Applications of heaps

Comp Sci 1575 Data Structures

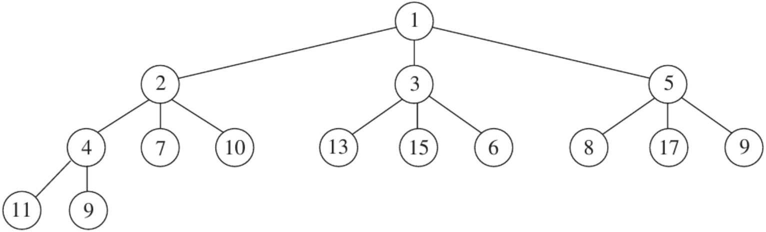
Missouri S&T Computer Science

Heap applications

Simplicity does not precede complexity, but follows it.
-Alan Perlis

Outline

D-heaps



- Like a binary heap except that all nodes have d children (thus, a binary heap is a 2-heap).

D-heaps

- Shallow, thus run time of inserts to $O(log_d N)$.
- For large d, *deleteMin* operation is more expensive, because even though the tree is shallower, the minimum of d children must be found, which takes $d - 1$ comparisons using a standard algorithm, raising the time for this operation to $O(d\ log_d N)$. If d is a constant, both running times are $O(logN)$.
- Multiplications and divisions to find children and parents are now by d, which, unless d is a power of 2, increasing the running time, because we can no longer implement division by a bit shift.
- Number of insertions is greater than the number of deleteMins.
- 4-heaps may outperform binary heaps in practice.

Outline

General applications

- Heaps are used for real simulations of some kinds of queues (patient priority, multi-tasking priority, etc).

  Note: For this type of heap, max is better, because the more important end (higher numbers) can have levels of importance added in constant time by just adding a higher priority, unlike a min-heap which requires adjusting all values in the heap to add more resolution.
- Heaps are used when one part of an algorithm requires producing an ordered stream. For this type of heap, min or max serve a similar purpose.
  - Graph path finding (more to come later)
  - Best-first search (like path finding)
  - Minimum spanning tree calculation on a graph
  - Huffman trees (overview today)
  - Bandwidth management
  - more?

Outline

Outline

Problem:

- ASCII coding scheme assigns a unique eight-bit value to each character.
- It takes a certain minimum number of bits to provide unique codes for each character.
- For example, it takes $\log 128$ (or seven bits to provide the 128 unique codes) needed to represent the 128 symbols of the ASCII character set.
- The requirement for $\log n$ bits to represent n unique code values assumes that all codes will be the same length, as are ASCII codes. This is called a **fixed-length coding scheme**.
- Compression?
- Variable-length coding scheme?

Outline

Letter frequencies in English

Letter	Frequency	Letter	Frequency
A	77	N	67
B	17	O	67
C	32	P	20
D	42	Q	5
E	120	R	59
F	24	S	67
G	17	T	85
H	50	U	37
I	76	V	12
J	4	W	22
K	7	X	4
L	42	Y	22
M	24	Z	2

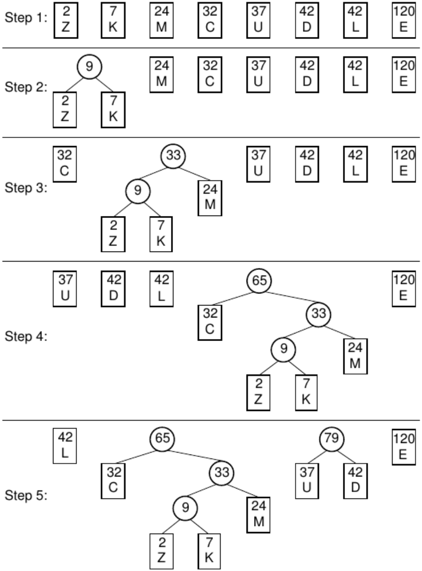
Outline

Store letters in a tree

- Shallower is faster, so store more frequent letters shallow in the tree
- Goal is to build a tree with the minimum external path weight.
- Define the weighted path length of a leaf to be its weight times its depth.
- Binary tree with minimum external path weight is the one with the minimum sum of weighted path lengths for the given set of leaves.
- A letter with high weight should have low depth, so that it will count the least against the total path length.
- As a result, another letter might be pushed deeper in the tree if it has less weight.

Outline

Building a Huffman coding tree



Why Q: why is the heap helpful here?

