CS 577- Intro to Algorithms

Dynamic Programming (Part 3)

Dieter van Melkebeek

September 29, 2020

Outline

Outline

Paradigm

Recursive approach such that:

1. The number of distinct subproblems in the recursion tree is small.
2. Each of those subproblems is solved only once.

Outline

Paradigm

Recursive approach such that:

1. The number of distinct subproblems in the recursion tree is small.
2. Each of those subproblems is solved only once.

Examples

- Computing Fibonacci numbers
- Weighted interval scheduling
- Knapsack problem
- RNA secondary structure

Outline

Paradigm

Recursive approach such that:

1. The number of distinct subproblems in the recursion tree is small.
2. Each of those subproblems is solved only once.

Examples

- Computing Fibonacci numbers
- Weighted interval scheduling
- Knapsack problem
- RNA secondary structure
- Sequence alignment / Edit distance

Subproblems – 1D tables

- Computing Fibonacci numbers:
  Fib(k): kth Fibonacci number F_k
  $0 \leq k \leq n$

- Weighted interval scheduling:
  OPT(k): max total weight achievable w/ meetings $\{k, \ldots, n\}$
  $1 \leq k \leq n + 1$

Subproblems – 2D tables

- Knapsack problem:
  OPT(k, w): max total value achievable with items $\{k, \ldots, n\}$ and weight limit w
  $1 \leq k \leq n+1$ and $0 \leq w \leq W$

- RNA secondary structure:
  OPT(i, j): max number of bonds that can be formed in $R[i, j]$
  $1 \leq i \leq j \leq n$

Sequence Alignment / Edit Distance

Sequence Alignment / Edit Distance

$A = \text{CCGUCUG}$
$B = \text{GCUCGC}$

Sequence Alignment / Edit Distance

$A =$ CCGUCUG
$B =$ GCUCGC

- Alignment of A and B minimizing the number of misses (non-matches and mismatches)

Sequence Alignment / Edit Distance

$A =$ CCGUCUG
$B =$ GCUCGC

- Alignment of A and B minimizing the number of misses (non-matches and mismatches)

$$\begin{array}{cccccccc} C & C & G & U & C & U & G & \\ G & C & & U & C & & G & C \end{array}$$

Sequence Alignment / Edit Distance

$A = \mathrm{CCGUCUG}$
$B = \mathrm{GCUCGC}$

- Alignment of A and B minimizing the number of misses (non-matches and mismatches)

$$\begin{array}{cccccccc} C & C & G & U & C & U & G & \\ G & C & & U & C & & G & C \end{array}$$

- Edit distance $d(A, B)$: minimum number of edits (deletions, substitutions, insertions) to transform A into B

Sequence Alignment / Edit Distance

$A =$ CCGUCUG
$B =$ GCUCGC

- Alignment of A and B minimizing the number of misses (non-matches and mismatches)

$$\begin{array}{cccccccc} C & C & G & U & C & U & G & \\ G & C & & U & C & & G & C \end{array}$$

- Edit distance $d(A, B)$: minimum number of edits (deletions, substitutions, insertions) to transform A into B

CCGUCUG

Sequence Alignment / Edit Distance

$A =$ CCGUCUG
$B =$ GCUCGC

- Alignment of A and B minimizing the number of misses (non-matches and mismatches)

$$\begin{array}{cccccccc} C & C & G & U & C & U & G & \\ G & C & & U & C & & G & C \end{array}$$

- Edit distance $d(A, B)$: minimum number of edits (deletions, substitutions, insertions) to transform A into B

CCGUCUG [insert C at end]
CCGUCUGC

Sequence Alignment / Edit Distance

$A =$ CCGUCUG
$B =$ GCUCGC

- Alignment of A and B minimizing the number of misses (non-matches and mismatches)

$$\begin{array}{cccccccc} C & C & G & U & C & U & G & \\ G & C & & U & C & & G & C \end{array}$$

- Edit distance $d(A, B)$: minimum number of edits (deletions, substitutions, insertions) to transform A into B

CCGUCUG [insert C at end]
CCGUCUGC [delete last U]
CCGUCGC

Sequence Alignment / Edit Distance

$A =$ CCGUCUG
$B =$ GCUCGC

- Alignment of A and B minimizing the number of misses (non-matches and mismatches)

  $$\begin{array}{cccccccc} C & C & G & U & C & U & G & \\ G & C & & U & C & & G & C \end{array}$$

- Edit distance $d(A, B)$: minimum number of edits (deletions, substitutions, insertions) to transform A into B

  CCGUCUG [insert C at end]
  CCGUCUGC [delete last U]
  CCGUCGC [delete first G]
  CCUCGC

