Lecture 19: Elements of Dynamic Programming II

COMS10007 - Algorithms

Dr. Christian Konrad

29.04.2019

Admin

Schedule

- Today: Dynamic programming: Maximum subarray problem
- Tomorrow: 2D peak finding
- Tomorrow: Exercise sheet on general problem solving
- No lecture on Monday, May 6th - Early May bank holiday
- Tuesday, May 7th: Repetition, exercises, etc.

  Email me `christian.konrad@bristol.ac.uk` *today* if you want me to repeat a specific topic

Exam:

- Mock exam will be put online next week
- Bookwork and skills

Solving a Problem with Dynamic Programming:

1. Identify optimal substructure

   > Problem **P** exhibits *optimal substructure* if:
   >
   > An optimal solution to **P** contains within it optimal solutions to subproblems of **P**.

2. Give recursive solution
   (inspired by optimal substructure)
3. Compute optimal costs
   (fill table, bottom-up or top-down)
4. Construct optimal solution
   (keep track of decisions when filling table)

Fibonacci Numbers:

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

Require: Integer $n \geq 0$
  if $n \leq 1$ **then**
    return n
  else
    $A \leftarrow$ array of size n
    $A[0] \leftarrow 1$, $A[1] \leftarrow 1$
    for $i \leftarrow 2 \dots n$ **do**
      $A[i] \leftarrow A[i-2] + A[i-1]$
    return $A[n]$

DynPrgFib(n)

Why is this a dynamic programming algorithm?

Fibonacci Numbers - Dynamic Programming

Identify Optimal Substructure:

- Recall: $F_n = F_{n-1} + F_{n-2}$
- (Optimal) solution to size n problem equals sum of (optimal) solutions to subproblems of sizes $n-1$ and $n-2$ ✓

Give Recursive Solution:

- Recursive solution is already given in the problem description
- $F_n = F_{n-1} + F_{n-2}$

Compute Optimal Costs & Compute Optimal Solution

- Cost and solution is identical for Fibonacci numbers
- There is no need to keep track of optimal choices, since there is only a single "choice"

Maximum Subarray Problem

Problem: MAXIMUM-SUBARRAY

- **Input:** Array A of n numbers
- **Output:** Indices $0 \leq i \leq j \leq n-1$ such that $\sum_{l=i}^{j} A[l]$ is maximum.

Example:

$$-25 \quad 20 \quad -3 \quad -16 \quad -23 \quad 18 \quad 20 \quad -7 \quad 12 \quad -5 \quad 1$$

Divide-and-Conquer Algorithm

- In lecture 7 we gave a divide-and-conquer algorithm with runtime $O(n \log n)$
- We will give now a faster dynamic programming algorithm

Maximum Subarray Problem

Problem: MAXIMUM-SUBARRAY

- **Input:** Array A of n numbers
- **Output:** Indices $0 \le i \le j \le n-1$ such that $\sum_{l=i}^{j} A[l]$ is maximum.

Example:

$-25 \quad 20 \quad -3 \quad -16 \quad -23 \quad 18 \quad 20 \quad -7 \quad 12 \quad -5 \quad 1$

Divide-and-Conquer Algorithm

- In lecture 7 we gave a divide-and-conquer algorithm with runtime $O(n \log n)$
- We will give now a faster dynamic programming algorithm

Dynamic Programming for MAXIMUM-SUBARRAY

Related Problem: MAXIMUM-SUFFIX-ARRAY

- **Input:** Array A of n numbers
- **Output:** Index $0 \leq i \leq n-1$ such that $\sum_{l=i}^{n-1} A[l]$ is maximum.

$-25 \quad 20 \quad -3 \quad -16 \quad -23 \quad 18 \quad 20 \quad -7 \quad 12 \quad -5 \quad 1$

Optimal Substructure for MAXIMUM-SUBARRAY:

- Let i, j be the indices of the optimal solution
- Then i is the optimal solution for MAXIMUM-SUFFIX-ARRAY on input $A[0 \ldots j]$

Dynamic Programming for MAXIMUM-SUBARRAY

Related Problem: MAXIMUM-SUFFIX-ARRAY

- **Input:** Array A of n numbers
- **Output:** Index $0 \leq i \leq n-1$ such that $\sum_{l=i}^{n-1} A[l]$ is maximum.

