Greedy Algorithms

Alexandra Stefan

Greedy Method for Optimization Problems

- Greedy:
  - take the action that is best now (out of the current options)
  - it may cause you to miss the optimal solution
  - You build the solution as you go. No need to backtrace it.
- Examples:
  - Knapsack
    - Pic the item with the largest value/weight ratio.
      - Other possible measures: largest value, smallest weight.
  - Interval scheduling
    - Pick the interval that finishes first
  - Huffman codes
    - Prefix codes: no code is also a prefix of another code.
    - Huffman tree to decode (the code for a character, x, takes you to the leaf with x)
  - Edit distance
    - – greedy? – not clear.

Knapsack versions

- Unbounded (electronics warehouse)
  - Unlimited amounts of each item.
  - Cannot take fractions of an item.
- Fractional (market: flour, rice, wine)
  - Limited amounts (e.g. 3 of item A)
  - Can take a fraction of an item.
- What would a greedy thief do?
  - Method:
    - Sort in decreasing order of value/weight ratio.
    - Pick as many of the largest ratio as possible. After that, try to take as many of the next ratio as possible and so on.
  - Does NOT give an optimal solution. See next page.
- Would any of the above problems be solved optimally using Greedy? (Prove or give counter-example)
  - Yes. The fractional version.

Greedy for Knapsack

Item	A	B	C	D	E
Value	4	5	11	14	15
Weight	3	4	7	8	9
Ratio	4/3= 1.3	5/4= 1.25	11/7 =1.57	14/8 =1.75	15/9 =1.67
Reordered decreasing by ratio: **D, E, C, A, B**					

Unbounded

Picked	D	D	A
Remaining weight	13 (=21-8)	5 (=13-8)	2 (=5-3)
Value: **14+14+4 = 32**			

Unbounded, fractional

Picked	D	D	5/8 of D
Remaining weight	13 (=21-8)	5 (=13-8)	0 (=5-5)
Value: **14+14+(5/8)*14 = 36.75**			

0/1

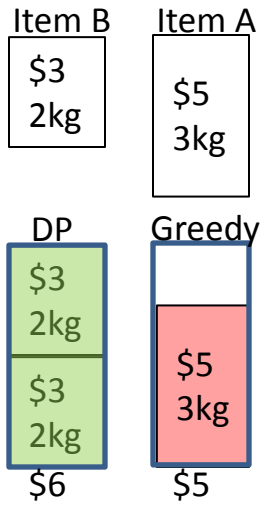
Picked	D	E	A
Remaining weight	13 (=21-8)	4 (=13-9)	1 (=4-3)
Value: **14+15+4 = 33**			

0/1 fractional

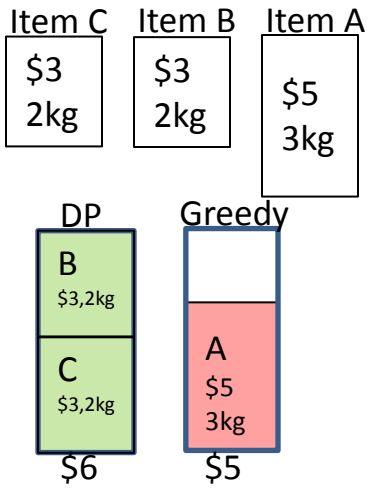
Picked	D	E	4/7 of C
Remaining weight	13 (=21-8)	4 (=13-9)	0 (=4-4)
Value: **14+15+(4/7)*11 = 35.28**			

Counter example used to prove that Greedy fails for Unbounded Knapsack

- Goal: construct an Unbounded Knapsack instance where Greedy does not give the optimal answer.
- Intuition: We want Greedy to pick only one item, when in fact two other items can be picked and together give a higher value:
  - Item B: weight 2 (can fit 2), value 3 => total value 6
  - Item A: weight 3, value 5 => total value 5
  - Knapsack max weight: 4
  - !!! You must double-check that Greedy would pick item A => check the ratios: 5/3 > 3/2 (1.66 > 1.5).
    - *If item A had value 4, Greedy would also have picked B.*



