Greedy (2)

Oct 25^{th}, 2018

Algorithm Design and Analysis

YUN-NUNG (VIVIAN) CHEN HTTP://ADA.MIULAB.TW

國立臺灣大學
National Taiwan University

Slides credited from Hsu-Chun Hsiao

Announcement

- Mini-HW 6 Released
  - Due on 11/01 (Thur) 14:20
- 11/01 Midterm Review QA Session
  - Optional participation
- Homework 2
  - Due on 11/06 (Tue) 18:00
- 11/08 Midterm Exam

Mini-HW 6

Piepie is an experienced cake baker. To bake a cake, he needs to mix all of the ingredients together into batter(麵糊). However, the process of stirring requests lots of body strength, which is equal to the sum of weight of what you mixed. And, **you can only mix two bowls of batter into one each time.**

Here's an example:

- If there are three ingredients, {eggs, butter, flour}, you can either (a.) mix eggs and butter together first, include the flour at last. Or, you can also (b.) mix eggs and flour together first, than include the eggs at last. But you **CAN'T** put all of the three things together in one sitting.
- If the weight of {eggs, butter, flour} are {3, 5, 10}, then the (a.) approach should require (3+5)+(8+10)=26 units of body strength, while the (b.) approach requires (3+10)+(13+8)=34 units of body strength.

Now, given an sorted sequence of the weight of N ingredients, please design a **greedy algorithm** to tell Piepie whats the order of stirring, so that he can **use least body strength** to complete the cake. (40%)

Please prove the correctness of your algorithm, including **Greedy-choice property** (30%) & **Optimal substructure** (30%). Your algorithm should have an $O(N \log N)$ time complexity.

Midterm!!!

- **Date: 11/08 (Thursday)**
- **Time: 14:20-17:20** (3 hours)
- **Location: R102, R103, R104** (check the seat assignment before entering the room)
- Content
  - Recurrence and Asymptotic Analysis
  - Divide and Conquer
  - Dynamic Programming
  - Greedy
- Based on slides, assignments, and some variations (practice via textbook exercises)
- Format: Yes/No, Multiple-Choice, Short Answer, Prove/Explanation
- Easy: ~60%, Medium: ~30%, Hard: ~10%
- Close book

Outline

- Greedy Algorithms
- Greedy #1: Activity-Selection / Interval Scheduling
- Greedy #2: Coin Changing
- Greedy #3: Fractional Knapsack Problem
- Greedy #4: Breakpoint Selection
- Greedy #5: Huffman Codes
- Greedy #6: Scheduling to Minimize Lateness
- Greedy #7: Task-Scheduling

6

Coin Changing

Textbook Exercise 16.1

Coin Changing Problem

- Input: n dollars and unlimited coins with values $\{v_i\}$ (1, 5, 10, 50)
- Output: the minimum number of coins with the total value n

- **Cashier's algorithm**: at each iteration, add the coin with the largest value no more than the current total

Does this algorithm return the OPT?

Step 1: Cast Optimization Problem

Coin Changing Problem
Input: n dollars and unlimited coins with values $\{v_i\}$ (1, 5, 10, 50)
Output: the minimum number of coins with the total value n

- Subproblems
  - `C(i)`: minimal number of coins for the total value i
  - Goal: `C(n)`

Step 2: Prove Optimal Substructure

Coin Changing Problem
Input: n dollars and unlimited coins with values $\{v_i\}$ (1, 5, 10, 50)
Output: the minimum number of coins with the total value n

- Suppose OPT is an optimal solution to `C(i)`, there are 4 cases:
  - Case 1: coin 1 in OPT
    - OPT\coin1 is an optimal solution of `C(i - v1)`
  - Case 2: coin 2 in OPT
    - OPT\coin2 is an optimal solution of `C(i - v2)`
  - Case 3: coin 3 in OPT
    - OPT\coin3 is an optimal solution of `C(i - v3)`
  - Case 4: coin 4 in OPT
    - OPT\coin4 is an optimal solution of `C(i - v4)`

$$C_i = \min_j (1 + C_{i-v_j})$$

Step 3: Prove Greedy-Choice Property

Coin Changing Problem
Input: n dollars and unlimited coins with values $\{v_i\}$ (1, 5, 10, 50)
Output: the minimum number of coins with the total value n

- Greedy choice: select the coin with the largest value no more than the current total
- Proof via contradiction (use the case $10 \leq i < 50$ for demo)
  - Assume that there is no OPT including this greedy choice (choose 10)

    → all OPT use 1, 5, 50 to pay i
    - 50 cannot be used
    - #coins with value 5 < 2 → otherwise we can use a 10 to have a better output
    - #coins with value 1 < 5 → otherwise we can use a 5 to have a better output
  - We cannot pay i with the constraints (at most $5 + 4 = 9$)

11

Fractional Knapsack Problem

Textbook Exercise 16.2-2

Knapsack Problem

- Input: n items where i-th item has value v_i and weighs w_i (v_i and w_i are positive integers)
- Output: the <u>maximum value</u> for the knapsack with capacity of W

- Variants of knapsack problem
  - 0-1 Knapsack Problem: 每項物品只能拿一個
  - Unbounded Knapsack Problem: 每項物品可以拿多個
  - Multidimensional Knapsack Problem: 背包空間有限
  - Multiple-Choice Knapsack Problem: 每一類物品最多拿一個
  - Fractional Knapsack Problem: 物品可以只拿部分

Knapsack Problem

- Input: n items where i-th item has value v_i and weighs w_i (v_i and w_i are positive integers)
- Output: the maximum value for the knapsack with capacity of W

- Variants of knapsack problem
  - 0-1 Knapsack Problem: 每項物品只能拿一個
  - Unbounded Knapsack Problem: 每項物品可以拿多個
  - Multidimensional Knapsack Problem: 背包空間有限
  - Multiple-Choice Knapsack Problem: 每一類物品最多拿一個
  - **Fractional Knapsack Problem: 物品可以只拿部分**

Fractional Knapsack Problem

- Input: n items where i-th item has value v_i and weighs w_i (v_i and w_i are positive integers)
- Output: the maximum value for the knapsack with capacity of W, where we can take **any fraction of items**
- Greedy algorithm: at each iteration, choose the item with the highest $\frac{v_i}{w_i}$ and continue when $W - w_i > 0$

Step 1: Cast Optimization Problem

Fractional Knapsack Problem
Input: n items where i-th item has value v_i and weighs w_i
Output: the max value within W capacity, where we can take **any fraction of items**

- Subproblems
  - `F-KP(i, w)`: fractional knapsack problem within w capacity for the first i items
  - Goal: `F-KP(n, W)`

Step 2: Prove Optimal Substructure

Fractional Knapsack Problem
Input: n items where i-th item has value v_i and weighs w_i
Output: the max value within W capacity, where we can take **any fraction of items**

- Suppose OPT is an optimal solution to `F-KP(i, w)`, there are 2 cases:
  - Case 1: full/partial item i in OPT
    - Remove w' of item i from OPT is an optimal solution of `F-KP(i - 1, w - w')`
  - Case 2: item i not in OPT
    - OPT is an optimal solution of `F-KP(i - 1, w)`

Step 3: Prove Greedy-Choice Property

Fractional Knapsack Problem
Input: n items where i-th item has value v_i and weighs w_i
Output: the max value within W capacity, where we can take **any fraction of items**

- Greedy choice: select the item with the highest $\frac{v_i}{w_i}$
- Proof via contradiction ($j = \underset{i}{\operatorname{argmax}} \frac{v_i}{w_i}$)
  - Assume that there is no OPT including this greedy choice
    - If $W \leq w_j$, we can replace all items in OPT with item j
    - If $W > w_j$, we can replace any item weighting w_j in OPT with item j
  - The total value must be equal or higher, because item j has the highest $\frac{v_i}{w_i}$

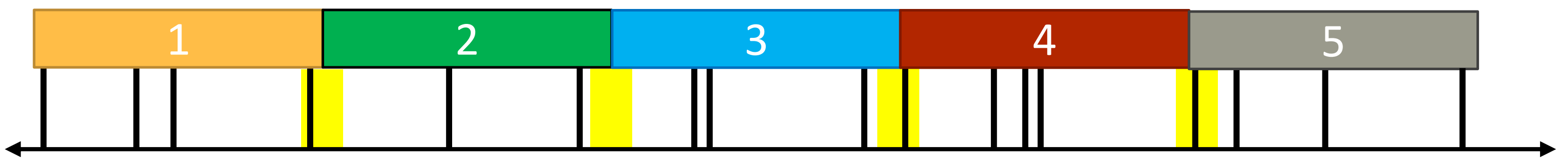
Do other knapsack problems have this property?

