Algorithms for Programming Contests - Week 8

Pranav Ashok, Michael Blondin, Philipp Meyer,
Christian Müller, Gregor Schwarz
conpra@in.tum.de

12.12.2017

Fibonacci Numbers

Definition (Fibonacci Numbers)

$fib(0) = 1$
$fib(1) = 1$
$fib(n) = fib(n-1) + fib(n-2)$

procedure $\textsc{Fib}(n)$
  if $n \leq 2$ **then return** 1
  else
    return $\textsc{Fib}(n-1) + \textsc{Fib}(n-2)$

Fibonacci Numbers

Definition (Fibonacci Numbers)

$fib(0) = 1$
$fib(1) = 1$
$fib(n) = fib(n-1) + fib(n-2)$

procedure $\textsc{Fib}(n)$
    if $n \leq 2$ **then return** 1
    if $DP[n] \neq 0$ **then**
        $DP[n] \leftarrow \textsc{Fib}(n-1) + \textsc{Fib}(n-2)$
    return $DP[n]$

Change Making

What is the minimum number of coins to make 40 cents?

1$

25¢

20¢

1¢

What are the subproblems?

Change Making

What is the minimum number of coins to make 40 cents?

1\$ 25¢ 20¢ 1¢

What are the subproblems?
Add 1 coin to the solutions for 40 − 25¢, 40 − 20¢, 40 − 1¢

Change Making

What is the minimum number of coins to make 40 cents?




1\$ 25¢ 20¢ 1¢

What are the subproblems?
Add 1 coin to the solutions for 40 − 25¢, 40 − 20¢, 40 − 1¢

Let dp[i] = "What is the least amount of coins I need to make *i* ¢?"

Change Making

What is the minimum number of coins to make 40 cents?

1\$ 25¢ 20¢ 1¢

What are the subproblems?
Add 1 coin to the solutions for $40 - 25$¢, $40 - 20$¢, $40 - 1$¢

Let dp[i] = "What is the least amount of coins I need to make i ¢?"

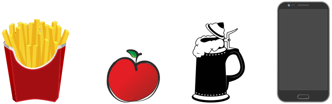
$$dp[i] = \min(dp[i-25], dp[i-20], dp[i-1]) + 1$$

General Approach

1. Find recursive subproblems (smaller numbers, fewer nodes, ...)
2. Solve subproblem, cache solution
3. Assemble bigger solution

0/1 Knapsack

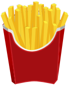
Maximum capacity $W = 10$



Values v_i:	10	40	30	50
Weights w_i:	5	4	6	3

0/1 Knapsack

Maximum capacity $W = 10$

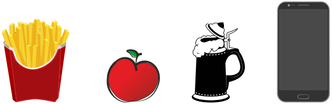
   

Values v_i:	10	40	30	50
Weights w_i:	5	4	6	3

Possible approach: Build 2-dimensional table $dp[n][W]$ where $dp[i][w]$ considers a backpack of size $w < W$ and items $1, \ldots, i$ only.

0/1 Knapsack

Maximum capacity $W = 10$

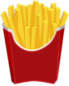


Values v_i:	10	40	30	50
Weights w_i:	5	4	6	3

Possible approach: Build 2-dimensional table $dp[n][W]$ where $dp[i][w]$ considers a backpack of size $w < W$ and items $1, \ldots, i$ only.

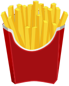
$$dp[0, w] \leftarrow 0 \text{ for all } w \leq W$$
$$dp[i, w] \leftarrow \mathtt{max}(dp[i-1, w], dp[i-1, w-w_i] + v_i)$$

0/1 Knapsack

					
v_i:		10	40	30	50
w_i:		5	4	6	3
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0

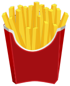
$$\texttt{max}\left(\begin{array}{l} dp[i-1, w], \\ dp[i-1, w-w_i]+v_i \end{array}\right)$$

0/1 Knapsack

					
v_i:		10	40	30	50
w_i:		5	4	6	3
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	50

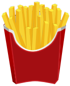
$$\texttt{max}\begin{pmatrix} dp[i-1, w], \\ dp[i-1, w-w_i] + v_i \end{pmatrix}$$

0/1 Knapsack

					
v_i:		10	40	30	50
w_i:		5	4	6	3
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	50
4	0	0	40	40	50

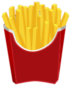
$$\texttt{max}\left(\begin{array}{l} dp[i-1, w], \\ dp[i-1, w-w_i]+v_i \end{array}\right)$$

0/1 Knapsack

					
v_i:		10	40	30	50
w_i:		5	4	6	3
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	50
4	0	0	40	40	50
5	0	10	40	40	50

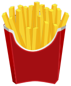
$$\mathtt{max}\left(\begin{array}{l} dp[i-1, w], \\ dp[i-1, w-w_i] + v_i \end{array}\right)$$

