

Greedy Algorithm

KOMS120403 / KOMS119602 - Design and Analysis of
Algorithm (2021/2022)

09 - Greedy Algorithm

Dewi Sintiar

Prodi S1 Ilmu Komputer
Universitas Pendidikan Ganesha

Week 4-8 April 2022

Table of contents

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

Optimization problem

An **optimization problem** is the problem of finding the best solution from all feasible solutions.

Types of optimization problems

- Maximization
 - Example: Integer knapsack problem
- Minimization
 - Example: Graph coloring problem, TSP

Standard form:

$$\begin{array}{ll}\text{minimize} & f(x) \\ \text{subject to} & g_i(x) \leq 0, \quad i = 1, \dots, m \\ & h_j(x) = 0, \quad j = 1, \dots, p\end{array}$$

Definition

Greedy algorithm is an algorithmic paradigm that builds up a solution piece by piece, always choosing the next piece that offers the most obvious and immediate benefit.

This is the most popular and the simplest algorithm to solve **optimization** problems (**maximization** and **minimization**).

- Greedy principal: **Take what you can get now!**
- Greedy algorithm builds the solution step-by-step.
- In each step, there are many possible choices. We take the best decision in each step, i.e. we choose the **local optimum**, in order to reach the **global optimum**.

Greedy algorithm

Example (Coin exchange problem)

We have a check of 42 in hand. There are coins of nominal 1, 5, 10, and 25. We want to exchange the money with the coins. What is the minimum number of coins needed in the exchange.

There are many possible combinations of coins. With brute-force algorithm, we can simply list all possibilities:

- $42 = 1 + 1 + \dots + 1 \rightarrow 42$ coins
- $42 = 5 + 1 + 1 + \dots + 1 \rightarrow 38$ coins
- ... etc.

With greedy algorithm, at each step, we take a coin with maximum value as many as possible.

$$42 = 25 + 10 + 5 + 1 + 1$$

So 5 coins are enough.

Scheme of greedy algorithm

Components of greedy algorithm:

- A **candidate set** - consists of candidate of solution that is created from the set.
- A **selection function** - used to choose the best candidate to be added to the solution.
- A **feasibility function** - used to determine whether a candidate can be included in the solution (feasible/unfeasible).
- An **objective function** - used to assign a value to a solution or a partial solution (maximizing or minimizing).
- A **solution function** - used to indicate whether a complete solution has been reached.

Components analysis of the coin exchange problem:

- **Candidate set**: the set of coins $\{1, 5, 10, 25\}$.
- **Selection function**: choose the coin that has maximum value.
- **Feasibility function**: check if the sum of coins after taking a new coin does not exceed the amount of money.
- **Objective function**: minimizing the number of coins used.
- **Solution function**: the set of selected coins $\{1, 10, 25\}$.

Scheme of greedy algorithm

Algorithm 1 General scheme of greedy algorithm

```
1: procedure GREEDY( $C$ : candidate set)
2:    $S \leftarrow \{\}$ 
3:   while (not SOLUTION( $S$ )) and ( $C \neq \{\}$ ) do
4:      $x \leftarrow$  SELECTION( $C$ )
5:      $C \leftarrow C - \{x\}$ 
6:     if FEASIBLE( $S \cup \{x\}$ ) then
7:        $S \leftarrow S \cup \{x\}$ 
8:     end if
9:   end while
10:  if SOLUTION( $S$ ) then return  $S$ 
11:  else print('No solution exists')
12:  end if
13: end procedure
```

▷ S is the solution function

- At the end of iteration ("if condition"), we have a local optimum solution.
- At the end of the while loop, we obtain the global optimum solution (if any)

Some examples:

- ① Coin exchange problem
- ② Activity selection problem
- ③ Time minimization in the system
- ④ Integer knapsack problem
- ⑤ Fractional knapsack problem
- ⑥ Huffman coding
- ⑦ Traveling Salesman Problem

1. Coin exchange problem

1. Coin exchange problem (1)

Problem formulation:

- The amount of money want to be exchanged: M
- The set of available coins: $\{c_1, c_2, \dots, c_n\}$
- The solution set: $X = \{x_1, x_2, \dots, x_n\}$, where $x_i = 1$ if a_i is chosen and $x_i = 0$ otherwise

The objective function:

$$\begin{aligned} \text{Minimize } F &= \sum_{i=1}^n x_i \\ \text{subject to } \sum_{i=1}^n c_i x_i &= M \end{aligned}$$

1. Coin exchange problem (2)

Solution by exhaustive search

- Since $X = \{x_1, x_2, \dots, x_n\}$ and $x_i \in \{0, 1\}$, then there are 2^n possible solutions.
- To evaluate the objective function for each solution candidate, we need $\mathcal{O}(n)$ -time
- So, the time complexity of the exhaustive search is: $\mathcal{O}(n \cdot 2^n)$.

1. Coin exchange problem (3)

Algorithm 2 General scheme of greedy algorithm

```
1: procedure COINEXCHANGE( $C$ : coin set,  $M$ : integer)
2:    $S \leftarrow \{\}$ 
3:   while ( $\sum(\text{all coins in } S) \neq M$ ) and ( $C \neq \{\}$ ) do
4:      $x \leftarrow$  coin of maximum value
5:      $C \leftarrow C - \{x\}$ 
6:     if  $\sum(\text{all coins in } S) + \text{value}(x) \leq M$  then
7:        $S \leftarrow S \cup \{x\}$ 
8:     end if
9:   end while
10:  if  $\sum(\text{all coins in } S) = M$  then
11:    return  $S$ 
12:  else
13:    print('No feasible solution')
14:  end if
15: end procedure
```

1. Coin exchange problem (4)

Time complexity:

- Choosing a coin with maximum value: $\mathcal{O}(n)$ (using brute-force to get max).
- The while loop is repeated n times (maximum), so the overall complexity is: $\mathcal{O}(n^2)$.
- If the coin is ordered in descending order, the complexity becomes $\mathcal{O}(n)$, because choosing a coin with max value only takes $\mathcal{O}(1)$ -time.

2. Activity selection problem

2. Activity selection problem (1)

Problem: given n activities $S = \{1, 2, \dots, n\}$, that will use a resource (for instance, meeting room, studio, processor, etc.).

Suppose that a resource can only be used to do one activity at one time. Each time an activity occupies a resource, then the other activities cannot use it until the first activity is finished.

Each activity i starts at time s_i and ends at time f_i , where $s_i \leq f_i$. Two activities i and j is called *compatible* if the interval $[s_i, f_i]$ and $[s_j, f_j]$ do not intersect.

Our goal is to choose as many activities as possible that can be served by a resource.

2. Activity selection problem (2)

Example (Activity selection problem)

Given $n = 11$ activities with the starting-ending time as given in the following table:

i	s_i	f_i
1	1	4
2	3	5
3	4	6
4	5	7
5	3	8
6	7	9
7	10	11
8	8	12
9	8	13
10	2	14
11	13	15

2. Activity selection problem (3)

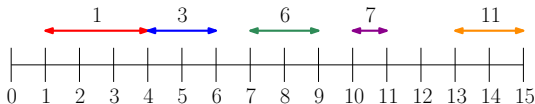
Solution with exhaustive search:

- 1 List all subsets of the set of n activities.
- 2 Evaluate each subset, check if the solution is compatible.
- 3 If yes, then the subset is a solution candidate.
- 4 Choose the solution candidate with the maximum number of activities
- 5 The time complexity of the algorithm is $\mathcal{O}(n \cdot 2^n)$. Why?

2. Activity selection problem (4)

The greedy approach:

- 1 Order the activities based on the end-time in ascending order.
- 2 At each step, choose the activity whose starting time is greater than or equal to the end-time of the activity that was chosen before



Strategy: at each step, we choose the activity of the smallest index that can be done in the available time-slot.