18 Breakpoint Selection

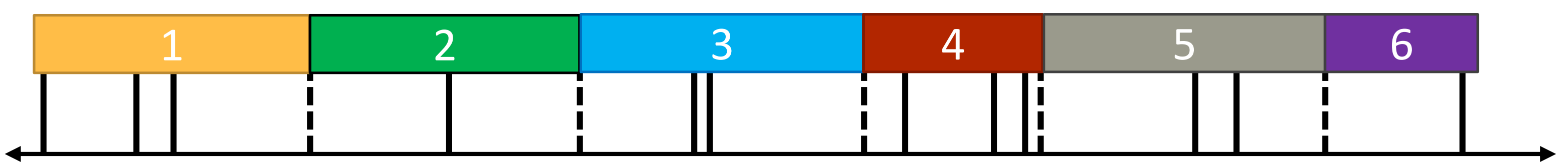
Breakpoint Selection Problem

- Input: a planned route with $n+1$ gas stations $b_0, \ldots, b_n$; the car can go at most C after refueling at a breakpoint
- Output: a refueling schedule ($b_0 \rightarrow b_n$) that minimizes the number of stops

Ideally: stop when out of gas



Actually: may not be able to find the gas station when out of gas



- Greedy algorithm: go as far as you can before refueling

Step 1: Cast Optimization Problem

Breakpoint Selection Problem
Input: $n + 1$ breakpoints $b_0, \dots, b_n$; gas storage is C
Output: a refueling schedule ($b_0 \rightarrow b_n$) that minimizes the number of stops

- Subproblems
  - `B(i)`: breakpoint selection problem from b_i to b_n
  - Goal: `B(0)`

Step 2: Prove Optimal Substructure

Breakpoint Selection Problem
Input: $n + 1$ breakpoints $b_0, \ldots, b_n$; gas storage is C
Output: a refueling schedule ($b_0 \rightarrow b_n$) that minimizes the number of stops

- Suppose OPT is an optimal solution to `B(i)` where j is the largest index satisfying $b_j - b_i \leq C$, there are $j - i$ cases
  - Case 1: stop at b_{i+1}
    - OPT+{b_{i+1}} is an optimal solution of `B(i + 1)`
  - Case 2: stop at b_{i+2}
    - OPT+{b_{i+2}} is an optimal solution of `B(i + 2)`

    :
  - Case $j - i$: stop at b_j
    - OPT+{b_j} is an optimal solution of `B(j)`

$$B_i = \min_{i<k\leq j}(1 + B_k)$$

Step 3: Prove Greedy-Choice Property

Breakpoint Selection Problem
Input: $n + 1$ breakpoints $b_0, \ldots, b_n$; gas storage is C
Output: a refueling schedule ($b_0 \rightarrow b_n$) that minimizes the number of stops

- Greedy choice: go as far as you can before refueling (select b_j)
- Proof via contradiction
  - Assume that there is no OPT including this greedy choice (after b_i then stop at b_k, $k \neq j$)
    - If $k > j$, we cannot stop at b_k due to out of gas
    - If $k < j$, we can replace the stop at b_k with the stop at b_j
  - The total value must be equal or higher, because we refuel later ($b_j > b_k$)

$$B_i = \min_{i<k\leq j}(1 + B_k) \Rightarrow B_i = 1 + B_j$$

Pseudo Code

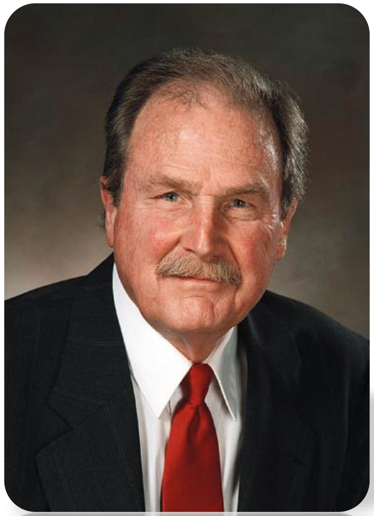
Breakpoint Selection Problem
Input: $n + 1$ breakpoints $b_0, \ldots, b_n$; gas storage is C
Output: a refueling schedule ($b_0 \rightarrow b_n$) that minimizes the number of stops

```
BP-Select(C, b)
  Sort(b) s.t. b[0] < b[1] < … < b[n]
  p = 0
  S = {0}
  for i = 1 to n - 1
    if b[i + 1] - b[p] > C
      if i == p
        return "no solution"
      A = A ∪ {i}
      p = i
return A
```

$$T(n) = \Theta(n \log n)$$

24

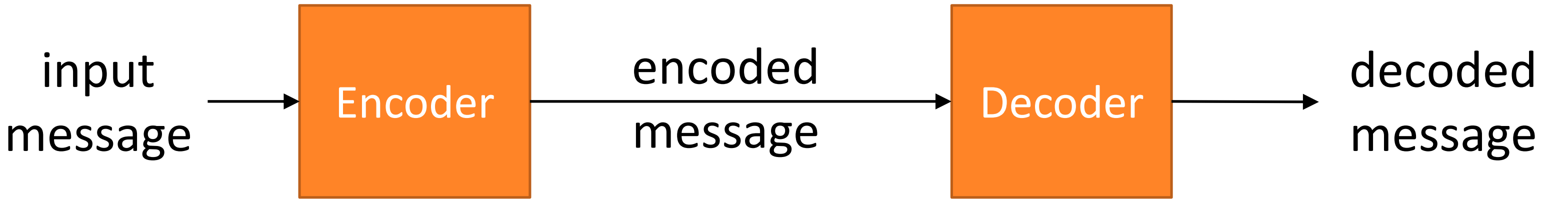
Huffman Codes



Textbook Chapter 16.3 – Huffman codes

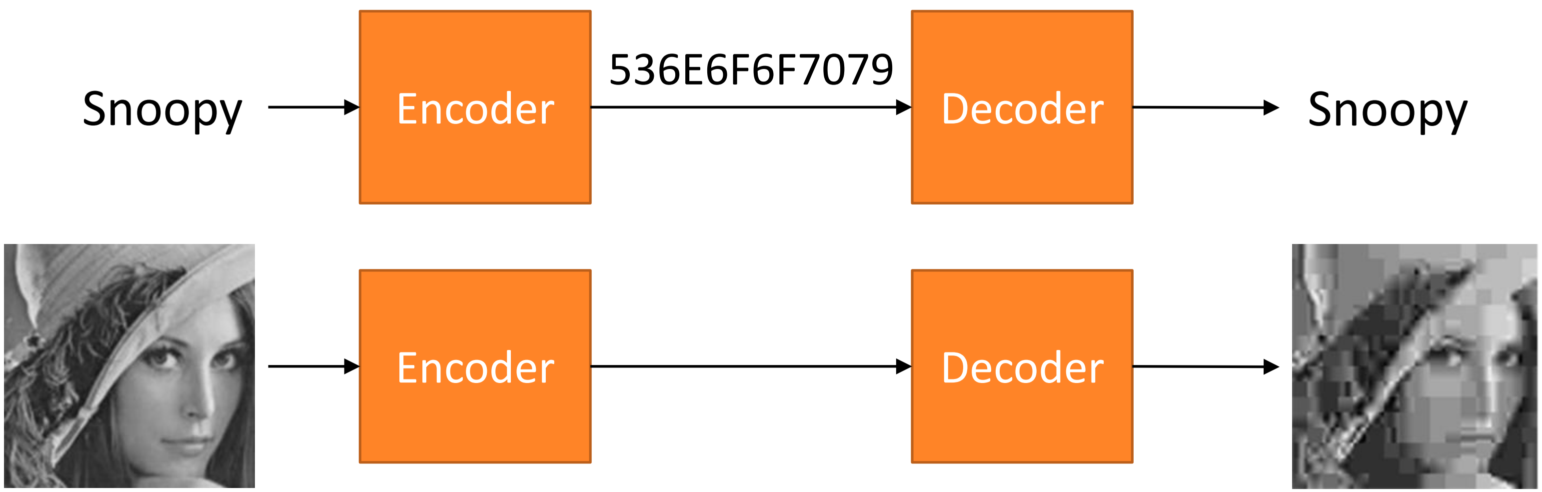
Encoding & Decoding

- **Code (編碼)** is a system of **rules to convert information**—such as a letter, word, sound, image, or gesture—into another, sometimes shortened or secret, form or representation for communication through a channel or storage in a medium.

