09 - Greedy Technique (part 2)

[KOMS119602] & [KOMS120403]

Design and Analysis of Algorithm (2021/2022)

Dewi Sintiari

Prodi S1 Ilmu Komputer
Universitas Pendidikan Ganesha

Week 11-15 April 2022

Table of contents

- Principal of Greedy algorithm
- Scheme of Greedy algorithm
- Some examples of Greedy implementation

Part 2:
4. Integer (1/0) knapsack problem

4. Integer (1/0) knapsack problem (1)

Problem: given n objects and a knapsack with capacity K. Every object has weight w_i and profit p_i.

How to chose the objects to be included in the knapsack s.t. the total profit is maximum? The total weight of the objects should not exceed the capacity of the knapsack.

The mathematical formulation of 1/0 knapsack problem:

$$\text{Maximize} \quad F = \sum_{i=1}^{n} p_i x_i$$

$$\text{subject to} \quad \sum_{i=1}^{n} w_i x_i \leq K$$

$$\text{and } x_i = 0 \text{ or } x_i = 1, \ \text{ for } i = 1, 2, \ldots, n$$

4. Integer (1/0) knapsack problem (2)

Recall that the time complexity with exhaustive search is $\mathcal{O}(n \cdot 2^n)$. Why?

The greedy approach:

- Include the object one-by-one to the knapsack. Once it is included, it cannot be undone.
- Some greedy-heuristically strategies that can be used to choose the objects in the knapsack:
  1. Greedy by profit: at each step, choose the object with maximum profit
  2. Greedy by weight: at each step, choose the object of minimum weight
  3. Greedy by density: at each step, choose the object with the maximum value of p_i/w_i
- However, none of the strategies above guarantees an optimal solution.

4. Integer (1/0) knapsack problem (3)

Example

Given four objects as follows, and a knapsack of capacity $M = 16$.

$$(w_1, p_1) = (6, 12);\ (w_2, p_2) = (5, 15)$$

$$(w_3, p_3) = (10, 50);\ (w_4, p_4) = (5, 10)$$

Object properties				Greedy by			Optimal solution
i	w_i	p_i	p_i/w_i	profit	weight	density	
1	6	12	2	0	1	0	0
2	5	15	3	1	1	1	1
3	10	50	5	1	0	1	1
4	5	10	2	0	1	0	0
Solution set				$\{3, 2\}$	$\{2, 4, 1\}$	$\{3, 2\}$	15
Total weight				20	20	20	15
Total profit				28.2	31.0	31.5	65

4. Integer (1/0) knapsack problem (4)

Example

Given six objects as follows:

$$(w_1, p_1) = (100, 40);\ (w_2, p_2) = (50, 35);\ (w_3, p_3) = (45, 18)$$

$$(w_4, p_4) = (20, 4);\ (w_5, p_5) = (10, 10);\ (w_6, p_6) = (5, 2)$$

and a knapsack of capacity $M = 100$.

Example

Given six objects as follows:

$$(w_1, p_1) = (100, 40);\ (w_2, p_2) = (50, 35);\ (w_3, p_3) = (45, 18)$$

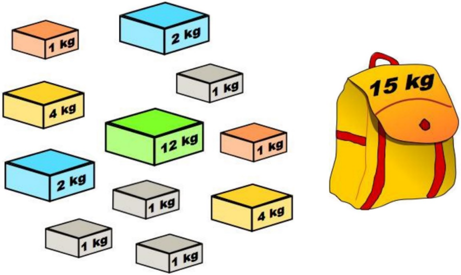
$$(w_4, p_4) = (20, 4);\ (w_5, p_5) = (10, 10);\ (w_6, p_6) = (5, 2)$$

and a knapsack of capacity $M = 100$.

