CS 577- Intro to Algorithms

Dynamic Programming (Part 2)

Dieter van Melkebeek

September 24, 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

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

Discrete Multivariate Optimization

Discrete Multivariate Optimization

Setting

Discrete Multivariate Optimization

Setting

- System consisting of n components.

Discrete Multivariate Optimization

Setting

- System consisting of n components.
- Each component can be in any of a finite number of states.

Discrete Multivariate Optimization

Setting

- System consisting of n components.
- Each component can be in any of a finite number of states.
- Want to set the states of the components so as to optimize an certain objective under certain constraints.

Discrete Multivariate Optimization

Setting

- System consisting of n components.
- Each component can be in any of a finite number of states.
- Want to set the states of the components so as to optimize an certain objective under certain constraints.

Notation

$\mathrm{OPT}(I)$ denotes the optimal *value* of the objective for instance I.

Discrete Multivariate Optimization

Setting

- System consisting of n components.
- Each component can be in any of a finite number of states.
- Want to set the states of the components so as to optimize an certain objective under certain constraints.

Notation

$\text{OPT}(I)$ denotes the optimal *value* of the objective for instance I.

Weighted interval scheduling

Discrete Multivariate Optimization

Setting

- System consisting of n components.
- Each component can be in any of a finite number of states.
- Want to set the states of the components so as to optimize an certain objective under certain constraints.

Notation

$\mathrm{OPT}(I)$ denotes the optimal *value* of the objective for instance I.

Weighted interval scheduling

Input: meetings $i \in [n]$ specified by start time $s_i \in \mathbb{R}$, end time $e_i \in \mathbb{R}$, and importance $w_i \in \mathbb{R}$.

Discrete Multivariate Optimization

Setting

- System consisting of n components.
- Each component can be in any of a finite number of states.
- Want to set the states of the components so as to optimize an certain objective under certain constraints.

Notation

$\mathrm{OPT}(I)$ denotes the optimal *value* of the objective for instance I.

Weighted interval scheduling

Input: meetings $i \in [n]$ specified by start time $s_i \in \mathbb{R}$, end time $e_i \in \mathbb{R}$, and importance $w_i \in \mathbb{R}$.

Ouput: $S \subseteq [n]$ such that no distinct intervals $[s_i, e_i)$ for $i \in S$ overlap and $\sum_{i \in S} w_i$ is maximized.

Principle of Optimality

Principle of Optimality

Principle

Optimal solution to I with additional constraint that component i^* is set to s^* can be expressed in terms of optimal solutions to instances I' without additional constraints and fewer components.

Principle of Optimality

Principle

Optimal solution to I with additional constraint that component i^* is set to s^* can be expressed in terms of optimal solutions to instances I' without additional constraints and fewer components.

Recursion on input a nontrivial instance I

1. Pick some component i^*.
2. For all possible states s^* for i^*, find optimal solution to I with i^* set to s^*.
3. Return best one.

Principle of Optimality

Principle

Optimal solution to I with additional constraint that component i^* is set to s^* can be expressed in terms of optimal solutions to instances I' without additional constraints and fewer components.

Recursion on input a nontrivial instance I

1. Pick some component i^*.
2. For all possible states s^* for i^*, find optimal solution to I with i^* set to s^*.
3. Return best one.

Weighted interval scheduling

$\mathrm{OPT}(I) = \max\left(\mathrm{OPT}(I \text{ without } i^*), w_{i^*} + \mathrm{OPT}(I \text{ without } C(i^*)\right)$
where $C(i^*)$: meetings that overlap with i^*.

Bounding the Number of Distinct Subproblems

Bounding the Number of Distinct Subproblems

Approach

Bounding the Number of Distinct Subproblems

Approach

- State reduction: What (minimal) information about decisions made suffices to continue the process?

Bounding the Number of Distinct Subproblems

Approach

- State reduction: What (minimal) information about decisions made suffices to continue the process?
- Explicit description of subproblems: What (minimal) set of parameters suffice to describe all subproblems?