Set of solution: $\{1, 3, 6, 7, 11\}$

2. Activity selection problem (5)

Algorithm 3 Greedy Activity Selector

```
1: procedure ACTIVITYSELECTOR( $(s_1, \dots, s_n), (f_1, \dots, f_n)$ )
2:    $n \leftarrow \text{length}(s)$   $\triangleright s = (s_1, \dots, s_n)$ 
3:    $A \leftarrow \{1\}$   $\triangleright A$  is the solution function
4:    $j \leftarrow 1$ 
5:   for  $i \leftarrow 2$  to  $n$  do
6:     if  $s_i \geq f_j$  then  $\triangleright s_i$ : starting time of activity  $i$ ,  $f(j)$ : finishing time of activity  $j$ 
7:        $A \leftarrow A \cup \{i\}$ 
8:        $j \leftarrow i$ 
9:     end if
10:  end for
11: end procedure
```

Time complexity: $\mathcal{O}(n)$. Why?

2. Activity selection problem (6)

Theorem

The greedy algorithm gives an optimal solution for the activity selection problem.

Proof.

Let $S = \{1, 2, \dots, n\}$ be the set of activities that are ordered based on the finishing time.

Suppose that $A \subseteq S$ be an optimal solution, where the elements in A are also ordered based on the finishing time, and the first element of A is k_1 .

If $k_1 = 1$, then A is begun with a greedy choice (as in the algorithm).

Otherwise, we show that $B = (A - \{k_1\}) \cup \{1\}$ (a solution begin with the greedy choice 1 is another optimal solution).

Since $f_1 \leq f_{k_1}$ (bcs 1 is the first element of S), and the activities in A are compatible, then the activities in B are also compatible. Since $|A| = |B|$, then B is an optimal solution. Hence, there is also an optimal solution that is begun with a greedy choice. □

2. Activity selection problem (7)

Proposition of greedy strategy: (*unproved*)

Choose the activity that has the shortest duration and the starting time \leq the finishing time of the activity that has been chosen.

i	s_i	f_i	duration
1	1	4	3
2	3	5	2
3	4	6	2
4	5	7	2
5	3	8	5
6	7	9	2
7	10	11	1
8	8	12	4
9	8	13	5
10	2	14	12
11	13	15	2

Solution: $\{7, 2, 4, 6, 11\}$

Homework: Can you check whether this strategy gives an optimal solution?

3. Time minimization in the system

3. Time minimization in the system (1)

Problem: a server (processor, cashier, customer service, etc.) has n clients that must be served. The time to serve client i is t_i . How to minimize the total time in the system (including the waiting time)?

$$T = \sum_{i=1}^n \text{time spent in the system}$$

Remark. This problem is equivalent to minimizing the average time of clients in the system.

3. Time minimization in the system (2)

Example

There are three clients with serving time: $t_1 = 5$, $t_2 = 10$, and $t_3 = 3$.

Solution

The possible order of the clients:

- **1,2,3:** $5 + (5 + 10) + (5 + 10 + 3) = 38$
- **1,3,2:** $5 + (5 + 3) + (5 + 3 + 10) = 31$
- **2,1,3:** $10 + (10 + 5) + (10 + 5 + 3) = 43$
- **2,3,1:** $10 + (10 + 3) + (10 + 3 + 5) = 41$
- **3,1,2:** $3 + (3 + 5) + (3 + 5 + 10) = 29$
- **3,2,1:** $3 + (3 + 10) + (3 + 10 + 5) = 34$

3. Time minimization in the system (3)

- Other problems similar to minimizing time in the system is **optimal storage on tapes** or music storage in a cassette tape (analog system with sequential storage system).
- The programs/musics are saved in the tape sequentially. The length of every song i is t_i (in second/minute). To retrieve and play a song, the tape is initially placed in the beginning.
- If the songs in the tape are saved in the order $X = \{x_1, x_2, \dots, x_n\}$, then the time needed to play the song x_j to the end is
$$T_j = \sum_{1 \leq k \leq j} t_{x_k}.$$
- If all songs are often played, the *mean retrieval time / MRT* is
$$\frac{1}{n} \sum_{1 \leq j \leq n} T_j.$$
- Here we are asked to find a permutation of n songs s.t. if the songs are saved in the tape, then the MRT is minimum. Minimizing MRT is equivalent to minimizing the following:

$$d(X) = \frac{1}{n} \sum_{1 \leq j \leq n} \sum_{1 \leq k \leq j} t_{x_k} \quad \text{or} \quad d(X) = \sum_{1 \leq j \leq n} \sum_{1 \leq k \leq j} t_{x_k}$$