Object properties				Greedy by			Optimal solution
i	w_i	p_i	p_i/w_i	profit	weight	density	
1	100	40	0.4	1	0	0	0
2	50	35	0.7	0	0	1	1
3	45	18	0.4	0	1	0	1
4	20	4	0.2	0	1	1	0
5	10	10	1.0	0	1	1	0
6	5	2	0.4	0	1	1	0
	Total weight			100	80	85	100
	Total profit			40	34	51	55

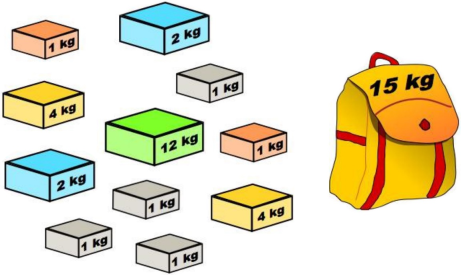
Conclusion:

Is greedy algorithm always able to find an optimal solution for the integer knapsack problem?



Conclusion:

Is greedy algorithm always able to find an optimal solution for the integer knapsack problem?



NO! **Homework:** Find an example where the three approaches do not give an optimum solution!

5. Fractional knapsack problem

The fractional knapsack problem is a variant of knapsack problem, but the solution is not necessarily integer, it can be in fraction.

Problem formulation:

$$\text{Maximize } F = \sum_{i=1}^{n} p_i x_i$$

$$\text{s.t. } \sum_{i=1}^{n} w_i x_i \leq K$$

$$\text{and } 0 \leq x_i \leq 1, \text{ for } i = 1, 2, \ldots, n$$

Question: is it possible to solve the problem with exhaustive search?

5. Fractional knapsack problem (2)

Question: is it possible to solve the problem with exhaustive search?

Since $0 \leq x_i \leq 1$, then there are an infinite number of possibilities of x_i.

This problem is not discrete, but a continuous problem, so it is not possible to solve with exhaustive search.

5. Fractional knapsack problem (3)

Question: is it possible to solve the problem with the greedy approach?

Example

Given three objects as follows:

$$(w_1, p_1) = (18, 25);\ (w_2, p_2) = (15, 24);\ (w_3, p_3) = (10, 15)$$

and a knapsack of capacity $M = 20$.

Object properties				Greedy by		
i	w_i	p_i	p_i/w_i	profit	weight	density
1	18	25	1.4	1	0	0
2	15	24	1.6	2/15	2/3	1
3	10	15	1.5	0	1	1/2
	Total weight			20	20	20
	Total profit			28.2	31.0	31.5

5. Fractional knapsack problem (4)

Theorem (Greedy by density gives an optimal solution)

If $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \cdots \geq \frac{p_n}{w_n}$, then the greedy algorithm with the strategy of choosing the maximum $\frac{p_i}{w_i}$ gives an optimal solution.

Proof.

Homework! (give a similar taste of proof as for the "Activity Selector Problem") $\square$

Algorithm:

- Compute $\frac{p_i}{w_i}$ for $i = 1, 2, \ldots, n$
- For this strategy to work, the $\frac{p_i}{w_i}$'s are ordered in descending order.

5. Fractional knapsack problem (5)

```
procedure FRACTIONALKNAPSACK(C: objects set, K: real)
    for i ← 1 to n do
        x[i] ← 0                                  ▷ x is the solution set
    end for
    i ← 0;  totalwt ← 0;  intFrac ← True   ▷ 'totalwt': total weight, and 'intFrac': boolean var indicating if current object can be included fully
    while (i ≤ n) and intFrac do
        i ← i + 1
        if totalwt + w[i] ≤ K then
            x[i] ← 1                              ▷ Include object i to knapsack
            totalwt ← totalwt + w[i]   ▷ Include the fraction of object i to total weight
        else
            intFrac ← False
            x[i] ← (K − totalwt) / w[i]   ▷ Only a fraction of object i can be included to the knapsack
        end if
    end while
    return x
end procedure
```

6. Huffman coding

6. Huffman coding (1)

The principal of encoding and decoding

Encoding/decoding is the translation of a message that is easily understood.

Encoding: the way any character is understood within the computer storage or transmission from one machine to another machine.

Decoding: the process of turning back an encoded message to the original message.

7. Huffman coding (2)

Fixed-length versus Variable-length codes

Fixed length encoding scheme uses a fixed number of bytes to represent different characters.

Variable length encoding scheme uses different number of bytes to represent different characters

Huffman coding is used for data compression.

