Greedy algorithms

David Kauchak
cs140
Spring 2024

1

Administrative

Assignment 6

Grades

Dr. Dave's grades

2

Greedy algorithms

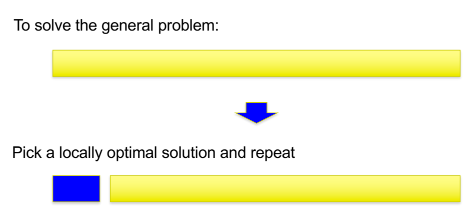
Algorithm that makes a local decision with the goal of creating a globally optimal solution

Method for solving problems where optimal solutions can be defined in terms of optimal solutions to sub-problems

3

Greedy

Greedy



4

Horn formula

A horn formula is a set of implications and negative clauses:

$$\Rightarrow x \qquad x \wedge u \Rightarrow z$$

$$\Rightarrow y \qquad \bar{x} \vee \bar{y} \vee \bar{z}$$

5

Horn formula

A horn formula is a set of **implications** and negative clauses:

$$\Rightarrow x \qquad x \wedge u \Rightarrow z$$

$$\Rightarrow y \qquad \bar{x} \vee \bar{y} \vee \bar{z}$$

LHS: positive literals anded
RHS: single positive literal

p	q	$p \Rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

6

Horn formula

A horn formula is a set of implications and **negative clauses**:

$$\Rightarrow x \qquad x \wedge u \Rightarrow z$$

$$\Rightarrow y \qquad \bar{x} \vee \bar{y} \vee \bar{z}$$

Negated literals ored

7

Goal

Given a horn formula, determine if the formula is satisfiable, i.e. an assignment of true/false to the variables that is consistent with all of the implications/causes

$$\Rightarrow x \qquad x \wedge u \Rightarrow z$$

$$\Rightarrow y \qquad \bar{x} \vee \bar{y} \vee \bar{z}$$

u	x	y	z
0	1	1	0

8

A greedy solution?

$\Rightarrow x$ $x \wedge z \Rightarrow w$ $w \wedge y \wedge z \Rightarrow x$

$x \Rightarrow y$ $x \wedge y \Rightarrow w$ $\overline{w} \vee \overline{x} \vee \overline{y}$

w	0
x	0
y	0
z	0

9

A greedy solution?

$\Rightarrow x$ $x \wedge z \Rightarrow w$ $w \wedge y \wedge z \Rightarrow x$

$x \Rightarrow y$ $x \wedge y \Rightarrow w$ $\overline{w} \vee \overline{x} \vee \overline{y}$

w	0
x	1
y	0
z	0

10

A greedy solution?

$\Rightarrow x$ $x \wedge z \Rightarrow w$ $w \wedge y \wedge z \Rightarrow x$

$x \Rightarrow y$ $x \wedge y \Rightarrow w$ $\overline{w} \vee \overline{x} \vee \overline{y}$

w	0
x	1
y	1
z	0

11

A greedy solution?

$\Rightarrow x$ $x \wedge z \Rightarrow w$ $w \wedge y \wedge z \Rightarrow x$

$x \Rightarrow y$ $x \wedge y \Rightarrow w$ $\overline{w} \vee \overline{x} \vee \overline{y}$

w	1
x	1
y	1
z	0

12

A greedy solution?

$\Rightarrow x$ $\quad x \wedge z \Rightarrow w$ $\quad w \wedge y \wedge z \Rightarrow x$

$x \Rightarrow y$ $\quad x \wedge y \Rightarrow w$ $\quad \overline{w} \vee \overline{x} \vee \overline{y}$

w	1
x	1
y	1
z	0

not satisfiable

13

A greedy solution

```
Horn(H)
set all variables to false
for all implications i
        if Empty(LHS(i))
                RHS(i) ← true
changed ← true
while changed
        changed ← false
        for all implications i
                if LHS(i) = true and !RHS(i) = true
                        RHS(i) ← true
                        changed = true
for all negative clauses c
        if c = false
                return false
return true
```

