Dynamic Programming

Christian Wulff-Nilsen
Fourth lecture
Algorithms and Data Structures
DIKU

February 15, 2023

Overview for today

- Introduction:

Overview for today

- Introduction:
    - Computing Fibonacci numbers

Overview for today

- Introduction:
  - Computing Fibonacci numbers
  - What is dynamic programming?

Overview for today

- Introduction:
  - Computing Fibonacci numbers
  - What is dynamic programming?
- Rod cutting

Overview for today

- Introduction:
  - Computing Fibonacci numbers
  - What is dynamic programming?
- Rod cutting
- Longest common subsequence

Computing Fibonacci numbers

- The Fibonacci numbers $F_0, F_1, \ldots$ are defined by the recurrence:

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_n &= F_{n-1} + F_{n-2} \text{ for } n \geq 2. \end{aligned}$$

Computing Fibonacci numbers

- The Fibonacci numbers $F_0, F_1, \ldots$ are defined by the recurrence:

$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_n &= F_{n-1} + F_{n-2} \text{ for } n \geq 2. \end{aligned}$$

- How fast can we compute the nth Fibonacci number F_n, $n \geq 0$?

Computing Fibonacci numbers

- The Fibonacci numbers $F_0, F_1, \ldots$ are defined by the recurrence:

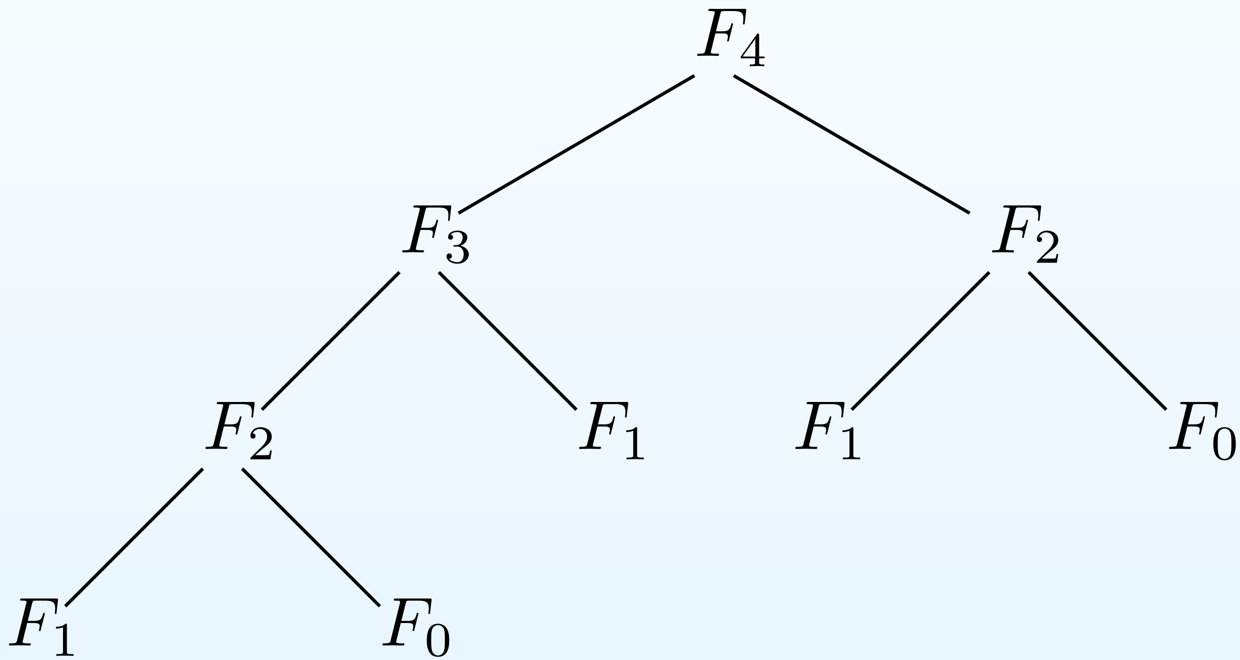
$$\begin{aligned} F_0 &= 0, \\ F_1 &= 1, \\ F_n &= F_{n-1} + F_{n-2} \text{ for } n \geq 2. \end{aligned}$$

- How fast can we compute the nth Fibonacci number F_n, $n \geq 0$?
- Simple algorithm (assume input $n \in \mathbb{N}_0$):

$\texttt{FIB}(n)$
1 if $n \leq 1$ return n
2 else return $\texttt{FIB}(n-1) + \texttt{FIB}(n-2)$

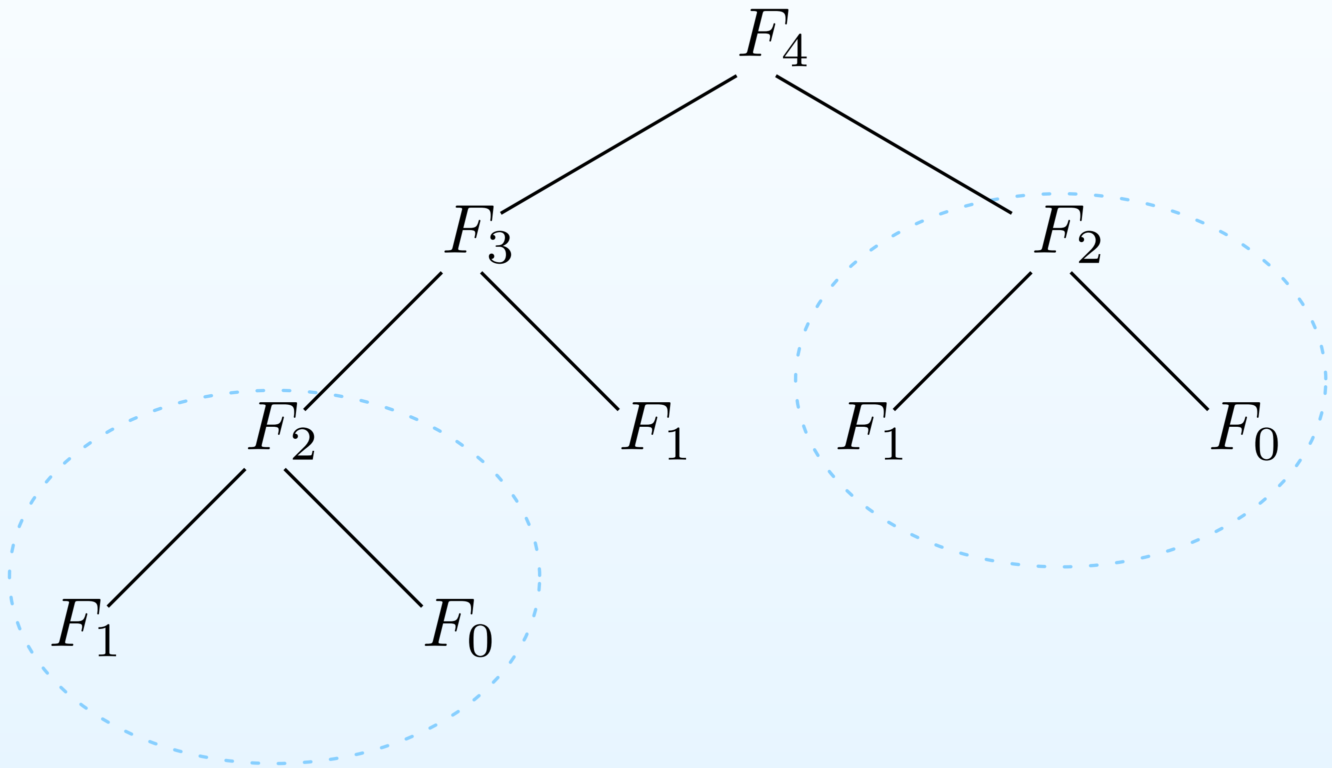
Subproblems solved for F_4

- Recursive calls executed by $\texttt{FIB}(4)$:



Subproblems solved for F_4

- Recursive calls executed by $\texttt{FIB}(4)$:



- Note that subproblem F_2 is solved twice.
- This overlap in subproblems gets much worse when computing bigger Fibonacci numbers.

Computing Fibonacci numbers

- Algorithm `FIB` recomputes many identical subproblems over and over.

Computing Fibonacci numbers

- Algorithm `FIB` recomputes many identical subproblems over and over.
- It can be shown to have $\Theta(1,618^n)$ running time.

Computing Fibonacci numbers

- Algorithm `FIB` recomputes many identical subproblems over and over.
- It can be shown to have $\Theta(1,618^n)$ running time.
- Can we do better?

Computing Fibonacci numbers

- Consider a different algorithm `FIBFAST` to find $F(n)$.

Computing Fibonacci numbers

- Consider a different algorithm `FIBFAST` to find $F(n)$.
- Initialize an array $F[0 \ldots n]$ where $F[0] = 0$, $F[1] = 1$, and $F[i] = -1$ for $2 \leq i \leq n$.

Computing Fibonacci numbers

- Consider a different algorithm FIBFAST to find $F(n)$.
- Initialize an array $F[0 \ldots n]$ where $F[0] = 0$, $F[1] = 1$, and $F[i] = -1$ for $2 \leq i \leq n$.
- Then make the call $\texttt{FIBFAST}(n, F)$, where:

```
FIBFAST(m, F)
if F[m] < 0
    F[m] = FIBFAST(m - 1, F) + FIBFAST(m - 2, F)
return F[m]
```

Computing Fibonacci numbers

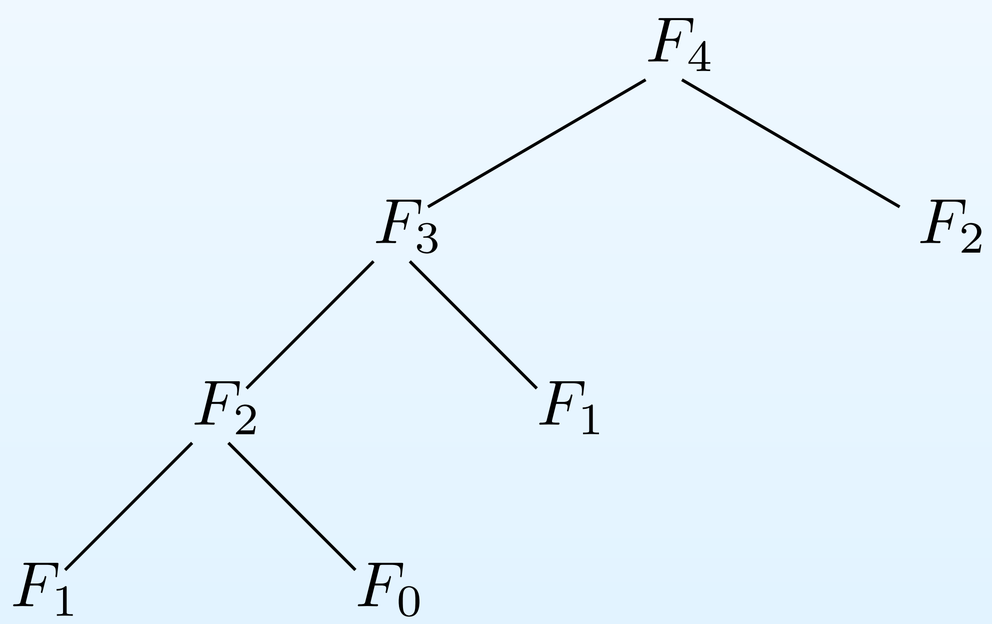
$\texttt{FIBFAST}(m, F)$
1 if $F[m] < 0$
2 $\quad F[m] = \texttt{FIBFAST}(m-1, F) + \texttt{FIBFAST}(m-2, F)$
3 return $F[m]$

- Recursive calls executed by $\texttt{FIBFAST}(4, F)$:

Computing Fibonacci numbers

```
FIBFAST(m, F)
if F[m] < 0
    F[m] = FIBFAST(m − 1, F) + FIBFAST(m − 2, F)
return F[m]
```

- Recursive calls executed by $\texttt{FIBFAST}(4, F)$:



Computing Fibonacci numbers

```
FIBFAST(m, F)
1  if F[m] < 0
2      F[m] = FIBFAST(m − 1, F) + FIBFAST(m − 2, F)
3  return F[m]
```

- Recursive calls executed by $\texttt{FIBFAST}(4, F)$:
- $\texttt{FIBFAST}(n, S)$ runs in $\Theta(n)$ time since we execute line 2 only once for each m between 2 and n.

Computing Fibonacci numbers

$\texttt{FIBFAST}(m, F)$
1 if $F[m] < 0$
2 $\quad F[m] = \texttt{FIBFAST}(m-1, F) + \texttt{FIBFAST}(m-2, F)$
3 return $F[m]$

- Recursive calls executed by $\texttt{FIBFAST}(4, F)$:
- $\texttt{FIBFAST}(n, S)$ runs in $\Theta(n)$ time since we execute line 2 only once for each m between 2 and n.
- Technical detail:
  - Assumes addition of big $\Theta(n)$-bit numbers can be done in constant time

Computing Fibonacci numbers

```
FIBFAST(m, F)
1  if F[m] < 0
2      F[m] = FIBFAST(m − 1, F) + FIBFAST(m − 2, F)
3  return F[m]
```

- Recursive calls executed by $\texttt{FIBFAST}(4, F)$:
- $\texttt{FIBFAST}(n, S)$ runs in $\Theta(n)$ time since we execute line 2 only once for each m between 2 and n.
- Technical detail:
  - Assumes addition of big $\Theta(n)$-bit numbers can be done in constant time
  - Without this assumption, the running time becomes $\Theta(n^2/\lg n)$ since a $\Theta(n)$-bit number can be stored in $\Theta(n/\lg n)$ words, allowing for an addition to be computed in $\Theta(n/\lg n)$ time.

Dynamic programming

- `FIBFAST` is an example of dynamic programming.

Dynamic programming

- `FIBFAST` is an example of dynamic programming.
- The idea of DP is to avoid recomputing identical subproblems.

Dynamic programming

- `FIBFAST` is an example of dynamic programming.
- The idea of DP is to avoid recomputing identical subproblems.
- This is done by storing the solution to a subproblem in a table.

Dynamic programming

- `FIBFAST` is an example of dynamic programming.
- The idea of DP is to avoid recomputing identical subproblems.
- This is done by storing the solution to a subproblem in a table.
- If that subproblem is encountered again, there is no need to recompute it as a simple table look-up will give the solution.