Fixed-length code

Given a message of length 100,000 characters with frequency of a letter appears in the message as the following:

Character	*a*	*b*	*c*	*d*	*e*	*f*
Frequency	45%	13%	12%	16%	9%	5%
Encoding	000	001	010	011	100	111

Example: the encoding of 'bad' is 001000011

With this method, the encoding of 100,000 characters needs 300,000 bits.

6. Huffman coding (3)

Huffman coding is used for data compression.

Fixed-length code

Given a message of length 100,000 characters with frequency of a letter appears in the message as the following:

Character	*a*	*b*	*c*	*d*	*e*	*f*
Frequency	45%	13%	12%	16%	9%	5%
Encoding	000	001	010	011	100	111

Example: the encoding of 'bad' is 001000011

With this method, the encoding of 100,000 characters needs 300,000 bits.

The principal of Huffman coding:

- the more often a character appears, the shortest its *encoding*, and vice versa.

Variable-length code (Huffman code)

Character	*a*	*b*	*c*	*d*	*e*	*f*
Frequency	45%	13%	12%	16%	9%	5%
Encoding	0	101	100	111	1100	1100

Example: the encoding of bad is 1010111

With this method, the encoding of 100,000 characters needs:

$$(0.45 \times 1 + 0.13 \times 3 + 0.12 \times 3 + 0.16 \times 3 + 0.09 \times 4 + 0.05 \times 4) \times 10^5$$
$$= 224,000 \text{ bits}$$

Ratio of compression $= \frac{300{,}000 - 224{,}000}{300{,}000} \times 100\% = 25,5\%.$

6. Huffman coding (5)

- The greedy algorithm to form Huffman coding aims to minimize the length of binary code for all characters in the message $(M_1, M_2, \ldots, M_n)$.
- We build a weighted binary tree. Every *leave node* indicates the character in the message, and *internal nodes* indicate the merging of those characters.
- Every edge in the tree is given label 0 or 1 consistently (e.g.: left given '0' and right given '1').
- Minimizing the binary code for every character is equivalent to minimizing the length of path from the root to the leaves.

Algorithm:

1. Compute the frequency of every character in the message. Represent every character by a tree with a single node, and every node is assigned with the frequency of the corresponding character.
2. We apply the greedy strategy: at each step, merge two trees that have the smallest frequencies in a root. The new root has frequency equals to the sum of the frequencies of the two trees that composed it.
3. We repeat the 2nd step until we finally obtain a single Huffman tree. It forms a binary tree.
4. We label every edge of the tree by 0 or 1 (e.g. left-oriented edge is labeled 0 and right-oriented edge is labeled 1.
5. Every path from the root the each leaf of the tree represents the binary string for every character, with frequency as indicated on the corresponding leaf.

What is the time complexity?
$\mathcal{O}(n \log n)$.

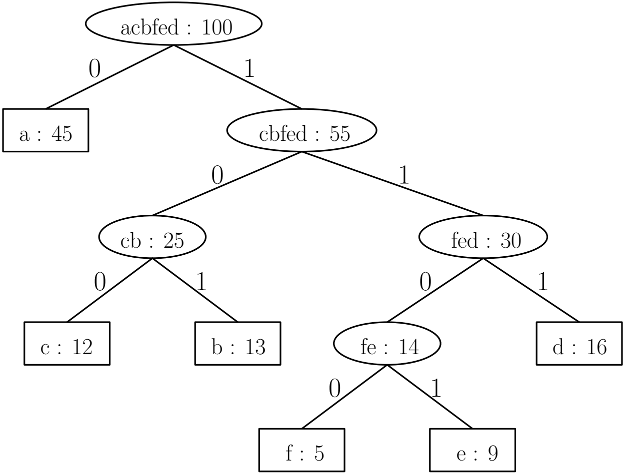
- Use a heap to store the weight of each tree, each iteration requires $\mathcal{O}(\log n)$-time to determine the cheapest weight and insert the new weight.
- There are $\mathcal{O}(n)$ iterations, one for each item.

Exercise: Given a message of length 100. The message is composed with letters a, b, c, d, e, f. The frequency of each letter in the message is as follows:

Character	a	b	c	d	e	f
Frequency	45%	13%	12%	16%	9%	5%

Find the Huffman code for every character in the message.

6. Huffman coding (10)



6. Huffman coding (11)

Huffman code:

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