0/1 Knapsack

					
v_i:		10	40	30	50
w_i:		5	4	6	3
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	50
4	0	0	40	40	50
5	0	10	40	40	50
6	0	10	40	40	50
7	0	10	40	40	90
8	0	10	50	50	90
9	0	10	50	50	90
10	0	10	50	70	**90**

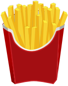
$$\mathtt{max}\begin{pmatrix} dp[i-1, w], \\ dp[i-1, w-w_i]+v_i \end{pmatrix}$$

0/1 Knapsack

					
v_i:		10	40	30	50
w_i:		5	4	6	3
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	50
4	0	0	40	40	50
5	0	10	40	40	50
6	0	10	40	40	50
7	0	10	40	40	90
8	0	10	50	50	90
9	0	10	50	50	90
10	0	10	50	70	**90**

$$\texttt{max}\begin{pmatrix} dp[i-1, w], \\ dp[i-1, w-w_i] + v_i \end{pmatrix}$$

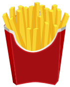
0/1 Knapsack

					
v_i:		10	40	30	50
w_i:		5	4	6	3
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	50
4	0	0	40	40	50
5	0	10	40	40	50
6	0	10	40	40	50
7	0	10	40	40	90
8	0	10	50	50	90
9	0	10	50	50	90
10	0	10	50	70	**90**

$$\texttt{max}\begin{pmatrix} dp[i-1, w], \\ dp[i-1, w-w_i] + v_i \end{pmatrix}$$

How to compute the solution?

0/1 Knapsack

					
v_i:		10	40	30	50
w_i:		5	4	6	3
0	0	0	0	0	0
1	0	0	0	0	0
2	0	0	0	0	0
3	0	0	0	0	50
4	0	0	40	40	50
5	0	10	40	40	50
6	0	10	40	40	50
7	0	10	40	40	90
8	0	10	50	50	90
9	0	10	50	50	90
10	0	10	50	70	**90**

$$\texttt{max}\begin{pmatrix} dp[i-1, w], \\ dp[i-1, w-w_i] + v_i \end{pmatrix}$$

How to compute the solution?
`predecessor` array storing incoming edge

Top-Down vs Bottom-Up

Top-Down - Memoization

- Recursive computation
- Save results as they appear (HashMap)

  Straight-forward to implement
  Only computing relevant subproblems
  Good for sparse statespace

Bottom-Up

- Fill table for all smaller subproblems first
- Save results in array

  Better cache locality
  Good for dense statespace

More Examples

- Floyd-Warshall: All Pairs Shortest Paths
- Dijkstra: Single Source Shortest Path
- Longest Common Subsequence of two strings
- Edit Distance of two strings

Longest Increasing Subsequence

Problem

Given a sequence of numbers

$$0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15$$

What is the length of the longest *increasing subsequence*?

Longest Increasing Subsequence

Problem

Given a sequence of numbers

$$0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15$$

What is the length of the longest *increasing subsequence*?

Answer

6. But how? What are the subproblems?

Longest Increasing Subsequence

Problem

Given a sequence of numbers

$$0, 8, 4, 12, 2, 10, 6, 14, 1, 9, 5, 13, 3, 11, 7, 15$$

What is the length of the longest *increasing subsequence*?

Answer

6. Compute the solution for shorter sequences and build up.

Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence with length s[i] that ends with the value v[i]."

v[i]	0,	8,	4,	12,	2,	10,	6,	14,	1,	9
s[i]	1									

Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence with length s[i] that ends with the value v[i]."

v[i]	0,	8,	4,	12,	2,	10,	6,	14,	1,	9
s[i]	1	2								

Observation

If there is a longest increasing subsequence at a smaller index i and $v[i] < v[j]$, then there is a sequence of length $s[i] + 1$ at index j.
$\Rightarrow s[j]$ is one longer than the maximum sequence ending at values smaller than $v[j]$.

Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence with length s[i] that ends with the value v[i]."

v[i]	0,	8,	4,	12,	2,	10,	6,	14,	1,	9
s[i]	1	2	2							

Observation

If there is a longest increasing subsequence at a smaller index i and $v[i] < v[j]$, then there is a sequence of length $s[i] + 1$ at index j.
$\Rightarrow s[j]$ is one longer than the maximum sequence ending at values smaller than $v[j]$.

Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence with length s[i] that ends with the value v[i]."

v[i]	0,	8,	4,	12,	2,	10,	6,	14,	1,	9
s[i]	1	2	2	3						

Observation

If there is a longest increasing subsequence at a smaller index i and $v[i] < v[j]$, then there is a sequence of length $s[i] + 1$ at index j.
$\Rightarrow s[j]$ is one longer than the maximum sequence ending at values smaller than $v[j]$.

Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence with length s[i] that ends with the value v[i]."

v[i]	0,	8,	4,	12,	2,	10,	6,	14,	1,	9
s[i]	1	2	2	3	2					

Observation

If there is a longest increasing subsequence at a smaller index i and $v[i] < v[j]$, then there is a sequence of length $s[i] + 1$ at index j.
$\Rightarrow s[j]$ is one longer than the maximum sequence ending at values smaller than $v[j]$.

Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence with length s[i] that ends with the value v[i]."

v[i]	0,	8,	4,	12,	2,	10,	6,	14,	1,	9
s[i]	1	2	2	3	2	3				

Observation

If there is a longest increasing subsequence at a smaller index i and $v[i] < v[j]$, then there is a sequence of length $s[i] + 1$ at index j.
$\Rightarrow s[j]$ is one longer than the maximum sequence ending at values smaller than $v[j]$.

Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence with length s[i] that ends with the value v[i]."

v[i]	0,	8,	4,	12,	2,	10,	6,	14,	1,	9
s[i]	1	2	2	3	2	3	3			

Observation

If there is a longest increasing subsequence at a smaller index i and $v[i] < v[j]$, then there is a sequence of length $s[i] + 1$ at index j.
$\Rightarrow s[j]$ is one longer than the maximum sequence ending at values smaller than $v[j]$.

Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence with length s[i] that ends with the value v[i]."

v[i]	0,	8,	4,	12,	2,	10,	6,	14,	1,	9
s[i]	1	2	2	3	2	3	3	4		

Observation

If there is a longest increasing subsequence at a smaller index i and $v[i] < v[j]$, then there is a sequence of length $s[i] + 1$ at index j.
$\Rightarrow s[j]$ is one longer than the maximum sequence ending at values smaller than $v[j]$.

Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence with length s[i] that ends with the value v[i]."

v[i]	0,	8,	4,	12,	2,	10,	6,	14,	1,	9
s[i]	1	2	2	3	2	3	3	4	2	

Observation

If there is a longest increasing subsequence at a smaller index i and $v[i] < v[j]$, then there is a sequence of length $s[i] + 1$ at index j.
$\Rightarrow s[j]$ is one longer than the maximum sequence ending at values smaller than $v[j]$.

Longest Increasing Subsequence

s[i] := “What is the length of the longest increasing subsequence with length s[i] that ends with the value v[i].”

v[i]	0,	8,	4,	12,	2,	10,	6,	14,	1,	9
s[i]	1	2	2	3	2	3	3	4	2	3

Observation

If there is a longest increasing subsequence at a smaller index i and $v[i] < v[j]$, then there is a sequence of length $s[i] + 1$ at index j.
$\Rightarrow s[j]$ is one longer than the maximum sequence ending at values smaller than $v[j]$.

Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence with length s[i] that ends with the value v[i]."

v[i]	0,	8,	4,	12,	2,	10,	6,	14,	1,	9
s[i]	1	2	2	3	2	3	3	4	2	3

Observation

If there is a longest increasing subsequence at a smaller index i and $v[i] < v[j]$, then there is a sequence of length $s[i] + 1$ at index j.
$\Rightarrow s[j]$ is one longer than the maximum sequence ending at values smaller than $v[j]$.

Easy algorithm

Compute $s[i]$ by looking at all smaller $s[i]$.

Longest Increasing Subsequence

s[i] := "What is the length of the longest increasing subsequence with length s[i] that ends with the value v[i]."

v[i]	0,	8,	4,	12,	2,	10,	6,	14,	1,	9
s[i]	1	2	2	3	2	3	3	4	2	3

Observation

If there is a longest increasing subsequence at a smaller index i and $v[i] < v[j]$, then there is a sequence of length $s[i] + 1$ at index j.
$\Rightarrow s[j]$ is one longer than the maximum sequence ending at values smaller than $v[j]$.

Easy algorithm

Compute $s[i]$ by looking at all smaller $s[i]$. $\mathcal{O}(n^2)$

Pseudocode

s[i] := “What is the length of the longest increasing subsequence with length s[i] that ends with the value v[i].”

Pseudocode

Keep track of representative sequences by ascending length.
m[i] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length i ending at $v[i]$."