Bounding the Number of Distinct Subproblems

Approach

- State reduction: What (minimal) information about decisions made suffices to continue the process?
- Explicit description of subproblems: What (minimal) set of parameters suffice to describe all subproblems?

Weighted interval scheduling

Bounding the Number of Distinct Subproblems

Approach

- State reduction: What (minimal) information about decisions made suffices to continue the process?
- Explicit description of subproblems: What (minimal) set of parameters suffice to describe all subproblems?

Weighted interval scheduling

- Meetings in given order:

Bounding the Number of Distinct Subproblems

Approach

- State reduction: What (minimal) information about decisions made suffices to continue the process?
- Explicit description of subproblems: What (minimal) set of parameters suffice to describe all subproblems?

Weighted interval scheduling

- Meetings in given order: subsets of meetings considered / to consider

Bounding the Number of Distinct Subproblems

Approach

- State reduction: What (minimal) information about decisions made suffices to continue the process?
- Explicit description of subproblems: What (minimal) set of parameters suffice to describe all subproblems?

Weighted interval scheduling

- Meetings in given order: subsets of meetings considered / to consider – $2^{\Theta(n)}$ distinct subproblems

Bounding the Number of Distinct Subproblems

Approach

- State reduction: What (minimal) information about decisions made suffices to continue the process?
- Explicit description of subproblems: What (minimal) set of parameters suffice to describe all subproblems?

Weighted interval scheduling

- Meetings in given order: subsets of meetings considered / to consider – $2^{\Theta(n)}$ distinct subproblems
- Meetings in order of earliest deadline first:

Bounding the Number of Distinct Subproblems

Approach

- State reduction: What (minimal) information about decisions made suffices to continue the process?
- Explicit description of subproblems: What (minimal) set of parameters suffice to describe all subproblems?

Weighted interval scheduling

- Meetings in given order: subsets of meetings considered / to consider – $2^{\Theta(n)}$ distinct subproblems
- Meetings in order of earliest deadline first: prefixes / suffixes

Bounding the Number of Distinct Subproblems

Approach

- State reduction: What (minimal) information about decisions made suffices to continue the process?
- Explicit description of subproblems: What (minimal) set of parameters suffice to describe all subproblems?

Weighted interval scheduling

- Meetings in given order: subsets of meetings considered / to consider – $2^{\Theta(n)}$ distinct subproblems
- Meetings in order of earliest deadline first: prefixes / suffixes – $\Theta(n)$ distinct subproblems

Knapsack Problem

Knapsack Problem

Problem

Knapsack Problem

Problem

Input: items $i \in [n]$ specified by weight $w_i \in \mathbb{Z}^+$ and value $v_i \in \mathbb{R}$; weight limit $W \in \mathbb{Z}^+$

Knapsack Problem

Problem

Input: items $i \in [n]$ specified by weight $w_i \in \mathbb{Z}^+$ and value $v_i \in \mathbb{R}$; weight limit $W \in \mathbb{Z}^+$

Ouput: $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is maximized.

Knapsack Problem

Problem

Input: items $i \in [n]$ specified by weight $w_i \in \mathbb{Z}^+$ and value $v_i \in \mathbb{R}$; weight limit $W \in \mathbb{Z}^+$

Ouput: $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is maximized.

Principle of optimality

Knapsack Problem

Problem

Input: items $i \in [n]$ specified by weight $w_i \in \mathbb{Z}^+$ and value $v_i \in \mathbb{R}$; weight limit $W \in \mathbb{Z}^+$

Ouput: $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is maximized.

Principle of optimality

- Case $i^* \notin S$:

Knapsack Problem

Problem

Input: items $i \in [n]$ specified by weight $w_i \in \mathbb{Z}^+$ and value $v_i \in \mathbb{R}$; weight limit $W \in \mathbb{Z}^+$

Ouput: $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is maximized.