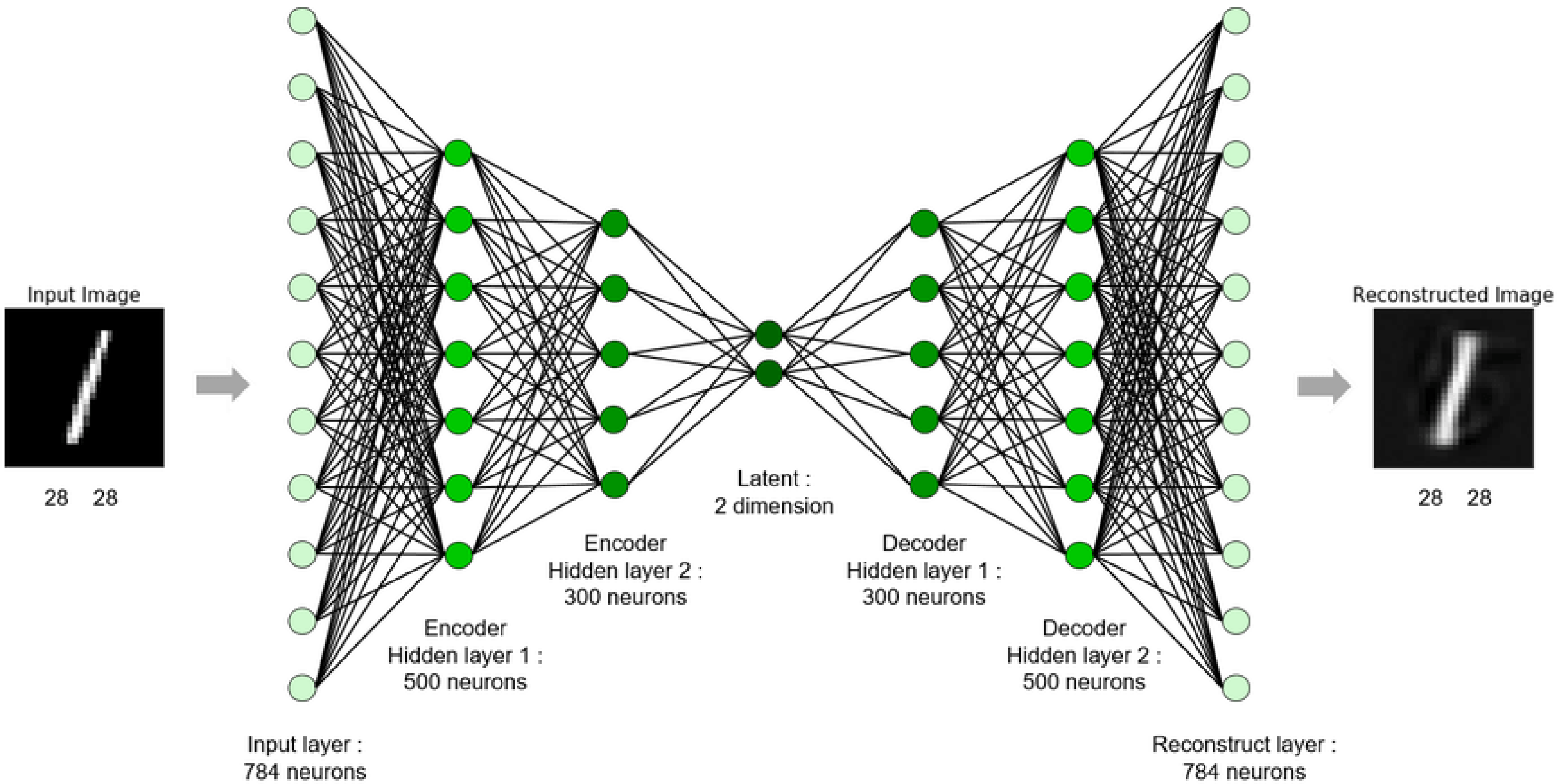


Encoding & Decoding

- Goal
  - Enable communication and storage
  - Detect or correct errors introduced during transmission
  - Compress data: lossy or lossless



Lossy Data Compression: Autoencoder



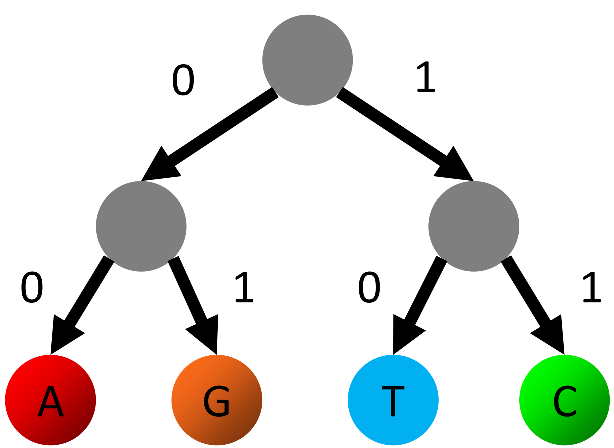
Lossless Data Compression

- Goal: encode each symbol using an unique binary code (w/o ambiguity)
  - How to represent symbols?
  - How to ensure decode(encode(x))=x?
  - How to minimize the number of bits?

Lossless Data Compression

- Goal: encode each symbol using an unique binary code (w/o ambiguity)
  - **How to represent symbols?**
  - How to ensure decode(encode(x))=x?
  - How to minimize the number of bits?

find a binary tree



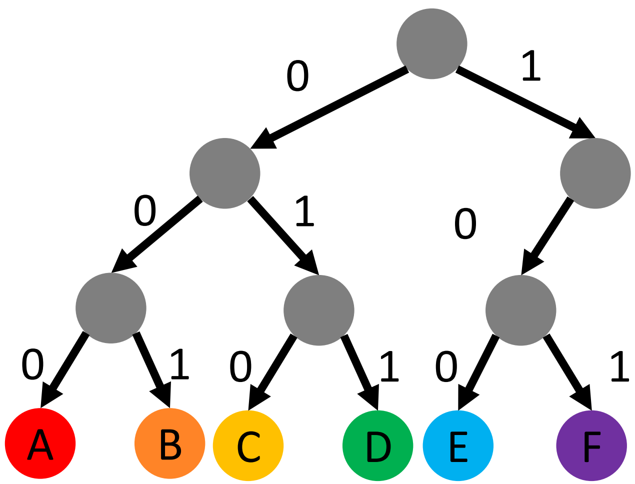
10101101011010100101010010

T T C G G T T T G G G A T

Code

Symbol	A	B	C	D	E	F
Frequency (K)	*45*	*13*	*12*	*16*	*9*	*5*
Fixed-length	000	001	010	011	100	101
Variable-length	0	101	100	111	1101	1100

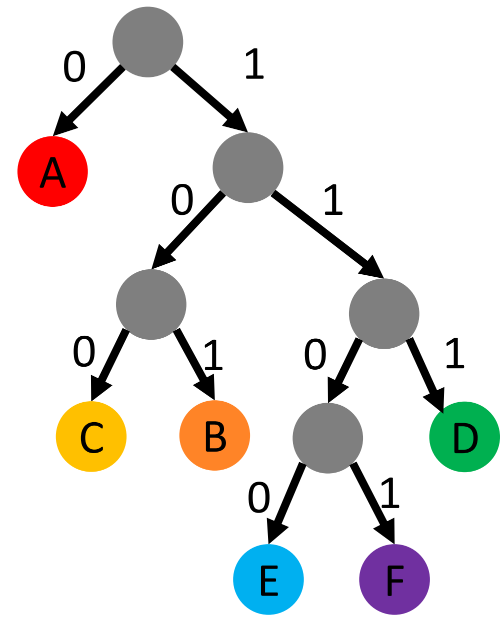
- **Fixed-length**: use the same number of bits for encoding every symbol
  - Ex. ASCII, Big5, UTF



- The length of this sequence is

$$(45+13+12+16+9+5)\cdot 3$$
$$= 300$$

- **Variable-length**: shorter codewords for more frequent symbols



- The length of this sequence is

$$45\cdot 1+(13+12+16)\cdot 3+(9+5)\cdot 4$$
$$= 224$$

Lossless Data Compression

- Goal: encode each symbol using an unique binary code (w/o ambiguity)
  - How to represent symbols?
  - **How to ensure decode(encode(x))=x?**
  - How to minimize the number of bits?

use codes that are uniquely decodable

Prefix Code

- Definition: a variable-length code where no codeword is a prefix of some other codeword

Symbol		A	B	C	D	E	F
Frequency (K)		*45*	*13*	*12*	*16*	*9*	*5*
Variable-length	Prefix code	0	101	100	111	1101	1100
	Not prefix code	0	**101**	**10**	111	1101	1100

