Greedy Design Technique

Introduction

- How do we define a greedy algorithm?

Introduction

- How do we define a greedy algorithm?
  - Construct solution piece by piece.
  - Choose what is the best for the moment
  - Typically works in stages

Introduction

- How do we define a greedy algorithm?
  - Construct solution piece by piece.
  - Choose what is the best for the moment
  - Typically works in stages
    - a decision made in one stage can’t be change later
    - the choice must lead to the feasible solution

Introduction

- How do we define a greedy algorithm?
  - Construct solution piece by piece.
  - Choose what is the best for the moment
  - Typically works in stages
    - a decision made in one stage can’t be change later
    - the choice must lead to the feasible solution
      - ✓ expected to be an optimal solution

Introduction…

- When solving optimization problems, we are given a set of constraints, and an optimization function.

Introduction…

- When solving optimization problems, we are given a set of constraints, and an optimization function.
- Feasible solutions: ?

Introduction…

- When solving optimization problems, we are given a set of constraints, and an optimization function.
- Feasible solutions: solutions that satisfy the constraints.

Introduction…

- When solving optimization problems, we are given a set of constraints, and an optimization function.
- Feasible solutions: solutions that satisfy the constraints.
- Optimal solution: ?

Introduction…

- When solving optimization problems, we are given a set of constraints, and an optimization function.
- Feasible solutions: solutions that satisfy the constraints.
- Optimal solution: A feasible solution for which the optimization function has the best possible value.

Introduction…

- When solving optimization problems, we are given a set of constraints, and an optimization function.
- Feasible solutions: solutions that satisfy the constraints.
- Optimal solution: A feasible solution for which the optimization function has the best possible value.
- Greedy is way to construct a feasible solution for such optimization problems, and, sometimes, it leads to an optimal one.

The control abstraction

```
Algorithm Greedy(Type a[], int n)
solution = EMPTY
i=1;
for i = 1 to n
    do
        Type x = select(a);
        if feasible(solution, x)
            solution = solution U x;
    done
return solution
```

The Thirsty Baby problem

The Thirsty Baby problem

- A very thirsty, and intelligent, baby wants to quench her thirst. She has access to a glass of water, a carton of milk, cans of various juices, etc., a total of n different kinds of liquids.

The Thirsty Baby problem

- A very thirsty, and intelligent, baby wants to quench her thirst. She has access to a glass of water, a carton of milk, cans of various juices, etc., a total of n different kinds of liquids.
- Let a_i be the amount of ounces in which the i^{th} liquid is available.

The Thirsty Baby problem

- A very thirsty, and intelligent, baby wants to quench her thirst. She has access to a glass of water, a carton of milk, cans of various juices, etc., a total of n different kinds of liquids.
- Let a_i be the amount of ounces in which the i^{th} liquid is available.
- Based on her experience of taste and desire for nutrition, she also assigns certain satisfying factor, s_i, to the i^{th} liquid.

The Thirsty Baby problem

- A very thirsty, and intelligent, baby wants to quench her thirst. She has access to a glass of water, a carton of milk, cans of various juices, etc., a total of n different kinds of liquids.
- Let a_i be the amount of ounces in which the i^{th} liquid is available.
- Based on her experience of taste and desire for nutrition, she also assigns certain satisfying factor, s_i, to the i^{th} liquid.
- If the baby needs to drink t ounces of liquid, how much of each liquid should she drink?

The Thirsty Baby problem …

- Let $x_i, 1 \leq i \leq n$, be the amount of the i^{th} liquid the baby will drink.