3. Time minimization in the system (4)

Solution with the exhaustive search

- The order of clients in the system is a permutation. The number of permutation of n elements is $n!$.
- To evaluate the objective function of a permutation needs $\mathcal{O}(n)$ -time
- The time complexity of exhaustive search is $\mathcal{O}(n \cdot n!)$.

3. Time minimization in the system (5)

Solution with the greedy algorithm

Strategy: At each step, choose the client that needs the minimum serving time among all clients that have not been served.

Algorithm 4 Clients Scheduling

```
1: procedure CLIENTSSCHEDULING( $n$ )
2:    $S \leftarrow \{\}$ 
3:   while  $C \neq \{\}$  do
4:      $i \leftarrow$  client with minimum  $t[i]$  in  $C$ 
5:      $C \leftarrow C - \{i\}$ 
6:      $S \leftarrow S \cup \{i\}$ 
7:   end while
8:   return  $S$ 
9: end procedure
```

▷ S is the solution function
▷ C is the solution candidate
▷ t_i is the serving time of client i

Time complexity: $\mathcal{O}(n^2)$. Why?

3. Time minimization in the system (6)

If the clients are ordered based on the serving time (in ascending order), then the complexity of the greedy algorithm is $\mathcal{O}(n)$.

Algorithm 5 Clients Scheduling

```
1: procedure CLIENTSSCHEDULING2( $n$ )
2:   input: clients  $(1, 2, \dots, n)$  ordered ascendingly based on  $t_i$ 
3:   for  $i \leftarrow 1$  to  $n$  do
4:     print( $i$ )
5:   end for
6: end procedure
```

Remark. The greedy algorithm for clients scheduling based on the serving time with ascending order always produces an optimal solution.

3. Time minimization in the system (7)

Theorem

If $t_1 \leq t_2 \leq \dots \leq t_n$, then the order $i_j = j$, $1 \leq j \leq n$ minimizes:

$$T = \sum_{k=1}^n \sum_{j=1}^k t_{i_j}$$

for all possible permutations of i_j , $1 \leq j \leq n$.

Proof available in Ellis Horowitz & Sartaj Sahni, Computer Algorithms, 1998). See next slide.

3. Time minimization in the system (8)

Proof: Let $I = i_1, i_2, \dots, i_n$ be any permutation of the index set $\{1, 2, \dots, n\}$. Then

$$d(I) = \sum_{k=1}^n \sum_{j=1}^k t_{i_j} = \sum_{k=1}^n (n - k + 1) t_{i_k}$$

If there exist a and b such that $a < b$ and $t_{i_a} > t_{i_b}$, then interchanging i_a and i_b results in a permutation I' with

$$d(I') = \left[\sum_{\substack{k \\ k \neq a \\ k \neq b}} (n - k + 1) t_{i_k} \right] + (n - a + 1) t_{i_b} + (n - b + 1) t_{i_a}$$

Subtracting $d(I')$ from $d(I)$, we obtain

$$\begin{aligned} d(I) - d(I') &= (n - a + 1)(t_{i_a} - t_{i_b}) + (n - b + 1)(t_{i_b} - t_{i_a}) \\ &= (b - a)(t_{i_a} - t_{i_b}) \\ &> 0 \end{aligned}$$

Hence, no permutation that is not in nondecreasing order of the t_i 's can have minimum d . It is easy to see that all permutations in nondecreasing order of the t_i 's have the same d value. Hence, the ordering defined by $i_j = j, 1 \leq j \leq n$, minimizes the d value. \square

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 choose 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:

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

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

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

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 $M = 100$.

