Greedy algorithms 2

David Kauchak
cs302
Spring 2012

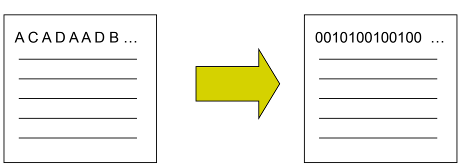
Knapsack problems: Greedy or not?

- **0-1 Knapsack** – A thief robbing a store finds n items worth $v_1, v_2, .., v_n$ dollars and weight $w_1, w_2, …, w_n$ pounds, where v_i and w_i are integers. The thief can carry at most W pounds in the knapsack. Which items should the thief take if he wants to maximize value.
- **Fractional knapsack problem** – Same as above, but the thief happens to be at the bulk section of the store and can carry fractional portions of the items. For example, the thief could take 20% of item i for a weight of $0.2w_i$ and a value of $0.2v_i$.

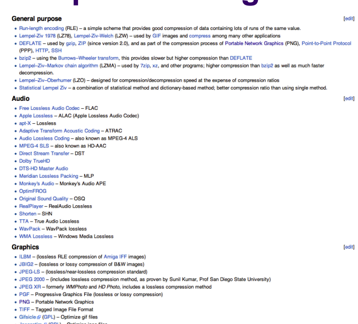
Data compression

- 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



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

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

Fixed length code

Use ceil($\log_2|\Sigma|$) bits for each character

A =
B =
C =
D =

Fixed length code

Use ceil($\log_2|\Sigma|$) bits for each character

Symbol	Frequency
A	70
B	3
C	20
D	37

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

260 bits

How many bits to encode the file?

Fixed length code

Use ceil($\log_2|\Sigma|$) bits for each character

Symbol	Frequency
A	70
B	3
C	20
D	37

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

260 bits

Can we do better?

Variable length code

What about:

Symbol	Frequency
A	70
B	3
C	20
D	37

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

153 bits

How many bits to encode the file?

Decoding a file

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

010100011010

What characters does this sequence represent?

Decoding a file

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

010100011010

A D or **B?**

What characters does this sequence represent?

Variable length code

What about:

Symbol	Frequency
A	70
B	3
C	20
D	37

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

213 bits
(18% reduction)

How many bits to encode the file?