Principle of optimality

- Case $i^* \notin S$:
  Remains to solve given instance with i^* removed.

Knapsack Problem

Problem

Input: items $i \in [n]$ specified by weight $w_i \in \mathbb{Z}^+$ and value $v_i \in \mathbb{R}$; weight limit $W \in \mathbb{Z}^+$

Ouput: $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is maximized.

Principle of optimality

- Case $i^* \notin S$:
  Remains to solve given instance with i^* removed.
- Case $i^* \in S$

Knapsack Problem

Problem

Input: items $i \in [n]$ specified by weight $w_i \in \mathbb{Z}^+$ and value $v_i \in \mathbb{R}$; weight limit $W \in \mathbb{Z}^+$

Ouput: $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is maximized.

Principle of optimality

- Case $i^* \notin S$:
  Remains to solve given instance with i^* removed.
- Case $i^* \in S$ (only an option if $w_{i^*} \leq W$):

Knapsack Problem

Problem

Input: items $i \in [n]$ specified by weight $w_i \in \mathbb{Z}^+$ and value $v_i \in \mathbb{R}$; weight limit $W \in \mathbb{Z}^+$

Ouput: $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is maximized.

Principle of optimality

- Case $i^* \notin S$:
  Remains to solve given instance with i^* removed.
- Case $i^* \in S$ (only an option if $w_{i^*} \leq W$):
  Remains to solve given instance with i^* removed and weight limit $W - w_{i^*}$.

Knapsack Problem

Problem

Input: items $i \in [n]$ specified by weight $w_i \in \mathbb{Z}^+$ and value $v_i \in \mathbb{R}$; weight limit $W \in \mathbb{Z}^+$

Ouput: $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is maximized.

Principle of optimality

- Case $i^* \notin S$:
  Remains to solve given instance with i^* removed.
- Case $i^* \in S$ (only an option if $w_{i^*} \leq W$):
  Remains to solve given instance with i^* removed and weight limit $W - w_{i^*}$.

$$\begin{aligned} \mathrm{OPT}(I) = \max\big(&\mathrm{OPT}(I \text{ without } i^*), \\ &v_{i^*} + \mathrm{OPT}(I \text{ without } i^* \text{ and weight limit } W - w_{i^*})\big) \end{aligned}$$

Knapsack Problem

Problem

Input: items $i \in [n]$ specified by weight $w_i \in \mathbb{Z}^+$ and value $v_i \in \mathbb{R}$; weight limit $W \in \mathbb{Z}^+$

Ouput: $S \subseteq [n]$ such that $\sum_{i \in S} w_i \leq W$ and $\sum_{i \in S} v_i$ is maximized.

Principle of optimality

- Case $i^* \notin S$:
  Remains to solve given instance with i^* removed.
- Case $i^* \in S$ (only an option if $w_{i^*} \leq W$):
  Remains to solve given instance with i^* removed and weight limit $W - w_{i^*}$.

$$\begin{aligned} \mathrm{OPT}(I) = \max\big(&\mathrm{OPT}(I \text{ without } i^*), \\ &v_{i^*} + \mathrm{OPT}(I \text{ without } i^* \text{ and weight limit } W - w_{i^*})\big) \end{aligned}$$

Analysis

Analysis

- State reduction:

Analysis

- State reduction: $\Theta(n \cdot W)$ states
  last item considered, total weight thus far

Analysis

- State reduction: $\Theta(n \cdot W)$ states
  last item considered, total weight thus far
- Subproblem specification:

Analysis

- State reduction: $\Theta(n \cdot W)$ states
  last item considered, total weight thus far
- Subproblem specification:
  $\mathrm{OPT}(k, w) = \mathrm{OPT}(\text{items } \{k, \ldots, n\} \text{ and weight limit } w)$