$-25 \quad 20 \quad -3 \quad -16 \quad -23 \quad 18 \quad 20 \quad -7 \quad 12 \quad -5 \quad 1$

Optimal Substructure for MAXIMUM-SUBARRAY:

- Let i, j be the indices of the optimal solution
- Then i is the optimal solution for MAXIMUM-SUFFIX-ARRAY on input $A[0 \ldots j]$

Dynamic Programming for Maximum-Subarray

Related Problem: Maximum-Suffix-Array

- **Input:** Array A of n numbers
- **Output:** Index $0 \leq i \leq n-1$ such that $\sum_{l=i}^{n-1} A[l]$ is maximum.

$-25 \quad 20 \quad -3 \quad -16 \quad -23 \quad 18 \quad 20 \quad -7 \quad 12 \quad -5 \quad 1$

Optimal Substructure for Maximum-Subarray:

- Let i, j be the indices of the optimal solution
- Then i is the optimal solution for Maximum-Suffix-Array on input $A[0 \ldots j]$

$-25 \quad 20 \quad -3 \quad -16 \quad -23 \quad 18 \quad 20 \quad -7 \quad 12 \quad -5 \quad 1$

Dynamic Programming for MAXIMUM-SUBARRAY

Related Problem: MAXIMUM-SUFFIX-ARRAY

- **Input:** Array A of n numbers
- **Output:** Index $0 \leq i \leq n-1$ such that $\sum_{l=i}^{n-1} A[l]$ is maximum.

$-25 \quad 20 \quad -3 \quad -16 \quad -23 \quad 18 \quad 20 \quad -7 \quad 12 \quad -5 \quad 1$

Optimal Substructure for MAXIMUM-SUBARRAY:

- Let i, j be the indices of the optimal solution
- Then i is the optimal solution for MAXIMUM-SUFFIX-ARRAY on input $A[0 \dots j]$

$-25 \quad 20 \quad -3 \quad -16 \quad -23 \quad 18 \quad 20 \quad -7 \quad 12$

Dynamic Programming for Maximum Suffix Array

Optimal Substructure:

Lemma

Let A be an array of length n. Let i be the optimal solution for Maximum-Suffix-Array *on A. If $i < n-1$ then the optimal solution to* Maximum-Suffix-Array *on $A[0 \dots n-2]$ is also i.*

$$A[0] \quad A[1] \quad \dots A[i] \quad A[i+1] \quad \dots \quad A[n-2] \quad A[n-1]$$

Proof. Suppose that the lemma is not true and suppose that $i' \neq i$ is the optimal solution to Maximum-Suffix-Array on $A[0 \dots n-2]$. Then,

$$\sum_{j=i'}^{n-2} A[j] > \sum_{j=i}^{n-2} A[j]$$

But then $\sum_{j=i'}^{n-1} A[j] > \sum_{j=i}^{n-1} A[j]$, a contradiction to the fact that i is optimal for A. □

Recursive Solution:

$$m[i] := \text{value of maximum suffix array of } A[0 \ldots i]$$

$$m[i] = \begin{cases} A[0] & \text{if } i = 0 \\ A[i] & \text{if } m[i-1] \leq 0 \\ m[i-1] + A[i] & \text{if } m[i-1] > 0 \ . \end{cases}$$

Example: Bottom-up Computation

	0	1	2	3	4	5	6	7	8	9	10
A	−25	20	−3	−16	−23	18	20	−7	12	−5	1
m											

Recursive Solution:

$$m[i] := \text{value of maximum suffix array of } A[0 \dots i]$$

$$m[i] = \begin{cases} A[0] & \text{if } i = 0 \\ A[i] & \text{if } m[i-1] \leq 0 \\ m[i-1] + A[i] & \text{if } m[i-1] > 0 \ . \end{cases}$$

