

EE360C: Algorithms

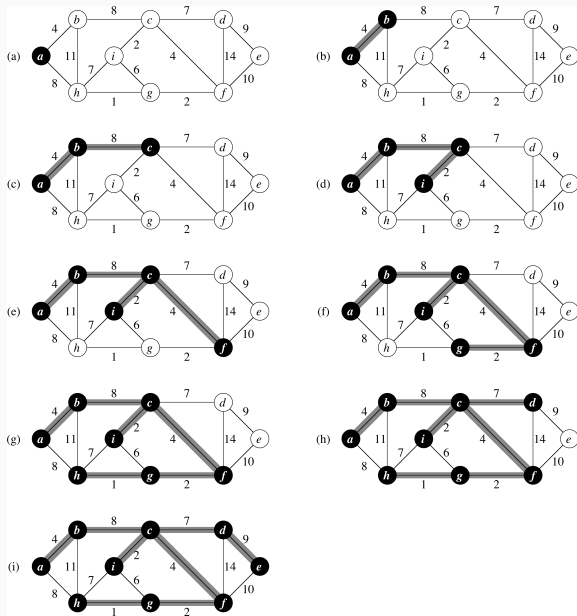
Greedy Algorithms

Part 4/4

Summer 2019

Department of Electrical and Computer Engineering
University of Texas at Austin

Prim's Algorithm Example



Huffman Codes

Huffman Codes

Huffman coding is a variable length data compression scheme that takes advantage of the fact that some characters are more common than others

- characters that are more common get shorter codes
- avoid ambiguity by requiring that no codeword is a prefix of another

For example, if the letters to encode are a, b, c, d, e, f, and the codes are, respectively, 0, 101, 100, 111, 1101, 1100:

- encoding is easy; just look up the code for each letter and concatenate them
- decoding is not bad since the coding is “prefix-free”
- e.g., decode 001011101

Prefix codes and Binary Trees

- Any binary prefix code can be described using a binary tree in which codewords are the *leaves* of the tree, and where a left branch means “0” and a right branch means “1”.
- An encoding of S constructed from T is a prefix code. Why? (Proof required for Programming Assignment)
- Decoding the prefix code: traverse the tree from root to leave, letting the input characters tell us which branch to take.

Huffman Codes

Let C be a set of n characters, each with frequency $f(c_i)$ in some file. Construct the optimal codebook of C that minimizes the number of bits needed to represent the file.

Given the codetree T , the number of bits needed to encode the file is:


$$B(T) = \sum_{c \in C} f(c) d_T(c)$$

where $d_T(c)$ is the depth of c 's leaf in T .

The Optimal Tree Structure

What is the tree structure for an optimal prefix code?

The binary tree corresponding to the optimal prefix code is full.

Proof:  by exchange argument.

- Let T be a binary tree corresponding to the optimal prefix code.
- Suppose it contains a node u with exactly one child v .
 - If u is a root node, then delete u , and use v as the root.
 - If u is not the root, let w be the parent of u . If u is deleted, and make v the child of w . By doing so, the depth of the subtree rooted at u is decreased by 1, and it does not effect the number of leaves.
- In both case, the new tree T' obtained by deleting u has a smaller average number of bits per leaf than T , contradicting the optimality of T .

The Optimal Tree Structure (contd.)

If someone gave us the optimal tree for a prefix code, how would we assign the elements of S to the leaves?

Suppose that u and v are leaves of T^* (an optimal prefix code tree), and $\text{depth}(u) < \text{depth}(v)$. Suppose that in a labeling of T^* corresponding to an optimal prefix code, leaf u is labeled with $y \in S$ and leaf v is labeled with $z \in S$. Then $f_y \geq f_z$.

Proof by exchange argument. (On white board)

Where are the most frequent characters? Where are the least frequent characters?

Constructing a Huffman Code

Approach

Start with $|C|$ leaves; perform $|C| - 1$ merging operations to create tree

- use a min heap keyed on f to merge the two least frequent objects
- the result is a new object with frequency that's the sum of the frequencies of the merged objects

HUFFMAN(C)

```
1   $n \leftarrow |C|$ 
2   $Q \leftarrow C$ 
3  for  $i \leftarrow 1$  to  $n - 1$ 
4      do allocate a new node  $z$ 
5           $\text{left}[z] \leftarrow x \leftarrow \text{EXTRACT-MIN}(Q)$ 
6           $\text{right}[z] \leftarrow y \leftarrow \text{EXTRACT-MIN}(Q)$ 
7           $f[z] \leftarrow f[x] + f[y]$ 
8           $\text{INSERT}(Q, z)$ 
9      return  $\text{EXTRACT-MIN}(Q)$ 
```

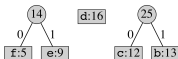
Huffman Code Algorithm Visualized

(a) f:5 e:9 c:12 b:13 d:16 a:45

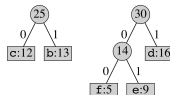
(b) c:12 b:13 14 d:16 a:45



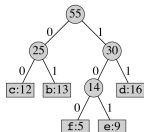
(c) 14 d:16 25 a:45



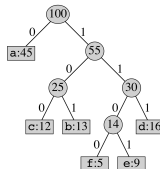
(d) 25 30 a:45



(e) a:45



(f)



Correctness of Huffman's Algorithm

What does this have to do with the greedy choice?