Analysis

- State reduction: $\Theta(n \cdot W)$ states
  last item considered, total weight thus far
- Subproblem specification:
  $\mathrm{OPT}(k, w) = \mathrm{OPT}(\text{items } \{k, \ldots, n\} \text{ and weight limit } w)$
  where $1 \leq k \leq n + 1$ and $0 \leq w \leq W$
- Recurrence: $\mathrm{OPT}(k, w) =$

Analysis

- State reduction: $\Theta(n \cdot W)$ states
  last item considered, total weight thus far
- Subproblem specification:
  $\mathrm{OPT}(k, w) = \mathrm{OPT}(\text{items } \{k, \ldots, n\} \text{ and weight limit } w)$
  where $1 \leq k \leq n + 1$ and $0 \leq w \leq W$
- Recurrence: $\mathrm{OPT}(k, w) =$
  $\max\left(\mathrm{OPT}(k+1, w), v_k + \mathrm{OPT}(k+1, w - w_k) \text{ only if } w_k \leq w\right)$

Analysis

- State reduction: $\Theta(n \cdot W)$ states
  last item considered, total weight thus far
- Subproblem specification:
  $\text{OPT}(k, w) = \text{OPT}(\text{items } \{k, \ldots, n\} \text{ and weight limit } w)$
  where $1 \leq k \leq n+1$ and $0 \leq w \leq W$
- Recurrence: $\text{OPT}(k, w) =$
  $\max\left(\text{OPT}(k+1, w), v_k + \text{OPT}(k+1, w - w_k) \text{ only if } w_k \leq w\right)$
- Base cases

Analysis

- State reduction: $\Theta(n \cdot W)$ states
  last item considered, total weight thus far
- Subproblem specification:
  $\text{OPT}(k, w) = \text{OPT}(\text{items } \{k, \ldots, n\} \text{ and weight limit } w)$
  where $1 \leq k \leq n+1$ and $0 \leq w \leq W$
- Recurrence: $\text{OPT}(k, w) =$
  $\max\left(\text{OPT}(k+1, w), v_k + \text{OPT}(k+1, w - w_k) \text{ only if } w_k \leq w\right)$
- Base cases ($k = n+1$):

Analysis

- State reduction: $\Theta(n \cdot W)$ states
  last item considered, total weight thus far
- Subproblem specification:
  $\text{OPT}(k, w) = \text{OPT}(\text{items } \{k, \ldots, n\} \text{ and weight limit } w)$
  where $1 \leq k \leq n+1$ and $0 \leq w \leq W$
- Recurrence: $\text{OPT}(k, w) =$
  $\max\left(\text{OPT}(k+1, w), v_k + \text{OPT}(k+1, w - w_k) \text{ only if } w_k \leq w\right)$
- Base cases $(k = n+1)$: $\text{OPT}(n+1, w) = 0$

Analysis

- State reduction: $\Theta(n \cdot W)$ states
  last item considered, total weight thus far
- Subproblem specification:
  $\mathrm{OPT}(k, w) = \mathrm{OPT}(\text{items } \{k, \ldots, n\} \text{ and weight limit } w)$
  where $1 \leq k \leq n+1$ and $0 \leq w \leq W$
- Recurrence: $\mathrm{OPT}(k, w) =$
  $\max\left(\mathrm{OPT}(k+1, w), v_k + \mathrm{OPT}(k+1, w - w_k) \text{ only if } w_k \leq w\right)$
- Base cases $(k = n+1)$: $\mathrm{OPT}(n+1, w) = 0$
- Answer: $\mathrm{OPT}(1, W)$

Retrieving the Solution

Retrieving the Solution

Pseudocode

```
procedure Retrieve-Solution
    S ← ∅
    w ← W
    for k = 1 to n do
        if w_k ≤ w cand OPT(k, w) = v_k + OPT(k + 1, w − w_k)
            then S ← S ∪ {k}; w ← w − w_k
    return S
```