14

A greedy solution

```
Horn(H)
set all variables to false
for all implications i
        if Empty(LHS(i))
                RHS(i) ← true
changed ← true
while changed
        changed ← false
        for all implications i
                if LHS(i) = true and !RHS(i) = true
                        RHS(i) ← true
                        changed = true
for all negative clauses c
        if c = false
                return false
return true
```

set all variables of the implications of the form "⇒x" to true

15

A greedy solution

```
Horn(H)
set all variables to false
for all implications i
        if Empty(LHS(i))
                RHS(i) ← true
changed ← true
while changed
        changed ← false
        for all implications i
                if LHS(i) = true and !RHS(i) = true
                        RHS(i) ← true
                        changed = true
for all negative clauses c
        if c = false
                return false
return true
```

if the all variables of the lhs of an implication are true, then set the rhs variable to true

16

A greedy solution

```
HORN(H)
set all variables to false
for all implications i
        if EMPTY(LHS(i))
                RHS(i) ← true
changed ← true
while changed
        changed ← false
        for all implications i
                if LHS(i) = true and !RHS(i) = true
                        RHS(i) ← true
                        changed = true
for all negative clauses c
        if c = false
                return false
return true
```

see if all of the negative clauses are satisfied

17

A greedy solution

```
HORN(H)
set all variables to false
for all implications i
        if EMPTY(LHS(i))
                RHS(i) ← true
changed ← true
while changed
        changed ← false
        for all implications i
                if LHS(i) = true and !RHS(i) = true
                        RHS(i) ← true
                        changed = true
for all negative clauses c
        if c = false
                return false
return true
```

How is this a greedy algorithm?

18

A greedy solution

```
HORN(H)
set all variables to false
for all implications i
        if EMPTY(LHS(i))
                RHS(i) ← true
changed ← true
while changed
        changed ← false
        for all implications i
                if LHS(i) = true and !RHS(i) = true
                        RHS(i) ← true
                        changed = true
for all negative clauses c
        if c = false
                return false
return true
```

How is this a greedy algorithm?

Make a greedy decision about which variables to set and then moves on

19

Correctness of greedy solution

Two parts:

- If our algorithm returns an assignment, is it a valid assignment?
- If our algorithm does not return an assignment, does an assignment exist?

20

Correctness of greedy solution

If our algorithm returns an assignment, is it a valid assignment?

```
Horn(H)
set all variables to false
for all implications i
        if Empty(LHS(i))
                RHS(i) ← true
changed ← true
while changed
        changed ← false
        for all implications i
                if LHS(i) = true and !RHS(i) = true
                        RHS(i) ← true
                        changed = true
for all negative clauses c
        if c = false
                return false
return true
```

21

Correctness of greedy solution

If our algorithm returns an assignment, is it a valid assignment?

```
Horn(H)
set all variables to false
for all implications i
        if Empty(LHS(i))
                RHS(i) ← true
changed ← true
while changed
        changed ← false
        for all implications i
                if LHS(i) = true and !RHS(i) = true
                        RHS(i) ← true
                        changed = true
for all negative clauses c
        if c = false
                return false
return true
```

explicitly check all negative clauses

22

Correctness of greedy solution

If our algorithm returns an assignment, is it a valid assignment?

```
Horn(H)
set all variables to false
for all implications i
        if Empty(LHS(i))
                RHS(i) ← true
changed ← true
while changed
        changed ← false
        for all implications i
                if LHS(i) = true and !RHS(i) = true
                        RHS(i) ← true
                        changed = true
for all negative clauses c
        if c = false
                return false
return true
```

don't stop until all implications with all lhs elements true have rhs true

23

Correctness of greedy solution

If our algorithm does not return an assignment, does an assignment exist?

```
Horn(H)
set all variables to false
for all implications i
        if Empty(LHS(i))
                RHS(i) ← true
changed ← true
while changed
        changed ← false
        for all implications i
                if LHS(i) = true and !RHS(i) = true
                        RHS(i) ← true
                        changed = true
for all negative clauses c
        if c = false
                return false
return true
```

Our algorithm is "stingy". It only sets those variables that **have** to be true. All others remain false.

