

Module 6

Optimization Problems

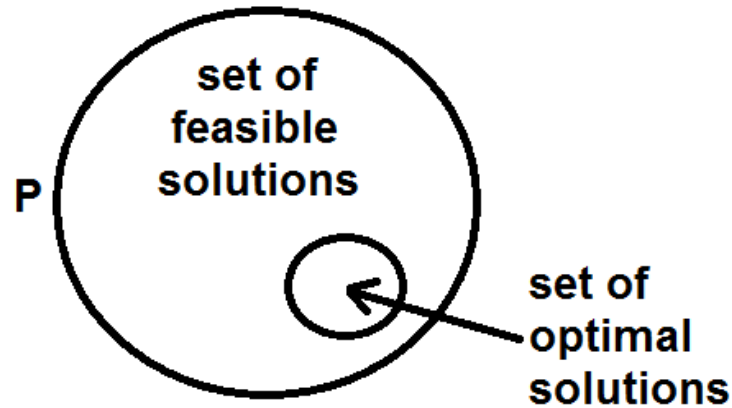
S. Lakshmivarahan
School of Computer Science
University of Oklahoma
USA-73019
Varahan@ou.edu

Optimization Problems

- Optimization problems occur naturally and they take various shapes and forms
- Examples include
 - Find the length of the shortest path from point A to point B
 - Find the least cost network to connect a set of n buildings in your college campus
 - Find the best algorithm (least computational complexity) to solve a given problem
 - Find the least cost tour for a salesman to use so that he can navigate a set of college towns in a state

Common Features of Optimization Problems

- Let P be the given problem
- Any solution that solves the problem is called a “feasible solution”



Feasible Set Examples

- In the case of short path problems, the set of all paths connecting A and B constitute the feasible set
- In the case of least cost network, the set of networks connecting the given set of n buildings form the feasible set
- In the case of best algorithms, the set of algorithms that solve the given problem is the feasible set
- For the travelling salesman problem, the set of all towns in a given graph constitutes the feasible set

Optimization Problems

- The optimal solution, in general, is not unique. That is, there exists a subset of solutions which may be optimal
- Our goal is to identify this subset of optimal solutions

Optimization Problems

- Two approaches
 1. Greedy Algorithms
 2. Dynamic Programming
- We will provide examples of optimization problems solved by these two methods

Greedy Algorithms

- Simple and efficient
- Myopic in the sense that it picks what is best for this moment
- Leads to optimal algorithms for many problems
- Examples include:
 - Design of compression code
 - Design of least cost network called the minimum spanning tree
- Does not lead to optimal solutions in all cases

Dynamic Programming

- Always leads to optimal solution
- This is a systematic procedure that exhaustively searches through all feasible solutions to arrive at the optimal solution
- It costs more to find the optimal solution but there is saving when optimal algorithm is used to solve the problem

Data Compression- Greedy Algorithm

- The need for digitizing data in all forms- print, voice, video, and subsequent compression of the digitized output arise from a number of different directions
- Library of Congress in Washington D.C. has the mandate to acquire and preserve data in all of the three forms.
- To minimize the storage space, the physical buildings and cost of maintaining them, one turns to digitizing and data compression techniques so that we can minimize storage and manage information in a cost effective manner

Data Compression- Greedy Algorithm

- A robot roaming the surface of Mars wants to transmit large volumes of digital pictures by compressing them so as to maximize the efficiency of transmission.

Data Compression- Greedy Algorithm

- Two forms of compression
 1. Information lossless compression
 - Data, music
 2. Lossey compression
 - Video

Data Compression- Greedy Algorithm

- Most of the compression algorithms exploit the redundancy in the language used to encode the information.
- The popular game show “Wheel of Fortune” comes to mind. Redundancy relates to the ability to recognize a word by knowing only certain portions of the word.

Data Compression- Greedy Algorithm

- Let us examine the encoding process.
- Thoughts, ideas, and information are rather abstract
- Using a natural language- English for example, we then translate the abstract thoughts and ideas into sentences in English, using only the legal words in the dictionary

Data Compression- Greedy Algorithm

- For storing/transmission, we then translate the English words/sentences/paragraphs/pages, etc into binary words using a fixed length encoding scheme where every letter is represented as binary string of fixed length, say 8.
- The standard ASCII code is an example of the fixed length code.