Dynamic programming

- We follow four steps when developing a DP algorithm:

Dynamic programming

- We follow four steps when developing a DP algorithm:
  - Characterize the structure of an optimal solution in terms of optimal solutions to subproblems (*optimal substructure*).

Dynamic programming

- We follow four steps when developing a DP algorithm:
  - Characterize the structure of an optimal solution in terms of optimal solutions to subproblems (*optimal substructure*).
  - Recursively define the value of an optimal solution.

Dynamic programming

- We follow four steps when developing a DP algorithm:
  - Characterize the structure of an optimal solution in terms of optimal solutions to subproblems (*optimal substructure*).
  - Recursively define the value of an optimal solution.
  - Compute the value of an optimal solution using, e.g., recursion.

Dynamic programming

- We follow four steps when developing a DP algorithm:
  - Characterize the structure of an optimal solution in terms of optimal solutions to subproblems (*optimal substructure*).
  - Recursively define the value of an optimal solution.
  - Compute the value of an optimal solution using, e.g., recursion.
  - Construct optimal solution from computed information.

Dynamic programming

- We follow four steps when developing a DP algorithm:
  - Characterize the structure of an optimal solution in terms of optimal solutions to subproblems (*optimal substructure*).
  - Recursively define the value of an optimal solution.
  - Compute the value of an optimal solution using, e.g., recursion.
  - Construct optimal solution from computed information.
- For dynamic programming to be useful, we need *overlapping subproblems* (the same subproblems are visited repeatedly).

The rod cutting problem

- Suppose we are given a rod R of length n.

The rod cutting problem

- Suppose we are given a rod R of length n.
- We want to cut R up into pieces of certain lengths and sell these pieces.

The rod cutting problem

- Suppose we are given a rod R of length n.
- We want to cut R up into pieces of certain lengths and sell these pieces.
- A piece of length i earns us p_i kr., where $i \in \{1, \ldots, n\}$.

The rod cutting problem

- Suppose we are given a rod R of length n.
- We want to cut R up into pieces of certain lengths and sell these pieces.
- A piece of length i earns us p_i kr., where $i \in \{1, \ldots, n\}$.
- There is no cost in cutting R.

The rod cutting problem

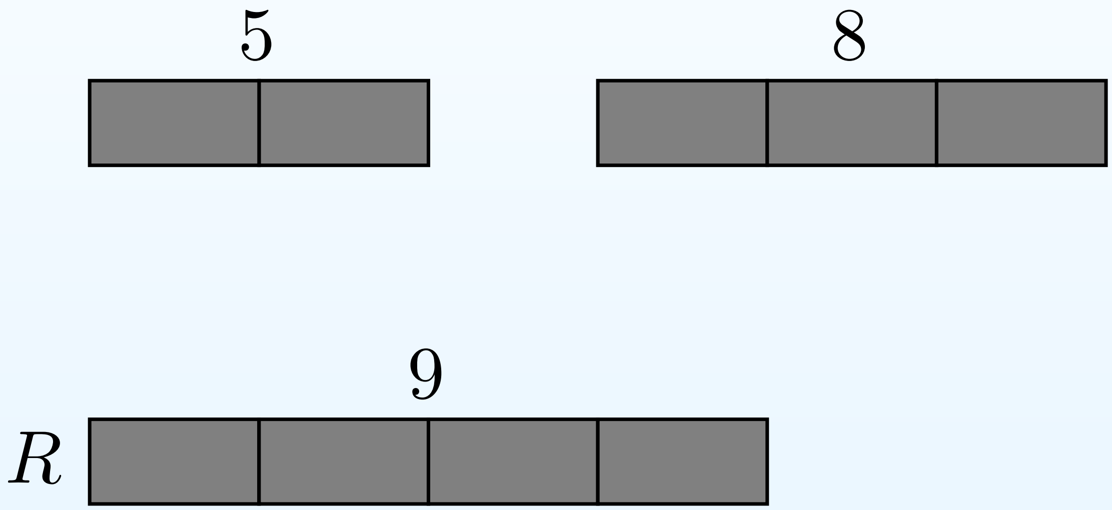
- Suppose we are given a rod R of length n.
- We want to cut R up into pieces of certain lengths and sell these pieces.
- A piece of length i earns us p_i kr., where $i \in \{1, \ldots, n\}$.
- There is no cost in cutting R.
- What is the maximum revenue r_n that we can obtain?

The rod cutting problem

- Suppose we are given a rod R of length n.
- We want to cut R up into pieces of certain lengths and sell these pieces.
- A piece of length i earns us p_i kr., where $i \in \{1, \ldots, n\}$.
- There is no cost in cutting R.
- What is the maximum revenue r_n that we can obtain?
- The input to an algorithm for the rod cutting problem is the rod length n and prices $p_1, p_2, \ldots, p_n$.

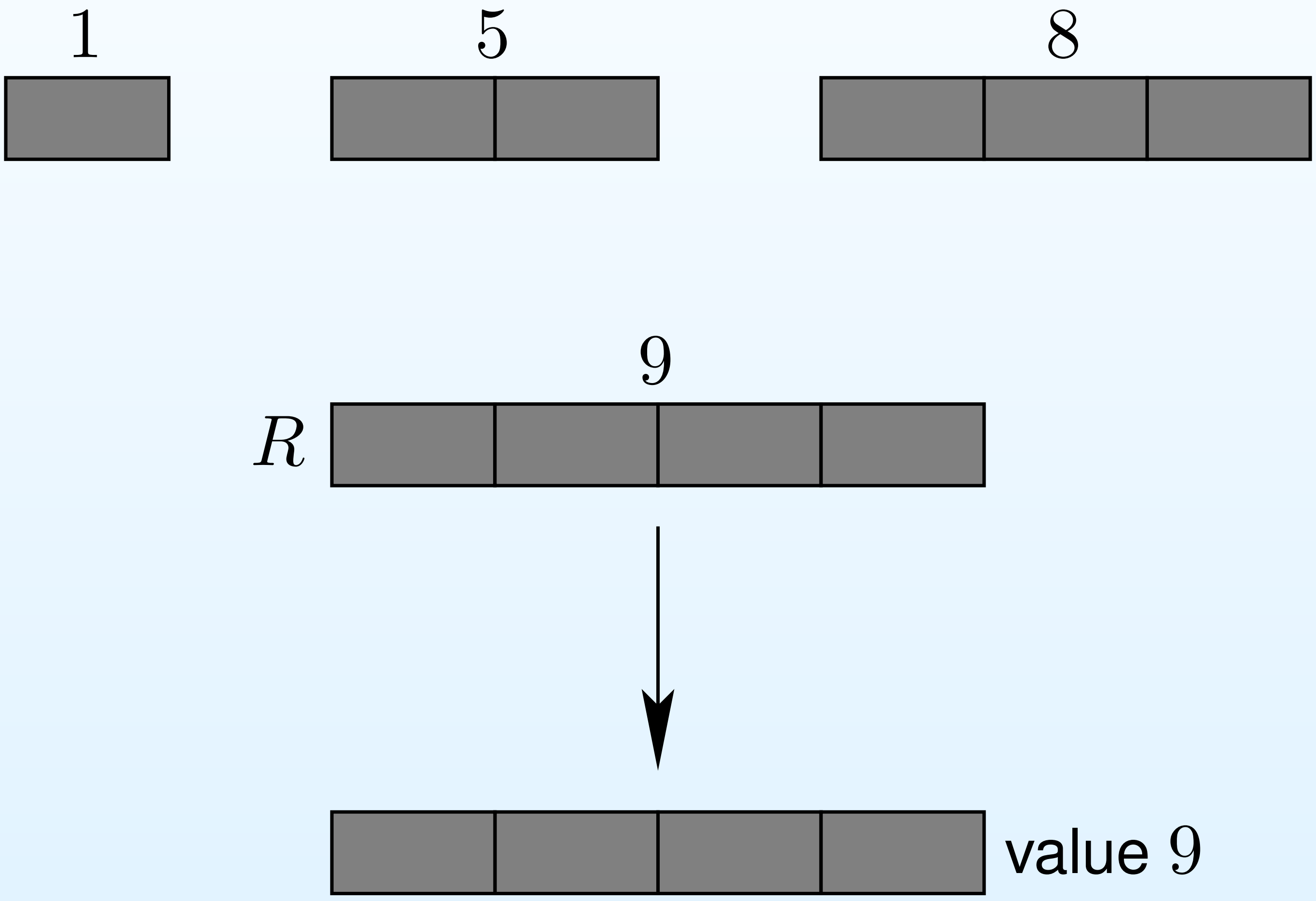
The rod cutting problem

- Example (numbers are prices):



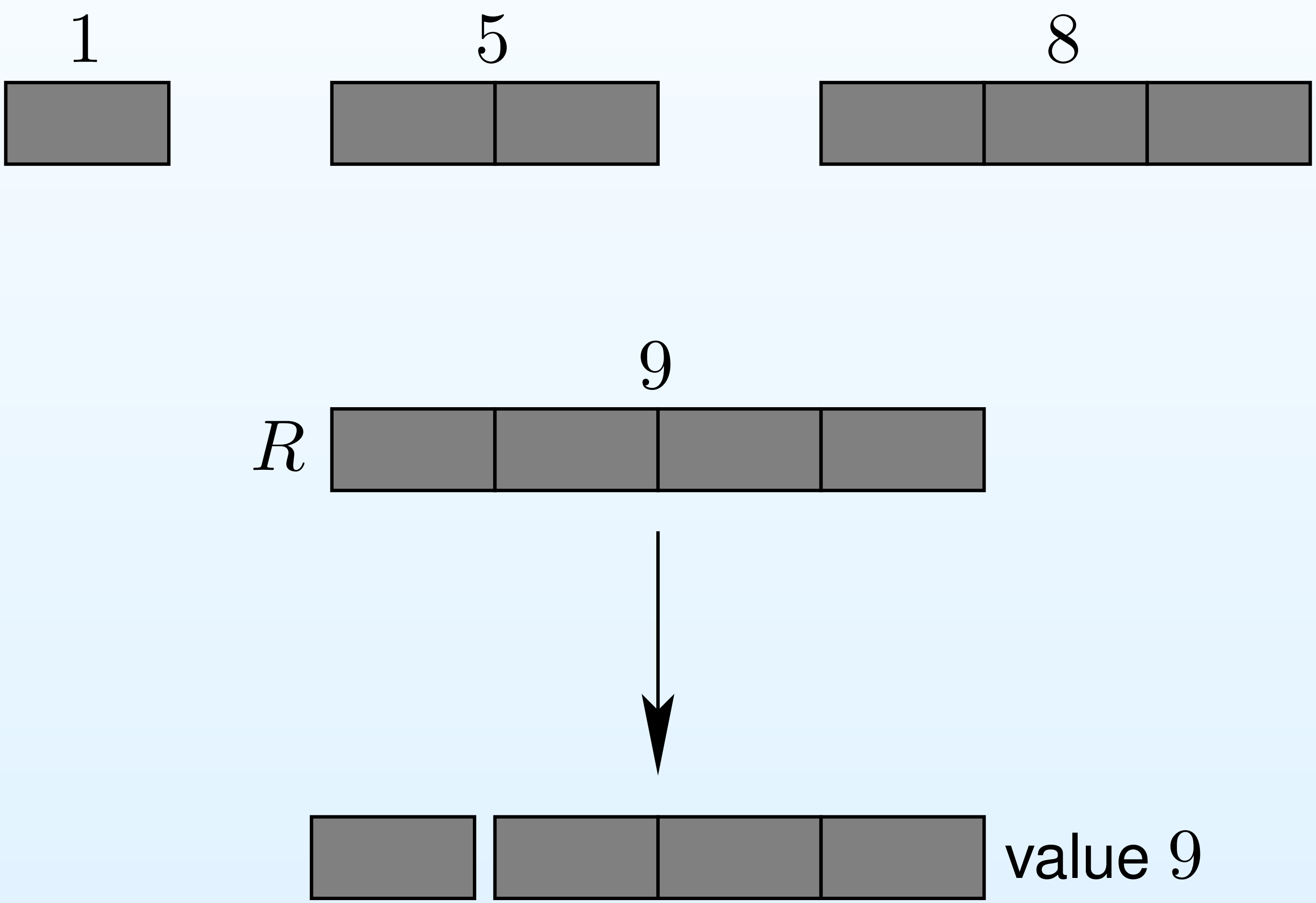
The rod cutting problem

- Example (numbers are prices):