24

Correctness of greedy solution

If our algorithm does not return an assignment, does an assignment exist?

```
HORN(H)
set all variables to false
for all implications i
        if EMPTY(LHS(i))
                RHS(i) ← true
changed ← true
while changed
        changed ← false
        for all implications i
                if LHS(i) = true and !RHS(i) = true
                        RHS(i) ← true
                        changed = true
for all negative clauses c
        if c = false
                return false
return true
```

25

Running time?

```
HORN(H)
set all variables to false
for all implications i
        if EMPTY(LHS(i))
                RHS(i) ← true
changed ← true
while changed
        changed ← false
        for all implications i
                if LHS(i) = true and !RHS(i) = true
                        RHS(i) ← true
                        changed = true
for all negative clauses c
        if c = false
                return false
return true
```

?

n = number of variables

m = number of formulas

26

Running time?

```
HORN(H)
set all variables to false
for all implications i
        if EMPTY(LHS(i))
                RHS(i) ← true
changed ← true
while changed
        changed ← false
        for all implications i
                if LHS(i) = true and !RHS(i) = true
                        RHS(i) ← true
                        changed = true
for all negative clauses c
        if c = false
                return false
return true
```

O(nm)

n = number of variables

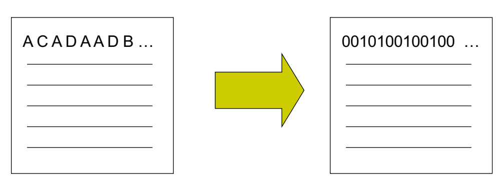
m = number of formulas

27

Data compression

Given a file containing some data of a fixed alphabet Σ (e.g. A, B, C, D), we would like to pick a binary character code that minimizes the number of bits required to represent the data.

minimize the size of the encoded file



28

Compression algorithms

General purpose [edit]

- Run-length encoding (RLE) – a simple scheme that provides good compression of data containing lots of runs of the same value.
- Lempel-Ziv 1978 (LZ78), Lempel-Ziv-Welch (LZW) – used by GIF images and compress among many other applications
- DEFLATE – used by gzip, ZIP (since version 2.0), and as part of the compression process of Portable Network Graphics (PNG), Point-to-Point Protocol (PPP), HTTP, SSH
- bzip2 – using the Burrows–Wheeler transform, this provides slower but higher compression than DEFLATE
- Lempel–Ziv–Markov chain algorithm (LZMA) – used by 7zip, xz, and other programs; higher compression than bzip2 as well as much faster decompression.
- Lempel–Ziv–Oberhumer (LZO) – designed for compression/decompression speed at the expense of compression ratios
- Statistical Lempel Ziv – a combination of statistical method and dictionary-based method; better compression ratio than using single method.

Audio [edit]

- Free Lossless Audio Codec – FLAC
- Apple Lossless – ALAC (Apple Lossless Audio Codec)
- apt-X – Lossless
- Adaptive Transform Acoustic Coding – ATRAC
- Audio Lossless Coding – also known as MPEG-4 ALS
- MPEG-4 SLS – also known as HD-AAC
- Direct Stream Transfer – DST
- Dolby TrueHD
- DTS-HD Master Audio
- Meridian Lossless Packing – MLP
- Monkey's Audio – Monkey's Audio APE
- OptimFROG
- Original Sound Quality – OSQ
- RealPlayer – RealAudio Lossless
- Shorten – SHN
- TTA – True Audio Lossless
- WavPack – WavPack lossless
- WMA Lossless – Windows Media Lossless

Graphics [edit]