Sequence Alignment / Edit Distance

$A =$ CCGUCUG
$B =$ GCUCGC

- Alignment of A and B minimizing the number of misses (non-matches and mismatches)

$$\begin{array}{cccccccc} C & C & G & U & C & U & G & \\ G & C & & U & C & & G & C \end{array}$$

- Edit distance $d(A, B)$: minimum number of edits (deletions, substitutions, insertions) to transform A into B

CCGUCUG [insert C at end]
CCGUCUGC [delete last U]
CCGUCGC [delete first G]
CCUCGC [replace first C by G]
GCUCGC

Problem Specifications

Sequence alignment

Input: strings $A[1, \ldots, n]$ and $B[1, \ldots, m]$

Ouput: alignment of A and B that minimizes the total number of misses (deletions, substitutions, insertions)

Problem Specifications

Sequence alignment

Input: strings $A[1, \ldots, n]$ and $B[1, \ldots, m]$

Ouput: alignment of A and B that minimizes the total number of misses (deletions, substitutions, insertions)

Edit distance

Input: strings $A[1, \ldots, n]$ and $B[1, \ldots, m]$

Ouput: $d(A, B) (= \mathrm{OPT}(A, B))$

Principle of optimality

Principle of optimality

Consider alignment at the end.

Principle of optimality

Consider alignment at the end.

- Case 1: Delete $A[n]$

Principle of optimality

Consider alignment at the end.

- Case 1: Delete $A[n]$
  Penalty: 1

Principle of optimality

Consider alignment at the end.

- Case 1: Delete $A[n]$
  Penalty: 1
  Remains to solve problem for $A[1, \ldots, n-1]$ and $B[1, \ldots, m]$

Principle of optimality

Consider alignment at the end.

- Case 1: Delete $A[n]$
  Penalty: 1
  Remains to solve problem for $A[1, \ldots, n-1]$ and $B[1, \ldots, m]$
- Case 2: Insert $B[m]$

Principle of optimality

Consider alignment at the end.

- Case 1: Delete $A[n]$
  Penalty: 1
  Remains to solve problem for $A[1, \ldots, n-1]$ and $B[1, \ldots, m]$
- Case 2: Insert $B[m]$
  Penalty: 1

Principle of optimality

Consider alignment at the end.

- Case 1: Delete $A[n]$
  Penalty: 1
  Remains to solve problem for $A[1, \ldots, n-1]$ and $B[1, \ldots, m]$
- Case 2: Insert $B[m]$
  Penalty: 1
  Remains to solve problem for $A[1, \ldots, n]$ and $B[1, \ldots, m-1]$

Principle of optimality

Consider alignment at the end.

- Case 1: Delete $A[n]$
  Penalty: 1
  Remains to solve problem for $A[1,\ldots,n-1]$ and $B[1,\ldots,m]$
- Case 2: Insert $B[m]$
  Penalty: 1
  Remains to solve problem for $A[1,\ldots,n]$ and $B[1,\ldots,m-1]$
- Case 3: Align $A[n]$ and $B[m]$

Principle of optimality

Consider alignment at the end.

- Case 1: Delete $A[n]$
  Penalty: 1
  Remains to solve problem for $A[1, \dots, n-1]$ and $B[1, \dots, m]$
- Case 2: Insert $B[m]$
  Penalty: 1
  Remains to solve problem for $A[1, \dots, n]$ and $B[1, \dots, m-1]$
- Case 3: Align $A[n]$ and $B[m]$
  Penalty: 1 if $A[n] \neq B[m]$, 0 otherwise

Principle of optimality

Consider alignment at the end.

- Case 1: Delete $A[n]$
  Penalty: 1
  Remains to solve problem for $A[1, \ldots, n-1]$ and $B[1, \ldots, m]$
- Case 2: Insert $B[m]$
  Penalty: 1
  Remains to solve problem for $A[1, \ldots, n]$ and $B[1, \ldots, m-1]$
- Case 3: Align $A[n]$ and $B[m]$
  Penalty: 1 if $A[n] \neq B[m]$, 0 otherwise
  Remains to solve problem for $A[1, \ldots, n-1]$ and
  $B[1, \ldots, m-1]$

Subproblems and Recurrence

Subproblems

$\mathrm{OPT}(i, j) = d(A[1, \ldots, i], B[1, \ldots, j])$

Subproblems and Recurrence

Subproblems

$\mathrm{OPT}(i, j) = d(A[1, \ldots, i], B[1, \ldots, j])$
$0 \leq i \leq n$ and $0 \leq j \leq m$