The Thirsty Baby problem …

- Let x_i, $1 \leq i \leq n$, be the amount of the i^{th} liquid the baby will drink.
- The solution for this thirsty baby problem is obtained by finding real numbers x_i, $1 \leq i \leq n$, that maximize $\sum_{i=1}^{n} s_i x_i$

The Thirsty Baby problem …

- Let x_i, $1 \leq i \leq n$, be the amount of the i^{th} liquid the baby will drink.
- The solution for this thirsty baby problem is obtained by finding real numbers x_i, $1 \leq i \leq n$, that maximize $\sum_{i=1}^{n} s_i x_i$
- Subject constraint: $\sum_{i=1}^{n} x_i = t$ and for all $1 \leq i \leq n$, $o \leq x_i \leq a_i$

The Thirsty Baby problem …

- Let x_i, $1 \leq i \leq n$, be the amount of the i^{th} liquid the baby will drink.
- The solution for this thirsty baby problem is obtained by finding real numbers x_i, $1 \leq i \leq n$, that maximize $\sum_{i=1}^{n} s_i x_i$
- Subject constraint: $\sum_{i=1}^{n} x_i = t$ and for all $1 \leq i \leq n$, $o \leq x_i \leq a_i$
- We notice that if $\sum_{i=1}^{n} a_i < t$, then this instance is not solvable.

The Thirsty Baby problem …

- Input: n, t, s_i, a_i; $1 \leq i \leq n$; n is integer

The Thirsty Baby problem …

- Input: $n, t, s_i, a_i; \quad 1 \leq i \leq n; \quad n$ is integer
- Output: Real no $x_i; \;\; 1 \leq i \leq n$

  Such that $\sum_{i=1}^{n} s_i x_i$ is maximum optimization function

The Thirsty Baby problem …

- Input: $n, t, s_i, a_i; \quad 1 \le i \le n; \quad n$ is integer
- Output: Real no $x_i; \ \ 1 \le i \le n$

  Such that $\sum_{i=1}^{n} s_i x_i$ is maximum optimization function.
- Every set of x_i that satisfies the constraints is a feasible solution, and it is optimal if it further maximizes $\sum_{i=1}^{n} s_i x_i$

The Thirsty Baby problem …

- How should we feed her?

The Thirsty Baby problem …

- How should we feed her?
  - If Baby is Thirsty,
  - Baby picks up and drink a liquid X she likes the most.
  - If liquid X is over, she picks up and drink the next liquid she likes the most.

The Thirsty Baby problem …

- How should we feed her?
  - If Baby is Thirsty,
  - Baby picks up and drink a liquid X she likes the most.
  - If liquid X is over, she picks up and drink the next liquid she likes the most.
- Any intelligent decisions?
  - Myopically (Short-sightedness)

The Thirsty Baby problem …

- How should we feed her?
  - If Baby is Thirsty,
  - Baby picks up and drink a liquid X she likes the most.
  - If liquid X is over, she picks up and drink the next liquid she likes the most.
- Any intelligent decisions?
  - Myopically (Short-sightedness)
- Any irrevocability?

Container Loading problem

Container Loading problem

- A large ship is to be loaded with containers of cargos. Different containers, although of equal size, will have different weights.

Container Loading problem

- A large ship is to be loaded with containers of cargos. Different containers, although of equal size, will have different weights.
- Let w_i be the weight of the i^{th}container, $1 \leq i \leq n$, and the capacity of the ship is c, we want to find out a way to load the ship with the maximum number of containers, without tipping over the ship.

Container Loading problem

- A large ship is to be loaded with containers of cargos. Different containers, although of equal size, will have different weights.
- Let w_i be the weight of the i^{th} container, $1 \leq i \leq n$, and the capacity of the ship is c, we want to find out a way to load the ship with the maximum number of containers, without tipping over the ship.
- Let $x_i \in \{0, 1\}$. If $x_i = 1$, we will load the i^{th} container, otherwise, we will not load it.

Container Loading problem

- A large ship is to be loaded with containers of cargos. Different containers, although of equal size, will have different weights.
- Let w_i be the weight of the i^{th}container, $1 \leq i \leq n$, and the capacity of the ship is c, we want to find out a way to load the ship with the maximum number of containers, without tipping over the ship.
- Let $x_i \in \{0, 1\}$. If $x_i = 1$, we will load the i^{th} container, otherwise, we will not load it.
- We wish to assign values to x_i's such that $\sum_{i=1}^{n} w_i \leq c$, and $\sum_{i=1}^{n} x_i$ is maximized.

