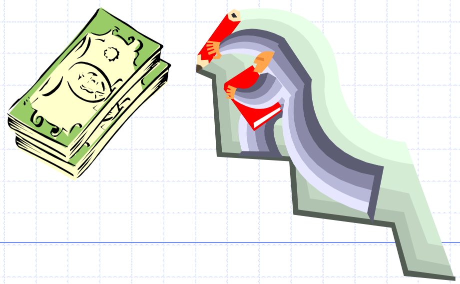
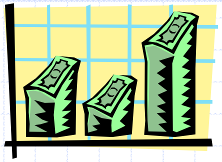
The Greedy Method and Text Compression



The Greedy Method Technique



- **The greedy method** is a general algorithm design paradigm, built on the following elements:
  - **configurations**: different choices, collections, or values to find
  - **objective function**: a score assigned to configurations, which we want to either maximize or minimize
- It works best when applied to problems with the **greedy-choice** property:
  - a globally-optimal solution can always be found by a series of local improvements from a starting configuration.

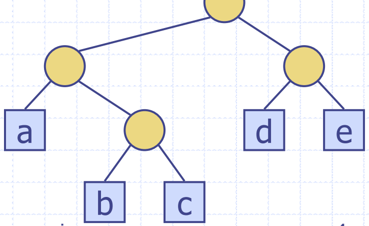
Text Compression

- Given a string X, efficiently encode X into a smaller string Y
  - Saves memory and/or bandwidth
- A good approach: **Huffman encoding**
  - Compute frequency f(c) for each character c.
  - Encode high-frequency characters with short code words
  - No code word is a prefix for another code
  - Use an optimal encoding tree to determine the code words

Encoding Tree Example

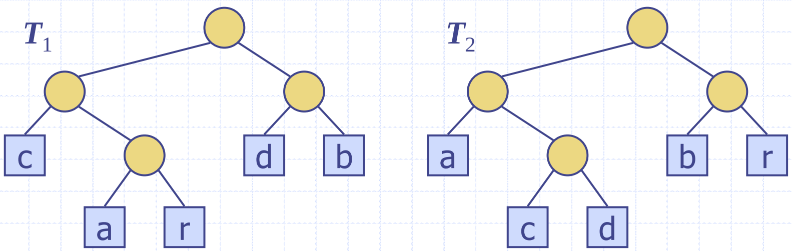
- A **code** is a mapping of each character of an alphabet to a binary code-word
- A **prefix code** is a binary code such that no code-word is the prefix of another code-word
- An **encoding tree** represents a prefix code
  - Each external node stores a character
  - The code word of a character is given by the path from the root to the external node storing the character (0 for a left child and 1 for a right child)

00	010	011	10	11
a	b	c	d	e



Encoding Tree Optimization

- Given a text string X, we want to find a prefix code for the characters of X that yields a small encoding for X
  - Frequent characters should have long code-words
  - Rare characters should have short code-words
- Example
  - X = abracadabra
  - T_1 encodes X into 29 bits
  - T_2 encodes X into 24 bits



Huffman's Algorithm

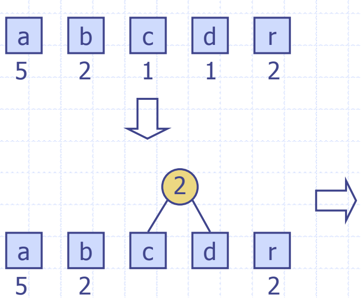
- Given a string X, Huffman's algorithm construct a prefix code the minimizes the size of the encoding of X
- It runs in time $O(n + d \log d)$, where n is the size of X and d is the number of distinct characters of X
- A heap-based priority queue is used as an auxiliary structure

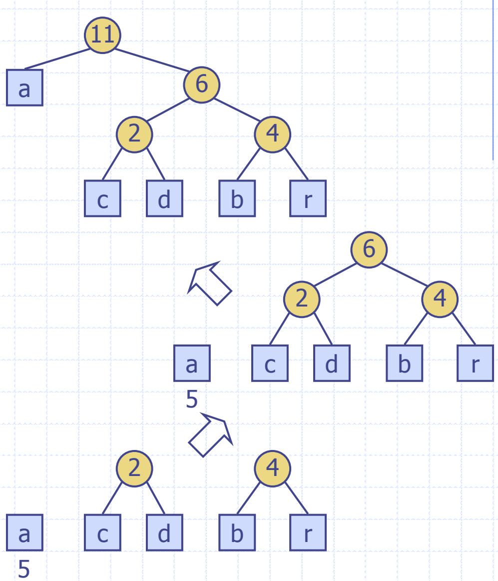
Algorithm *HuffmanEncoding*(X)
  Input string X of size n
  Output optimal encoding trie for X
  $C \leftarrow$ *distinctCharacters*(X)
  computeFrequencies(C, X)
  $Q \leftarrow$ new empty heap
  for all $c \in C$
    $T \leftarrow$ new single-node tree storing c
    Q.*insert*(*getFrequency*(c), T)
  while Q.*size*() > 1
    $f_1 \leftarrow Q$.*minKey*()
    $T_1 \leftarrow Q$.*removeMin*()
    $f_2 \leftarrow Q$.*minKey*()
    $T_2 \leftarrow Q$.*removeMin*()
    $T \leftarrow$ *join*(T_1, T_2)
    Q.*insert*($f_1 + f_2$, T)
  return Q.*removeMin*()