- ILBM – (lossless RLE compression of Amiga IFF images)
- JBIG2 – (lossless or lossy compression of B&W images)
- JPEG-LS – (lossless/near-lossless compression standard)
- JPEG 2000 – (includes lossless compression method, as proven by Sunil Kumar, Prof San Diego State University)
- JPEG XR – formerly *WMPhoto* and *HD Photo*, includes a lossless compression method
- PGF – Progressive Graphics File (lossless or lossy compression)
- PNG – Portable Network Graphics
- TIFF – Tagged Image File Format
- Gifsicle (GPL) – Optimize gif files
- Jpegoptim (GPL) – Optimize jpeg files

http://en.wikipedia.org/wiki/Lossless_data_compression

29

Simplifying assumption: frequency only

Assume that we only have character frequency information for a file

=

Symbol	Frequency
A	70
B	3
C	20
D	37

30

Fixed length code

Use $\lceil log_2|\Sigma| \rceil$ bits for each character

A =
B =
C =
D =

31

Fixed length code

Use $\lceil log_2|\Sigma| \rceil$ bits for each character

A = 00   2 x 70 +
B = 01   2 x 3 +
C = 10   2 x 20 +
D = 11   2 x 37 =

260 bits

Symbol	Frequency
A	70
B	3
C	20
D	37

How many bits to encode the file?

32

Fixed length code

Use $\lceil log_2|\Sigma|\rceil$ bits for each character

A = 00 2 x 70 +
B = 01 2 x 3 +
C = 10 2 x 20 +
D = 11 2 x 37 =

260 bits

Symbol	Frequency
A	70
B	3
C	20
D	37

Can we do better?

33

Variable length code

What about:

A = 0 1 x 70 +
B = 01 2 x 3 +
C = 10 2 x 20 +
D = 1 1 x 37 =

153 bits

Symbol	Frequency
A	70
B	3
C	20
D	37

How many bits to encode the file?

34

Decoding a file

A = 0
B = 01
C = 10
D = 1

010100011010

What characters does this sequence represent?

35

Decoding a file

A = 0
B = 01
C = 10
D = 1

010100011010

A D or **B?**

What characters does this sequence represent?

36

Variable length code

What about:

A = 0
B = 100
C = 101
D = 11

Is it decodeable?

Symbol	Frequency
A	70
B	3
C	20
D	37

37

Variable length code

What about:

A = 0 1 x 70 +
B = 100 3 x 3 +
C = 101 3 x 20 +
D = 11 2 x 37 =

213 bits
(18% reduction)

Symbol	Frequency
A	70
B	3
C	20
D	37

How many bits to encode the file?

38

Prefix codes

A prefix code is a set of codes where no codeword is a **prefix** of any other codeword

A = 0
B = 01
C = 10
D = 1

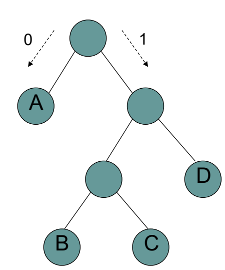
A = 0
B = 100
C = 101
D = 11

39

Prefix tree

We can encode a prefix code using a **full** binary tree where each leaf represents an encoding of a symbol

A = 0
B = 100
C = 101
D = 11

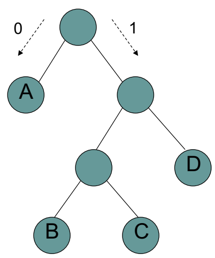


40

Decoding using a prefix tree

To decode, we traverse the graph until a leaf node is reached and output the symbol