Subproblems and Recurrence

Subproblems

$\mathrm{OPT}(i, j) = d(A[1, \ldots, i], B[1, \ldots, j])$
$0 \leq i \leq n$ and $0 \leq j \leq m$

Recursive case

$\mathrm{OPT}(i, j) =$

Subproblems and Recurrence

Subproblems

$\mathrm{OPT}(i, j) = d(A[1, \ldots, i], B[1, \ldots, j])$
$0 \le i \le n$ and $0 \le j \le m$

Recursive case

$\mathrm{OPT}(i, j) = \min($
$1 + \mathrm{OPT}(i-1, j)$ [deletion, only for $i \ge 1$]
$1 + \mathrm{OPT}(i, j-1)$ [insertion, only for $j \ge 1$]
$(1 - \delta_{A[i], B[j]}) + \mathrm{OPT}(i-1, j-1)$ [alignment, only for $i, j \ge 1$])

where $\delta_{a,b} \doteq \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$

Subproblems and Recurrence

Subproblems

$\mathrm{OPT}(i,j) = d(A[1,\ldots,i], B[1,\ldots,j])$
$0 \leq i \leq n$ and $0 \leq j \leq m$

Recursive case

$\mathrm{OPT}(i,j) = \min($

$1 + \mathrm{OPT}(i-1,j)$ [deletion, only for $i \geq 1$]

$1 + \mathrm{OPT}(i,j-1)$ [insertion, only for $j \geq 1$]

$(1 - \delta_{A[i],B[j]}) + \mathrm{OPT}(i-1,j-1)$ [alignment, only for $i,j \geq 1$])

where $\delta_{a,b} \doteq \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$

Base case: $\mathrm{OPT}(0,0) = 0$

Subproblems and Recurrence

Subproblems

$\mathrm{OPT}(i,j) = d(A[1,\ldots,i], B[1,\ldots,j])$
$0 \le i \le n$ and $0 \le j \le m$

Recursive case

$\mathrm{OPT}(i,j) = \min($
$\quad 1 + \mathrm{OPT}(i-1,j)$ [deletion, only for $i \ge 1$]
$\quad 1 + \mathrm{OPT}(i,j-1)$ [insertion, only for $j \ge 1$]
$\quad (1 - \delta_{A[i],B[j]}) + \mathrm{OPT}(i-1,j-1)$ [alignment, only for $i,j \ge 1$])

where $\delta_{a,b} \doteq \begin{cases} 1 & \text{if } a = b \\ 0 & \text{otherwise} \end{cases}$

Base case: $\mathrm{OPT}(0,0) = 0$

Answer: $\mathrm{OPT}(n,m)$

Complexity Analysis

Complexity Analysis

Time

- $O(1)$ per cell
- $O(nm)$ cells
- $O(nm)$ total

Space

- $O(\min(n, m))$ for edit distance
- $O(nm)$ for alignment

Reducing Space Complexity for Alignment

Reducing Space Complexity for Alignment

- Need to find shortest path from $(0,0)$ to (n,m).
- Path must touch column $m/2$ somewhere, say in row i^*.
- Once we know i^*, remains to find:
  (a) shortest path from $(0,0)$ to $(i^*, m/2)$, and
  (b) shortest path from $(i^*, m/2)$ to (n,m).

  Both (a) and (b) are instances of the same problem.
- To find i^* compute for each $i \in [n]$:
  (a) $f(i)$: length of shortest path from $(0,0)$ to $(i, m/2)$
  (b) $g(i)$: length of shortest path from $(i, m/2)$ to (n,m)

  Then set i^* to an $i \in [n]$ that minimizes $f(i) + g(i)$.
- As $f(i) = \mathrm{OPT}(i, m/2)$, all of f can be computed in time $O(nm)$ and space $O(n)$ using original algorithm.
- Same applies to g by symmetry (reverse direction of edges).
- Thus, i^* can be computed in time $O(nm)$ and space $O(n)$.

Recursion Tree – space analysis

Recursion Tree – space analysis

- $O(n+m)$ for path [global]
- $O(n)$ for computing i^* [local, reused]
- $O(1)$ per level of recursion [recursion stack]
- Total: $O(n+m) + O(n) + O(\log m) = O(n+m)$

Recursion Tree – time analysis

Recursion Tree – time analysis

Consider instance of dimension $n \times m$

- local work: $c \cdot n \cdot m$
- dimension of children: $i^* \times m/2$ and $(n - i^*) \times m/2$
- local work at children:
  $c \cdot i^* \cdot m/2 + c \cdot (n - i^*) \cdot m/2 = c \cdot (i^* + (n - i^*)) \cdot m/2 = \frac{1}{2} c \cdot n \cdot m$
- Total: $O(nm)$