CS 577- Intro to Algorithms

Dynamic Programming

Dieter van Melkebeek

September 22, 2020

Paradigm

Paradigm

Divide and Conquer

Recursive approach such that subproblems decrease significantly in size.

Paradigm

Divide and Conquer

Recursive approach such that subproblems decrease significantly in size.

Dynamic Programming

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.

Paradigm

Divide and Conquer

Recursive approach such that subproblems decrease significantly in size.

Dynamic Programming

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.

Realizing property 2

Paradigm

Divide and Conquer

Recursive approach such that subproblems decrease significantly in size.

Dynamic Programming

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.

Realizing property 2

- Memoization: remember all computed solutions

Paradigm

Divide and Conquer

Recursive approach such that subproblems decrease significantly in size.

Dynamic Programming

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.

Realizing property 2

- Memoization: remember all computed solutions
  top-down, generic

Paradigm

Divide and Conquer

Recursive approach such that subproblems decrease significantly in size.

Dynamic Programming

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.

Realizing property 2

- Memoization: remember all computed solutions
  top-down, generic
- Iteration: build up table of solved instances from easier to harder

Paradigm

Divide and Conquer

Recursive approach such that subproblems decrease significantly in size.

Dynamic Programming

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.

Realizing property 2

- Memoization: remember all computed solutions
  top-down, generic
- Iteration: build up table of solved instances from easier to harder
  bottom-up, ad hoc

Computing Fibonacci Numbers

Computing Fibonacci Numbers

Definition

- $F_0 = 0$
- $F_1 = 1$

Computing Fibonacci Numbers

Definition

- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$ for $n \in \mathbb{N}$ with $n \geq 2$

Computing Fibonacci Numbers

Definition

- $F_0 = 0$
- $F_1 = 1$
- $F_n = F_{n-1} + F_{n-2}$ for $n \in \mathbb{N}$ with $n \geq 2$

Recursive algorithm

Input: $n \in \mathbb{N}$
Output: F_n

```
procedure FIB-REC(n)
    if n ≤ 1 then
        return n
    else
        return FIB-REC(n − 1) + FIB-REC(n − 2)
```

Recursion Tree

DP Algorithm

DP Algorithm

Memoization

DP Algorithm

Memoization

- time: $O(n)$, space: $O(n)$

DP Algorithm

Memoization

- time: $O(n)$, space: $O(n)$

Iteration

DP Algorithm

Memoization

- time: $O(n)$, space: $O(n)$

Iteration

```
procedure FIB-IT(n)
    Fib[0 ··· n] ← a new array of size n + 1
    for i = 0 to n do
        if i ≤ 1 then
            Fib[i] ← i
        else
            Fib[i] ← Fib[i − 1] + Fib[i − 2]
    return Fib[n]
```

- time: $O(n)$, space: $O(n) \to O(1)$

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.

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.

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.

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

Recursive Algorithm

Recursive Algorithm

Subproblem specification

For $J \subseteq [n]$, let $\mathrm{OPT}(J)$ denote the maximum total importance achievable for the subproblem defined by the meetings in J

Recursive Algorithm

Subproblem specification

For $J \subseteq [n]$, let $\mathrm{OPT}(J)$ denote the maximum total importance achievable for the subproblem defined by the meetings in J

Recurrence

- For $J \neq \emptyset$, let j^* denote the first meeting in J and $C(j^*)$ the meetings that overlap with j^*.

Recursive Algorithm

Subproblem specification

For $J \subseteq [n]$, let $\mathrm{OPT}(J)$ denote the maximum total importance achievable for the subproblem defined by the meetings in J

Recurrence

- For $J \neq \emptyset$, let j^* denote the first meeting in J and $C(j^*)$ the meetings that overlap with j^*.
- $\mathrm{OPT}(J) =$

Bounding the Number of Distinct Subproblems

Bounding the Number of Distinct Subproblems

- State reduction: compression of decision history

Bounding the Number of Distinct Subproblems

- State reduction: compression of decision history
- Explicit description of subproblems: few parameters with small ranges

Improved Algorithm

Improved Algorithm

Idea

Sort the meetings earliest start time first, then run prior algorithm.

Improved Algorithm

Idea

Sort the meetings earliest start time first, then run prior algorithm.

Subproblem specification

For $k \in [n+1]$, let $\mathrm{OPT}(k)$ denote the maximum total importance achievable for the subproblem defined by the meetings in $\{k, k+1, \ldots, n\}$

Improved Algorithm

Idea

Sort the meetings earliest start time first, then run prior algorithm.

Subproblem specification

For $k \in [n+1]$, let $\mathrm{OPT}(k)$ denote the maximum total importance achievable for the subproblem defined by the meetings in $\{k, k+1, \ldots, n\}$

Recurrence

- For $k \in [n]$, let $\mathrm{next}(k)$ denote the smallest $i \geq k$ such that $s_i \geq e_k$ (where $s_{n+1} \doteq \infty$).

Improved Algorithm

Idea

Sort the meetings earliest start time first, then run prior algorithm.

Subproblem specification

For $k \in [n+1]$, let $\mathrm{OPT}(k)$ denote the maximum total importance achievable for the subproblem defined by the meetings in $\{k, k+1, \ldots, n\}$

Recurrence

- For $k \in [n]$, let $\mathrm{next}(k)$ denote the smallest $i \geq k$ such that $s_i \geq e_k$ (where $s_{n+1} \doteq \infty$).
- $\mathrm{OPT}(k) = \max\left(\mathrm{OPT}(k+1), w_k + \mathrm{OPT}(\mathrm{next}(k))\right)$

Analysis

Analysis

Running time

Analysis

Running time

- Sorting: $O(n \log n)$

Analysis

Running time

- Sorting: $O(n \log n)$
- Number of subproblems: $n + 1$

Analysis

Running time

- Sorting: $O(n \log n)$
- Number of subproblems: $n + 1$
- Amount of work per subproblem: $O(\log n)$ for finding next(k) using binary search.

Analysis

Running time

- Sorting: $O(n \log n)$
- Number of subproblems: $n + 1$
- Amount of work per subproblem: $O(\log n)$ for finding next(k) using binary search.
- Total: $O(n \log n)$

Analysis

Running time

- Sorting: $O(n \log n)$
- Number of subproblems: $n + 1$
- Amount of work per subproblem: $O(\log n)$ for finding next(k) using binary search.
- Total: $O(n \log n)$

Memory space

- $O(n)$

Retrieving the Solution

Retrieving the Solution

Recursively

Return both the value and a solution achieving it.

Retrieving the Solution

Recursively

Return both the value and a solution achieving it.

Iteratively

procedure RETRIEVE-SOLUTION
    $S \leftarrow \emptyset$
    $i \leftarrow 1$
    while $i \leq n$ **do**
        if $\mathrm{OPT}(i) = \mathrm{OPT}(i+1)$ **then**
            $i \leftarrow i + 1$
        else
            $S \leftarrow S \cup \{i\}$
            $i \leftarrow \mathrm{next}(i)$
    return S