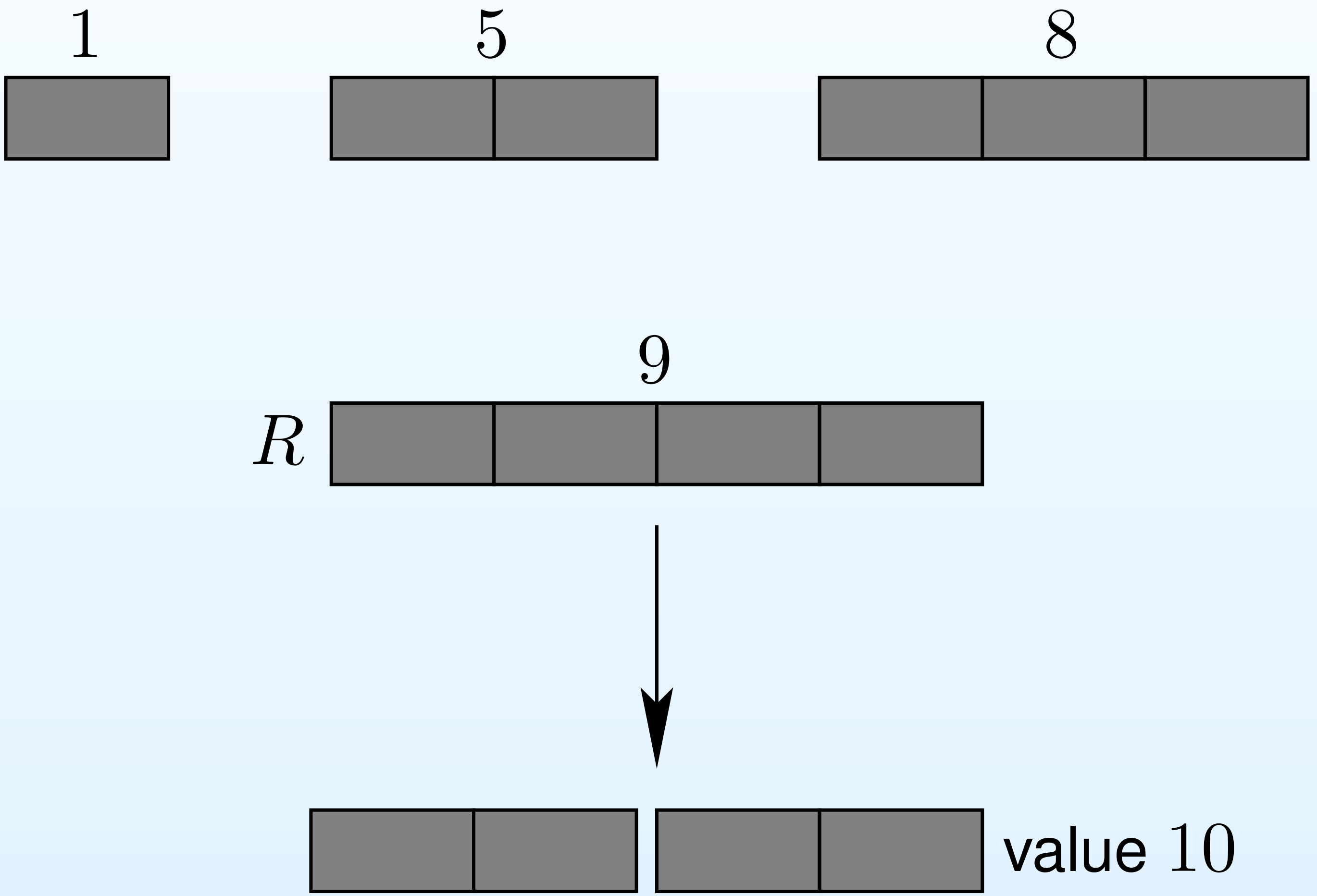
The rod cutting problem

- Example (numbers are prices):



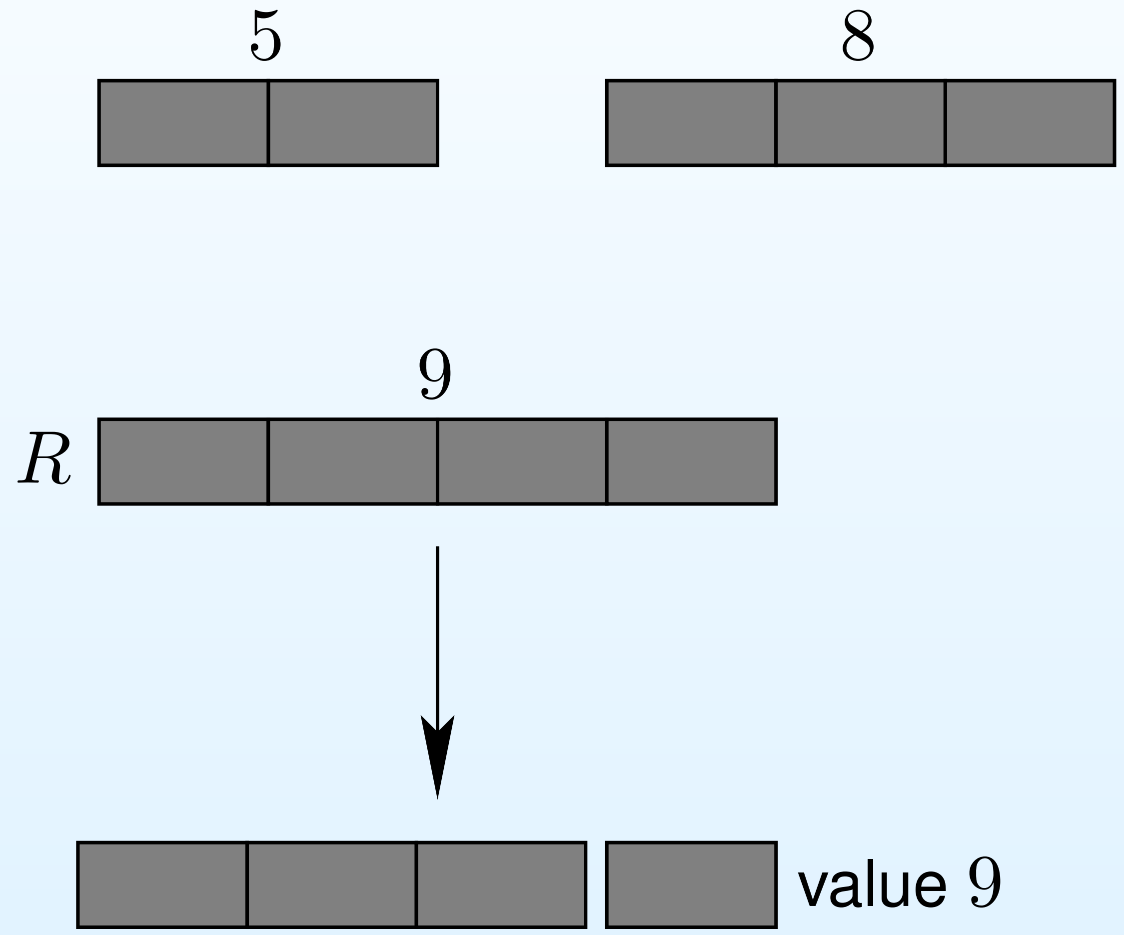
The rod cutting problem

- Example (numbers are prices):



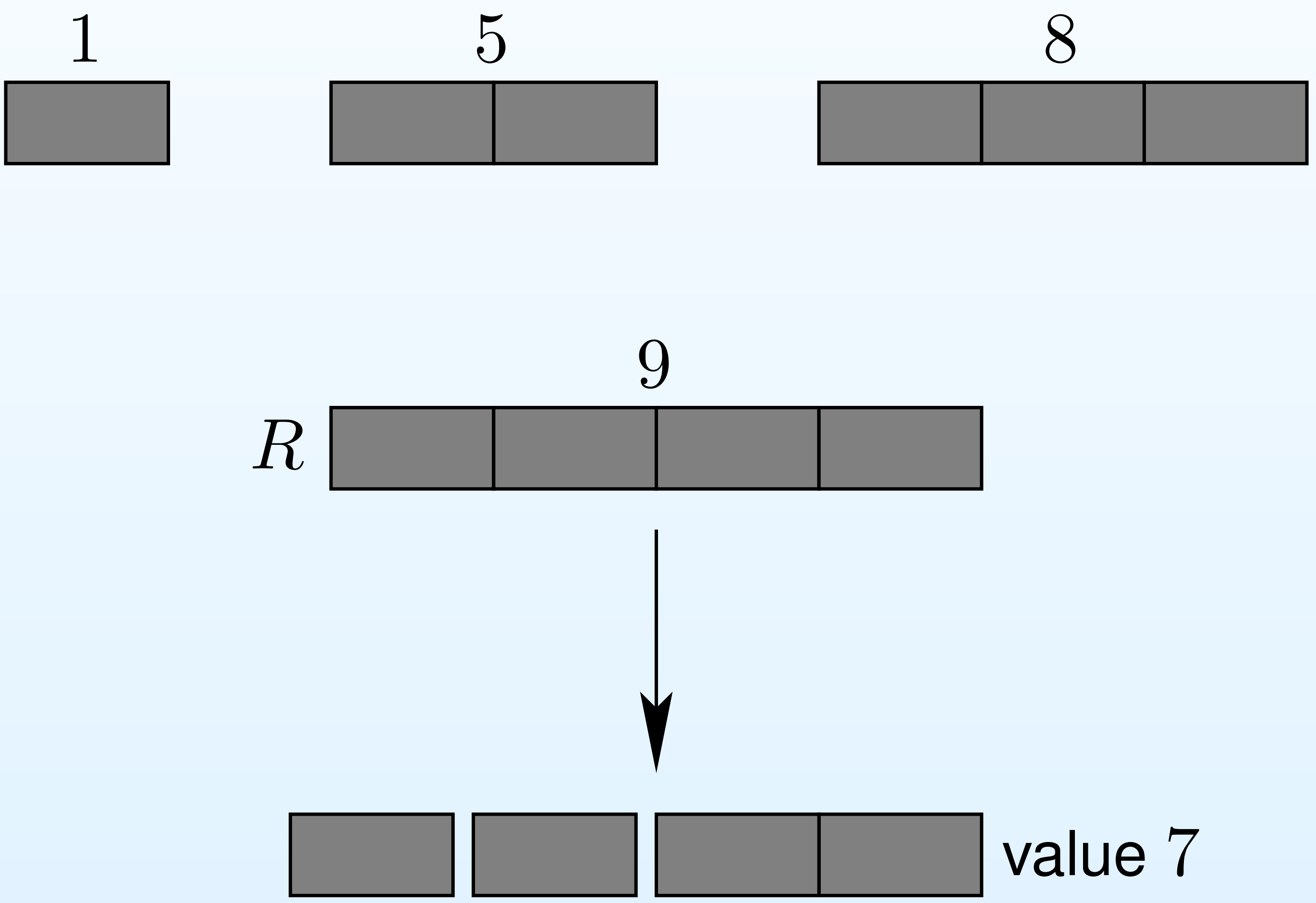
The rod cutting problem

- Example (numbers are prices):



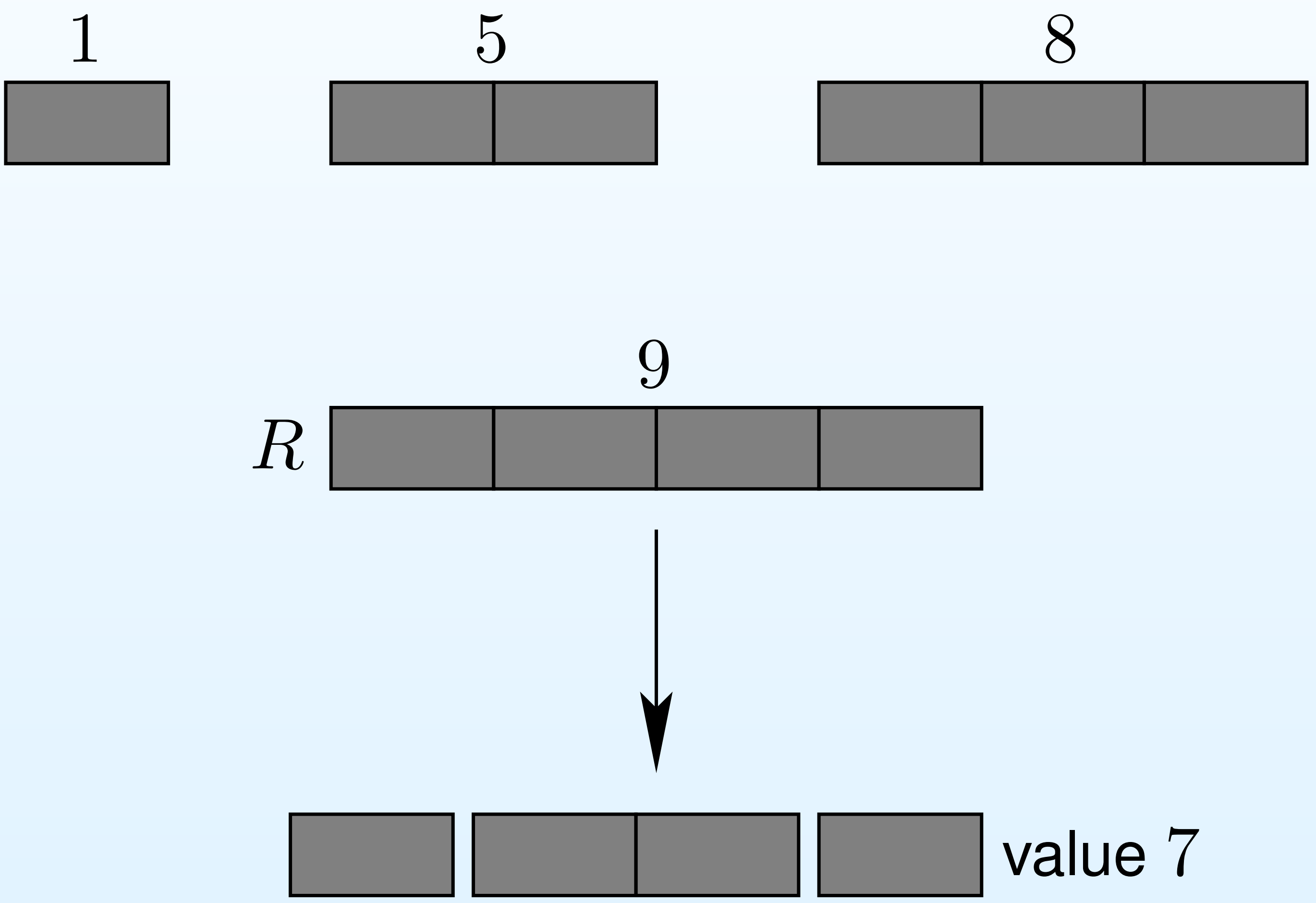
The rod cutting problem

- Example (numbers are prices):