Knapsack problem

Fractional Knapsack Problem

- A thief robbing a store finds n items.

Fractional Knapsack Problem

- A thief robbing a store finds n items.
- The i^{th} item is worth v_i dollars or p_i profit and weighs w_i pounds, where v_i, p_i and w_i are integers.

Fractional Knapsack Problem

- A thief robbing a store finds n items.
- The i^{th} item is worth v_i dollars or p_i profit and weighs w_i pounds, where v_i, p_i and w_i are integers.
- The thief wants to take as valuable a load as possible, but he can carry at most W pounds in his knapsack, for some integer W.

Fractional Knapsack Problem

- A thief robbing a store finds n items.
- The i^{th} item is worth v_i dollars or p_i profit and weighs w_i pounds, where v_i, p_i and w_i are integers.
- The thief wants to take as valuable a load as possible, but he can carry at most W pounds in his knapsack, for some integer W.
- Which items should he take?

Fractional Knapsack Problem

- Some greedy strategy where the items are arranged in some order which is based on some greedy criterion.

Fractional Knapsack Problem

- Some greedy strategy where the items are arranged in some order which is based on some greedy criterion.
- **Greedy Criterion I :**
  - In this criterion the items are arranged by their values or profit. Here the item with maximum value or profit is selected first, and process continue till the minimum value.

Fractional Knapsack Problem

- Some greedy strategy where the items are arranged in some order which is based on some greedy criterion.
- **Greedy Criterion I :**
  - In this criterion the items are arranged by their values or profit. Here the item with maximum value or profit is selected first, and process continue till the minimum value.
  - Illustration :
    - $W = 15$
    - $(p_1, p_2, p_3, p_4, p_5, p_6, p_7) = (10, 5, 15, 7, 6, 18, 3)$
    - $(w_1, w_2, w_3, w_4, w_5, w_6, w_7) = (2, 3, 5, 7, 1, 4, 1)$

Fractional Knapsack Problem

- Some greedy strategy where the items are arranged in some order which is based on some greedy criterion.
- **Greedy Criterion I :**
  - In this criterion the items are arranged by their values or profit. Here the item with maximum value or profit is selected first, and process continue till the minimum value.
  - Illustration :
    - $W = 15$
    - $(p_1, p_2, p_3, p_4, p_5, p_6, p_7) = (10, 5, 15, 7, 6, 18, 3)$
    - $(w_1, w_2, w_3, w_4, w_5, w_6, w_7) = (2, 3, 5, 7, 1, 4, 1)$
    - The solution set is $\left(1, 0, 1, \frac{4}{7}, 0, 1, 0\right)$ and profit $= 47$ units

Fractional Knapsack Problem

- **Greedy Criterion II :**
  - In this criterion the items are arranged by their weights. Here the item with lightest weight is selected first, and process continue till the maximum value.

Fractional Knapsack Problem

- **Greedy Criterion II :**
    - In this criterion the items are arranged by their weights. Here the item with lightest weight is selected first, and process continue till the maximum value.
        - Illustration :
            - $W = 15$
            - $(p_1, p_2, p_3, p_4, p_5, p_6, p_7) = (10, 5, 15, 7, 6, 18, 3)$
            - $(w_1, w_2, w_3, w_4, w_5, w_6, w_7) = (2, 3, 5, 7, 1, 4, 1)$

Fractional Knapsack Problem

- **Greedy Criterion II :**
  - In this criterion the items are arranged by their weights. Here the item with lightest weight is selected first, and process continue till the maximum value.
    - Illustration :
      - $W = 15$
      - $(p_1, p_2, p_3, p_4, p_5, p_6, p_7) = (10, 5, 15, 7, 6, 18, 3)$
      - $(w_1, w_2, w_3, w_4, w_5, w_6, w_7) = (2, 3, 5, 7, 1, 4, 1)$
      - The solution set is $\left(1, 1, \frac{4}{5}, 0, 1, 1, 1\right)$ and profit $= 54$ units

Fractional Knapsack Problem …

- **Greedy Criterion III :**
  - In this criterion the items are arranged by certain ratio, P_i, where P_i is the ratio of value over weight. Here selection proceeds from maximum ratio to minimum ratio.