- Ambiguity: decode(1011100) can be 'BF' or 'CDAA'

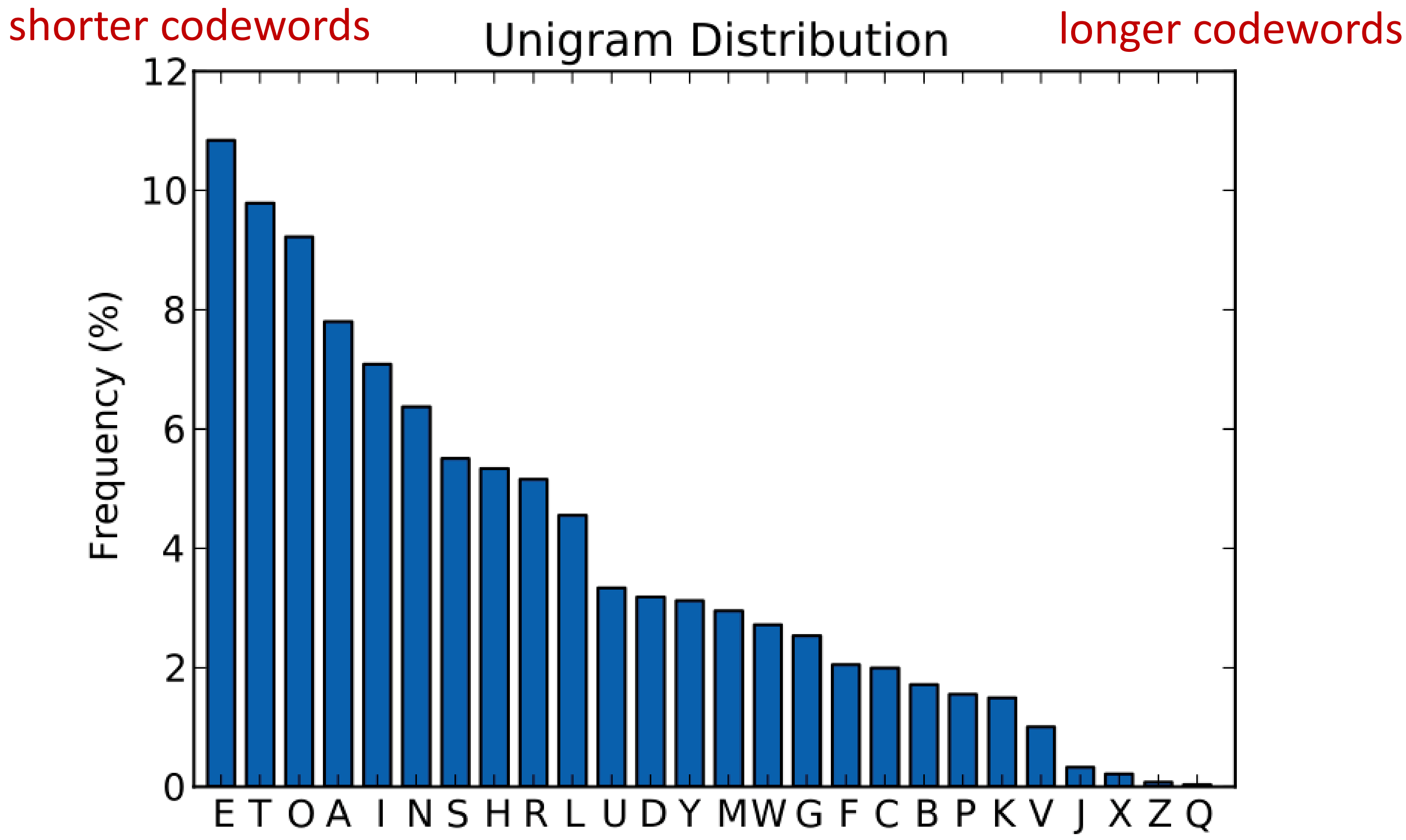
prefix codes are uniquely decodable

Lossless Data Compression

- Goal: encode each symbol using an unique binary code (w/o ambiguity)
  - How to represent symbols?
  - How to ensure decode(encode(x))=x?
  - **How to minimize the number of bits?**

more frequent symbols should use shorter codewords

Letter Frequency Distribution



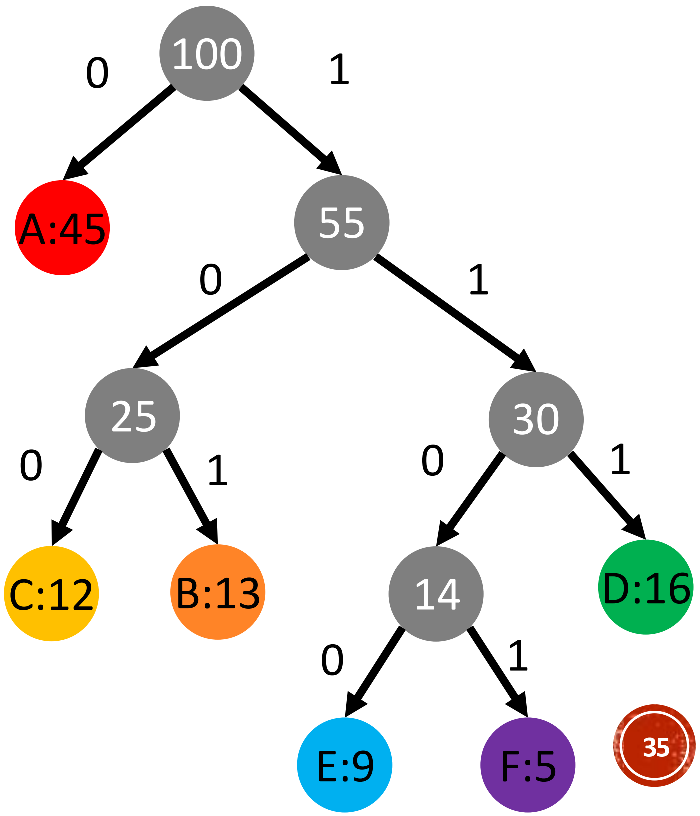
Total Length of Codes

- The weighted depth of a leaf = weight of a leaf (freq) × depth of a leaf
- Total length of codes = Total weighted depth of leaves
- Cost of the tree T

$$B(T) = \sum_{c \in C} \text{freq}(c) \cdot d_T(c)$$

- Average bits per character

$$\frac{B(T)}{100} = \sum_{c \in C} \text{relative-freq}(c) \cdot d_T(c)$$



How to find the **optimal prefix code** to **minimize the cost**?

Prefix Code Problem

- Input: n positive integers $w_1, w_2, \ldots, w_n$ indicating word frequency
- Output: a binary tree of n leaves, whose weights form $w_1, w_2, \ldots, w_n$ s.t. the cost of the tree is minimized

$$T^* = \arg\min_T B(T) = \arg\min_T \sum_{c \in C} \mathrm{freq}(c) \cdot d_T(c)$$