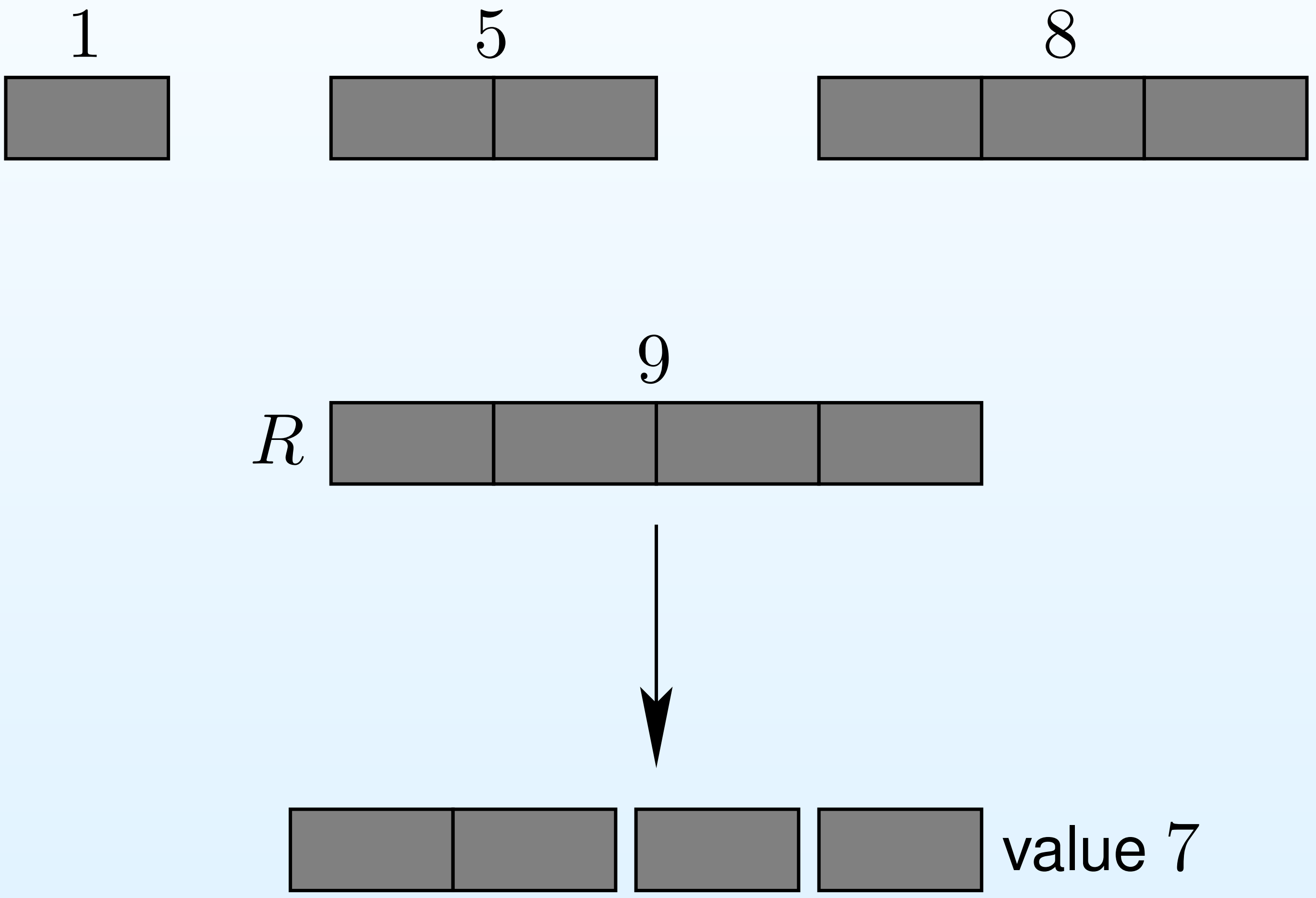
The rod cutting problem

- Example (numbers are prices):



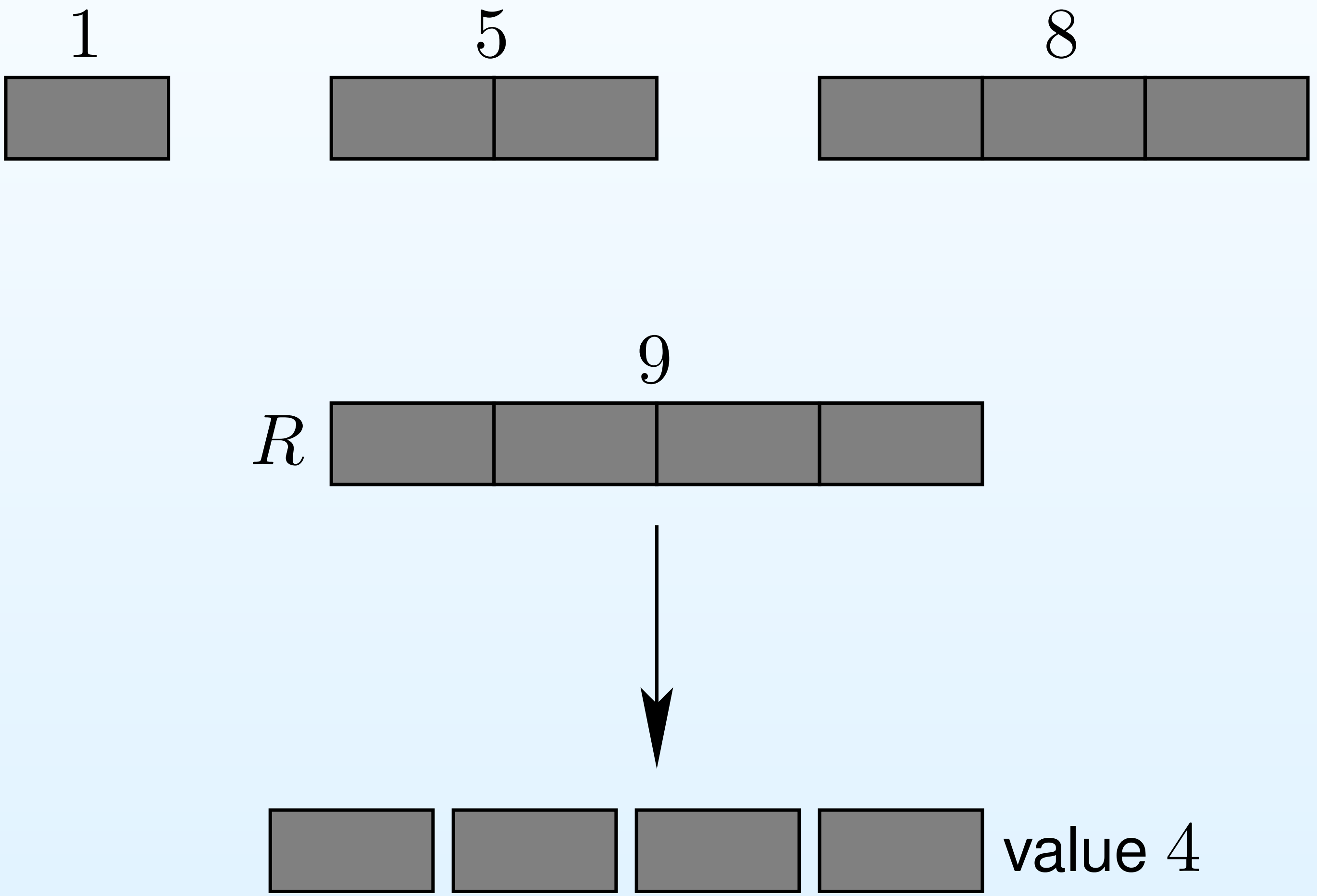
The rod cutting problem

- Example (numbers are prices):



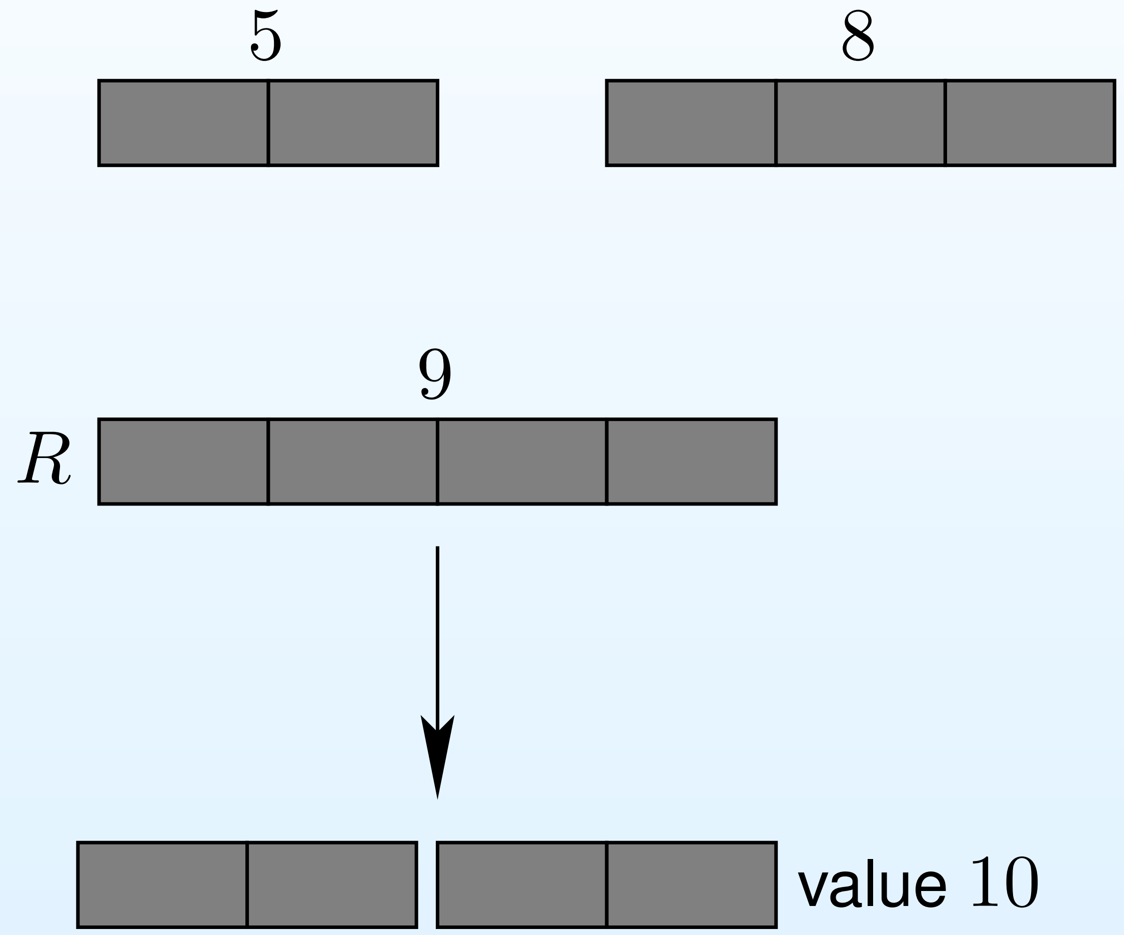
The rod cutting problem

- Example (numbers are prices):



The rod cutting problem

- Example (numbers are prices):



Naive algorithm

- Try all ways of cutting up R.

Naive algorithm

- Try all ways of cutting up R.
- Pick one giving the maximum revenue.

Naive algorithm

- Try all ways of cutting up R.
- Pick one giving the maximum revenue.
- There are 2^{n-1} ways of cutting up R.

Naive algorithm

- Try all ways of cutting up R.
- Pick one giving the maximum revenue.
- There are 2^{n-1} ways of cutting up R.
- Hence, this algorithm runs in exponential time.

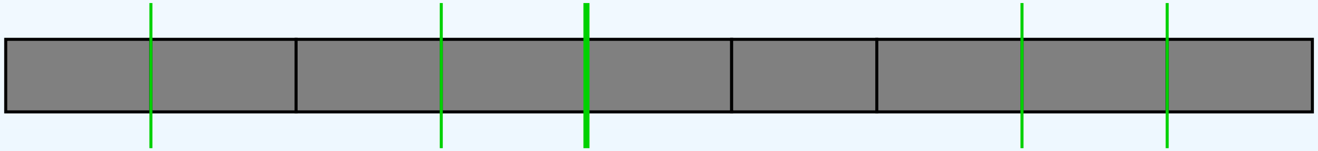
Naive algorithm

- Try all ways of cutting up R.
- Pick one giving the maximum revenue.
- There are 2^{n-1} ways of cutting up R.
- Hence, this algorithm runs in exponential time.
- We will give a much faster algorithm that uses dynamic programming.

Characterizing the structure of an optimal solution

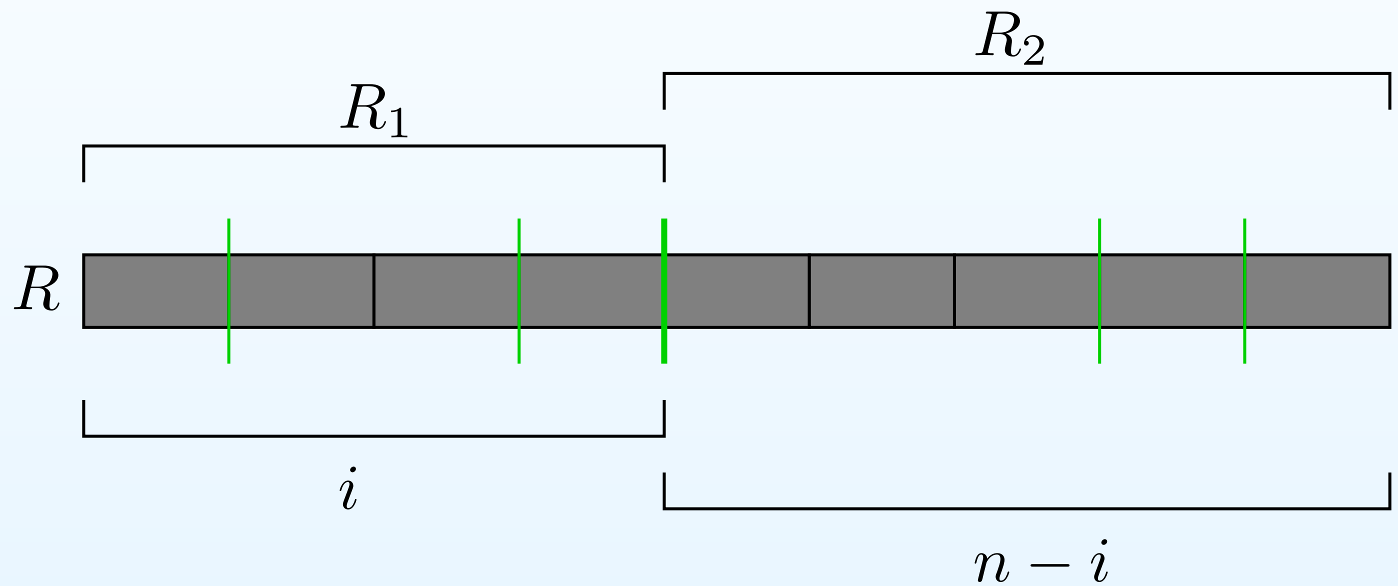
- Suppose someone told us one of the cuts of R in an optimal solution OPT for R (OPT consists of green cuts with the given cut highlighted):

R



Characterizing the structure of an optimal solution

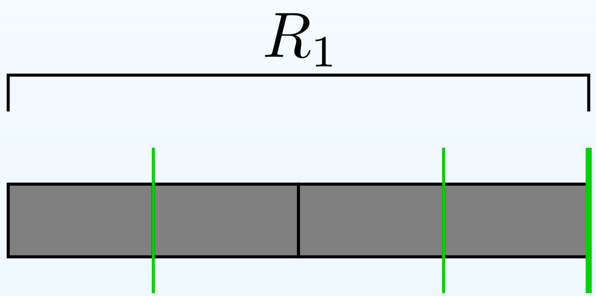
- Suppose someone told us one of the cuts of R in an optimal solution OPT for R (OPT consists of green cuts with the given cut highlighted):



- This cut of OPT partitions R into two smaller rods R_1 and R_2, one of some length i, the other of length $n-i$.