Data Compression- Greedy Algorithm

- The basic idea behind the data compression scheme is that the distribution of the 26 alphabets in all the legal words in English is not uniform- The letter E occurs with larger frequency than the letter Z

Data Compression- Greedy Algorithm

- Thus, one can obtain effective data compression by making the length of the binary code to be inversely proportional to its frequency
- The resulting variable length code indeed achieves considerable compression

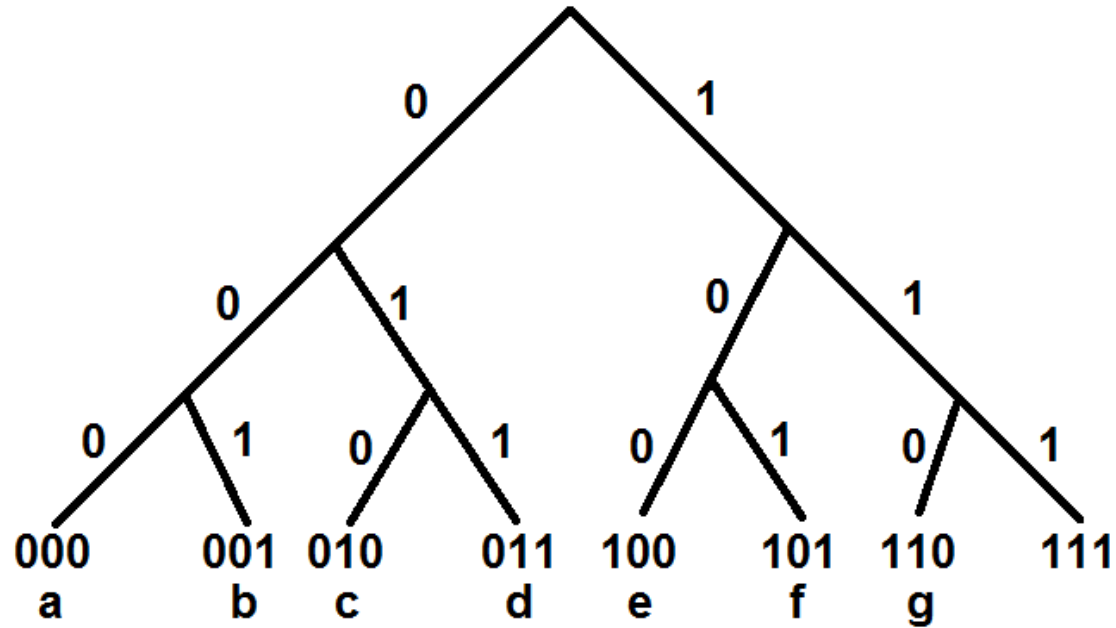
Data Compression - Huffman Code

- We now illustrate a special class of variable length code called Huffman code
- Consider a hypothetical alphabet set with, say, 7 letters which for convenience are denoted by a through g .
- Assume that there is a text in which these letters occur with frequencies given as follows.

Data Compression- Greedy Algorithm

Letters:	a	b	c	d	e	f	g
Frequency:	30	5	16	9	12	13	15
Fixed Length Code:	000	001	010	011	100	101	110

A Fixed Length Code



Data Compression- Greedy Algorithm

- Since there are 7 alphabets we need a minimum of 3-bit to encode them in binary.
- There are 8 possible strings of length 3.
- Construct a complete binary tree of depth 3 and associate a 0 for the left branch and 1 for the right branch

Data Compression- Greedy Algorithm

- Concatenate the strings along the path to the leafs- we get the eight binary strings attached to the eight leafs.
- Now can choose any subset of 7 out of these 8 strings. There are $\binom{8}{7} = \binom{8}{1} = 8$ ways.
- Associate the 7 alphabets with any chosen subset of 7 binary strings.
- One potential association is shown in the previous figure.