Retrieving the Solution

Pseudocode

procedure RETRIEVE-SOLUTION
    $S \leftarrow \emptyset$
    $w \leftarrow W$
    for $k = 1$ to n **do**
        if $w_k \leq w$ **cand** $\mathrm{OPT}(k, w) = v_k + \mathrm{OPT}(k+1, w - w_k)$
            then $S \leftarrow S \cup \{k\}$; $w \leftarrow w - w_k$
    return S

Analysis

Retrieving the Solution

Pseudocode

```
procedure RETRIEVE-SOLUTION
    S ← ∅
    w ← W
    for k = 1 to n do
        if w_k ≤ w cand OPT(k, w) = v_k + OPT(k + 1, w − w_k)
            then S ← S ∪ {k}; w ← w − w_k
    return S
```

Analysis

- Time: $O(n \cdot W)$ with or without retrieval.

Retrieving the Solution

Pseudocode

procedure RETRIEVE-SOLUTION
    $S \leftarrow \emptyset$
    $w \leftarrow W$
    for $k = 1$ to n **do**
        if $w_k \leq w$ **cand** $\mathrm{OPT}(k, w) = v_k + \mathrm{OPT}(k+1, w - w_k)$
            then $S \leftarrow S \cup \{k\}$; $w \leftarrow w - w_k$
    return S

Analysis

- Time: $O(n \cdot W)$ with or without retrieval.
- Space: $O(n \cdot W)$ with retrieval; $O(W)$ without.

A Little Bio

A Little Bio

DNA

A Little Bio

DNA

- String over $\{A, C, G, T\}$
- Complementary strands: $A \sim T$ and $C \sim G$

A Little Bio

DNA

- String over $\{A, C, G, T\}$
- Complementary strands: $A \sim T$ and $C \sim G$

RNA

A Little Bio

DNA

- String over $\{A, C, G, T\}$
- Complementary strands: $A \sim T$ and $C \sim G$

RNA

- String over $\{A, C, G, U\}$
- Single strand
- Self-stabilizes forming bonds $A \sim U$ and $C \sim G$

A Little Bio

DNA

- String over $\{A, C, G, T\}$
- Complementary strands: $A \sim T$ and $C \sim G$

RNA

- String over $\{A, C, G, U\}$
- Single strand
- Self-stabilizes forming bonds $A \sim U$ and $C \sim G$
- Example: Escherichia coli

RNA Secondary Structure

RNA Secondary Structure

Input:

string $R[1, \ldots, n]$ over alphabet $\{A, C, G, U\}$

RNA Secondary Structure

Input:

string $R[1, \ldots, n]$ over alphabet $\{A, C, G, U\}$

Output:

set S of pairs $(i, j) \in [n] \times [n]$ with $i < j$ of maximum size $|S|$ s.t.:

RNA Secondary Structure

Input:

string $R[1, \ldots, n]$ over alphabet $\{A, C, G, U\}$

Output:

set S of pairs $(i, j) \in [n] \times [n]$ with $i < j$ of maximum size $|S|$ s.t.:

- [Complementarity] For each $(i, j) \in S$, $R[i] \sim R[j]$.

RNA Secondary Structure

Input:

string $R[1,\ldots,n]$ over alphabet $\{A, C, G, U\}$

Output:

set S of pairs $(i,j) \in [n] \times [n]$ with $i < j$ of maximum size $|S|$ s.t.:

- [Complementarity] For each $(i,j) \in S$, $R[i] \sim R[j]$.
- [Matching] Each $i \in [n]$ appears in at most one pair of S.

RNA Secondary Structure

Input:

string $R[1, \ldots, n]$ over alphabet $\{A, C, G, U\}$

Output:

set S of pairs $(i, j) \in [n] \times [n]$ with $i < j$ of maximum size $|S|$ s.t.:

- [Complementarity] For each $(i, j) \in S$, $R[i] \sim R[j]$.
- [Matching] Each $i \in [n]$ appears in at most one pair of S.
- [No sharp turns] For each $(i, j) \in S$, $j \geq i + 5$.