Characterizing the structure of an optimal solution

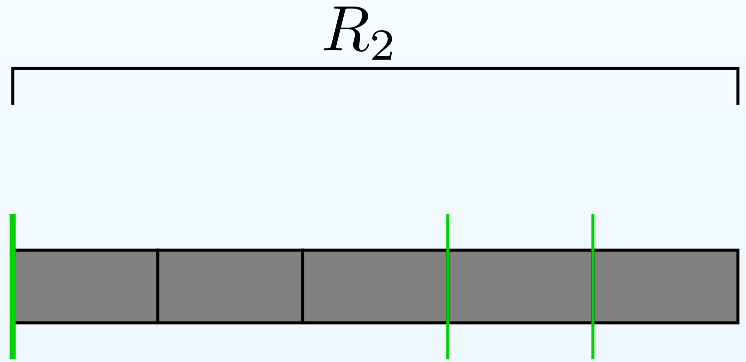
- Suppose someone told us one of the cuts of R in an optimal solution OPT for R (OPT consists of green cuts with the given cut highlighted):



- This cut of OPT partitions R into two smaller rods R_1 and R_2, one of some length i, the other of length $n - i$.
- Restricting OPT to R_1 gives an optimal solution to R_1. (why?)

Characterizing the structure of an optimal solution

- Suppose someone told us one of the cuts of R in an optimal solution OPT for R (OPT consists of green cuts with the given cut highlighted):



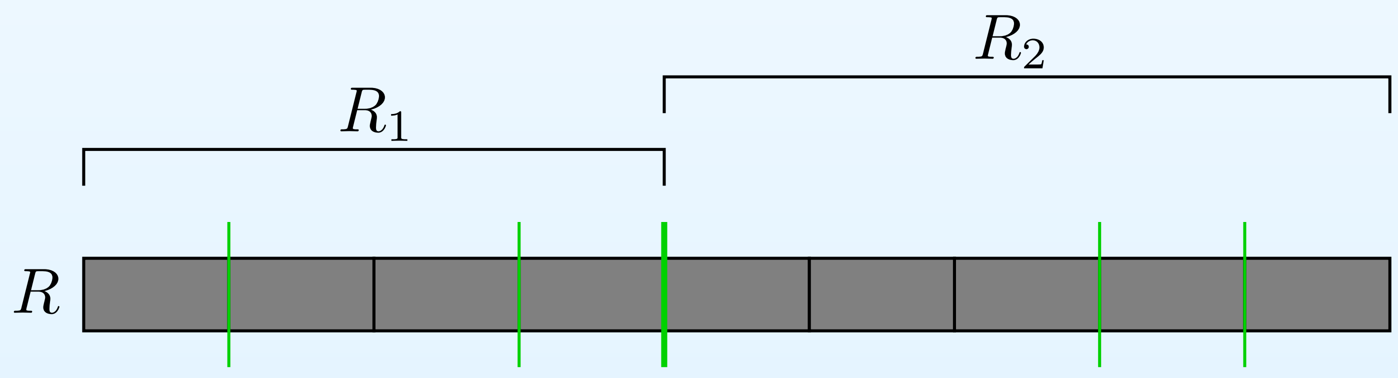
- This cut of OPT partitions R into two smaller rods R_1 and R_2, one of some length i, the other of length $n - i$.
- Restricting OPT to R_1 gives an optimal solution to R_1. (why?)
- Similarly, restricting OPT to R_2 gives an optimal solution to R_2. (why?)

Characterizing the structure of an optimal solution

- It follows that the rod cutting problem exhibits optimal substructure:
  - An optimal solution to the whole problem consists of optimal solutions to subproblems which can be solved independently.

Characterizing the structure of an optimal solution

- It follows that the rod cutting problem exhibits optimal substructure:
  - An optimal solution to the whole problem consists of optimal solutions to subproblems which can be solved independently.
- From the previous slide, solving the rod cutting problem separately for R_1 and for R_2 gives an optimal solution for R as the union of the optimal solutions for the two subproblems:



Characterizing the solution recursively

- Recall: r_n is the maximum revenue we can get from a rod of length n.

Characterizing the solution recursively

- Recall: r_n is the maximum revenue we can get from a rod of length n.
- From the previous slides, there is an i with $1 \leq i \leq n-1$ such that:

$$r_n = r_i + r_{n-i}.$$

Characterizing the solution recursively

- Recall: r_n is the maximum revenue we can get from a rod of length n.
- From the previous slides, there is an i with $1 \leq i \leq n-1$ such that:

$$r_n = r_i + r_{n-i}.$$

- However, we do not know i; this value was given to us by someone.

Characterizing the solution recursively

- Recall: r_n is the maximum revenue we can get from a rod of length n.
- From the previous slides, there is an i with $1 \leq i \leq n-1$ such that:

$$r_n = r_i + r_{n-i}.$$

- However, we do not know i; this value was given to us by someone.
- We also need to handle the case where R should not be cut at all in an optimal solution.

Characterizing the solution recursively

- Recall: r_n is the maximum revenue we can get from a rod of length n.
- From the previous slides, there is an i with $1 \leq i \leq n-1$ such that:

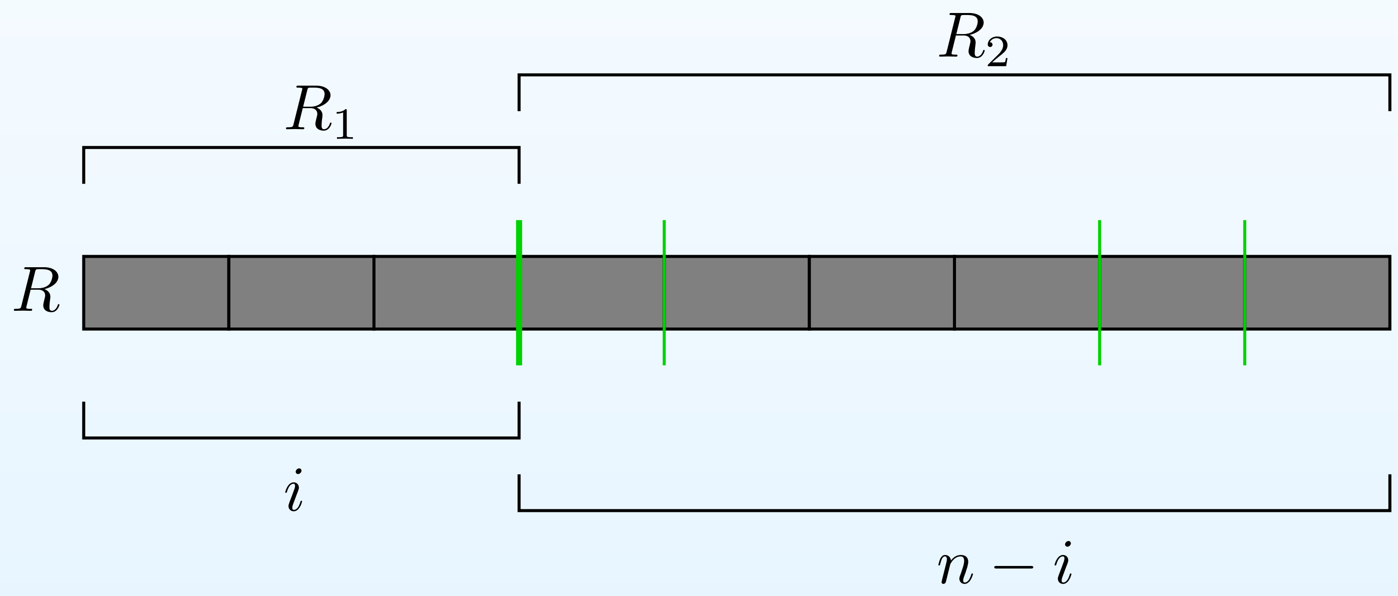
$$r_n = r_i + r_{n-i}.$$

- However, we do not know i; this value was given to us by someone.
- We also need to handle the case where R should not be cut at all in an optimal solution.
- Since our goal is to maximize revenue, we thus get:

$$r_n = \max\{p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \ldots, r_{n-1} + r_1\}.$$

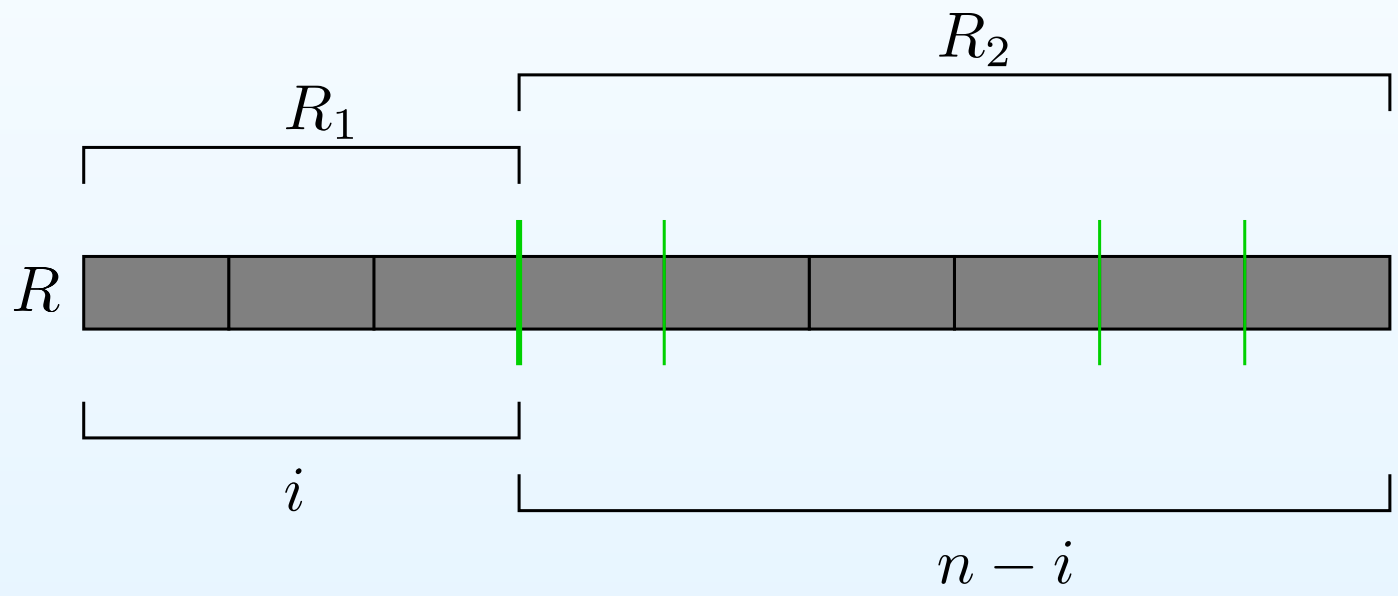
A simpler formulation

- Instead of someone giving us an *arbitrary* cut in an optimal solution, suppose instead we are given the *first* cut from left to right.



A simpler formulation

- Instead of someone giving us an *arbitrary* cut in an optimal solution, suppose instead we are given the *first* cut from left to right.



- If the left side of this cut has length i,

$$r_n = p_i + r_{n-i}.$$

A simpler formulation

- We have shown that

$$r_n = p_i + r_{n-i}.$$

A simpler formulation

- We have shown that

$$r_n = p_i + r_{n-i}.$$

- Now r_n depends on only one related subproblem instead of two.

A simpler formulation

- We have shown that

$$r_n = p_i + r_{n-i}.$$

- Now r_n depends on only one related subproblem instead of two.
- Since we do not know i or whether R should be cut at all, we get

$$r_n = \max_{1 \leq i \leq n} \{p_i + r_{n-i}\},$$

where we define $r_0 = 0$.

A recursive algorithm to compute r_n

- We saw that

$$r_n = \max_{1 \le i \le n} \{p_i + r_{n-i}\}.$$

A recursive algorithm to compute r_n

- We saw that
$$r_n = \max_{1 \le i \le n} \{p_i + r_{n-i}\}.$$

- This suggests the following recursive algorithm:

$\texttt{CUT-ROD}(p, n)$
if $n == 0$ return 0
$q = -\infty$
for $i = 1$ to n
    $q = \max\{q, p[i] + \texttt{CUT-ROD}(p, n-i)\}$
return q

- Here p is an array of length n where $p[i] = p_i$.

Running time of CUT-ROD

- CUT-ROD tries all ways of partitioning the rod:

```
CUT-ROD(p, n)
if n == 0 return 0
q = −∞
for i = 1 to n
    q = max{q, p[i] + CUT-ROD(p, n − i)}
return q
```

Running time of CUT-ROD

- CUT-ROD tries all ways of partitioning the rod:

```
CUT-ROD(p, n)
if n == 0 return 0
q = −∞
for i = 1 to n
    q = max{q, p[i] + CUT-ROD(p, n − i)}
return q
```

- There are 2^{n-1} ways of doing this.

Running time of `CUT-ROD`

- `CUT-ROD` tries all ways of partitioning the rod:

```
CUT-ROD(p, n)
if n == 0 return 0
q = -∞
for i = 1 to n
    q = max{q, p[i] + CUT-ROD(p, n - i)}
return q
```

- There are 2^{n-1} ways of doing this.
- Hence, `CUT-ROD` runs in exponential time.