A = 0
B = 100
C = 101
D = 11

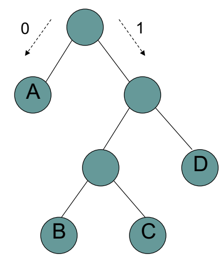


41

Decoding using a prefix tree

Traverse the graph until a leaf node is reached and output the symbol

1000111010100



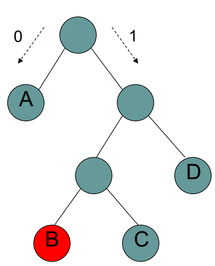
42

Decoding using a prefix tree

Traverse the graph until a leaf node is reached and output the symbol

100|0111010100

B



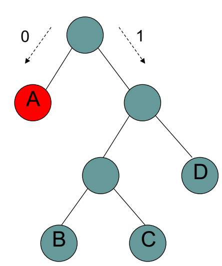
43

Decoding using a prefix tree

Traverse the graph until a leaf node is reached and output the symbol

100|0|111010100

B A



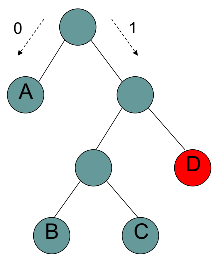
44

Decoding using a prefix tree

Traverse the graph until a leaf node is reached and output the symbol

1000111010100

B A D



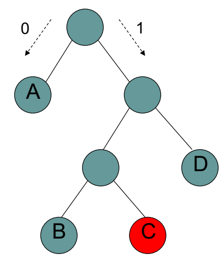
45

Decoding using a prefix tree

Traverse the graph until a leaf node is reached and output the symbol

1000111010100

B A D C



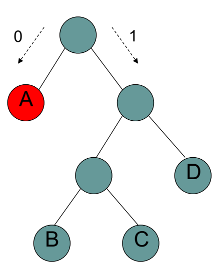
46

Decoding using a prefix tree

Traverse the graph until a leaf node is reached and output the symbol

1000111010100

B A D CA



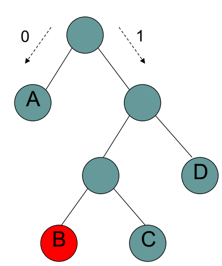
47

Decoding using a prefix tree

Traverse the graph until a leaf node is reached and output the symbol

1000111010100

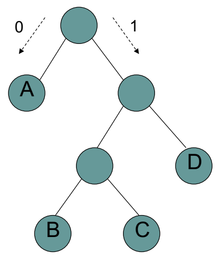
B A D CA B



48

Determining the cost of a file

Symbol	Frequency
A	70
B	3
C	20
D	37

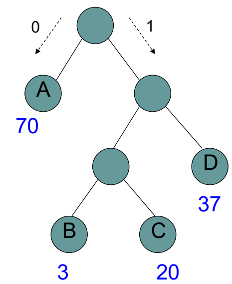


49

Determining the cost of a file

Symbol	Frequency
A	70
B	3
C	20
D	37

$$\mathrm{cost}(T) = \sum_{i=1}^{n} f_i \, \mathrm{depth}(i)$$

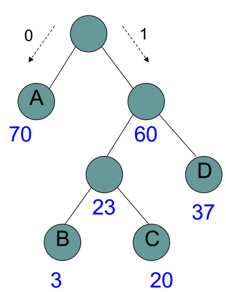


50

Determining the cost of a file

Symbol	Frequency
A	70
B	3
C	20
D	37

If we label the internal nodes with the sum of the children…

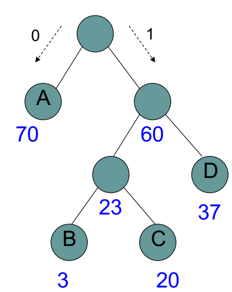


51

Determining the cost of a file

Symbol	Frequency
A	70
B	3
C	20
D	37

Cost is equal to the sum of the internal nodes (excluding the root) and the leaf nodes

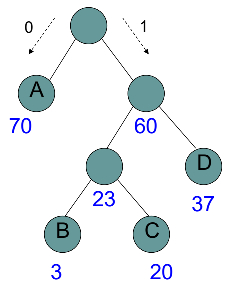


52

Determining the cost of a file

As we move down the tree, one bit gets read for every nonroot node

- 70 times we see a 0 by itself
- 60 times we see a prefix that starts with a 1
- of those, 37 times we see an additional 1
- the remaining 23 times we see an additional 0
- of these, 20 times we see a last 1 and 3 times a last 0

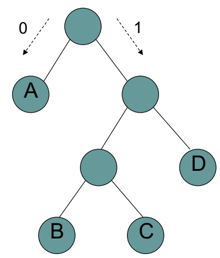


53

A greedy algorithm?

Given file frequencies, can we come up with a prefix-free encoding (i.e. build a prefix tree) that minimizes the number of bits?