Step 1: Cast Optimization Problem

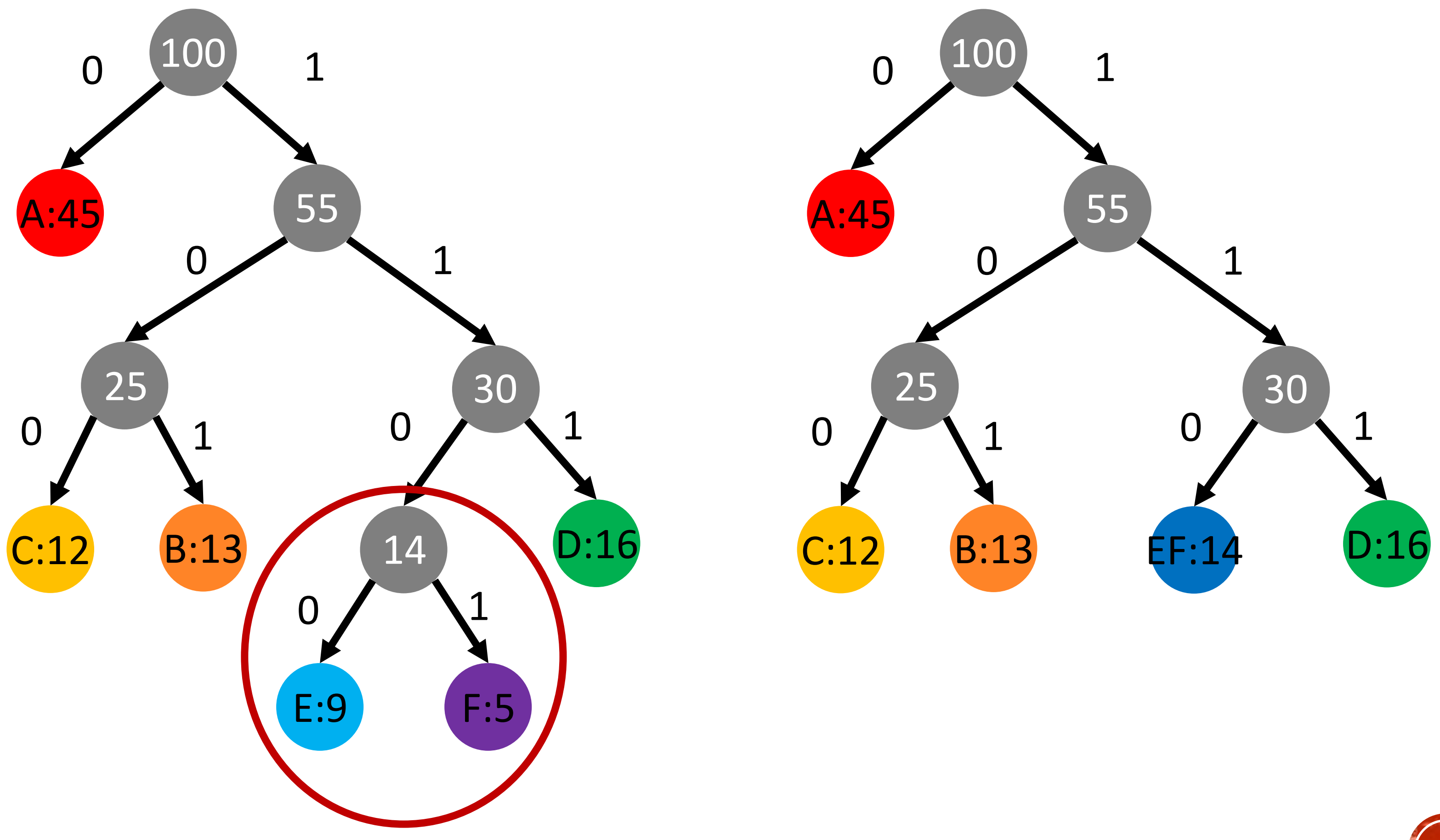
Prefix Code Problem
Input: n positive integers $w_1, w_2, \dots, w_n$ indicating word frequency
Output: a binary tree of n leaves with minimal cost

- Subproblem: merge two characters into a new one whose weight is their sum
  - `PC(i)`: prefix code problem for i leaves
  - Goal: `PC(n)`

`PC(n)` → `PC(n - 1)`

- Issues
  - It is not the subproblem of the original problem
  - The cost of two merged characters should be considered

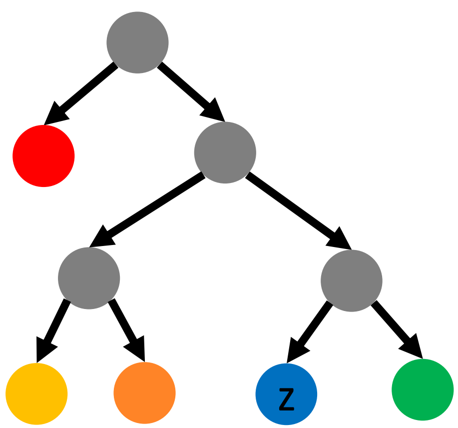
Example


100
0
1
A:45
55
0
1
25
30
0
1
0
1
C:12
B:13
14
D:16
0
1
E:9
F:5
100
0
1
A:45
55
0
1
25
30
0
1
0
1
C:12
B:13
EF:14
D:16

Step 2: Prove Optimal Substructure

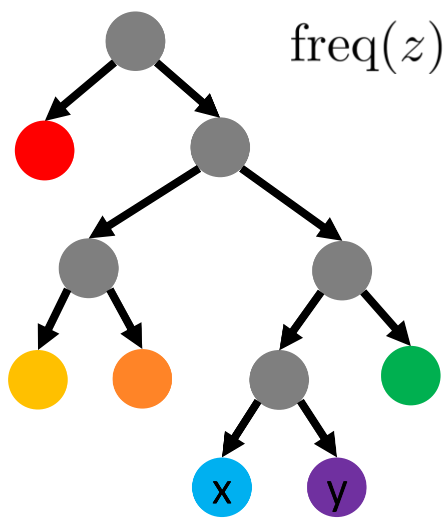
Prefix Code Problem
Input: n positive integers $w_1, w_2, \ldots, w_n$ indicating word frequency
Output: a binary tree of n leaves with minimal cost

- Suppose T' is an optimal solution to `PC(i, {w_{1…i-1}, z})`



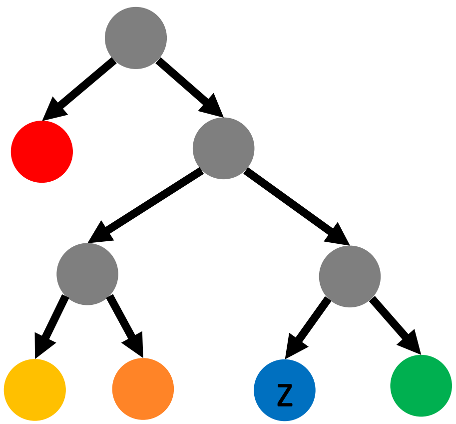
- T is an optimal solution to `PC(i+1, {w_{1…i-1}, x, y})`