Greedy Algorithm Steps

1. Arrange the alphabets in the increasing order of frequency
 - Example: b:5, d:9, e:12, f:13, g:15, c:16, a:30
2. Merge the two letters with the two least frequencies to create a new alphabet which is the concatenation of these two letters and whose frequency is the sum of their frequencies
 - Example: bd:14, e:12, f:13, g:15, c:16, a:30
3. Arrange the alphabets in the increasing order of frequency
 - Example: e:12, f:13, bd:14, g:15, c:16, a:30

Greedy Algorithm Steps

4. Merge the two letters with the two least frequencies
 - Example: ef:25, bd:14, g:15, c:16, a:30
 5. Arrange the alphabets in the increasing order of frequency
 - Example: bd:14, g:15, c:16, ef:25, a:30
 6. Merge the two letters with the two least frequencies
 - Example: bdg:29, c:16, ef:25, a:30
 8. Arrange the alphabets in the increasing order of frequency
 - Example: c:16, ef:25, bdg:29, a:30
-

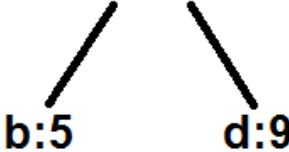
Greedy Algorithm Steps

8. Merge the two letters with the two least frequencies
 - Example: cef:41, bdg:29, a:30
 9. Arrange the alphabets in the increasing order of frequency
 - Example: bdg:29, a:30, cef:41
 10. Merge the two letters with the two least frequencies
 - Example: bdga:59, cef:41
 11. Arrange the alphabets in the increasing order of frequency and merge
 - Example: cefbdga:100
-

Pictorial Representation of the Merging Process


1) **b:5** **d:9** **e:12** **f:13** **g:15** **c:16** **a:30**

2) **e:12** **f:13** **bd:14** **g:15** **c:16** **a:30**



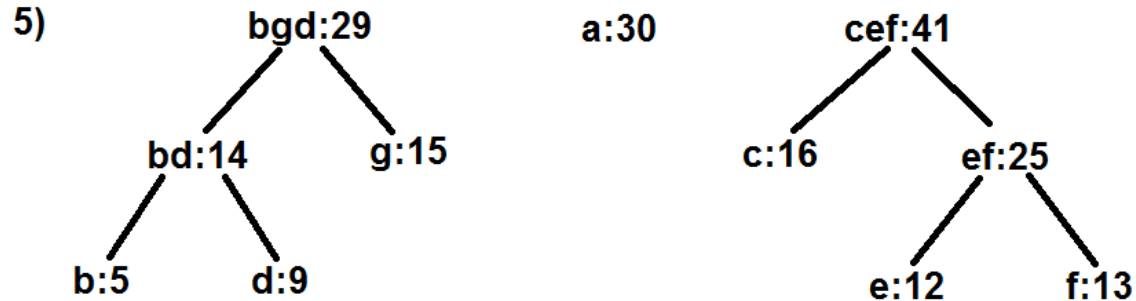
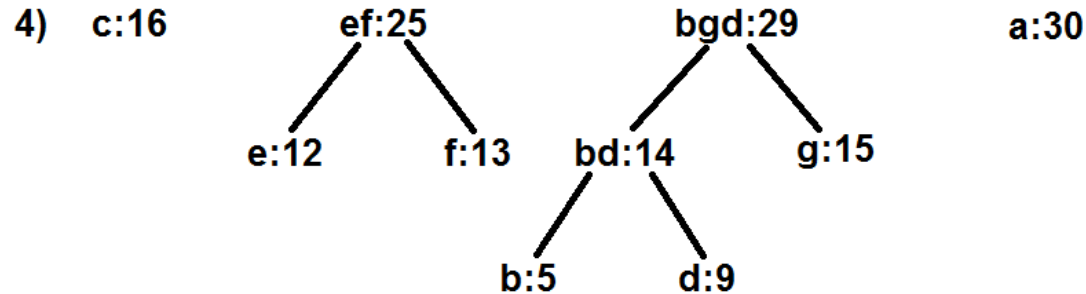
```
graph TD; bd14["bd:14"] --- b5["b:5"]; bd14 --- d9["d:9"]
```