Fractional Knapsack Problem …

- **Greedy Criterion III :**
  - In this criterion the items are arranged by certain ratio, P_i, where P_i is the ratio of value over weight. Here selection proceeds from maximum ratio to minimum ratio.
    - Illustration :
      - $W = 15$
      - $(p_1, p_2, p_3, p_4, p_5, p_6, p_7) = (10, 5, 15, 7, 6, 18, 3)$
      - $(w_1, w_2, w_3, w_4, w_5, w_6, w_7) = (2, 3, 5, 7, 1, 4, 1)$

Fractional Knapsack Problem …

- **Greedy Criterion III :**
  - In this criterion the items are arranged by certain ratio, P_i, where P_i is the ratio of value over weight. Here selection proceeds from maximum ratio to minimum ratio.
    - Illustration :
      - $W = 15$
      - $(p_1, p_2, p_3, p_4, p_5, p_6, p_7) = (10, 5, 15, 7, 6, 18, 3)$
      - $(w_1, w_2, w_3, w_4, w_5, w_6, w_7) = (2, 3, 5, 7, 1, 4, 1)$
      - $\left(\frac{p_1}{w_1}, \frac{p_2}{w_2}, \frac{p_3}{w_3}, \frac{p_4}{w_4}, \frac{p_5}{w_5}, \frac{p_6}{w_6}, \frac{p_7}{w_7}\right) = (5,\ 1.66,\ 3,\ 1,\ 6,\ 4.5,\ 3)$

Fractional Knapsack Problem …

- **Greedy Criterion III :**
  - In this criterion the items are arranged by certain ratio, P_i, where P_i is the ratio of value over weight. Here selection proceeds from maximum ratio to minimum ratio.
    - Illustration :
      - $W = 15$
      - $(p_1, p_2, p_3, p_4, p_5, p_6, p_7) = (10, 5, 15, 7, 6, 18, 3)$
      - $(w_1, w_2, w_3, w_4, w_5, w_6, w_7) = (2, 3, 5, 7, 1, 4, 1)$
      - $\left(\frac{p_1}{w_1}, \frac{p_2}{w_2}, \frac{p_3}{w_3}, \frac{p_4}{w_4}, \frac{p_5}{w_5}, \frac{p_6}{w_6}, \frac{p_7}{w_7}\right) = (5,\ 1.66,\ 3,\ 1,\ 6,\ 4.5,\ 3)$
      - The solution set is $\left(1,\ \frac{2}{3}, 1, 0, 1, 1, 1\right)$ and profit $= 55.33$ units

Communication and compression

Communication and compression

- Messages in English/Hindi/Gujarati/… are transmitted between computers in binary.

Communication and compression

- Messages in English/Hindi/Gujarati/… are transmitted between computers in binary.
- Encode letters $\{a, b, \ldots, z\}$ as strings over $\{0,1\}$.

Communication and compression

- Messages in English/Hindi/Gujarati/… are transmitted between computers in binary.
- Encode letters $\{a, b, \dots, z\}$ as strings over $\{0,1\}$
  - 26 letters, $2^5 = 32$, use strings of length 5?

Communication and compression

- Messages in English/Hindi/Gujarati/… are transmitted between computers in binary.
- Encode letters $\{a, b, \dots, z\}$ as strings over $\{0,1\}$
  - 26 letters, $2^5 = 32$, use strings of length 5?
- Suppose we have a $100,000$ - character data file that we wish to store compactly. We observe that the characters in the file occur with the frequencies given by Figure. That is, only 6 different characters appear, and the character a occurs $45,000$ times.