Example: Bottom-up Computation

	0	1	2	3	4	5	6	7	8	9	10
A	-25	20	-3	-16	-23	18	20	-7	12	-5	1
m	-25										

Recursive Solution to Maximum Suffix Array

Recursive Solution:

$$m[i] := \text{value of maximum suffix array of } A[0 \dots i]$$

$$m[i] = \begin{cases} A[0] & \text{if } i = 0 \\ A[i] & \text{if } m[i-1] \le 0 \\ m[i-1] + A[i] & \text{if } m[i-1] > 0 \ . \end{cases}$$

Example: Bottom-up Computation

	0	1	2	3	4	5	6	7	8	9	10
A	-25	20	-3	-16	-23	18	20	-7	12	-5	1
m	-25	20									

Recursive Solution to Maximum Suffix Array

Recursive Solution:

$$m[i] := \text{value of maximum suffix array of } A[0 \ldots i]$$

$$m[i] = \begin{cases} A[0] & \text{if } i = 0 \\ A[i] & \text{if } m[i-1] \leq 0 \\ m[i-1] + A[i] & \text{if } m[i-1] > 0 \ . \end{cases}$$

Example: Bottom-up Computation

	0	1	2	3	4	5	6	7	8	9	10
A	-25	20	-3	-16	-23	18	20	-7	12	-5	1
m	-25	20	17								

Recursive Solution to Maximum Suffix Array

Recursive Solution:

$$m[i] := \text{value of maximum suffix array of } A[0 \dots i]$$

$$m[i] = \begin{cases} A[0] & \text{if } i = 0 \\ A[i] & \text{if } m[i-1] \leq 0 \\ m[i-1] + A[i] & \text{if } m[i-1] > 0 \ . \end{cases}$$

Example: Bottom-up Computation

	0	1	2	3	4	5	6	7	8	9	10
A	-25	20	-3	-16	-23	18	20	-7	12	-5	1
m	-25	20	17	1							

Recursive Solution to Maximum Suffix Array

Recursive Solution:

$$m[i] := \text{value of maximum suffix array of } A[0\dots i]$$

$$m[i] = \begin{cases} A[0] & \text{if } i = 0 \\ A[i] & \text{if } m[i-1] \leq 0 \\ m[i-1] + A[i] & \text{if } m[i-1] > 0 \ . \end{cases}$$

Example: Bottom-up Computation

	0	1	2	3	4	5	6	7	8	9	10
A	-25	20	-3	-16	-23	18	20	-7	12	-5	1
m	-25	20	17	1	-22						

Recursive Solution to Maximum Suffix Array

Recursive Solution:

$$m[i] := \text{value of maximum suffix array of } A[0 \ldots i]$$

$$m[i] = \begin{cases} A[0] & \text{if } i = 0 \\ A[i] & \text{if } m[i-1] \leq 0 \\ m[i-1] + A[i] & \text{if } m[i-1] > 0 \ . \end{cases}$$

Example: Bottom-up Computation

	0	1	2	3	4	5	6	7	8	9	10
A	-25	20	-3	-16	-23	18	20	-7	12	-5	1
m	-25	20	17	1	-22	18					

Recursive Solution to Maximum Suffix Array

Recursive Solution:

$$m[i] := \text{value of maximum suffix array of } A[0 \dots i]$$

$$m[i] = \begin{cases} A[0] & \text{if } i = 0 \\ A[i] & \text{if } m[i-1] \leq 0 \\ m[i-1] + A[i] & \text{if } m[i-1] > 0 \ . \end{cases}$$

Example: Bottom-up Computation

	0	1	2	3	4	5	6	7	8	9	10
A	−25	20	−3	−16	−23	18	20	−7	12	−5	1
m	−25	20	17	1	−22	18	38				

Recursive Solution to Maximum Suffix Array

Recursive Solution:

$$m[i] := \text{value of maximum suffix array of } A[0 \dots i]$$

$$m[i] = \begin{cases} A[0] & \text{if } i = 0 \\ A[i] & \text{if } m[i-1] \leq 0 \\ m[i-1] + A[i] & \text{if } m[i-1] > 0 \ . \end{cases}$$