Running time of CUT-ROD

- CUT-ROD tries all ways of partitioning the rod:

  CUT-ROD(p, n)
  if $n == 0$ return 0
  $q = -\infty$
  for $i = 1$ to n
  $\quad q = \max\{q, p[i] + \texttt{CUT-ROD}(p, n - i)\}$
  return q

- There are 2^{n-1} ways of doing this.
- Hence, CUT-ROD runs in exponential time.
- We will improve this to $\Theta(n^2)$ time using DP.

Memoization: a top-down approach

- We solve the problem top-down using recursion.

Memoization: a top-down approach

- We solve the problem top-down using recursion.
- When a subproblem is solved, it is *memoized*, i.e., it is “remembered”/stored in memory.

Memoization: a top-down approach

- We solve the problem top-down using recursion.
- When a subproblem is solved, it is *memoized*, i.e., it is "remembered"/stored in memory.
- If a memoized subproblem is encountered, it is not solved again but simply looked up.

Solving rod-cutting using memoization

- Initialize an array $r[0 \ldots n]$ where each $r[i] = -\infty$.

Solving rod-cutting using memoization

- Initialize an array $r[0 \ldots n]$ where each $r[i] = -\infty$.
- We solve the rod cutting problem with the call MEM-CUT-ROD(p, n, r), where:

```
MEM-CUT-ROD(p, m, r)
if r[m] ≥ 0 then return r[m]
if m == 0 then q = 0
else
    q = −∞
    for i = 1 to m
        q = max{q, p[i] + MEM-CUT-ROD(p, m − i, r)}
r[m] = q
return q
```

Running time of `MEM-CUT-ROD`

```
MEM-CUT-ROD(p, m, r)
1  if r[m] ≥ 0 then return r[m]
2  if m == 0 then q = 0
3  else
4      q = −∞
5      for i = 1 to m
6          q = max{q, p[i] + MEM-CUT-ROD(p, m − i, r)}
7  r[m] = q
8  return q
```

- Over *all* recursive calls, lines 2–8 are executed only once for each $m \in \{0, 1, \ldots, n\}$, for a total of $n + 1$ times.

Running time of `MEM-CUT-ROD`

```
MEM-CUT-ROD(p, m, r)
1  if r[m] ≥ 0 then return r[m]
2  if m == 0 then q = 0
3  else
4      q = −∞
5      for i = 1 to m
6          q = max{q, p[i] + MEM-CUT-ROD(p, m − i, r)}
7  r[m] = q
8  return q
```

- Over *all* recursive calls, lines 2–8 are executed only once for each $m \in \{0, 1, \ldots, n\}$, for a total of $n + 1$ times.
- Executing these lines once takes $O(n)$ time, excluding the time spent in recursive calls in line 6.

Running time of `MEM-CUT-ROD`

```
MEM-CUT-ROD(p, m, r)
1  if r[m] ≥ 0 then return r[m]
2  if m == 0 then q = 0
3  else
4      q = −∞
5      for i = 1 to m
6          q = max{q, p[i] + MEM-CUT-ROD(p, m − i, r)}
7  r[m] = q
8  return q
```

- Over *all* recursive calls, lines 2–8 are executed only once for each $m \in \{0, 1, \ldots, n\}$, for a total of $n + 1$ times.
- Executing these lines once takes $O(n)$ time, excluding the time spent in recursive calls in line 6.
- This gives a total running time of $\Theta(n^2)$.

A bottom-up approach

- Memoization solves a problem top-down.

A bottom-up approach

- Memoization solves a problem top-down.
- We can also solve a problem bottom-up with DP.

A bottom-up approach

- Memoization solves a problem top-down.
- We can also solve a problem bottom-up with DP.
- The idea is to sort subproblems by increasing “size”.

A bottom-up approach

- Memoization solves a problem top-down.
- We can also solve a problem bottom-up with DP.
- The idea is to sort subproblems by increasing “size”.
- Then these subproblems are solved in this order and their solutions are stored.

A bottom-up approach

- Memoization solves a problem top-down.
- We can also solve a problem bottom-up with DP.
- The idea is to sort subproblems by increasing “size”.
- Then these subproblems are solved in this order and their solutions are stored.
- When solving a particular subproblem, the smaller subproblems that its solution depends on have already been solved.

A bottom-up approach

- Memoization solves a problem top-down.
- We can also solve a problem bottom-up with DP.
- The idea is to sort subproblems by increasing “size”.
- Then these subproblems are solved in this order and their solutions are stored.
- When solving a particular subproblem, the smaller subproblems that its solution depends on have already been solved.
- Hence, we do not need any recursion, only look-ups.

Solving rod-cutting bottom-up

- The following algorithm solves the problem bottom-up:

```
BOTTOM-UP-CUT-ROD(p, n)
let r[0 ... n] be a new array
r[0] = 0
for j = 1 to n
    q = -∞
    for i = 1 to j
        q = max{q, p[i] + r[j - i]}
    r[j] = q
return r[n]
```

Running time of `BOTTOM-UP-CUT-ROD`

```
BOTTOM-UP-CUT-ROD(p, n)
1  let r[0 . . . n] be a new array
2  r[0] = 0
3  for j = 1 to n
4      q = −∞
5      for i = 1 to j
6          q = max{q, p[i] + r[j − i]}
7      r[j] = q
8  return r[n]
```

- There are n iterations of the for-loop in lines 3–7.

Running time of `BOTTOM-UP-CUT-ROD`

```
BOTTOM-UP-CUT-ROD(p, n)
1  let r[0 . . . n] be a new array
2  r[0] = 0
3  for j = 1 to n
4      q = −∞
5      for i = 1 to j
6          q = max{q, p[i] + r[j − i]}
7      r[j] = q
8  return r[n]
```

- There are n iterations of the for-loop in lines 3–7.
- For each of these iterations, there are at most n iterations of the for-loop in lines 5–6.

Running time of `BOTTOM-UP-CUT-ROD`

```
BOTTOM-UP-CUT-ROD(p, n)
1  let r[0 . . . n] be a new array
2  r[0] = 0
3  for j = 1 to n
4      q = −∞
5      for i = 1 to j
6          q = max{q, p[i] + r[j − i]}
7      r[j] = q
8  return r[n]
```

- There are n iterations of the for-loop in lines 3–7.
- For each of these iterations, there are at most n iterations of the for-loop in lines 5–6.
- Total running time: $\Theta(n^2)$.

Pros and cons of the two types of DP

- Bottom-up DP:

Pros and cons of the two types of DP

- Bottom-up DP:
  - Has the advantage of avoiding recursion.

Pros and cons of the two types of DP

- Bottom-up DP:
  - Has the advantage of avoiding recursion.
  - For-loops typically give smaller constant factors in the running time.

Pros and cons of the two types of DP

- Bottom-up DP:
  - Has the advantage of avoiding recursion.
  - For-loops typically give smaller constant factors in the running time.
- Top-down DP with memoization:

Pros and cons of the two types of DP

- Bottom-up DP:
  - Has the advantage of avoiding recursion.
  - For-loops typically give smaller constant factors in the running time.
- Top-down DP with memoization:
  - Only solves the subproblems that are needed.

Pros and cons of the two types of DP

- Bottom-up DP:
  - Has the advantage of avoiding recursion.
  - For-loops typically give smaller constant factors in the running time.
- Top-down DP with memoization:
  - Only solves the subproblems that are needed.
  - This is different from bottom-up DP which solves all subproblems.

Pros and cons of the two types of DP

- Bottom-up DP:
  - Has the advantage of avoiding recursion.
  - For-loops typically give smaller constant factors in the running time.
- Top-down DP with memoization:
  - Only solves the subproblems that are needed.
  - This is different from bottom-up DP which solves all subproblems.
- Conclusion: whether to use bottom-up or top-down DP depends on the problem considered.

Finding a solution

- So far, we have only given algorithms that find the revenue r_n of an optimal solution to the rod-cutting problem.

Finding a solution

- So far, we have only given algorithms that find the revenue r_n of an optimal solution to the rod-cutting problem.
- We would also like to know the solution itself, i.e., how to cut the rod.

Finding a solution

- So far, we have only given algorithms that find the revenue r_n of an optimal solution to the rod-cutting problem.
- We would also like to know the solution itself, i.e., how to cut the rod.
- We do this by recording the choices made by the DP algorithm.

Finding a solution

- The following extension of `BOTTOM-UP-CUT-ROD` returns also an array $s[0 \ldots n]$.

Finding a solution

- The following extension of `BOTTOM-UP-CUT-ROD` returns also an array $s[0 \ldots n]$.
- $s[i]$ indicates where to make the first cut in a rod of length i:

Finding a solution

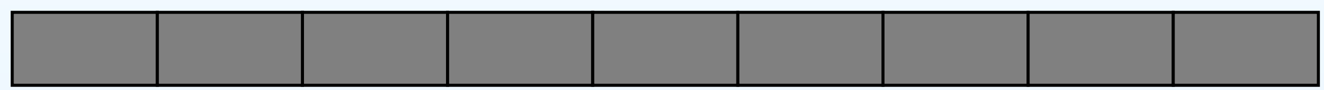
- The following extension of `BOTTOM-UP-CUT-ROD` returns also an array $s[0 \ldots n]$.
- $s[i]$ indicates where to make the first cut in a rod of length i:

```
EXT-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
    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
```

Using the s-array to find a solution

- Having computed the s-array with EXT-BOTTOM-UP-CUT-ROD, we can use s to find an optimal way of cutting up the rod:

R

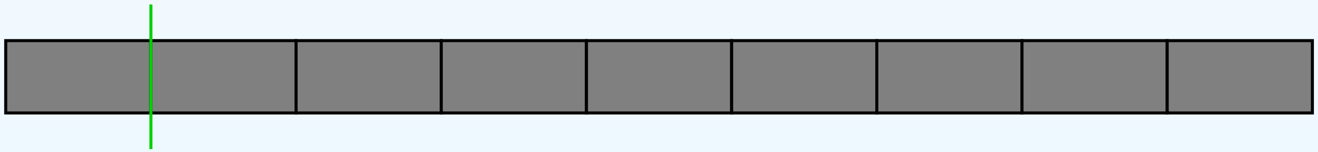


Using the s-array to find a solution

- Having computed the s-array with EXT-BOTTOM-UP-CUT-ROD, we can use s to find an optimal way of cutting up the rod:

$$s[9] = 1$$

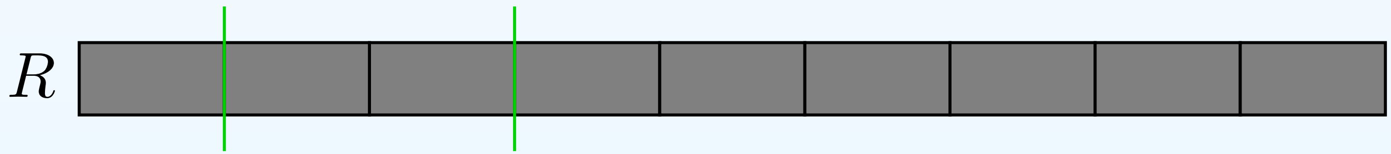
R



Using the s-array to find a solution

- Having computed the s-array with EXT-BOTTOM-UP-CUT-ROD, we can use s to find an optimal way of cutting up the rod:

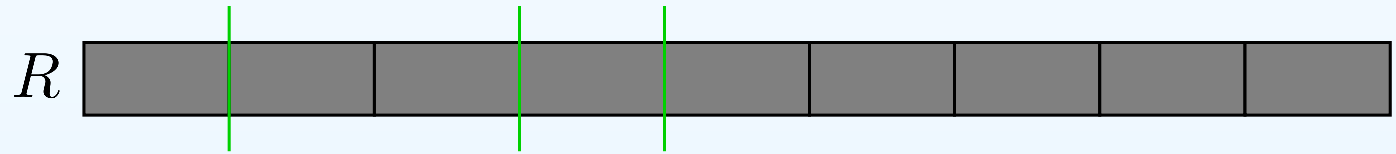
$$s[8] = 2$$



Using the s-array to find a solution

- Having computed the s-array with EXT-BOTTOM-UP-CUT-ROD, we can use s to find an optimal way of cutting up the rod:

$$s[6] = 1$$



Using the s-array to find a solution

- Having computed the s-array with EXT-BOTTOM-UP-CUT-ROD, we can use s to find an optimal way of cutting up the rod:

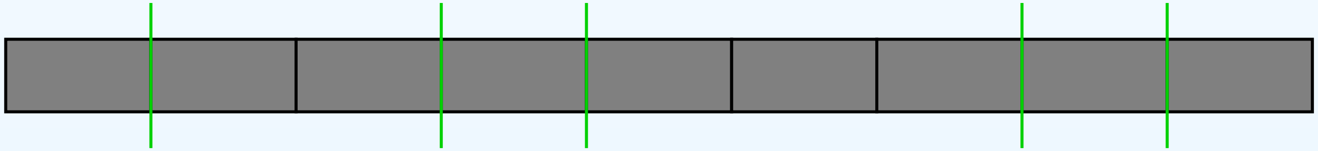
$$s[5] = 3$$

Using the s-array to find a solution

- Having computed the s-array with EXT-BOTTOM-UP-CUT-ROD, we can use s to find an optimal way of cutting up the rod:

$$s[2] = 1$$

R



Comparing DNA strands

- The ability to compare DNA sequences is important in biology.

Comparing DNA strands

- The ability to compare DNA sequences is important in biology.
- A DNA strand can be represented by a sequence of letters from $\{A, C, G, T\}$.

Comparing DNA strands

- The ability to compare DNA sequences is important in biology.
- A DNA strand can be represented by a sequence of letters from $\{A, C, G, T\}$.
- Example:

$$S_1 = ACGCTAC \text{ and } S_2 = CTGACA.$$

Comparing DNA strands

- The ability to compare DNA sequences is important in biology.
- A DNA strand can be represented by a sequence of letters from $\{A, C, G, T\}$.
- Example:

$$S_1 = ACGCTAC \text{ and } S_2 = CTGACA.$$

- We want a measure for how similar S_1 and S_2 are.

Comparing DNA strands

- The ability to compare DNA sequences is important in biology.
- A DNA strand can be represented by a sequence of letters from $\{A, C, G, T\}$.
- Example:

$$S_1 = ACGCTAC \text{ and } S_2 = CTGACA.$$

- We want a measure for how similar S_1 and S_2 are.
- One measure is the length of a *longest common subsequence* (LCS).