Pseudocode

m[i] := “What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length i ending at $v[i]$.”

i	0	1	2	3	4	5	6	7	8	9
v[i]	0	8	4	12	2	10	6	14	1	9
m[i]										

Pseudocode

m[i] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length i ending at $v[i]$."

i	0	1	2	3	4	5	6	7	8	9
v[i]	0	8	4	12	2	10	6	14	1	9
m[i]	0									

```
parent[0] = -1
maxlength = 0

for i in [0..n-1]:
  # Binary Search for largest j, $s.t.
  # v[m[j]] < v[i] and j < i
  j = search()
  parent[i] = m[j]
  m[j + 1] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

Pseudocode

m[i] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length i ending at $v[i]$."

i	0	1	2	3	4	5	6	7	8	9
v[i]	0	8	4	12	2	10	6	14	1	9
m[i]	0	0								

```
parent[0] = -1
maxlength = 0

for i in [0..n-1]:
  # Binary Search for largest j, $s.t.
  # v[m[j]] < v[i] and j < i
  j = search()
  parent[i] = m[j]
  m[j + 1] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

Pseudocode

m[i] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length i ending at $v[i]$."

i	0	1	2	3	4	5	6	7	8	9
v[i]	0	8	4	12	2	10	6	14	1	9
m[i]	0	0	1							

```
parent[0] = -1
maxlength = 0

for i in [0..n-1]:
  # Binary Search for largest j, $s.t.
  # v[m[j]] < v[i] and j < i
  j = search()
  parent[i] = m[j]
  m[j + 1] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

Pseudocode

m[i] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length i ending at $v[i]$."

i	0	1	2	3	4	5	6	7	8	9
v[i]	0	8	4	12	2	10	6	14	1	9
m[i]	0	0	2							

```
parent[0] = -1
maxlength = 0

for i in [0..n-1]:
  # Binary Search for largest j, $s.t.
  # v[m[j]] < v[i] and j < i
  j = search()
  parent[i] = m[j]
  m[j + 1] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

Pseudocode

m[i] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length i ending at $v[i]$."

i	0	1	2	3	4	5	6	7	8	9
v[i]	0	8	4	12	2	10	6	14	1	9
m[i]	0	0	2	3						

```
parent[0] = -1
maxlength = 0

for i in [0..n-1]:
  # Binary Search for largest j, $s.t.
  # v[m[j]] < v[i] and j < i
  j = search()
  parent[i] = m[j]
  m[j + 1] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

Pseudocode

m[i] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length i ending at $v[i]$."

i	0	1	2	3	4	5	6	7	8	9
v[i]	0	8	4	12	2	10	6	14	1	9
m[i]	0	0	4	3						

```
parent[0] = -1
maxlength = 0

for i in [0..n-1]:
  # Binary Search for largest j, $s.t.
  # v[m[j]] < v[i] and j < i
  j = search()
  parent[i] = m[j]
  m[j + 1] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

Pseudocode

m[i] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length i ending at $v[i]$."

i	0	1	2	3	4	5	6	7	8	9
v[i]	0	8	4	12	2	10	6	14	1	9
m[i]	0	0	4	5						

```
parent[0] = -1
maxlength = 0

for i in [0..n-1]:
  # Binary Search for largest j, $s.t.
  # v[m[j]] < v[i] and j < i
  j = search()
  parent[i] = m[j]
  m[j + 1] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

Pseudocode

m[i] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length i ending at $v[i]$."

i	0	1	2	3	4	5	6	7	8	9
v[i]	0	8	4	12	2	10	6	14	1	9
m[i]	0	0	4	6						

```
parent[0] = -1
maxlength = 0

for i in [0..n-1]:
  # Binary Search for largest j, $s.t.
  # v[m[j]] < v[i] and j < i
  j = search()
  parent[i] = m[j]
  m[j + 1] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

Pseudocode

m[i] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length i ending at $v[i]$."

i	0	1	2	3	4	5	6	7	8	9
v[i]	0	8	4	12	2	10	6	14	1	9
m[i]	0	0	4	6	7					

```
parent[0] = -1
maxlength = 0

for i in [0..n-1]:
  # Binary Search for largest j, $s.t.
  # v[m[j]] < v[i] and j < i
  j = search()
  parent[i] = m[j]
  m[j + 1] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

Pseudocode

m[i] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length i ending at $v[i]$."

i	0	1	2	3	4	5	6	7	8	9
v[i]	0	8	4	12	2	10	6	14	1	9
m[i]	0	8	4	6	7					

```
parent[0] = -1
maxlength = 0

for i in [0..n-1]:
  # Binary Search for largest j, $s.t.
  # v[m[j]] < v[i] and j < i
  j = search()
  parent[i] = m[j]
  m[j + 1] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```

Pseudocode

m[i] := "What is the index j with a minimum value $v[j]$, s.t. there is a LIS of length i ending at $v[i]$."

i	0	1	2	3	4	5	6	7	8	9
v[i]	0	8	4	12	2	10	6	14	1	9
m[i]	0	8	4	9	7					

```
parent[0] = -1
maxlength = 0

for i in [0..n-1]:
  # Binary Search for largest j, $s.t.
  # v[m[j]] < v[i] and j < i
  j = search()
  parent[i] = m[j]
  m[j + 1] = i
  if j + 1 > maxlength:
    maxlength = j + 1
return maxlength
```