Example

X = abracadabra
Frequencies

a	b	c	d	r
5	2	1	1	2

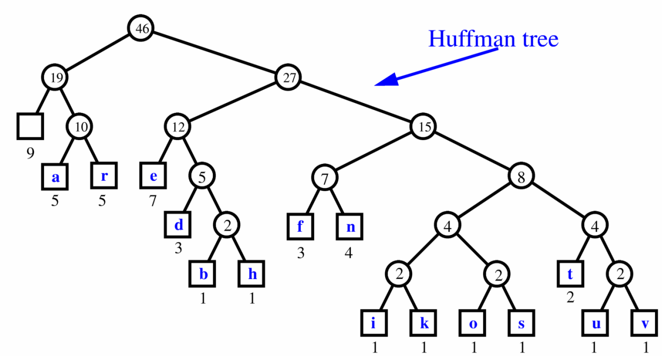




Extended Huffman Tree Example

String: **a fast runner need never be afraid of the dark**

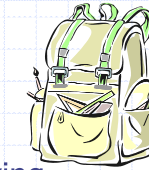
Character		a	b	d	e	f	h	i	k	n	o	r	s	t	u	v
Frequency	9	5	1	3	7	3	1	1	1	4	1	5	1	2	1	1



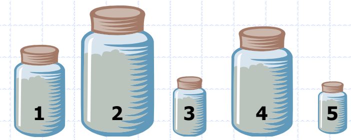
The Fractional Knapsack Problem (not in book)

- Given: A set S of n items, with each item i having
  - b_i - a positive benefit
  - w_i - a positive weight
- Goal: Choose items with maximum total benefit but with weight at most W.
- If we are allowed to take fractional amounts, then this is the **fractional knapsack problem**.
  - In this case, we let x_i denote the amount we take of item i
  - Objective: maximize $$\sum_{i \in S} b_i (x_i / w_i)$$
  - Constraint: $$\sum_{i \in S} x_i \le W$$

Example



- Given: A set S of n items, with each item i having
  - b_i - a positive benefit
  - w_i - a positive weight
- Goal: Choose items with maximum total benefit but with weight at most W.



Items:	1	2	3	4	5
Weight:	4 ml	8 ml	2 ml	6 ml	1 ml
Benefit:	\$12	\$32	\$40	\$30	\$50
Value: ($ per ml)	3	4	20	5	50

"knapsack"

10 ml

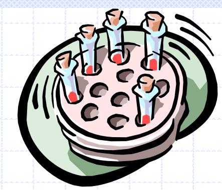
Solution:
- 1 ml of 5
- 2 ml of 3
- 6 ml of 4
- 1 ml of 2

The Fractional Knapsack Algorithm

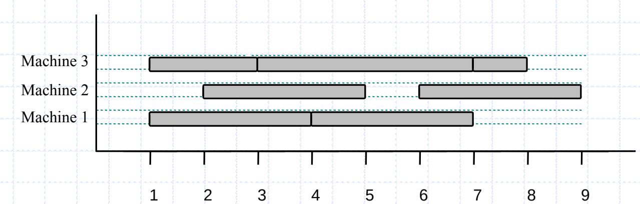
- Greedy choice: Keep taking item with highest value (benefit to weight ratio)
  - Since $\sum_{i \in S} b_i (x_i / w_i) = \sum_{i \in S} (b_i / w_i) x_i$
  - Run time: O(n log n). Why?
- Correctness: Suppose there is a better solution
  - there is an item i with higher value than a chosen item j, but $x_i<w_i$, $x_j>0$ and $v_i<v_j$
  - If we substitute some i with j, we get a better solution
  - How much of i: $\min\{w_i-x_i, x_j\}$
  - Thus, there is no better solution than the greedy one

```
Algorithm fractionalKnapsack(S, W)
  Input: set S of items w/ benefit b_i
      and weight w_i; max. weight W
  Output: amount x_i of each item i
      to maximize benefit w/ weight
      at most W
  for each item i in S
      x_i ← 0
      v_i ← b_i / w_i          {value}
  w ← 0                        {total weight}
  while w < W
      remove item i w/ highest v_i
      x_i ← min{w_i , W - w}
      w ← w + min{w_i , W - w}
```


Task Scheduling (not in book)



- Given: a set T of n tasks, each having:
  - A start time, s_i
  - A finish time, f_i (where $s_i < f_i$)
- Goal: Perform all the tasks using a minimum number of "machines."



Task Scheduling Algorithm

- Greedy choice: consider tasks by their start time and use as few machines as possible with this order.
  - Run time: O(n log n). Why?
- Correctness: Suppose there is a better schedule.
  - We can use k-1 machines
  - The algorithm uses k
  - Let i be first task scheduled on machine k
  - Machine i must conflict with k-1 other tasks
  - But that means there is no non-conflicting schedule using k-1 machines

Algorithm *taskSchedule*(T)
  Input: set T of tasks w/ start time s_i and finish time f_i
  Output: non-conflicting schedule with minimum number of machines
  $m \leftarrow 0$ {no. of machines}
  while *T is not empty*
    remove task i w/ smallest s_i
    if *there's a machine j for i* **then**
      schedule i on machine j
    else
      $m \leftarrow m + 1$
      schedule i on machine m

Example

- Given: a set T of n tasks, each having:
  - A start time, s_i
  - A finish time, f_i (where $s_i < f_i$)
  - [1,4], [1,3], [2,5], [3,7], [4,7], [6,9], [7,8] (ordered by start)
- Goal: Perform all tasks on min. number of machines