Comparing DNA strands

- The ability to compare DNA sequences is important in biology.
- A DNA strand can be represented by a sequence of letters from $\{A, C, G, T\}$.
- Example:

$$S_1 = ACGCTAC \text{ and } S_2 = CTGACA.$$

- We want a measure for how similar S_1 and S_2 are.
- One measure is the length of a *longest common subsequence* (LCS).
- This is a longest string which is a subsequence of both S_1 and S_2.

Comparing DNA strands

- The ability to compare DNA sequences is important in biology.
- A DNA strand can be represented by a sequence of letters from $\{A, C, G, T\}$.
- Example:

$$S_1 = ACGCTAC \text{ and } S_2 = CTGACA.$$

- We want a measure for how similar S_1 and S_2 are.
- One measure is the length of a *longest common subsequence* (LCS).
- This is a longest string which is a subsequence of both S_1 and S_2.
- For the case above, an LCS of S_1 and S_2 is $CGCA$.

Comparing DNA strands

- The ability to compare DNA sequences is important in biology.
- A DNA strand can be represented by a sequence of letters from $\{A, C, G, T\}$.
- Example:

$$S_1 = ACGCTAC \text{ and } S_2 = CTGACA.$$

- We want a measure for how similar S_1 and S_2 are.
- One measure is the length of a *longest common subsequence* (LCS).
- This is a longest string which is a subsequence of both S_1 and S_2.
- For the case above, an LCS of S_1 and S_2 is $CGCA$.
- Notation: $\mathrm{LCS}(S_1, S_2)$ denotes some LCS of S_1 and S_2.

The longest common subsequence problem

- We are given two sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$.

The longest common subsequence problem

- We are given two sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$.
- Goal: find $\mathrm{LCS}(X, Y)$.

The longest common subsequence problem

- We are given two sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$.
- Goal: find $\mathrm{LCS}(X, Y)$.
- Notation used in the following:
  - Let $Z = \langle z_1, z_2 \ldots, z_k \rangle = \mathrm{LCS}(X, Y)$.

The longest common subsequence problem

- We are given two sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$.
- Goal: find $\mathrm{LCS}(X, Y)$.
- Notation used in the following:
  - Let $Z = \langle z_1, z_2 \ldots, z_k \rangle = \mathrm{LCS}(X, Y)$.
  - Denote by X_i the subsequence $\langle x_1, x_2, \ldots, x_i \rangle$ of X.

The longest common subsequence problem

- We are given two sequences $X = \langle x_1, x_2, \ldots, x_m \rangle$ and $Y = \langle y_1, y_2, \ldots, y_n \rangle$.
- Goal: find $\mathrm{LCS}(X, Y)$.
- Notation used in the following:
  - Let $Z = \langle z_1, z_2 \ldots, z_k \rangle = \mathrm{LCS}(X, Y)$.
  - Denote by X_i the subsequence $\langle x_1, x_2, \ldots, x_i \rangle$ of X.
  - Define Y_i and Z_i similarly.

Optimal substructure

- We have the following optimal substructure:

Optimal substructure

- We have the following optimal substructure:
  1. $x_m = y_n \Rightarrow z_k = x_m = y_n \wedge Z_{k-1} = \mathrm{LCS}(X_{m-1}, Y_{n-1})$.

Optimal substructure

- We have the following optimal substructure:
  1. $x_m = y_n \Rightarrow z_k = x_m = y_n \wedge Z_{k-1} = \mathrm{LCS}(X_{m-1}, Y_{n-1})$.
  2. $x_m \neq y_n \wedge z_k \neq x_m \Rightarrow Z = \mathrm{LCS}(X_{m-1}, Y)$.

Optimal substructure

- We have the following optimal substructure:
    1. $x_m = y_n \Rightarrow z_k = x_m = y_n \wedge Z_{k-1} = \mathrm{LCS}(X_{m-1}, Y_{n-1})$.
    2. $x_m \neq y_n \wedge z_k \neq x_m \Rightarrow Z = \mathrm{LCS}(X_{m-1}, Y)$.
    3. $x_m \neq y_n \wedge z_k \neq y_n \Rightarrow Z = \mathrm{LCS}(X, Y_{n-1})$.

Optimal substructure

- We have the following optimal substructure:
  1. $x_m = y_n \Rightarrow z_k = x_m = y_n \wedge Z_{k-1} = \mathrm{LCS}(X_{m-1}, Y_{n-1})$.
  2. $x_m \neq y_n \wedge z_k \neq x_m \Rightarrow Z = \mathrm{LCS}(X_{m-1}, Y)$.
  3. $x_m \neq y_n \wedge z_k \neq y_n \Rightarrow Z = \mathrm{LCS}(X, Y_{n-1})$.
- Example, case 1:

$$
\begin{aligned}
X &= ****A & X_{m-1} &= **** \\
Y &= ******A & Y_{n-1} &= ****** \\
Z &= ***A & Z_{k-1} &= ***
\end{aligned}
$$

Optimal substructure

- We have the following optimal substructure:

    1. $x_m = y_n \Rightarrow z_k = x_m = y_n \wedge Z_{k-1} = \mathrm{LCS}(X_{m-1}, Y_{n-1})$.
    2. $x_m \neq y_n \wedge z_k \neq x_m \Rightarrow Z = \mathrm{LCS}(X_{m-1}, Y)$.
    3. $x_m \neq y_n \wedge z_k \neq y_n \Rightarrow Z = \mathrm{LCS}(X, Y_{n-1})$.

- Example, case 1:

$$
\begin{aligned}
X &= ****A & \qquad X_{m-1} &= **** \\
Y &= ******A & \qquad Y_{n-1} &= ****** \\
Z &= ***A & \qquad Z_{k-1} &= ***
\end{aligned}
$$

- Example, case 2:

$$
\begin{aligned}
X &= ****A & \qquad X_{m-1} &= **** \\
Y &= ******B & \qquad Y &= ******B \\
Z &= ***C & \qquad Z &= ***C
\end{aligned}
$$

Optimal substructure

- We have the following optimal substructure:

  1. $x_m = y_n \Rightarrow z_k = x_m = y_n \wedge Z_{k-1} = \mathrm{LCS}(X_{m-1}, Y_{n-1})$.
  2. $x_m \neq y_n \wedge z_k \neq x_m \Rightarrow Z = \mathrm{LCS}(X_{m-1}, Y)$.
  3. $x_m \neq y_n \wedge z_k \neq y_n \Rightarrow Z = \mathrm{LCS}(X, Y_{n-1})$.

- Example, case 1:

$$
\begin{aligned}
X &= ****A & X_{m-1} &= **** \\
Y &= ******A & Y_{n-1} &= ****** \\
Z &= ***A & Z_{k-1} &= ***
\end{aligned}
$$

- Example, case 2:

$$
\begin{aligned}
X &= ****A & X_{m-1} &= **** \\
Y &= ******B & Y &= ******B \\
Z &= ***C & Z &= ***C
\end{aligned}
$$

- We prove parts 1 and 2 (part 3 is symmetric to part 2).

Optimal substructure: proof of part 1

- We first show $x_m = y_n \Rightarrow z_k = x_m = y_n$.

Optimal substructure: proof of part 1

- We first show $x_m = y_n \Rightarrow z_k = x_m = y_n$.
- Assume for contradiction that $x_m = y_n$ but $z_k \neq x_m = y_n$.

Optimal substructure: proof of part 1

- We first show $x_m = y_n \Rightarrow z_k = x_m = y_n$.
- Assume for contradiction that $x_m = y_n$ but $z_k \neq x_m = y_n$.
- In words, Z does not match the last symbol x_m in X with the last symbol y_n in Y.

Optimal substructure: proof of part 1

- We first show $x_m = y_n \Rightarrow z_k = x_m = y_n$.
- Assume for contradiction that $x_m = y_n$ but $z_k \neq x_m = y_n$.
- In words, Z does not match the last symbol x_m in X with the last symbol y_n in Y.
- Example:

$$X = ****A$$
$$Y = ******A$$
$$Z = ***B$$

Optimal substructure: proof of part 1

- We first show $x_m = y_n \Rightarrow z_k = x_m = y_n$.
- Assume for contradiction that $x_m = y_n$ but $z_k \neq x_m = y_n$.
- In words, Z does not match the last symbol x_m in X with the last symbol y_n in Y.
- Example:

$$X = ****A$$
$$Y = ******A$$
$$Z = ***B$$

- But then Zx_m is a common subsequence of X and Y of length $|Z| + 1 > |Z|$, contradicting that $Z = \mathrm{LCS}(X, Y)$.

Optimal substructure: proof of part 1

- We first show $x_m = y_n \Rightarrow z_k = x_m = y_n$.
- Assume for contradiction that $x_m = y_n$ but $z_k \neq x_m = y_n$.
- In words, Z does not match the last symbol x_m in X with the last symbol y_n in Y.
- Example:

$$X = ****A$$
$$Y = ******A$$
$$Z = ***B$$

- But then Zx_m is a common subsequence of X and Y of length $|Z| + 1 > |Z|$, contradicting that $Z = \mathrm{LCS}(X, Y)$.
- For our example above, $Zx_m = ***BA$.

Optimal substructure: proof of part 1

- We first show $x_m = y_n \Rightarrow z_k = x_m = y_n$.
- Assume for contradiction that $x_m = y_n$ but $z_k \neq x_m = y_n$.
- In words, Z does not match the last symbol x_m in X with the last symbol y_n in Y.
- Example:

$$X = ****A$$
$$Y = ******A$$
$$Z = ***B$$

- But then Zx_m is a common subsequence of X and Y of length $|Z| + 1 > |Z|$, contradicting that $Z = \mathrm{LCS}(X, Y)$.
- For our example above, $Zx_m = ***BA$.
- We conclude that $x_m = y_n \Rightarrow z_k = x_m = y_n$.

Optimal substructure: proof of part 1

- Next we show that $x_m = y_n \Rightarrow Z_{k-1} = \mathrm{LCS}(X_{m-1}, Y_{n-1})$.

Optimal substructure: proof of part 1

- Next we show that $x_m = y_n \Rightarrow Z_{k-1} = \mathrm{LCS}(X_{m-1}, Y_{n-1})$.
- Assume for contradiction that $x_m = y_n$ but that Z_{k-1} is not $\mathrm{LCS}(X_{m-1}, Y_{n-1})$.

Optimal substructure: proof of part 1

- Next we show that $x_m = y_n \Rightarrow Z_{k-1} = \mathrm{LCS}(X_{m-1}, Y_{n-1})$.
- Assume for contradiction that $x_m = y_n$ but that Z_{k-1} is not $\mathrm{LCS}(X_{m-1}, Y_{n-1})$.
- Let $W = \mathrm{LCS}(X_{m-1}, Y_{n-1})$.

Optimal substructure: proof of part 1

- Next we show that $x_m = y_n \Rightarrow Z_{k-1} = \mathrm{LCS}(X_{m-1}, Y_{n-1})$.
- Assume for contradiction that $x_m = y_n$ but that Z_{k-1} is not $\mathrm{LCS}(X_{m-1}, Y_{n-1})$.
- Let $W = \mathrm{LCS}(X_{m-1}, Y_{n-1})$.
- Since $z_k = x_m = y_n$, Z_{k-1} is a common subsequence of X_{m-1} and Y_{n-1}.

Optimal substructure: proof of part 1

- Next we show that $x_m = y_n \Rightarrow Z_{k-1} = \mathrm{LCS}(X_{m-1}, Y_{n-1})$.
- Assume for contradiction that $x_m = y_n$ but that Z_{k-1} is not $\mathrm{LCS}(X_{m-1}, Y_{n-1})$.
- Let $W = \mathrm{LCS}(X_{m-1}, Y_{n-1})$.
- Since $z_k = x_m = y_n$, Z_{k-1} is a common subsequence of X_{m-1} and Y_{n-1}.
- Since Z_{k-1} is not $\mathrm{LCS}(X_{m-1}, Y_{n-1})$, we get $|W| > |Z_{k-1}|$.

Optimal substructure: proof of part 1

- Next we show that $x_m = y_n \Rightarrow Z_{k-1} = \mathrm{LCS}(X_{m-1}, Y_{n-1})$.
- Assume for contradiction that $x_m = y_n$ but that Z_{k-1} is not $\mathrm{LCS}(X_{m-1}, Y_{n-1})$.
- Let $W = \mathrm{LCS}(X_{m-1}, Y_{n-1})$.
- Since $z_k = x_m = y_n$, Z_{k-1} is a common subsequence of X_{m-1} and Y_{n-1}.
- Since Z_{k-1} is not $\mathrm{LCS}(X_{m-1}, Y_{n-1})$, we get $|W| > |Z_{k-1}|$.
- Example:

$$
\begin{aligned}
X &= ****A & X_{m-1} &= **** \\
Y &= ******A & Y_{n-1} &= ****** \\
Z &= ***A & Z_{k-1} &= *** \\
& & W = \mathrm{LCS}(X_{m-1}, Y_{n-1}) &= ****
\end{aligned}
$$

Optimal substructure: proof of part 1

- Example:

$$
\begin{aligned}
X &= ****A & X_{m-1} &= **** \\
Y &= ******A & Y_{n-1} &= ****** \\
Z &= ***A & Z_{k-1} &= *** \\
& & W &= \mathrm{LCS}(X_{m-1}, Y_{n-1}) = ****
\end{aligned}
$$

Optimal substructure: proof of part 1

- Example:

$$X = ****A \qquad X_{m-1} = ****$$
$$Y = ******A \qquad Y_{n-1} = ******$$
$$Z = ***A \qquad Z_{k-1} = ***$$
$$W = \mathrm{LCS}(X_{m-1}, Y_{n-1}) = ****$$

- Wx_m is a common subsequence of X and Y of length

$$|Wx_m| = |W| + 1 > |Z_{k-1}| + 1 = |Z|.$$

Optimal substructure: proof of part 1

- Example:

$$
\begin{aligned}
X &= ****A & X_{m-1} &= **** \\
Y &= ******A & Y_{n-1} &= ****** \\
Z &= ***A & Z_{k-1} &= *** \\
& & W = \mathrm{LCS}(X_{m-1}, Y_{n-1}) &= ****
\end{aligned}
$$

- Wx_m is a common subsequence of X and Y of length

$$|Wx_m| = |W| + 1 > |Z_{k-1}| + 1 = |Z|.$$

- In our example above, $Wx_m = ****A$.