RNA Secondary Structure

Input:

string $R[1, \ldots, n]$ over alphabet $\{A, C, G, U\}$

Output:

set S of pairs $(i, j) \in [n] \times [n]$ with $i < j$ of maximum size $|S|$ s.t.:

- [Complementarity] For each $(i, j) \in S$, $R[i] \sim R[j]$.
- [Matching] Each $i \in [n]$ appears in at most one pair of S.
- [No sharp turns] For each $(i, j) \in S$, $j \geq i + 5$.
- [No crossings] For no $(i, j), (k, \ell) \in S$, $i < k < j < \ell$.

Algorithm

Algorithm

Principle of optimality

Algorithm

Principle of optimality

- Case position 1 is not matched:

Algorithm

Principle of optimality

- Case position 1 is not matched:
  Remains to solve problem for $R[2, \ldots, n]$.

Algorithm

Principle of optimality

- Case position 1 is not matched:
  Remains to solve problem for $R[2, \ldots, n]$.
- Case position 1 is matched with k

Algorithm

Principle of optimality

- Case position 1 is not matched:
  Remains to solve problem for $R[2, \ldots, n]$.
- Case position 1 is matched with k
  (only an option if $k \geq 5$ and $R[1] \sim R[k]$):

Algorithm

Principle of optimality

- Case position 1 is not matched:
  Remains to solve problem for $R[2, \ldots, n]$.
- Case position 1 is matched with k
  (only an option if $k \geq 5$ and $R[1] \sim R[k]$):
  Remains to solve problem for $R[2, \ldots, k-1]$ and for $R[k+1, \ldots, n]$.

Algorithm

Principle of optimality

- Case position 1 is not matched:
  Remains to solve problem for $R[2, \ldots, n]$.
- Case position 1 is matched with k
  (only an option if $k \geq 5$ and $R[1] \sim R[k]$):
  Remains to solve problem for $R[2, \ldots, k-1]$ and for $R[k+1, \ldots, n]$.

Subproblem specification

Algorithm

Principle of optimality

- Case position 1 is not matched:
  Remains to solve problem for $R[2,\ldots,n]$.
- Case position 1 is matched with k
  (only an option if $k \geq 5$ and $R[1] \sim R[k]$):
  Remains to solve problem for $R[2,\ldots,k-1]$ and for $R[k+1,\ldots,n]$.

Subproblem specification

$\mathrm{OPT}(i,j) = \mathrm{OPT}(R[i,\ldots,j])$

Algorithm

Principle of optimality

- Case position 1 is not matched:
  Remains to solve problem for $R[2, \ldots, n]$.
- Case position 1 is matched with k
  (only an option if $k \geq 5$ and $R[1] \sim R[k]$):
  Remains to solve problem for $R[2, \ldots, k-1]$ and for $R[k+1, \ldots, n]$.

Subproblem specification

$\mathrm{OPT}(i, j) = \mathrm{OPT}(R[i, \ldots, j])$ where $1 \leq i \leq j \leq n$.

Algorithm

Principle of optimality

- Case position 1 is not matched:
  Remains to solve problem for $R[2, \ldots, n]$.
- Case position 1 is matched with k
  (only an option if $k \geq 5$ and $R[1] \sim R[k]$):
  Remains to solve problem for $R[2, \ldots, k-1]$ and for $R[k+1, \ldots, n]$.

Subproblem specification

$\mathrm{OPT}(i, j) = \mathrm{OPT}(R[i, \ldots, j])$ where $1 \leq i \leq j \leq n$.