$$\mathrm{freq}(z) = \mathrm{freq}(x) + \mathrm{freq}(y)$$

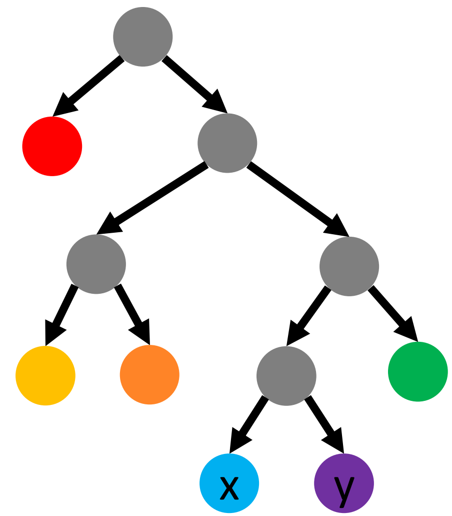


Step 2: Prove Optimal Substructure

- *T'*



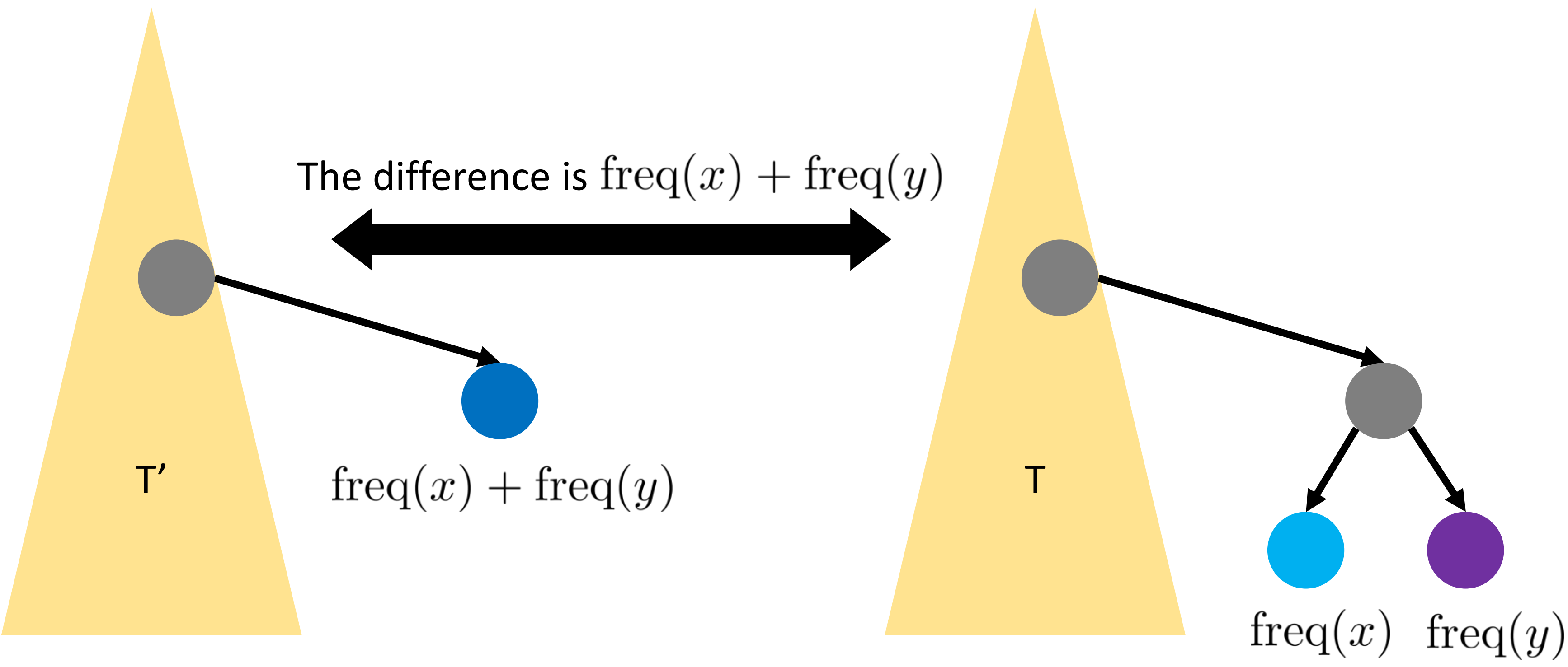
- *T*



$$
\begin{aligned}
B(T) &= B(T') - \text{freq}(z)d_{T'}(z) + \text{freq}(x)d_T(x) + \text{freq}(y)d_T(y) \\
&= B(T') - (\text{freq}(x) + \text{freq}(y))d_{T'}(z) + \text{freq}(x)(1 + d_{T'}(z)) + \text{freq}(y)(1 + d_{T'}(z)) \\
&= B(T') + \text{freq}(x) + \text{freq}(y)
\end{aligned}
$$

Step 2: Prove Optimal Substructure

- Optimal substructure: T' is OPT if and only if T is OPT

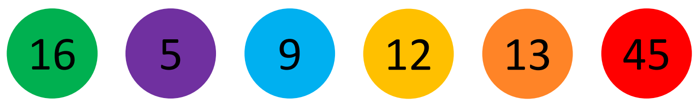


Greedy Algorithm Design

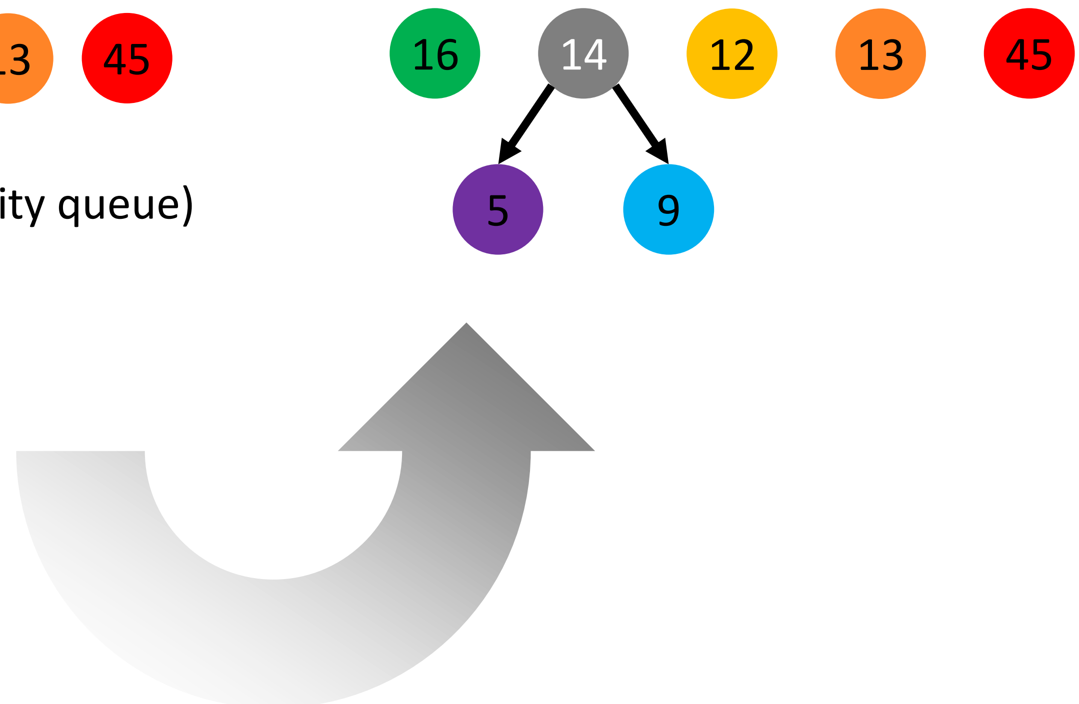
Prefix Code Problem
Input: n positive integers $w_1, w_2, \ldots, w_n$ indicating word frequency
Output: a binary tree of n leaves with minimal cost

- Greedy choice: merge repeatedly until one tree left
  - Select two trees x, y with minimal frequency roots $\mathrm{freq}(x)$ and $\mathrm{freq}(y)$
  - Merge into a single tree by adding root z with the frequency $\mathrm{freq}(x) + \mathrm{freq}(y)$