Symbol	Frequency
A	70
B	3
C	20
D	37



Where should the highest frequency items be?

54

A greedy algorithm?

Given file frequencies, can we come up with a prefix-free encoding (i.e. build a prefix tree) that minimizes the number of bits?

```
HUFFMAN(F)
Q ← MAKEHEAP(F)
for i ← 1 to |Q| − 1
        allocate a new node z
        left[z] ← x ← EXTRACTMIN(Q)
        right[z] ← y ← EXTRACTMIN(Q)
        f[z] ← f[x] + f[y]
        INSERT(Q, z)
return EXTRACTMIN(Q)
```

55

```
HUFFMAN(F)
Q ← MAKEHEAP(F)
for i ← 1 to |Q| − 1
        allocate a new node z
        left[z] ← x ← EXTRACTMIN(Q)
        right[z] ← y ← EXTRACTMIN(Q)
        f[z] ← f[x] + f[y]
        INSERT(Q, z)
return EXTRACTMIN(Q)
```

Symbol	Frequency
A	70
B	3
C	20
D	37

Heap

56

```
HUFFMAN(F)
  Q ← MAKEHEAP(F)
  for i ← 1 to |Q| − 1
        allocate a new node z
        left[z] ← x ← EXTRACTMIN(Q)
        right[z] ← y ← EXTRACTMIN(Q)
        f[z] ← f[x] + f[y]
        INSERT(Q, z)
  return EXTRACTMIN(Q)
```

Symbol	Frequency
A	70
B	3
C	20
D	37

Heap

B 3
C 20
D 37
A 70

57

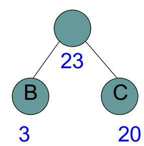
```
HUFFMAN(F)
  Q ← MAKEHEAP(F)
  for i ← 1 to |Q| − 1
        allocate a new node z
        left[z] ← x ← EXTRACTMIN(Q)
        right[z] ← y ← EXTRACTMIN(Q)
        f[z] ← f[x] + f[y]
        INSERT(Q, z)
  return EXTRACTMIN(Q)
```

Symbol	Frequency
A	70
B	3
C	20
D	37

Heap

merging with this node will incur an additional cost of 23 →

BC 23
D 37
A 70



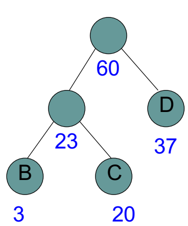
58

```
HUFFMAN(F)
  Q ← MAKEHEAP(F)
  for i ← 1 to |Q| − 1
        allocate a new node z
        left[z] ← x ← EXTRACTMIN(Q)
        right[z] ← y ← EXTRACTMIN(Q)
        f[z] ← f[x] + f[y]
        INSERT(Q, z)
  return EXTRACTMIN(Q)
```

Symbol	Frequency
A	70
B	3
C	20
D	37

Heap

BCD 60
A 70



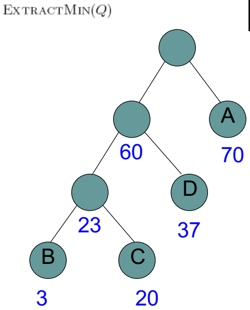
59

```
HUFFMAN(F)
  Q ← MAKEHEAP(F)
  for i ← 1 to |Q| − 1
        allocate a new node z
        left[z] ← x ← EXTRACTMIN(Q)
        right[z] ← y ← EXTRACTMIN(Q)
        f[z] ← f[x] + f[y]
        INSERT(Q, z)
  return EXTRACTMIN(Q)
```

Symbol	Frequency
A	70
B	3
C	20
D	37

Heap

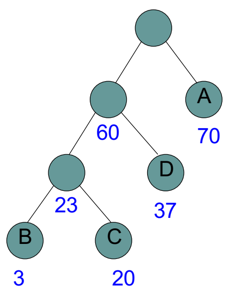
ABCD 130



60

What is the code (assume left = 0)?

