

Memoizing

"memoized" f

Dynamic Programming in a recursive (top-down) framework

- Whenever you compute the value of a subproblem, store it for later lookup

f becomes mf plus MEMO table

```
mf(x)
  if MEMO[x] defined then {}
  else MEMO[x] ← f(x)
  return MEMO[x]
```

Replace all calls to f by calls to mf

- includes ones in the body of f
- but not the one in mf

Problem 16-2

TEX

Pretty Paragraphs

words w_1, w_2, \dots, w_n *in Courier*
 lengths l_1, l_2, \dots, l_n
 separated by spaces on lines of length M

If w_i, \dots, w_j on a line, then extra spaces

$$\text{pad}(i, j) = M - (j - i) - \sum_{k=i}^j l_k$$

Minimize $(\text{pad}(i, j))^3$ over all lines but the last

$$\text{pad}(i, n) =$$

CS 410/584, Algorithm Design & Analysis: Lecture 2

Algorithm Design & Analysis

$M = 14$

Example

this is an
example group
of words

pad(1, 3) = 4 $(4)^3 = 64$
 pad(4, 5) = 1 $(1)^3 = 1$
 padding penalty $\frac{64}{1} = 65$

$M = 14$

Lecture 2

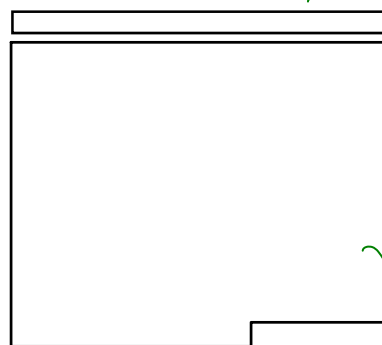
David Maier

3

Algorithm Design & Analysis

Recursive Solution

Decompose



first line

rest of words

BEST

Lecture 2

David Maier

4

Calculating Cost

Let $BEST[i, n]$ be the cost of the best layout of words $w_i \dots w_n$ (on multiple lines)

$BEST[i, n]$

$$\min_{i \leq k \leq p} (\text{pad}(i, k)) + BEST[k+1, n]$$

How big is largest p we must consider?

$$\left\lceil \frac{M}{2} \right\rceil$$

Strategy

Compute

$BEST[n, n]$

$BEST[n-1, n]$

$BEST[n-2, n]$

...

$BEST[1, n]$

helps if $BEST[n+1, n] = 0$

How long does each one take?

- k ranges over $O(n)$
- $BEST[k+1, n]$ look up
- $\text{pad}(i, k)$ might seem like $O(n)$

Computing $\text{pad}(i,j)$

But notice

each takes constant time

$$\begin{aligned} \text{pad}(i,i) &= M - l_i \\ \text{pad}(i,i+1) &= \text{pad}(i,i) - l_{i+1} - 1 \\ \text{pad}(i,i+2) &= \text{pad}(i,i+1) - l_{i+2} - 1 \\ &\dots \\ \text{pad}(i,p) &= \text{pad}(i,p-1) - l_p - 1 \end{aligned}$$

so time complexity is $O(n \cdot M)$

Greedy Algorithm

A series of locally optimum choices produces a globally optimum solution

What is needed:

- Greedy choice property — of all the optimal solutions, at least one *contains the initial greedy choice*
- Optimal substructure — as with *dynamic programming*

Greedy Example

Activity Selection

Have a set of possible activities to do

What is *most*, you can attend *without*

Greedy choice:

Have start and finish times

Order by *end times*

pick remaining item w/ earliest end time

LIST	A	B	C	D	E	F	G
	✓	✗	✗	✓	✗	✓	✗
	2-5	1-6	3-6	6-8	5-9	10-11	10-12

Sched = { *A, D, F* }

Schedule Algorithm

Sched $\leftarrow \emptyset$

(in order of end time)

for each ACT in LIST **do**

if ACT compatible with Sched

then Sched \leftarrow Sched \cup {ACT}

$O(\text{sort} + n)$

Complexity: \uparrow

Correct?

Let LIST = A_1, A_2, \dots, A_n

Greedy Choice

- Suppose $Sched$ is a maximal legal schedule that
Does not include A_1 (greedy choice)
- Let A_i be the activity in $Sched$ with
earliest end time
- Then $A_1.finish \leq A_i.finish$
- So $sched' = sched - \{A_i\} \cup \{A_1\}$ is also a
maximum legal schedule.



Optimal Substructure

- Assume $Sched$ is maximal legal
schedule for A_1, A_2, \dots, A_n
that includes A_1
- Let A_i be the first activity with
with $A_i.start \geq A_1.finish$
- Then $Sched - \{A_1\}$ is a maximal legal
schedule for A_i, \dots, A_n

CS 410/584, Algorithm Design & Analysis: Lecture 2

Algorithm Design & Analysis

Huffman Codes

◇	A	E	H	N	R	S	T	
000	001	010	011	100	101	110	111	$\begin{matrix} 28 \\ +3 \\ \hline 84 \text{ bits} \end{matrix}$

THE◇RAT◇SAT◇AT◇THE◇HEN◇REST◇ blank

Frequencies

7	3	4	3	1	2	2	6
◇	A	E	H	N	R	S	T

00	100	01	1010	10110	10110	10111	11	
----	-----	----	------	-------	-------	-------	----	--

→ 14 9 8 12 6 10 12 12 = 83 bits
 Contribution (in bits)