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

Example

Given six objects as follows:

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

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

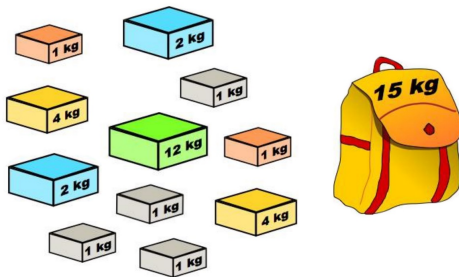
and a knapsack of capacity $M = 100$.

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

4. Integer (1/0) knapsack problem (5)

Conclusion:

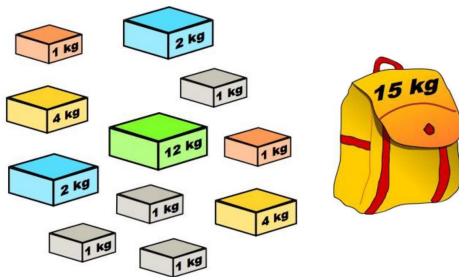
Is greedy algorithm always able to find an optimal solution for the integer knapsack problem?



4. Integer (1/0) knapsack problem (5)

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

5. Fractional knapsack problem (1)

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

Problem formulation:

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

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

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

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 $\frac{p_1}{w_1} \geq \frac{p_2}{w_2} \geq \dots \geq \frac{p_n}{w_n}$, 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”)



Algorithm:

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

5. Fractional knapsack problem (5)

```
1: procedure FRACTIONALKNAPSACK( $C$ : objects set,  $K$ : real)
2:   for  $i \leftarrow 1$  to  $n$  do
3:      $x[i] \leftarrow 0$   $\triangleright x$  is the solution set
4:   end for
5:    $i \leftarrow 0$ ; totalwt  $\leftarrow 0$ ; intFrac  $\leftarrow$  True  $\triangleright$  'totalwt': total weight, and 'intFrac':
boolean var indicating if current object can be included fully
6:   while ( $i \leq n$ ) and intFrac do
7:      $i \leftarrow i + 1$ 
8:     if totalwt +  $w[i] \leq K$  then
9:        $x[i] \leftarrow 1$   $\triangleright$  Include object  $i$  to knapsack
10:      totalwt  $\leftarrow$  totalwt +  $w[i]$   $\triangleright$  Include the fraction of object  $i$  to total weight
11:    else
12:      intFrac  $\leftarrow$  False
13:       $x[i] \leftarrow \frac{K - \text{totalwt}}{w[i]}$   $\triangleright$  Only a fraction of object  $i$  can be included to the knapsack
14:    end if
15:  end while
16:  return  $x$ 
17: 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

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	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
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	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
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.

6. Huffman coding (4)

Variable-length code (Huffman code)

Character	<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	<i>e</i>	<i>f</i>
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:

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

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

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, \dots, 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.

6. Huffman coding (6)

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.

6. Huffman coding (7)

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.

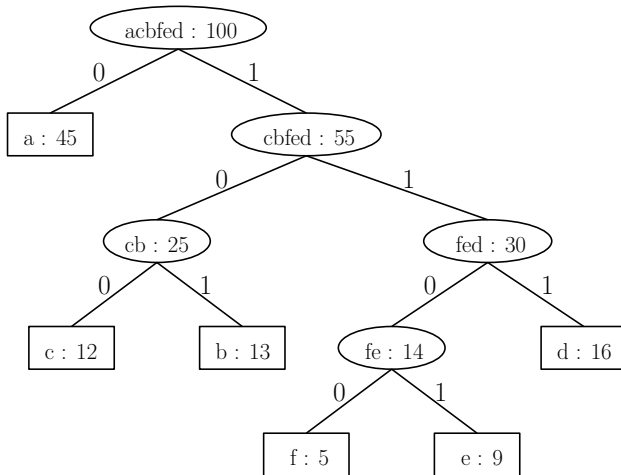
6. Huffman coding (8)

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