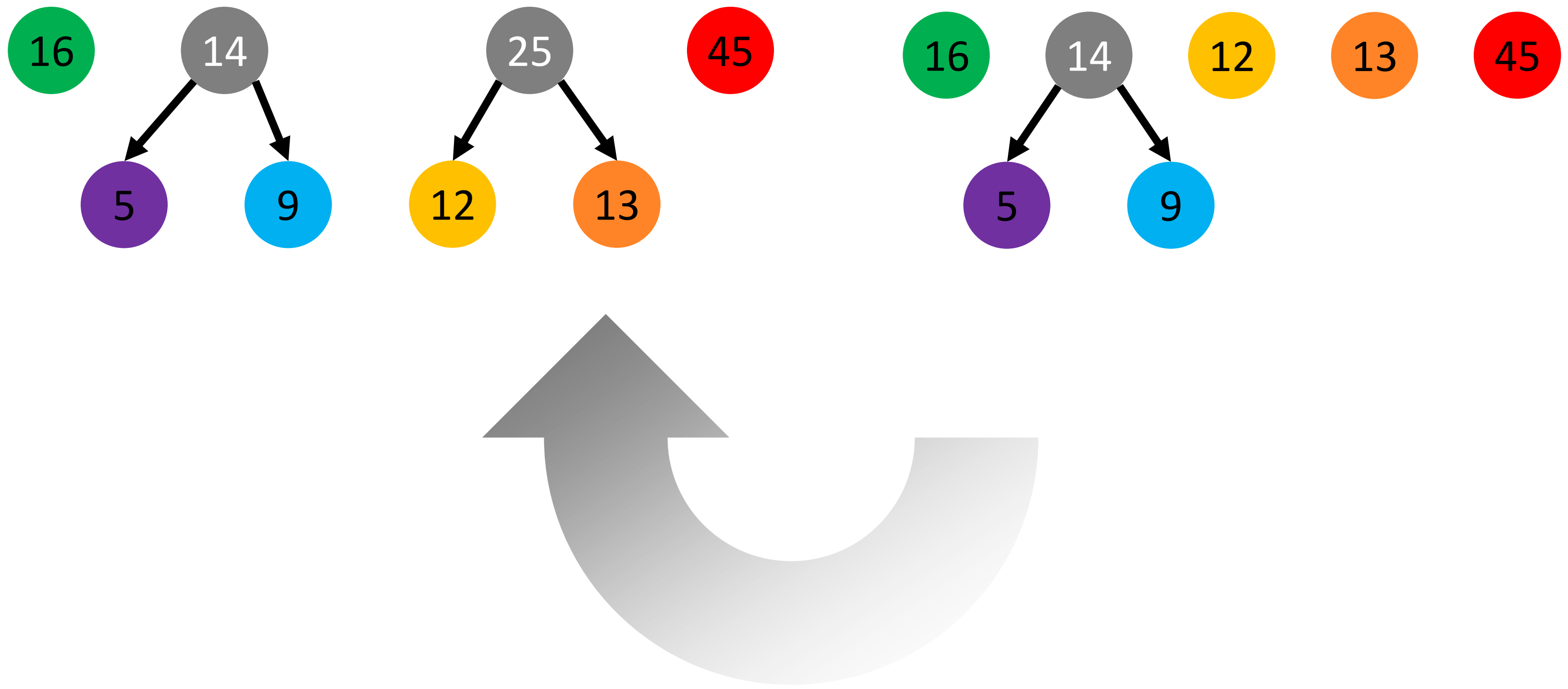
Example



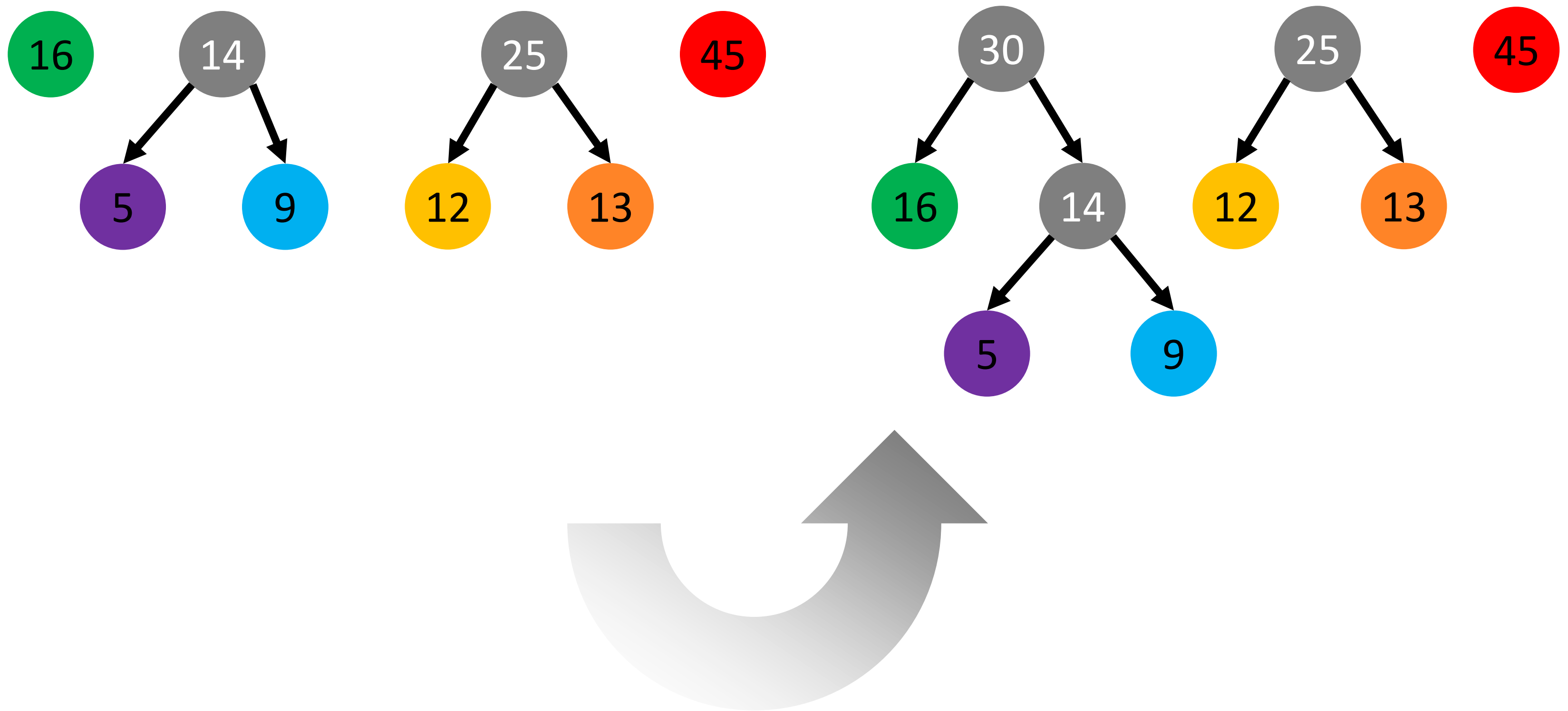
Initial set (store in a priority queue)