By choosing x and y with the lowest frequencies, Huffman picks the least cost merge at each step.

- Is this algorithm correct? In other words, does it minimize the expected coding length?
- Our approach will be to show that any tree that differs from the Huffman tree, can be converted into the Huffman tree without increasing the cost.

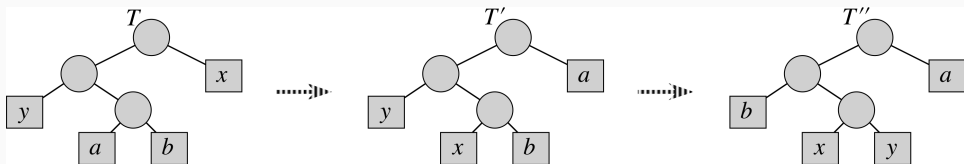
Correctness of Huffman's Algorithm

Lemma

Let C be an alphabet in which each $c \in C$ has frequency $f[c]$. Let x and y be two characters in C having the lowest frequencies. Then there exists an optimal prefix code for C in which the codewords for x and y have the same length and differ only in the last bit.

The idea of the proof is to take the tree T representing an arbitrary optimal prefix code and modify it to make a tree representing another optimal prefix code such that the characters x and y appear as sibling leaves of maximum depth in the new tree. If we can construct such a tree, then the codewords for x and y will have the same length and differ only in the last bit.

Proof Visualized



Correctness of Huffman's Algorithm: Substructure

Lemma

Let C be an alphabet with $f[c]$ defined for each $c \in C$. Let x and y be characters in C with lowest frequencies. Let $C' = C - \{x, y\} \cup z$ where $f[z] = f[x] + f[y]$. Let T' be any optimal prefix code for C' . Then the tree T obtained by replacing the leaf node for Z in T' with an internal node having x and y as children is an optimal prefix code for C .

Proof of Huffman Substructure

Let $d_T(c)$ be the depth of c in T (i.e., the length of c 's code). The cost with respect to all nodes other than x and y is unchanged from T to T' .

$d_T(x) = d_T(y) = d_{T'}(z) + 1$. So

$$\begin{aligned} f[x]d_T(x) + f[y]d_T(y) &= (f[x] + f[y])(d_{T'}(z) + 1) \\ &= f[z]d_{T'}(z) + (f[x] + f[y]) \end{aligned}$$

$B(T) = B(T') + f[x] + f[y]$. Rewriting, $B(T') = B(T) - f[x] - f[y]$.

Proof by Contradiction. Suppose T was not an optimal prefix code for C . Then there is some T'' , $B(T'') < B(T)$. Because x and y have the smallest frequencies in C , x and y are siblings in T'' , too (that's the greedy choice property). Construct T''' that is exactly T'' except that the subtree rooted at the parent of x and y is replaced by a node z , whose frequency is the sum of the frequencies of x and y .

$$\begin{aligned} B(T''') &= B(T'') - f[x] - f[y] \\ &< B(T) - f[x] - f[y] \\ &< B(T') \end{aligned}$$

But that's a contradiction, since T' was supposed to be optimal for C' , and now we've found something (T''') for the same C' that's better.

Notes on the Greedy Strategy

The Greedy Strategy in General

As a general rule, we are pretty direct in applying the greedy method:

1. Cast the optimization problem into one in which we make a choice and are left with only a single subproblem to solve.
2. Prove that there is always an optimal solution to the original problem that makes the greedy choice (i.e., that the greedy choice is *safe*)
3. Demonstrate that combining a solution to the remaining subproblem with the greedy choice yields an optimal solution to the original problem

Identifying Greedy Problems

In general, problems that can be solved by a greedy approach exhibit:

- optimal substructure — if an optimal solution to a problem contains within it optimal solutions to subproblems
- greedy-choice property — a globally optimal solution can be arrived at by making a locally optimal choice

The Knapsack Problem

0-1 Knapsack

A thief robbing a store finds n items; the i^{th} item is worth v_i dollars and weighs w_i pounds (both integers). The thief wants to take as valuable a load as possible but can carry only W pounds. Which items should he take?

Fractional Knapsack

The problem is the same as the 0-1 Knapsack problem, but the thief can take fractions of items; he doesn't have to take all or none of a particular item

Fractional Knapsack Problem

Does the problem display the optimal substructure property?

Yes. Take w of item j ; then we're left with finding an optimal solution for weight $W - w$ given the remaining $n - 1$ items plus $w_j - w$ of item j

Solution

- calculate v_i/w_i for each item (the value per pound)
- sort the items by value per pound
- take items from the front of the list until the knapsack is full

0-1 Knapsack Problem

Does the problem display the optimal substructure property?

Yes. If we take item j , we're left with finding an optimal solution for weight $W - w_j$ given items $\{1, 2, \dots, j-1, j+1, \dots, n\}$.

Does the problem satisfy the greedy-choice property?

No. Consider a knapsack that can hold 50 pounds and a set of three possible items: one that weighs 10 pounds and is worth \$60, one that weighs 20 pounds and is worth \$100, and one that weighs 30 pounds and is worth \$120.

0-1 Knapsack Problem (cont.)

What happened?

By choosing greedily, we didn't fill up the knapsack entirely, resulting in a lower effective value per pound of the knapsack.

What we should have done was compare the value of the subproblem that included the greedy choice with the value of the subproblem that *didn't*.

Questions