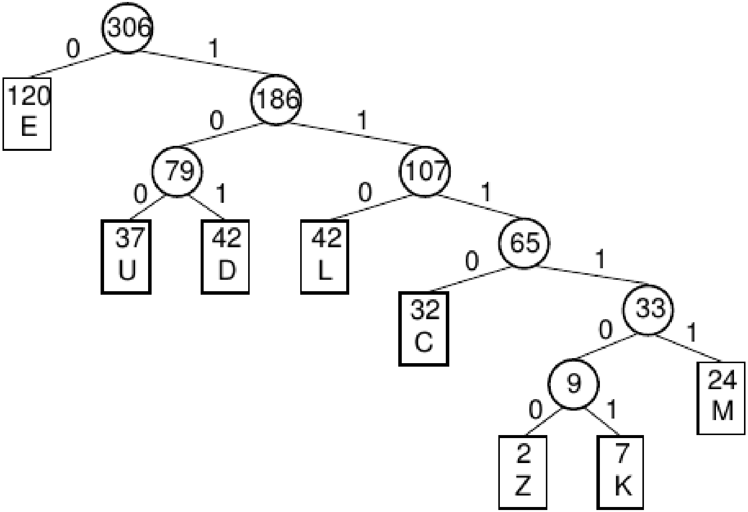
Outline

Building a Huffman coding tree

1. Create a collection of n initial Huffman trees, each of which is a single leaf node containing one of the letters.
2. Put the n partial trees onto a priority queue organized by weight (frequency).
3. Remove the first two trees (the ones with lowest weight) from the priority queue.
4. Join these two trees together to create a new tree whose root has the two trees as children with the weight of the root as the sum of the weights of the two trees.
5. Put this root / tree back into the priority queue
6. Repeat until all of the partial Huffman trees have been combined into one.

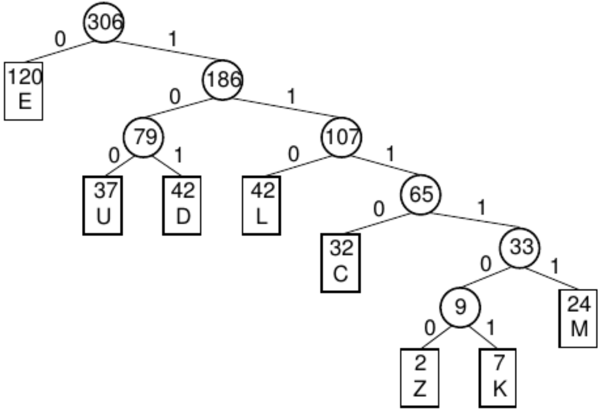
Outline

Huffman tree after complete creation



- Higher frequency letters stored more shallowly
- Why Q: What does this help?

Huffman tree after complete creation



Once the Huffman tree has been constructed, it is an easy matter to assign codes to individual letters. Beginning at the root, we assign either a ‘0’ or a ‘1’ to each edge in the tree. ‘0’ is assigned to edges connecting a node with its left child, and ‘1’ to edges connecting a node with its right child.

Outline

Huffman encoding scheme

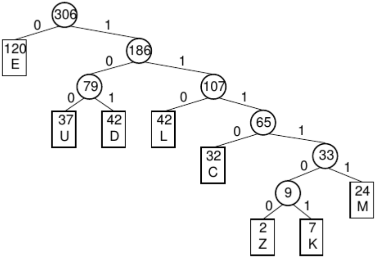
Letter	Freq	Code	Bits
C	32	1110	4
D	42	101	3
E	120	0	1
K	7	111101	6
L	42	110	3
M	24	11111	5
U	37	100	3
Z	2	111100	6

Once we have the encoding scheme, we can use any lookup method for encoding/decoding:

- the original tree: **encode**: searching for freq key, **decode**: traversing L-0 R-1)
- or any associative array like BST, hash table, to store letter-code mappings
- etc.

Huffman encoding scheme: does it work?

Letter	Freq	Code	Bits
C	32	1110	4
D	42	101	3
E	120	0	1
K	7	111101	6
L	42	110	3
M	24	11111	5
U	37	100	3
Z	2	111100	6



Ambiguous parses only occur on internal nodes! A set of codes

Efficiency

- Huffman tree building is an example of a greedy algorithm. At each step, the algorithm makes a "greedy" decision to merge the two subtrees with least weight.
- In theory, it is an optimal coding method whenever the true frequencies are known, and the frequency of a letter is independent of the context of that letter in the message.
- In practice, the frequencies of letters in an English text document do change depending on context. For example, while E is the most commonly used letter of the alphabet in English documents, T is more common as the first letter of a word.
- This is why most commercial compression utilities do not use Huffman coding as their primary coding method, but instead use techniques that take advantage of the context for the letters.
- In general, Huffman coding does better when there is large variation in the frequencies of letters.

Outline

Sorting with a heap??

- How can we sort with a heap?
- Check out sort video of other sorts: `https://www.youtube.com/watch?v=WaNLJf8xzC4`

Sorting with a heap

Name	Priority Queue Implementation	Best	Average	Worst
Heapsort	Heap	$n\log(n)$	$n\log(n)$	$n\log(n)$
Smoothsort	Leonardo Heap	n	$n\log(n)$	$n\log(n)$
Selection sort	Unordered Array	n^2	n^2	n^2
Insertion sort	Ordered Array	n	n^2	n^2
Tree sort	Self-balancing binary search tree	$n\log(n)$	$n\log(n)$	$n\log(n)$

Heap was invented for heapsort, and the priority queue is equivalent to sorting in some senses. Smoothsort is one of the best all-round sorts (along with blocksort).