Recurrence (for $i < j$)

$\mathrm{OPT}(i, j) = \max\left(\mathrm{OPT}(i+1, j),\right.$

Algorithm

Principle of optimality

- Case position 1 is not matched:
  Remains to solve problem for $R[2, \ldots, n]$.
- Case position 1 is matched with k
  (only an option if $k \geq 5$ and $R[1] \sim R[k]$):
  Remains to solve problem for $R[2, \ldots, k-1]$ and for $R[k+1, \ldots, n]$.

Subproblem specification

$\mathrm{OPT}(i, j) = \mathrm{OPT}(R[i, \ldots, j])$ where $1 \leq i \leq j \leq n$.

Recurrence (for $i < j$)

$$\mathrm{OPT}(i, j) = \max\big(\mathrm{OPT}(i+1, j), \\ \max_{i+5 \leq k \leq j, R[i] \sim R[k]} (1 + \mathrm{OPT}(i+1, k-1) + \mathrm{OPT}(k+1, j))\big)$$

Analysis

Analysis

Subproblem specification

$\mathrm{OPT}(i, j) = \mathrm{OPT}(R[i, \ldots, j])$ where $1 \leq i \leq j \leq n$.

Recurrence (for $i < j$)

$\mathrm{OPT}(i, j) = \max\big(\mathrm{OPT}(i + 1, j),$
$\max_{i+5 \leq k \leq j, R[i] \sim R[k]} (1 + \mathrm{OPT}(i + 1, k - 1) + \mathrm{OPT}(k + 1, j))\big)$

Analysis

Subproblem specification

$\mathrm{OPT}(i, j) = \mathrm{OPT}(R[i, \ldots, j])$ where $1 \le i \le j \le n$.

Recurrence (for $i < j$)

$\mathrm{OPT}(i, j) = \max\big(\mathrm{OPT}(i+1, j),$
$\max_{i+5 \le k \le j, R[i] \sim R[k]}(1 + \mathrm{OPT}(i+1, k-1) + \mathrm{OPT}(k+1, j))\big)$

Time

Analysis

Subproblem specification

$\mathrm{OPT}(i, j) = \mathrm{OPT}(R[i, \ldots, j])$ where $1 \leq i \leq j \leq n$.

Recurrence (for $i < j$)

$\mathrm{OPT}(i, j) = \max\big(\mathrm{OPT}(i+1, j),$
$\max_{i+5 \leq k \leq j, R[i] \sim R[k]}(1 + \mathrm{OPT}(i+1, k-1) + \mathrm{OPT}(k+1, j))\big)$

Time

- $\Theta(n^2)$ table entries
- $O(n)$ operations to evaluate recurrence for a given table entry
- $O(n^3)$ time overall

Analysis

Subproblem specification

$\mathrm{OPT}(i,j) = \mathrm{OPT}(R[i,\ldots,j])$ where $1 \le i \le j \le n$.

Recurrence (for $i < j$)

$\mathrm{OPT}(i,j) = \max\big(\mathrm{OPT}(i+1,j),$
$\max_{i+5 \le k \le j, R[i] \sim R[k]} (1 + \mathrm{OPT}(i+1,k-1) + \mathrm{OPT}(k+1,j))\big)$

Time

- $\Theta(n^2)$ table entries
- $O(n)$ operations to evaluate recurrence for a given table entry
- $O(n^3)$ time overall

Space

Analysis

Subproblem specification

$\mathrm{OPT}(i, j) = \mathrm{OPT}(R[i, \ldots, j])$ where $1 \leq i \leq j \leq n$.

Recurrence (for $i < j$)

$\mathrm{OPT}(i, j) = \max\left(\mathrm{OPT}(i+1, j),\right.$
$\left.\max_{i+5 \leq k \leq j, R[i] \sim R[k]} (1 + \mathrm{OPT}(i+1, k-1) + \mathrm{OPT}(k+1, j))\right)$

Time

- $\Theta(n^2)$ table entries
- $O(n)$ operations to evaluate recurrence for a given table entry
- $O(n^3)$ time overall

Space

$O(n^2)$ with or without retrieval.