Communication and compression

- Messages in English/Hindi/Gujarati/… are transmitted between computers in binary.
- Encode letters $\{a, b, \ldots, z\}$ as strings over $\{0,1\}$
  - 26 letters, $2^5 = 32$, use strings of length 5?
- Suppose we have a $100{,}000$ - character data file that we wish to store compactly. We observe that the characters in the file occur with the frequencies given by Figure. That is, only 6 different characters a

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5

Communication and compression …

- We have many options for how to represent such a file of information. Here, we consider the problem of designing a ***binary character code*** in which each character is represented by a unique binary string, which we call a ***code word***.

Communication and compression …

- We have many options for how to represent such a file of information. Here, we consider the problem of designing a ***binary character code*** in which each character is represented by a unique binary string, which we call a ***code word***.
- If we use a ***fixed-length code***, we need 3 bits to represent 6 characters: $a = 000, b = 001, \ldots, f = 101$.

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101

Communication and compression …

- We have many options for how to represent such a file of information. Here, we consider the problem of designing a ***binary character code*** in which each character is represented by a unique binary string, which we call a ***code word***.
- If we use a ***fixed-length code***, we need 3 bits to represent 6 characters: $a = 000, b = 001, \ldots, f = 101$.
- This method requires $300{,}000$ bits to code the entire file.

Communication and compression …

- Can we optimize the amount of data to transfer?
  - Can we do better?
    - Use shorter strings for more frequent letters?

Communication and compression …

- Variable length encoding

Communication and compression …

- Variable length encoding
  - Morse code

Communication and compression …

- Variable length encoding
  - Morse code
    - Encode letters using dots (0) and dashes (1)

Communication and compression …

- Variable length encoding
    - Morse code
        - Encode letters using dots (0) and dashes (1)
        - Encoding of e is 0, t is 1, a is 01

Communication and compression …

- Variable length encoding
  - Morse code
    - Encode letters using dots (0) and dashes (1)
    - Encoding of e is 0, t is 1, a is 01
    - Decode 0101 ?

Communication and compression …

- Variable length encoding
  - Morse code
    - Encode letters using dots (0) and dashes (1)
    - Encoding of e is 0, t is 1, a is 01
    - Decode 0101 — $etet, aa, eta, aet$?

Communication and compression …

- Variable length encoding
  - Morse code
    - Encode letters using dots (0) and dashes (1)
    - Encoding of e is 0, t is 1, a is 01
    - Decode 0101 — $etet, aa, eta, aet$?
    - Use pauses between letters to distinguish
      - Like an extra symbol in encoding

Communication and compression …

- Prefix code

Communication and compression …

- Prefix code
  - We consider here only codes in which no code word is also a prefix of some other code word. Such codes are called ***prefix codes***.

Communication and compression …

- Prefix code
  - We consider here only codes in which no code word is also a prefix of some other code word. Such codes are called ***prefix codes***.

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Communication and compression …

- Prefix code
  - We consider here only codes in which no code word is also a prefix of some other code word. Such codes are called ***prefix codes***.

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

  - here the 1-bit string 0 represents a, and the 4-bit string 1100 represents f. This code requires $(45 \times 1 \ + 13 \times 3 \ + 12 \times 3 + 16 \times 3 + 9 \times 4 + 5 \times 4) \times 1000 = 224{,}000$ bits.

Communication and compression …

- Prefix code
  - prefix code can always achieve the optimal data compression among any character code, and so we suffer no loss of generality by restricting our attention to prefix codes.

Communication and compression …

- Prefix code
  - prefix code can always achieve the optimal data compression among any character code, and so we suffer no loss of generality by restricting our attention to prefix codes.
  - Encoding is always simple for any binary character code; we just concatenate the code words representing each character of the file.