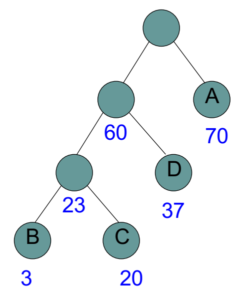
Symbol	Frequency
A	70
B	3
C	20
D	37



61

What is the code (assume left = 0)?

Symbol	Frequency
A	70
B	3
C	20
D	37



A: 1
B: 000
C: 001
D: 01

62

Proving correctness

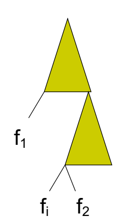
The algorithm selects the symbols with the two smallest frequencies first (call them f_1 and f_2)

63

Proving correctness: proof by contradiction

The algorithm selects the symbols with the two smallest frequencies first (call them f_1 and f_2)

Consider a tree that did not do this:



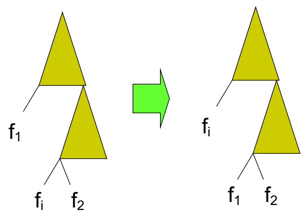
Is it optimal?

64

Proving correctness

The algorithm selects the symbols with the two smallest frequencies first (call them f_1 and f_2)

Consider a tree that did not do this:



$$\text{cost}(T) = \sum_{i=1}^{n} f_i \,\text{depth}(i)$$

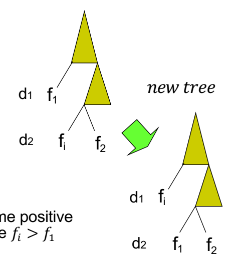
- frequencies don't change
- cost will **decrease** since $f_1 < f_i$

contradiction

65

$$\begin{aligned}
&\textit{original tree} - \textit{new tree} = \\
&f_1 d_1 + f_i d_2 + f_2 d_2 - (f_1 d_2 + f_2 d_2 + f_i d_1) = \\
&f_1 d_1 + f_i d_2 + f_2 d_2 - f_1 d_2 - f_2 d_2 - f_i d_1 = \\
&f_1 d_1 + f_i d_2 - f_1 d_2 - f_i d_1 = \\
&f_1 d_1 - f_i d_1 + f_i d_2 - f_1 d_2 = \\
&(f_1 - f_i) d_1 + (f_i - f_1) d_2 = \\
&-c d_1 + c d_2
\end{aligned}$$

where c is some positive constant, since $f_i > f_1$



Since $d_1 < d_2$ then $-cd_1 + cd_2 > 0$ which shows that cost of the new tree is less than the cost of the original tree

66

Runtime?

```
HUFFMAN(F)
Q ← MAKEHEAP(F)
for i ← 1 to |Q| − 1
        allocate a new node z
        left[z] ← x ← EXTRACTMIN(Q)
        right[z] ← y ← EXTRACTMIN(Q)
        f[z] ← f[x] + f[y]
        INSERT(Q, z)
return EXTRACTMIN(Q)
```

1 call to MakeHeap

2(n-1) calls ExtractMin

n-1 calls Insert

O(n log n)

67

Non-optimal greedy algorithms

All the greedy algorithms we've looked at so far give the optimal answer

Some of the most common greedy algorithms generate good, but non-optimal solutions

- set cover
- clustering
- hill-climbing
- relaxation

68

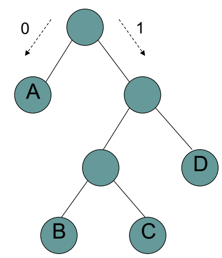
Handout

69

Decoding using a prefix tree

Traverse the graph until a leaf node is reached and output the symbol

1000111010100



70

```
HUFFMAN(F)
Q ← MAKEHEAP(F)
for i ← 1 to |Q| − 1
        allocate a new node z
        left[z] ← x ← EXTRACTMIN(Q)
        right[z] ← y ← EXTRACTMIN(Q)
        f[z] ← f[x] + f[y]
        INSERT(Q, z)
return EXTRACTMIN(Q)
```

Symbol	Frequency
A	5
B	20
C	10
D	13
E	9

Heap

What is the tree?

What is the encoding?

How many bits to encode the file?

71