Prefix codes

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

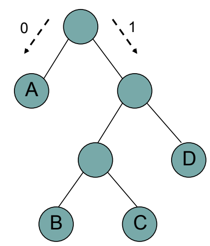
A = 0
B = 01
C = 10
D = 1

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

Prefix tree

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

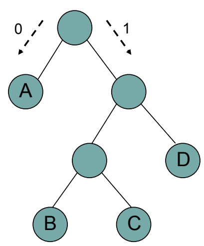
A = 0
B = 100
C = 101
D = 11



Decoding using a prefix tree

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

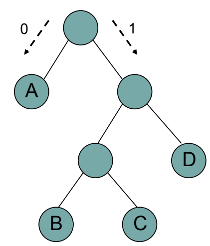
A = 0
B = 100
C = 101
D = 11



Decoding using a prefix tree

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

1000111010100

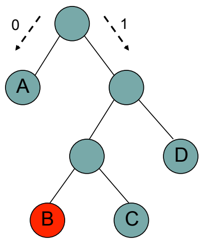


Decoding using a prefix tree

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

1000111010100

B

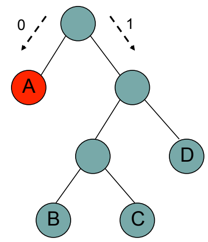


Decoding using a prefix tree

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

1000111010100

B A

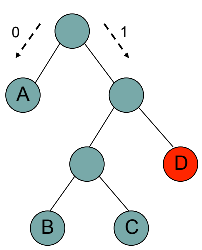


Decoding using a prefix tree

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

1000111010100

B A D

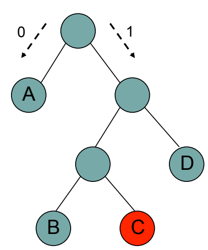


Decoding using a prefix tree

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

1000111010100

B A D C

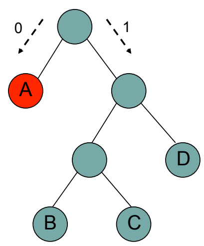


Decoding using a prefix tree

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

1000111010100

B A D C A

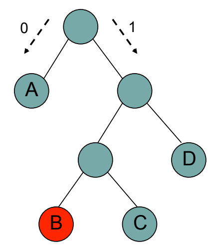


Decoding using a prefix tree

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

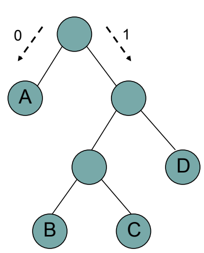
1000111010100

B A D C A B



Determining the cost of a file

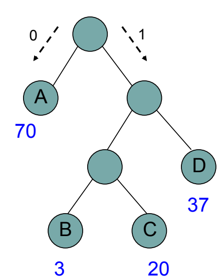
Symbol	Frequency
A	70
B	3
C	20
D	37



Determining the cost of a file

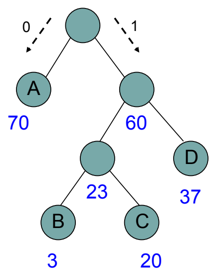
Symbol	Frequency
A	70
B	3
C	20
D	37

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



Determining the cost of a file

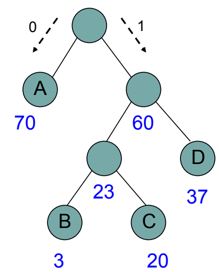
Symbol	Frequency
A	70
B	3
C	20
D	37



What if we label the internal nodes with the sum of the children?

Determining the cost of a file

Symbol	Frequency
A	70
B	3
C	20
D	37

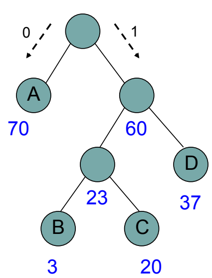


Cost is equal to the sum of the internal nodes and the leaf nodes

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



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)
```

```
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


```
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

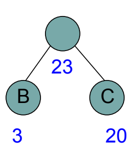

```
HUFFMAN(F)
Q ← MAKEHEAP(F)
for i ← 1 to |Q| − 1
        allocate a new node z
        left[z] ← x ← EXTRACTMIN(Q)
        right[z] ← y ← EXTRACTMIN(Q)
        f[z] ← f[x] + f[y]
        INSERT(Q, z)
return EXTRACTMIN(Q)
```

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

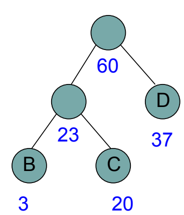



```
HUFFMAN(F)
Q ← MAKEHEAP(F)
for i ← 1 to |Q| − 1
        allocate a new node z
        left[z] ← x ← EXTRACTMIN(Q)
        right[z] ← y ← EXTRACTMIN(Q)
        f[z] ← f[x] + f[y]
        INSERT(Q, z)
return EXTRACTMIN(Q)
```

Symbol	Frequency
A	70
B	3
C	20
D	37

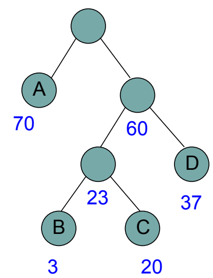
Heap

BCD 60
A 70



```
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

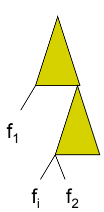
ABCD 130

Is it correct?

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

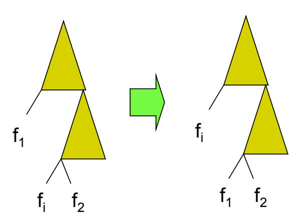
Is it correct?

- 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 correct?

- 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

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)

Non-optimal greedy algorithms

- All the greedy algorithms we've looked at today give the optimal answer
- Some of the most common greedy algorithms generate good, but non-optimal solutions
  - set cover
  - clustering
  - hill-climbing
  - relaxation