- Same example can be modified to work for 0/1 Knapsack:
  - Item A: 3kg, $5
  - Item B: 2kg, $3
  - Item C: 2kg, $3



Interval scheduling versions Worksheet

- With values (job = (start, end, value)):
  - Each interval has a value. Goal: maximize sum of values of picked non-overlapping jobs.
  - Greedy solution?
    - Algorithm
    - Is it optimal? Give proof or counter example.
- Without values (job = start, end)):
  - Pick the largest number of non-overlapping intervals.
  - "Activity selection problem" in CLRS (page 415)
  - Greedy solution?
    - Algorithm
    - Is it optimal? Give proof or counter example. (CLRS proof at page 418, proof of Theorem 16.1)

- Which of the two versions is more general?
  - Is one a special case (or special instance) of the other?
  - If you have a program to solve problems of one type, can you easily use it to solve problems of the other type? Which type should the program solve (with value, or without value)?

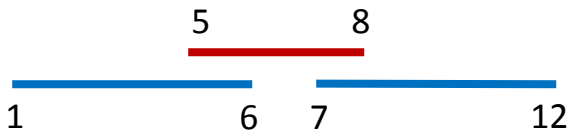
Interval scheduling Greedy measures

- Problem version: All jobs have the SAME value. => maximize number of jobs you can pick.

- Optimal:
  - job with that finishes first
  - job with latest start time

- NOT optimal:
  - Shortest duration
  - Least overlaps
  - Earliest start time

Example showing that Shortest duration
Does not give an optimal solution.



Greedy will pick the red job. Nothing else fits.
Optimal: the 2 blue jobs.

Huffman code

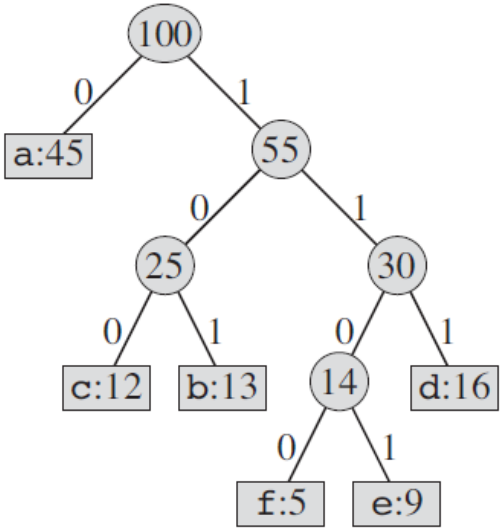
- Online resource (shows all steps): http://www.ida.liu.se/opendsa/OpenDSA/Books/OpenDSA/html/Huffman.html
- Application: file compression
- Example from CLRS:
  - File with 100,000 characters.
  - Characters: a,b,c,d,e,f
  - Frequency in thousands (e.g. the frequency of b is 13000):
  - Goal: binary encoding that requires less memory.

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

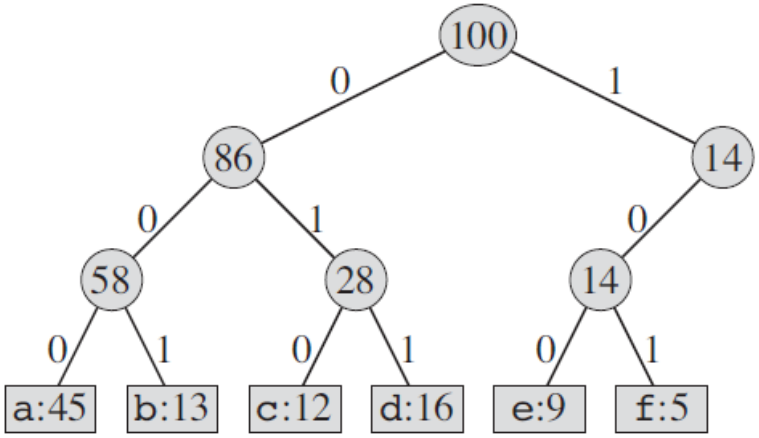
Huffman codes

- Internal nodes contain the sum of probabilities of the leaves in that subtree.