Lecture 2

David Maier

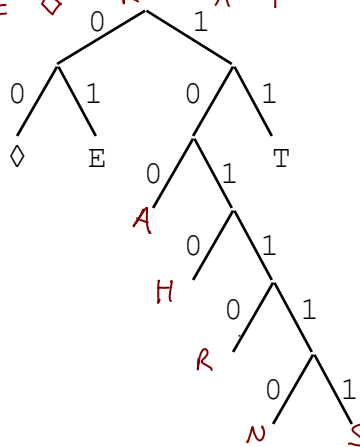
13

Algorithm Design & Analysis

Prefix Code

No code word is prefix of another

111010010010111010011 ...
 T H E ◇ R A T



Lecture 2

David Maier

14

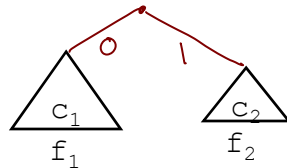
CS 410/584, Algorithm Design & Analysis: Lecture 2

Algorithm Design & Analysis

Greedy Algorithm for Code Generation

Given Frequencies of Characters

Start with n separate 1-character codes
Merge two codes with lowest frequencies



Frequency of merged code is $f_1 + f_2$

Until *there is only one code*

Lecture 2

David Maier

15

Algorithm Design & Analysis

Huffman Code Example

7	3	4	3	1	+	2	2	6
◇	A	E	H	N		R	S	T

7	<u>3</u>	4	3	3			<u>2</u>	6
◇	A	E	H	N		R	S	T

7	<u>5</u>	4	3	3				6
◇	A	E	H	N		R	S	T

Lecture 2

David Maier

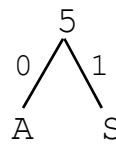
16

Algorithm Design & Analysis

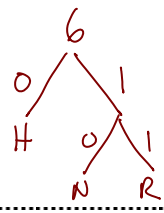
Huffman Code Example 2

7

◇



6



4

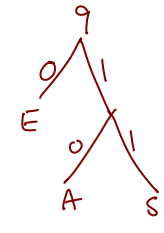
6

E

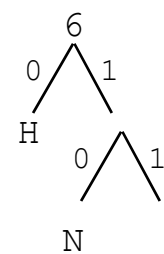
T

7

◇



6



6

T

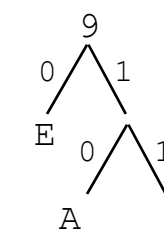
Lecture 2 David Maier **17**

Algorithm Design & Analysis

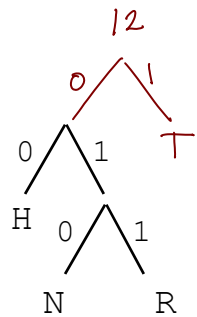
Huffman Code Example 3

7

◇



12



Lecture 2 David Maier **18**

Algorithm Design & Analysis

Huffman Code Example 4

Lecture 2 David Maier **19**

Algorithm Design & Analysis

Huffman Code Example 5

Handwritten annotations:
 14 7 diamond 12 4 E 12 3 A 8 2
 12 6 T 9 3 H 4 1 N 2 8 R
 $O(n \lg n)$
 = 79 bits

Why does it work?

Lecture 2 David Maier **20**

Algorithm Design & Analysis

Greedy Choice

• Greedy choice — There is an optimal code where $s_1 + s_2$ differ only in the last bit

d_1 - depth of s_1
 d_2 - depth of s_2
 d_{max} - greatest depth of any node

$d_{max} \geq d_1$ and $d_{max} \geq d_2$

character s_1 s_2 s_3 s_4
 frequencies f_1 f_2 f_3 f_4
 $f_1 \leq f_3$ $f_2 \leq f_4$

Lecture 2 David Maier **21**

Algorithm Design & Analysis

Greedy Choice 2

$f_3 \cdot d_{max} + f_4 \cdot d_{max} + f_1 \cdot d_1 + f_2 \cdot d_2$ *old*

$f_1 \cdot d_{max} + f_2 \cdot d_{max} + f_3 \cdot d_1 + f_4 \cdot d_2$ *new*

$s_1 - s_3$ switch:

old	$(f_3 \cdot d_{max} + f_1 \cdot d_1)$
new	$(f_1 \cdot d_{max} + f_3 \cdot d_1)$
subtract	$(f_3 - f_1) \cdot d_{max} + (f_1 - f_3) \cdot d_1$
	$(f_3 - f_1) \cdot d_{max} - (f_3 - f_1) \cdot d_1$
	$(f_3 - f_1) (d_{max} - d_1) \geq 0$

Lecture 2 David Maier **22**

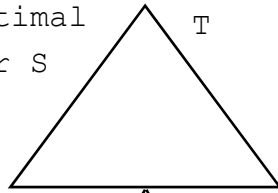
CS 410/584, Algorithm Design & Analysis: Lecture 2

Algorithm Design & Analysis

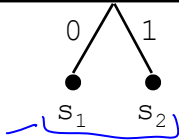
Optimal Substructure

Let S be the set of symbols

optimal
for S



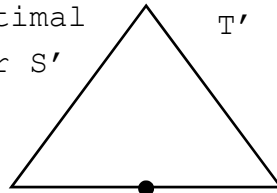
least
frequent
characters



$$\text{cost}(T) = \text{cost}(T') + \text{freq}[s_1] + \text{freq}[s_2]$$

If cheaper tree T'' for S' , gives a cheaper tree for S

optimal
for S'



$$S' = S - \{s_1, s_2\} \cup \{s_{12}\}$$

$$\text{freq}(s_{12}) = \text{freq}(s_1) + \text{freq}(s_2)$$

Lecture 2

David Maier

23