Example: Bottom-up Computation

	0	1	2	3	4	5	6	7	8	9	10
A	−25	20	−3	−16	−23	18	20	−7	12	−5	1
m	−25	20	17	1	−22	18	38	31			

Recursive Solution to Maximum Suffix Array

Recursive Solution:

$$m[i] := \text{value of maximum suffix array of } A[0 \dots i]$$

$$m[i] = \begin{cases} A[0] & \text{if } i = 0 \\ A[i] & \text{if } m[i-1] \leq 0 \\ m[i-1] + A[i] & \text{if } m[i-1] > 0 \ . \end{cases}$$

Example: Bottom-up Computation

	0	1	2	3	4	5	6	7	8	9	10
A	-25	20	-3	-16	-23	18	20	-7	12	-5	1
m	-25	20	17	1	-22	18	38	31	43		

Recursive Solution to Maximum Suffix Array

Recursive Solution:

$$m[i] := \text{value of maximum suffix array of } A[0 \dots i]$$

$$m[i] = \begin{cases} A[0] & \text{if } i = 0 \\ A[i] & \text{if } m[i-1] \leq 0 \\ m[i-1] + A[i] & \text{if } m[i-1] > 0 \ . \end{cases}$$

Example: Bottom-up Computation

	0	1	2	3	4	5	6	7	8	9	10
A	-25	20	-3	-16	-23	18	20	-7	12	-5	1
m	-25	20	17	1	-22	18	38	31	43	38	

Recursive Solution to Maximum Suffix Array

Recursive Solution:

$$m[i] := \text{value of maximum suffix array of } A[0 \ldots i]$$

$$m[i] = \begin{cases} A[0] & \text{if } i = 0 \\ A[i] & \text{if } m[i-1] \leq 0 \\ m[i-1] + A[i] & \text{if } m[i-1] > 0 \ . \end{cases}$$

Example: Bottom-up Computation

	0	1	2	3	4	5	6	7	8	9	10
A	−25	20	−3	−16	−23	18	20	−7	12	−5	1
m	−25	20	17	1	−22	18	38	31	43	38	39

Recursive Solution to Maximum Suffix Array

Recursive Solution:

$$m[i] := \text{value of maximum suffix array of } A[0 \dots i]$$

$$m[i] = \begin{cases} A[0] & \text{if } i = 0 \\ A[i] & \text{if } m[i-1] \leq 0 \\ m[i-1] + A[i] & \text{if } m[i-1] > 0 \ . \end{cases}$$

Example: Bottom-up Computation

	0	1	2	3	4	5	6	7	8	9	10
A	−25	20	−3	−16	−23	18	20	−7	12	−5	1
m	−25	20	17	1	−22	18	38	31	**43**	38	39

Maximum constitutes optimal solution to MAXIMUM-SUBARRAY!

Dynamic Programming Algorithm for Maximum Subarray

Algorithm: Input is an array A of integers of length n

1. Compute dyn. prog. table for MAXIMUM-SUFFIX-ARRAY
2. Return the maximum value in the table

```
Require: Array A of n integers
  Let m[0 ... n − 1] be a new array
  m[0] ← A[0]
  q ← A[0]
  for i = 1 ... n − 1 do
    if m[i − 1] < 0 then
      m[i] ← A[i]
    else
      m[i] ← A[i] + m[i − 1]
    q ← max{q, m[i]}
  return q
```

Kadane's Algorithm for MAXIMUM-SUBARRAY

Summary

Kadane's Algorithm

- Runtime: $O(n)$ (n subproblems, only one subproblem needed to compute current value)
- Recall that Divide-and-Conquer solution has a runtime of $O(n \log n)$
- Observe that for MAXIMUM-SUBARRAY Dynamic Programming and Divide-and-Conquer is applicable

Challenges:

- Compute max. subarray of size at most k, for some k
- Compute subarray $A[i, j]$ such that

$$\frac{\sum_{k=i}^{j} A[k]}{\sqrt{j - i + 1}}$$

is maximized.