Communication and compression …

- Prefix code
  - prefix code can always achieve the optimal data compression among any character code, and so we suffer no loss of generality by restricting our attention to prefix codes.
  - Encoding is always simple for any binary character code; we just concatenate the code words representing each character of the file.
  - For example, we code the 3-character file abc as $0.101.100 = 0101100$, where “ . ” denotes concatenation.

Communication and compression …

- Prefix code
  - prefix code can always achieve the optimal data compression among any character code, and so we suffer no loss of generality by restricting our attention to prefix codes.
  - Encoding is always simple for any binary character code; we just concatenate the code words representing each character of the file.
  - For example, we code the 3-character file abc as $0.101.100 = 0101100$, where " . " denotes concatenation.
  - Prefix codes are desirable because they simplify decoding. Since no code word is a prefix of any other, the code word that begins an encoded file is unambiguous.

Communication and compression …

- Prefix code
  - prefix code can always achieve the optimal data compression among any character code, and so we suffer no loss of generality by restricting our attention to prefix codes.
  - Encoding is always simple for any binary character code; we just concatenate the code words representing each character of the file.
  - For example, we code the 3-character file abc as $0.101.100 = 0101100$, where “ . ” denotes concatenation.
  - Prefix codes are desirable because they simplify decoding. Since no code word is a prefix of any other, the code word that begins an encoded file is unambiguous.
  - We can simply identify the initial code word, translate it back to the original character and repeat the decoding process on the remainder of the encoded file. In our example, the string 001011101 parses uniquely as $0.0.101.1101$, which decodes to $aabe$.

Communication and compression …

- Prefix code
  - prefix code can always achieve the optimal data compression among any character code, and so we suffer no loss of generality by restricting our attention to prefix codes.
  - Encoding is always simple for any binary character code; we just concatenate the code words representing each character of the file.
  - For example, we code the 3-character file abc as $0.101.100 = 0101100$, where " . " denotes concatenation.
  - Prefix codes are desirable because they simplify decoding. Since no code word is a prefix of any other, the code word that begins an encoded file is unambiguous.
  - We can simply identify the initial code word, translate it back to the original character and repeat the decoding process on the remainder of the encoded file. In our example, the string 001011101 parses uniquely as $0.0.101.1101$, which decodes to $aabe$.

Communication and compression …

- Prefix code
  - The decoding process needs a convenient representation for the prefix code so that we can easily pick off the initial code word. A binary tree whose leaves are the given characters provides one such representation. We interpret the binary code word for a character as the simple path from the root to that character, where 0 means “go to the left child” and 1 means “go to the right child.” Figure shows the trees for the two codes of our example.