3) **bd:14** **g:15** **c:16** **ef:25** **a:30**



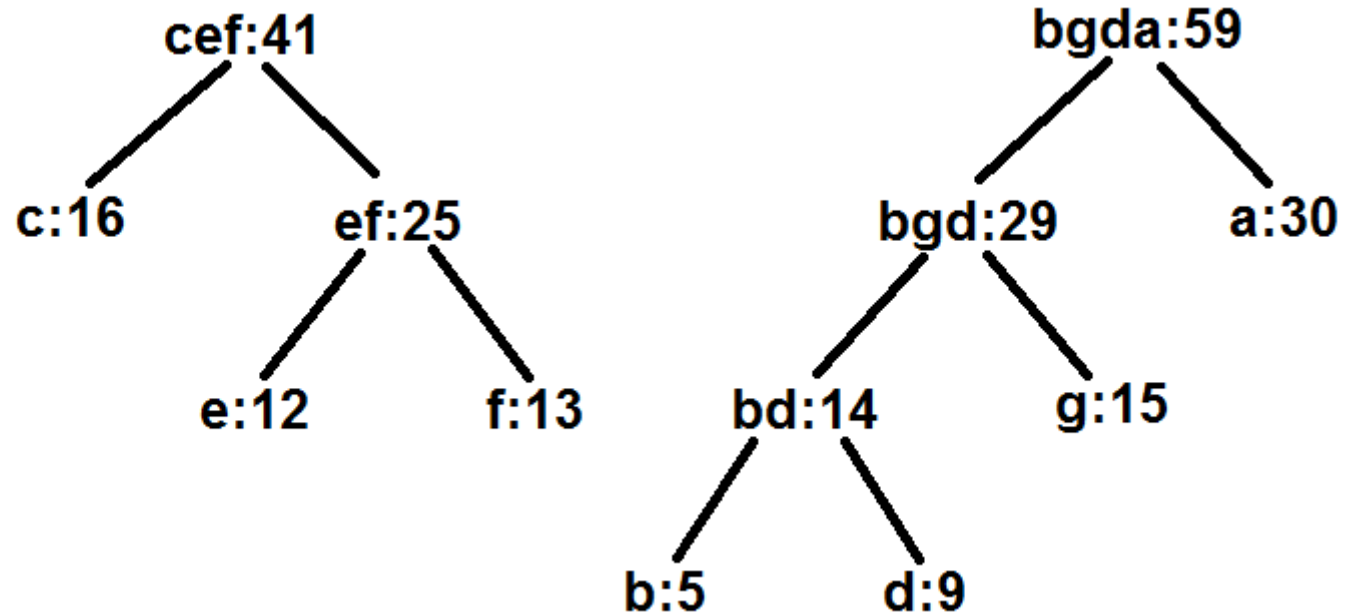
```
graph TD; bd14["bd:14"] --- b5["b:5"]; bd14 --- d9["d:9"]; ef25["ef:25"] --- e12["e:12"]; ef25 --- f13["f:13"]
```