Optimal substructure: proof of part 1

- Example:

$$X = ****A \qquad X_{m-1} = ****$$
$$Y = ******A \qquad Y_{n-1} = ******$$
$$Z = ***A \qquad Z_{k-1} = ***$$
$$W = \mathrm{LCS}(X_{m-1}, Y_{n-1}) = ****$$

- Wx_m is a common subsequence of X and Y of length

$$|Wx_m| = |W| + 1 > |Z_{k-1}| + 1 = |Z|.$$

- In our example above, $Wx_m = ****A$.
- This is a contradiction since $Z = \mathrm{LCS}(X, Y)$.

Optimal substructure: proof of part 1

- Example:

$$
\begin{aligned}
X &= ****A & X_{m-1} &= **** \\
Y &= ******A & Y_{n-1} &= ****** \\
Z &= ***A & Z_{k-1} &= *** \\
& & W &= \mathrm{LCS}(X_{m-1}, Y_{n-1}) = ****
\end{aligned}
$$

- Wx_m is a common subsequence of X and Y of length

$$|Wx_m| = |W| + 1 > |Z_{k-1}| + 1 = |Z|.$$

- In our example above, $Wx_m = ****A$.
- This is a contradiction since $Z = \mathrm{LCS}(X, Y)$.
- We conclude that $x_m = y_n \Rightarrow Z_{k-1} = \mathrm{LCS}(X_{m-1}, Y_{n-1})$.

Optimal substructure: proof of part 2

- We need to show $x_m \neq y_n \wedge z_k \neq x_m \Rightarrow Z = \mathrm{LCS}(X_{m-1}, Y)$.

Optimal substructure: proof of part 2

- We need to show $x_m \neq y_n \wedge z_k \neq x_m \Rightarrow Z = \mathrm{LCS}(X_{m-1}, Y)$.
- Assume for contradiction that $x_m \neq y_n$ and $z_k \neq x_m$ but that Z is not $\mathrm{LCS}(X_{m-1}, Y)$.

Optimal substructure: proof of part 2

- We need to show
  $x_m \neq y_n \wedge z_k \neq x_m \Rightarrow Z = \mathrm{LCS}(X_{m-1}, Y)$.
- Assume for contradiction that $x_m \neq y_n$ and $z_k \neq x_m$ but that Z is not $\mathrm{LCS}(X_{m-1}, Y)$.
- Let $W = \mathrm{LCS}(X_{m-1}, Y)$.

Optimal substructure: proof of part 2

- We need to show $x_m \neq y_n \wedge z_k \neq x_m \Rightarrow Z = \mathrm{LCS}(X_{m-1}, Y)$.
- Assume for contradiction that $x_m \neq y_n$ and $z_k \neq x_m$ but that Z is not $\mathrm{LCS}(X_{m-1}, Y)$.
- Let $W = \mathrm{LCS}(X_{m-1}, Y)$.
- Example:

$$
\begin{aligned}
X &= ******A & X_{m-1} &= ****** \\
Y &= *******B & Y &= *******B \\
Z &= ***C & W &= \mathrm{LCS}(X_{m-1}, Y) = *****
\end{aligned}
$$

Optimal substructure: proof of part 2

- We need to show
  $x_m \neq y_n \wedge z_k \neq x_m \Rightarrow Z = \mathrm{LCS}(X_{m-1}, Y)$.
- Assume for contradiction that $x_m \neq y_n$ and $z_k \neq x_m$ but that Z is not $\mathrm{LCS}(X_{m-1}, Y)$.
- Let $W = \mathrm{LCS}(X_{m-1}, Y)$.
- Example:

$$
\begin{array}{ll}
X = ******A & X_{m-1} = ****** \\
Y = *******B & Y = *******B \\
Z = ***C & W = \mathrm{LCS}(X_{m-1}, Y) = *****
\end{array}
$$

- Since $z_k \neq x_m$, Z is a common subsequence of X_{m-1} and Y.

Optimal substructure: proof of part 2

- We need to show $x_m \neq y_n \wedge z_k \neq x_m \Rightarrow Z = \mathrm{LCS}(X_{m-1}, Y)$.
- Assume for contradiction that $x_m \neq y_n$ and $z_k \neq x_m$ but that Z is not $\mathrm{LCS}(X_{m-1}, Y)$.
- Let $W = \mathrm{LCS}(X_{m-1}, Y)$.
- Example:

$$
\begin{aligned}
X &= {*}{*}{*}{*}{*}{*}A & X_{m-1} &= {*}{*}{*}{*}{*}{*} \\
Y &= {*}{*}{*}{*}{*}{*}{*}B & Y &= {*}{*}{*}{*}{*}{*}{*}B \\
Z &= {*}{*}{*}C & W &= \mathrm{LCS}(X_{m-1}, Y) = {*}{*}{*}{*}{*}
\end{aligned}
$$

- Since $z_k \neq x_m$, Z is a common subsequence of X_{m-1} and Y.
- Since we assume it is not the longest, we have $|W| > |Z|$.

Optimal substructure: proof of part 2

- We need to show $x_m \neq y_n \wedge z_k \neq x_m \Rightarrow Z = \mathrm{LCS}(X_{m-1}, Y)$.
- Assume for contradiction that $x_m \neq y_n$ and $z_k \neq x_m$ but that Z is not $\mathrm{LCS}(X_{m-1}, Y)$.
- Let $W = \mathrm{LCS}(X_{m-1}, Y)$.
- Example:

$$
\begin{aligned}
X &= {*}{*}{*}{*}{*}{*}A & \qquad X_{m-1} &= {*}{*}{*}{*}{*}{*} \\
Y &= {*}{*}{*}{*}{*}{*}{*}B & Y &= {*}{*}{*}{*}{*}{*}{*}B \\
Z &= {*}{*}{*}C & W &= \mathrm{LCS}(X_{m-1}, Y) = {*}{*}{*}{*}{*}
\end{aligned}
$$

- Since $z_k \neq x_m$, Z is a common subsequence of X_{m-1} and Y.
- Since we assume it is not the longest, we have $|W| > |Z|$.
- But W is also a common subsequence of X and Y.

Optimal substructure: proof of part 2

- We need to show
  $x_m \neq y_n \wedge z_k \neq x_m \Rightarrow Z = \mathrm{LCS}(X_{m-1}, Y)$.
- Assume for contradiction that $x_m \neq y_n$ and $z_k \neq x_m$ but that Z is not $\mathrm{LCS}(X_{m-1}, Y)$.
- Let $W = \mathrm{LCS}(X_{m-1}, Y)$.
- Example:

$$
\begin{array}{ll}
X = ******A & X_{m-1} = ****** \\
Y = *******B & Y = *******B \\
Z = ***C & W = \mathrm{LCS}(X_{m-1}, Y) = *****
\end{array}
$$

- Since $z_k \neq x_m$, Z is a common subsequence of X_{m-1} and Y.
- Since we assume it is not the longest, we have $|W| > |Z|$.
- But W is also a common subsequence of X and Y.
- This is a contradiction since $|W| > |Z|$ and $Z = \mathrm{LCS}(X, Y)$.

Optimal substructure: proof of part 2

- We need to show $x_m \neq y_n \wedge z_k \neq x_m \Rightarrow Z = \mathrm{LCS}(X_{m-1}, Y)$.
- Assume for contradiction that $x_m \neq y_n$ and $z_k \neq x_m$ but that Z is not $\mathrm{LCS}(X_{m-1}, Y)$.
- Let $W = \mathrm{LCS}(X_{m-1}, Y)$.
- Example:

$$
\begin{array}{ll}
X = {*}{*}{*}{*}{*}{*}A & X_{m-1} = {*}{*}{*}{*}{*}{*} \\
Y = {*}{*}{*}{*}{*}{*}{*}B & Y = {*}{*}{*}{*}{*}{*}{*}B \\
Z = {*}{*}{*}C & W = \mathrm{LCS}(X_{m-1}, Y) = {*}{*}{*}{*}{*}
\end{array}
$$

- Since $z_k \neq x_m$, Z is a common subsequence of X_{m-1} and Y.
- Since we assume it is not the longest, we have $|W| > |Z|$.
- But W is also a common subsequence of X and Y.
- This is a contradiction since $|W| > |Z|$ and $Z = \mathrm{LCS}(X, Y)$.
- This shows $x_m \neq y_n \wedge z_k \neq x_m \Rightarrow Z = \mathrm{LCS}(X_{m-1}, Y)$.

Expressing the problem recursively

- For each $i \in \{0, 1, \ldots, m\}$ and $j \in \{0, 1, \ldots, n\}$, define

$$c[i, j] = |\,\mathrm{LCS}(X_i, Y_j)|.$$

Expressing the problem recursively

- For each $i \in \{0, 1, \ldots, m\}$ and $j \in \{0, 1, \ldots, n\}$, define

$$c[i, j] = |\,\mathrm{LCS}(X_i, Y_j)|.$$

- We consider different cases:

Expressing the problem recursively

- For each $i \in \{0, 1, \ldots, m\}$ and $j \in \{0, 1, \ldots, n\}$, define

$$c[i, j] = |\,\mathrm{LCS}(X_i, Y_j)|.$$

- We consider different cases:
  - $i = 0$ or $j = 0$:

$$c[i, j] = 0$$

Expressing the problem recursively

- For each $i \in \{0, 1, \ldots, m\}$ and $j \in \{0, 1, \ldots, n\}$, define

$$c[i, j] = |\,\mathrm{LCS}(X_i, Y_j)|.$$

- We consider different cases:
  - $i = 0$ or $j = 0$:

$$c[i, j] = 0$$

  - $i, j > 0$ and $x_i = y_j$:

$$c[i, j] = c[i - 1, j - 1] + 1$$

Expressing the problem recursively

- For each $i \in \{0, 1, \ldots, m\}$ and $j \in \{0, 1, \ldots, n\}$, define

$$c[i, j] = |\,\mathrm{LCS}(X_i, Y_j)|.$$

- We consider different cases:
  - $i = 0$ or $j = 0$:

$$c[i, j] = 0$$

  - $i, j > 0$ and $x_i = y_j$:

$$c[i, j] = c[i-1, j-1] + 1$$

  - $i, j > 0$ and $x_i \neq y_j$:

$$c[i, j] = \max\{c[i, j-1], c[i-1, j]\}$$

A fast DP algorithm

- A $\Theta(mn)$ time bottom-up DP algorithm:

```
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] = "↖"
        else if 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
```

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

$i \backslash j$		0	1	2	3	4	5	6
			B	D	C	A	B	A
0								
1	A							
2	B							
3	C							
4	B							
5	D							
6	A							
7	B							

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0						
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

j		0	1	2	3	4	5	6
i			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0						
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

	j	0	1	2	3	4	5	6
i			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0					
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

$i \backslash j$		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0				
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0			
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

$i \backslash j$		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1		
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0						
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1					
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1				
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1			
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1		
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0						
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1					
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB$, $Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1				
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

$i \backslash j$		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2			
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2		
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

	j	0	1	2	3	4	5	6
i			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

	j	0	1	2	3	4	5	6
i			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0						
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

$i \backslash j$		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1					
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

$i \backslash j$		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1				
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2			
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB$, $Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2		
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1 B	2 D	3 C	4 A	5 B	6 A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0						
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1					
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2				
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB$, $Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2			
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2		
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB$, $Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0						
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB$, $Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1					
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2				
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

$i \backslash j$		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2			
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3		
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB$, $Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7	B	0						

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7	B	0	$\nwarrow$ 1					

A fast DP algorithm

- Example with $X = ABCBDAB$, $Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7	B	0	$\nwarrow$ 1	$\uparrow$ 2				

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7	B	0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2			

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7	B	0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3		

A fast DP algorithm

- Example with $X = ABCBDAB$, $Y = BDCABA$:

i \ j	0	1 B	2 D	3 C	4 A	5 B	6 A
0	0	0	0	0	0	0	0
1 A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2 B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3 C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4 B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5 D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6 A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7 B	0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow 1$	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow 1$	$\leftarrow 1$	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow 2$
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow 2$	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow 3$
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7	B	0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

A fast DP algorithm

- Example with $X = ABCBDAB$, $Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7	B	0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

A fast DP algorithm

- Example with $X = ABCBDAB$, $Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7	B	0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

A fast DP algorithm

- Example with $X = ABCBDAB$, $Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D	0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7		0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5		0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7		0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C	A	B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow 0$	$\uparrow 0$	$\uparrow 0$	$\nwarrow 1$	$\leftarrow 1$	$\nwarrow 1$
2	B	0	$\nwarrow 1$	$\leftarrow 1$	$\leftarrow 1$	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$
3	C	0	$\uparrow 1$	$\uparrow 1$	$\nwarrow 2$	$\leftarrow 2$	$\uparrow 2$	$\uparrow 2$
4	B	0	$\nwarrow 1$	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\nwarrow 3$	$\leftarrow 3$
5		0	$\uparrow 1$	$\nwarrow 2$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\uparrow 3$
6	A	0	$\uparrow 1$	$\uparrow 2$	$\uparrow 2$	$\nwarrow 3$	$\uparrow 3$	$\nwarrow 4$
7		0	$\nwarrow 1$	$\uparrow 2$	$\uparrow 2$	$\uparrow 3$	$\nwarrow 4$	$\uparrow 4$

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C		B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5		0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7		0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B	D	C		B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5		0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7		0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B		C		B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5		0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7		0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

A fast DP algorithm

- Example with $X = ABCBDAB, Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B		C		B	A
0		0	0	0	0	0	0	0
1	A	0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5		0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7		0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

A fast DP algorithm

- Example with $X = ABCBDAB$, $Y = BDCABA$:

i \ j		0	1	2	3	4	5	6
			B		C		B	A
0		0	0	0	0	0	0	0
1		0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B	0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C	0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B	0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5		0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A	0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7		0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 4

Plan for the lecture on February 20

- Greedy algorithms

Plan for the lecture on February 20

- Greedy algorithms
- We solve two problems with greedy algorithms:
  - Activity selection
  - Huffman codes