Communication and compression …

- Prefix code

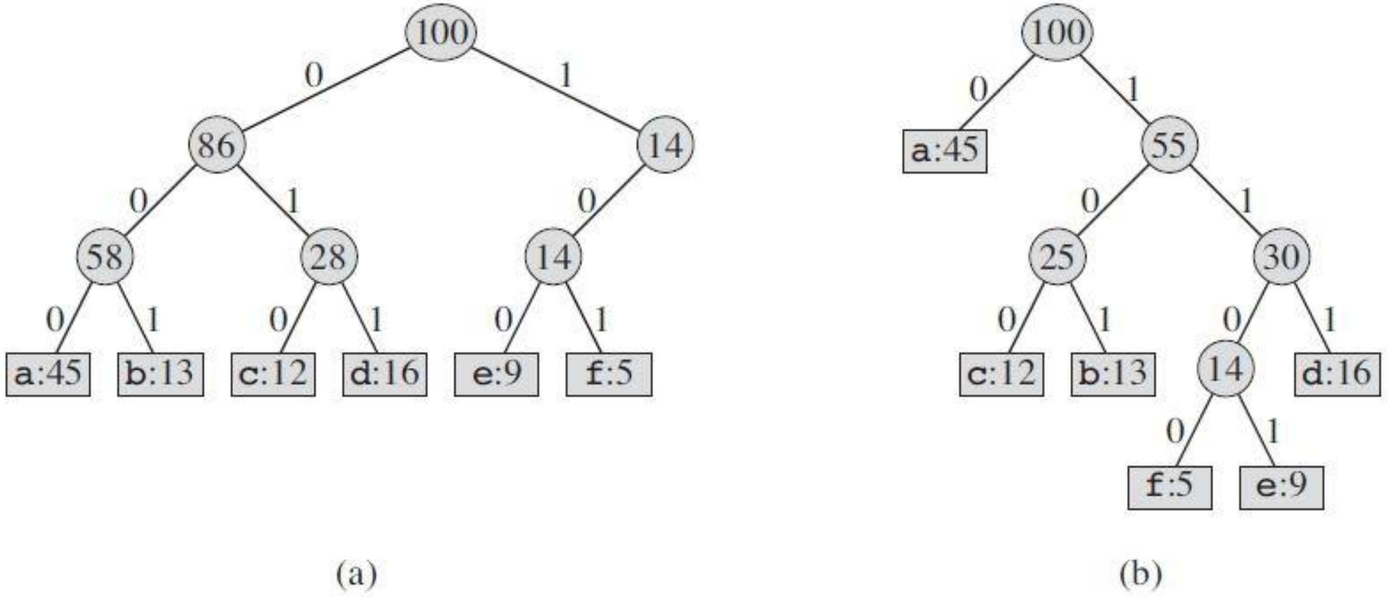


Figure Trees corresponding to the coding schemes in Figure . Each leaf is labeled with a character and its frequency of occurrence. Each internal node is labeled with the sum of the frequencies of the leaves in its subtree. **(a)** The tree corresponding to the fixed-length code a = 000, . . . , f = 101. **(b)** The tree corresponding to the optimal prefix code a = 0, b = 101, . . . , f = 1100.

Communication and compression …

- Prefix code
  - Given a tree T corresponding to a prefix code, we can easily compute the number of bits required to encode a file. For each character c in the alphabet C, let the attribute $c.freq$ denote the frequency of c in the file and let $d_T(c)$ denote the depth of c's leaf in the tree. Note that $d_T(c)$ is also the length of the code word for character c. The number of bits required to encode a file is thus

  $B(T) = \sum_{c \epsilon C} c.freq \, . d_T(c)$, which we define as the cost of tree T.

  - But, How to construct optimal prefix code?

Communication and compression …

- Huffman code

Communication and compression …

- Huffman code
  - Huffman invented a greedy algorithm that constructs an optimal prefix code called a ***Huffman code***.

Communication and compression …

- Huffman code
  - In the pseudo code that follows, we assume that C is a set of n characters and that each character $c \epsilon C$ is an object with an attribute $c.freq$ giving its frequency. The algorithm builds the tree T corresponding to the optimal code in a bottom-up manner. It begins with a set of $|C|$ leaves and performs a sequence of $|C| - 1$ “merging” operations to create the final tree. The algorithm uses a min-priority queue Q, keyed on the $freq$ attribute, to identify the two least-frequent objects to merge together. When we merge two objects, the result is a new object whose frequency is the sum of the frequencies of the two objects that were merged.