Optimal prefix codeword tree (Huffman tree) – optimal encoding



Fixed-length codeword tree



Compute the number of bits needed for the whole file using each of these encodings.

45,000*1 + 13,000 * 3 + 12,000*3 +
+16,000*3 + 9,000 * 4 + 5,000 * 4
= 224,000

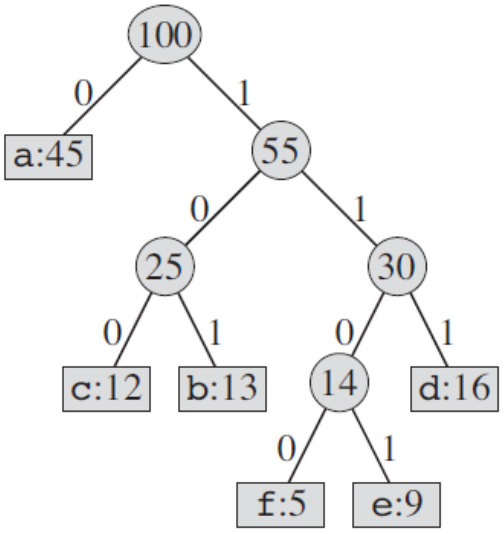
100,000 * 3 = 300,000

(Images from CLRS.)

Huffman codes

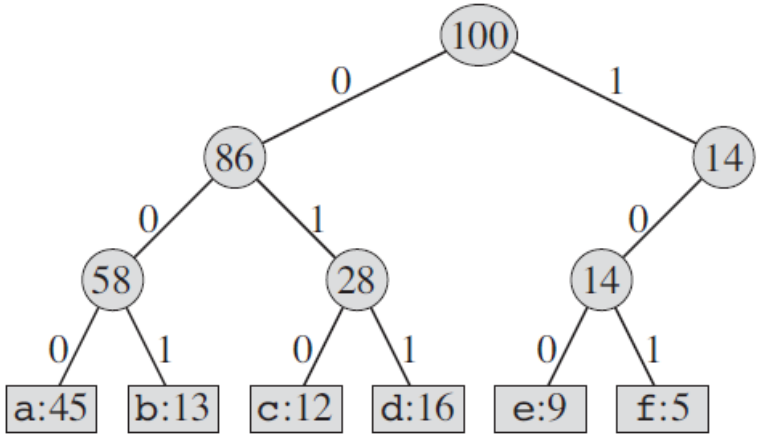
- Internal nodes contain the sum of probabilities of the leaves in that subtree.

Optimal prefix codeword tree (Huffman tree) – optimal encoding



a->0
b->101
c->100
d->111
e->1101
f->1100

Fixed-length codeword tree



Compute the number of bits needed for the whole file using each of these encodings.

Number of bits in code

45,000*1 + 13,000 * 3 + 12,000*3 + 16,000*3 + 9,000 * 4 + 5,000 * 4 = 224,000

100,000 * 3 = 300,000

(Images from CLRS.)

Building the Huffman Tree

1. Make ‘leaves’ with letters and their frequency and arrange them s.t. they are always in increasing order (of frequency).
2. Repeat until there is only one tree:
   1. Create a new tree from the two leftmost trees (with the smallest frequencies) and
   2. Put it in its place.
3. Label left/right branches with 0/1

Encoding of char = path from root to leaf that has that char

See book or lecture video for the step-by-step solution.
In exam or hw, you must use this method.

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

Huffman codes

- Tree properties:
  - Left node is always smaller or equal than the right node.
  - Every internal node has two children

Greedy Algorithms

- Greedy algorithms do not always guarantee optimal solution. It depends on the problem.
- Difference between Greedy and Dynamic Programming:
  - In DP typically you solve all the problems and then make your choice.
  - In greedy, you make the greedy choice and you are left with only one problem.