 2: Try brute force. How many possible solutions are there?
- Step 3: Define a natural subproblem.
- Step 4: find a recursive formula for $OPT(i)$
  - Could you quickly find a solution to the problem if you magically had solutions to all the smaller subproblems? Can you capture this in a formula, like we did for rod-cutting?
  - Imagine that magically, you have all the values for $OPT(j)$, for all $j < i$.
- Step 5: How would you turn the formula into an algorithm? What are the two techniques you can use? Write down the pseudocode and analyze its running time.
- Step 6: Modify your algorithm to return the optimal solution itself, not just its value.

Optimal substructure

An optimal solution to a problem (instance) contains optimal solutions to subproblems.

```
LCS-LENGTH(X, Y)
m = X.length
n = Y.length
let b[1..m, 1..n] and c[0..m, 0..n] be new tables
for i = 1 to m
    c[i, 0] = 0
for j = 0 to n
    c[0, j] = 0
for i = 1 to m
    for j = 1 to n
        if x_i == y_j
            c[i, j] = c[i − 1, j − 1] + 1
            b[i, j] = "↖"
        elseif c[i − 1, j] ≥ c[i, j − 1]
            c[i, j] = c[i − 1, j]
            b[i, j] = "↑"
        else c[i, j] = c[i, j − 1]
            b[i, j] = "←"
return c and b
```

```
PRINT-LCS(b, X, i, j)
if i == 0 or j == 0
    return
if b[i, j] == "↖"
    PRINT-LCS(b, X, i − 1, j − 1)
    print x_i
elseif b[i, j] == "↑"
    PRINT-LCS(b, X, i − 1, j)
else PRINT-LCS(b, X, i, j − 1)
```

Dynamic-programming algorithm

IDEA:
Compute the table bottom-up.

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Dynamic-programming algorithm

IDEA:
Compute the table bottom-up.

Time = $\Theta(mn)$.

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Dynamic-programming algorithm

IDEA:
Compute the table bottom-up.

Time = $\Theta(mn)$.

Reconstruct LCS by tracing backwards.

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Dynamic-programming algorithm

IDEA:
Compute the table bottom-up.

Time = $\Theta(mn)$.

Reconstruct LCS by tracing backwards.

Multiple solutions are possible.

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Dynamic-programming algorithm

IDEA:
Compute the table bottom-up.

Time = $\Theta(mn)$.

Reconstruct LCS by tracing backwards.

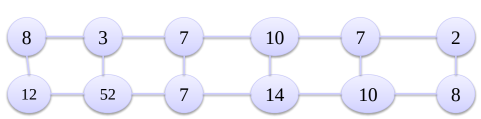
Space = $\Theta(mn)$.

With tweaks:
Space $O(\min\{m, n\})$

		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Weighted IS on a 2 x n grid

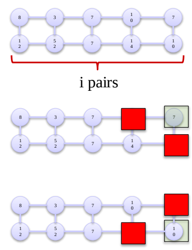
- Weighted independent set on the 2 by n grid.
  - Input: a grid graph of size 2 by n, with values $t_1,..,t_n$ and $b_1,..,b_n$
  - Goal: find a heaviest independent set



First attempt

- Let grid(i) be the solution to a grid of size 2 by i

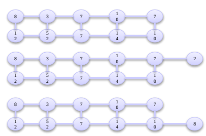
- Problem: subproblems that arise are not of the type grid(i).



Second attempt

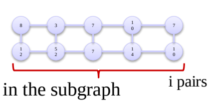
- Solution: further break down the type of subproblems

- Three types of subproblems:
  – grid(i)
  – gridTop(i)
  – gridBottom(i)

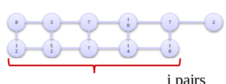


Exercise

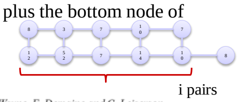
– grid(i): maximum independent set in the subgraph consisting of only the first i pairs of nodes



– gridTop(i): maximum independent set in the subgraph consisting of the first i pairs of nodes plus the top node of the (i+1)-st pair

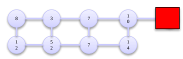


– gridBottom(i): maximum independent set in the subgraph consisting of the first i pairs of nodes plus the bottom node of the (i+1)-st pair

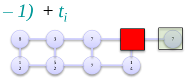


Recursive formulas for subproblems

- Lets solve gridTop(i) first
  - We can either not include the top ($i + 1$) node
    - Then gridTop(i) = grid(i)



  - Or, we can include it.
    - Then gridTop(i) = gridBottom($i - 1$) + t_i

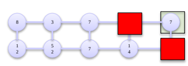


  - gridBottom(i) is symmetric to gridTop

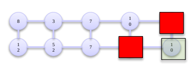
based on slides by S. Raskhodnikova, A. Smith, K. Wayne, E. Demaine and C. Leiserson

Recursive formulas for subproblems

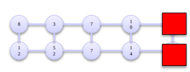
- grid(i)
  - We can either include top i node
    - grid(i) = t_i + gridBottom(i-2)



  - Or, include bottom i node
    - grid(i) = gridTop($i - 2$) + b_i



  - Or, include neither
    - grid(i) = grid($i - 1$)



based on slides by S. Raskhodnikova, A. Smith, K. Wayne, E. Demaine and C. Leiserson

Recursive formulas for subproblems

- t_i, b_i are the weights of the nodes in the i^{th} pair
- grid(i) = max(b_1 , t_1) if i = 1
  max(t_i + gridBottom(i-2), otherwise
  gridTop($i - 2$) + b_i ,
  grid($i - 1$))
- gridTop(i) = t_1 if i = 0
  max(grid(i), otherwise
  gridBottom($i - 1$) + t_i)
- gridBottom(i) = b_1 if i = 0
  max(grid(i), otherwise
  gridTop($i - 1$) + b_i)
- Bottom-up algorithm takes O(n) time.

based on slides by S. Raskhodnikova, A. Smith, K. Wayne, E. Demaine and C. Leiserson