Pictorial Representation of the Merging Process

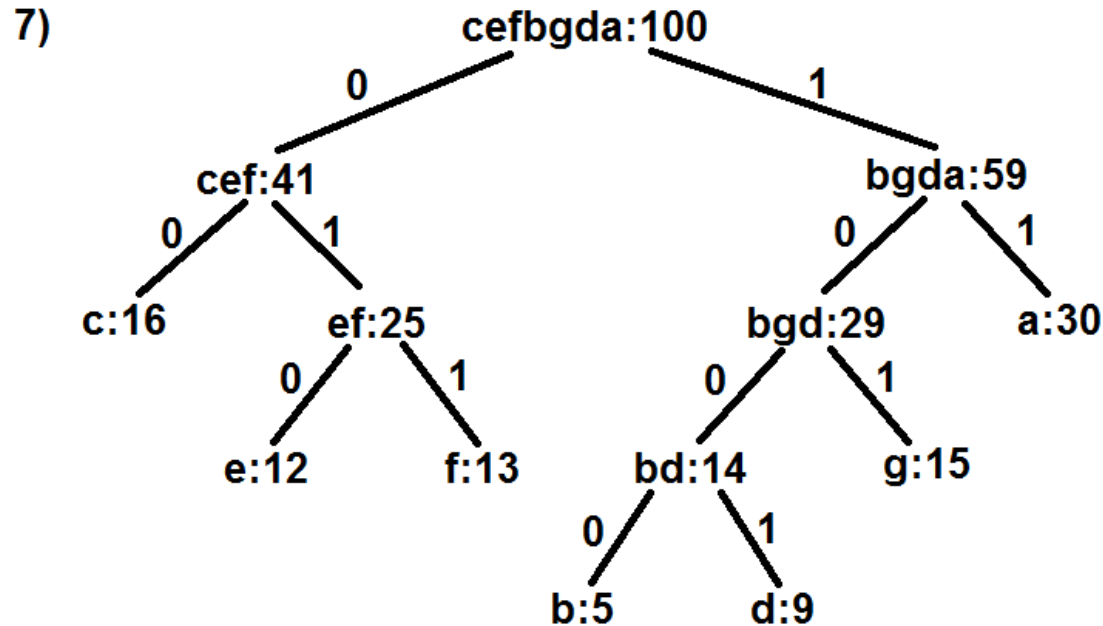


Pictorial Representation of the Merging Process

6)



Pictorial Representation of the Merging Process



Huffman Code

8. The variable length code is obtained by concatenating the bits along the root to the leafs. This resulting code is called the Huffman code

Letters:	a	b	c	d	e	f	g
Huffman Code:	11	1000	00	1001	010	011	101

Huffman Code

- The size of the binary file using fixed length code

$$\begin{aligned} L_f &= 3 * \text{sum of all the frequencies} \\ &= 3 * 100 * 100 = 3 * 10^5 \text{ bits} \end{aligned}$$

- The size of the binary file using variable length Huffman code

$$\begin{aligned} L_v &= 1000 \left[\begin{array}{l} \text{sum of the product of frequency of a letter} \\ \text{and the length of its Huffman code} \end{array} \right] \\ &= 1000[30(2) + 5(4) + 16(2) + 9(4) + 12(3) + 13(3) + 15(3)] \\ &= 2.68 * 10^5 \text{ bits} \end{aligned}$$

Huffman Code

- Compression Ratio:

$$\frac{L_f - L_v}{L_f} * 100 = \frac{(3 - 2.68)}{3} = 10.67\%$$

- I.e. we have achieved about 11% reduction in size

Huffman Code

- Note that if there is a larger difference in the frequencies, there will be a larger compression ratio
- If all the frequencies are the same, then we cannot achieve any compression using this scheme.

Huffman Code

- An important property of the Huffman code is that no code word is the prefix of any other code word
- In view of this, Huffman code is also called a prefix code to emphasize the fact that no code is a prefix of any other code word
- This prefix property is critical to unique decoding

Huffman Code

- Let the sender and receiver use the same tree as given above
- Suppose the sender wants to sent a message m =fade
- The encoder encodes m letter by letter using the Huffman code to get $\bar{m} = 011111001010$
- The receiver receives \bar{m} (assuming no error in transmission)

Huffman Code

- The decoder then scans \bar{m} from left to right, starting from the root, depending on 0 or 1 until a leaf is reached. Then it prints out that letter corresponding to the leaf and starts at the root until the whole of \bar{m} is scanned.

Huffman Code

- Thus,
 - 011 gives f
 - 11 gives a
 - 1001 gives d
 - 010 gives egiving back m=fade
- Note: It can be proven that Huffman code is optimal. For proof, refer to books listed in the reference

Homework

1. Generalize Huffman code using three alphabets- say $\{0, 1, 2\}$ and obtain the ternary Huffman code
2. Let the alphabets occur with frequencies that correspond to Fibonacci numbers:

a:1, b:1, c:2, d:3, e:5, f:8, g:13

Find the binary Huffman code. Generalize to n letters