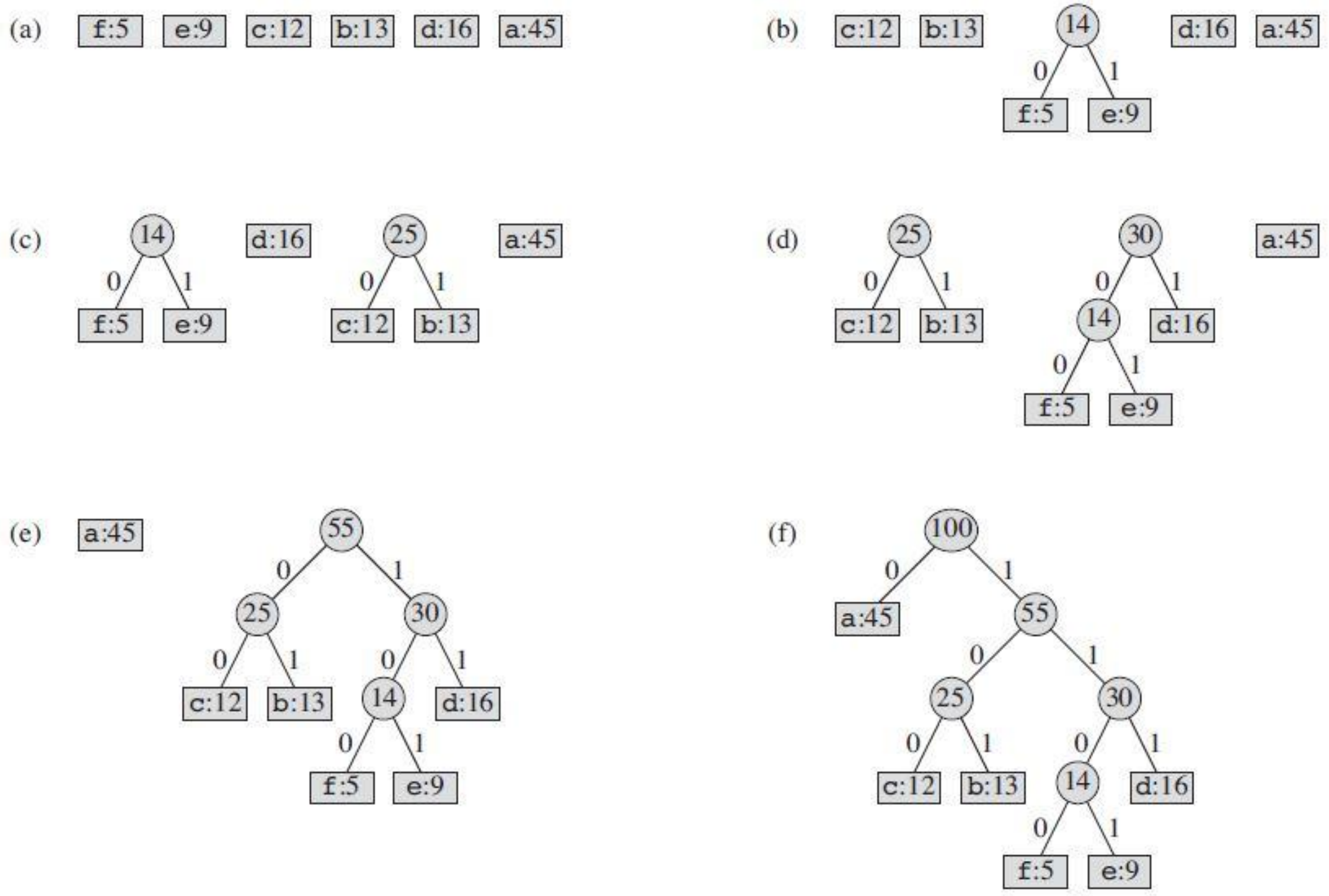
Communication and compression …

- Huffman code

```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n − 1
    allocate a new node z
    z.left = x = EXTRACT-MIN(Q)
    z.right = y = EXTRACT-MIN(Q)
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)      // return the root of the tree
```

Communication and compression …

- Huffman code



Communication and compression …

- Huffman's Algorithm Analysis
  - we assume that Q is implemented as a binary min-heap.
  - For a set C of n characters, we can initialize Q in line 2 in $O(n)$ time using the BUILD-MIN-HEAP procedure.
  - The for loop in lines $3-8$ executes exactly $n-1$ times, and since each heap operation requires time $O(\lg n)$, the loop contributes $O(n \lg n)$ to the running time.
  - Thus, the total running time of HUFFMAN on a set of n characters is $O(n \lg n)$.