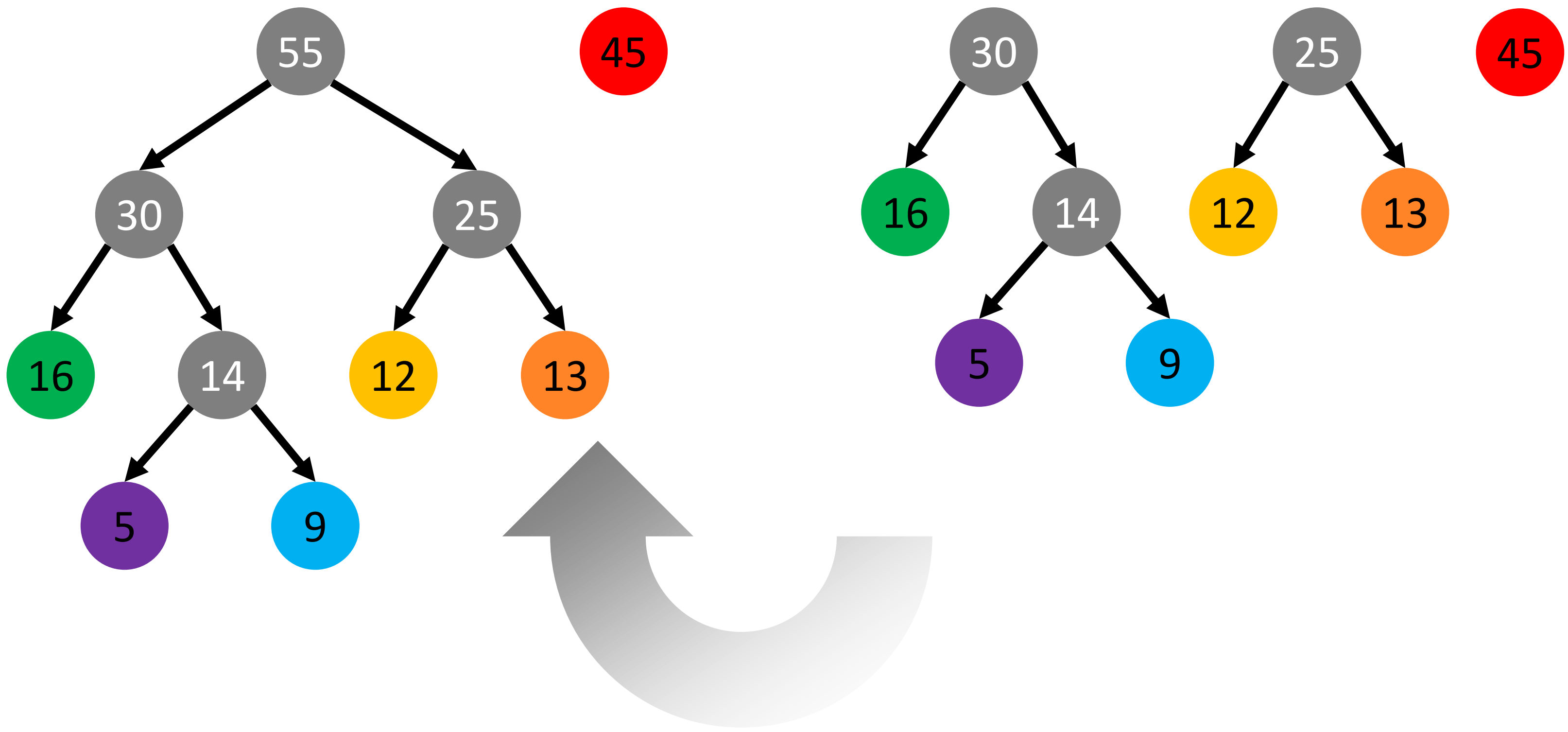
Example



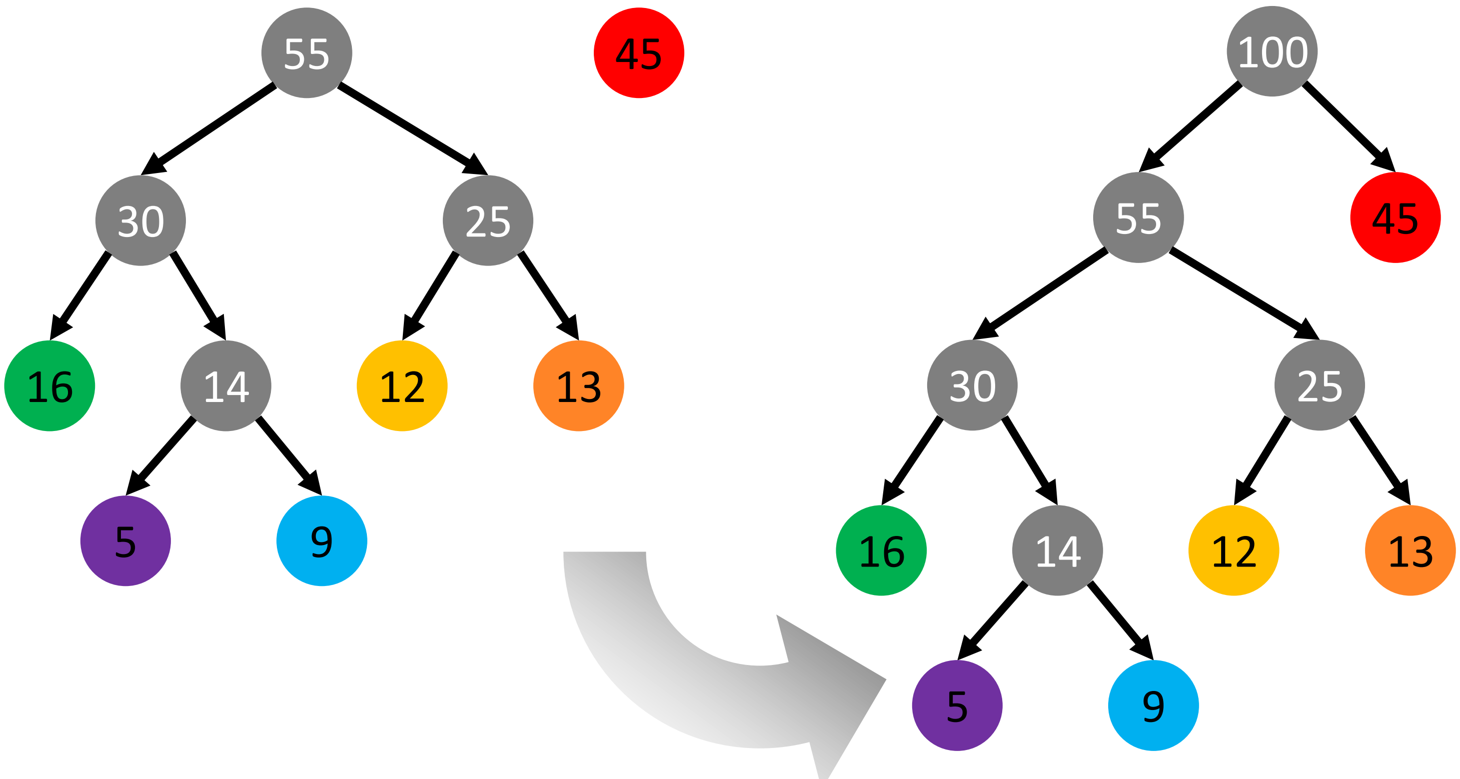
Example



Example



Example

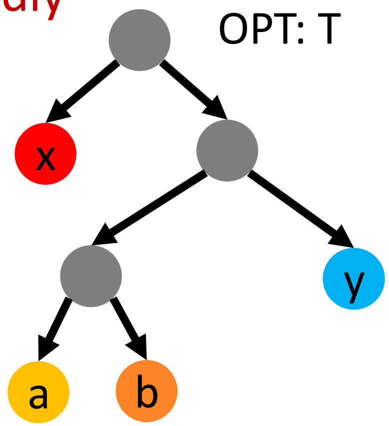


Step 3: Prove Greedy-Choice Property

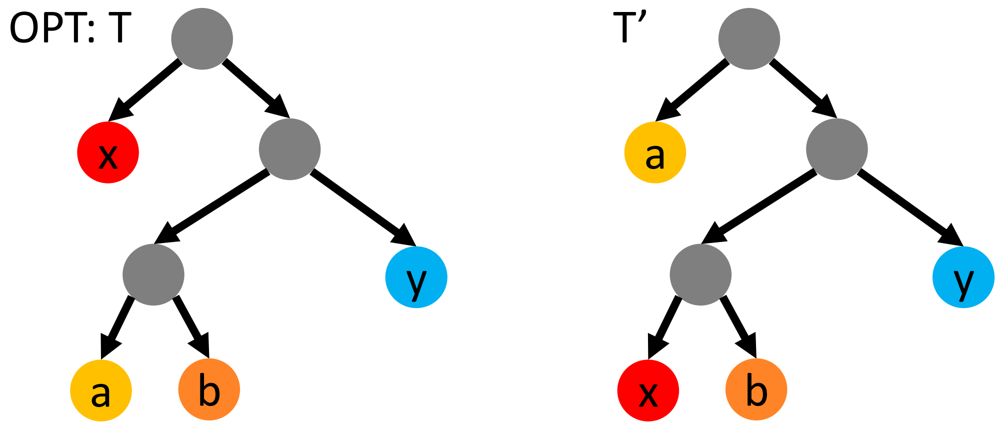
Prefix Code Problem
Input: n positive integers $w_1, w_2, \ldots, w_n$ indicating word frequency
Output: a binary tree of n leaves with minimal cost

- Greedy choice: merge two nodes with min weights repeatedly
- Proof via contradiction
  - Assume that there is no OPT including this greedy choice
    - x and y are two symbols with lowest frequencies
    - a and b are siblings with largest depths
    - WLOG, assume $\text{freq}(a) \le \text{freq}(b)$ and $\text{freq}(x) \le \text{freq}(y)$

    → $\text{freq}(x) \le \text{freq}(a)$ and $\text{freq}(y) \le \text{freq}(b)$
  - Exchanging a with x and then b with y can make the tree equally or better



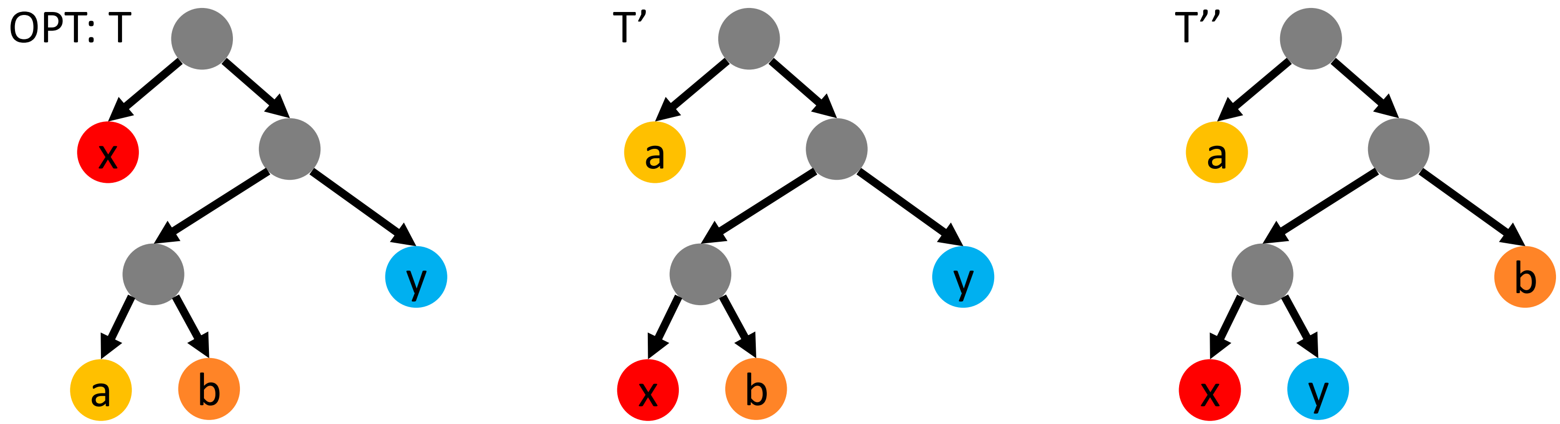
Step 3: Prove Greedy-Choice Property



$$
\begin{aligned}
B(T) - B(T') &= \textstyle\sum_{s \in S} \mathrm{freq}(s) d_T(s) - \sum_{s \in S} \mathrm{freq}(s) d_{T'}(s) \\
&= \mathrm{freq}(x) d_T(x) + \mathrm{freq}(a) d_T(a) - \mathrm{freq}(x) d_{T'}(x) - \mathrm{freq}(a) d_{T'}(a) \\
&= \mathrm{freq}(x) d_T(x) + \mathrm{freq}(a) d_T(a) - \mathrm{freq}(x) d_T(a) - \mathrm{freq}(a) d_T(x) \\
&= (\mathrm{freq}(a) - \mathrm{freq}(x))(d_T(a) - d_T(x)) \ \geq 0 \quad \because \mathrm{freq}(x) \leq \mathrm{freq}(a)
\end{aligned}
$$

- Because T is OPT, T' must be another optimal solution.

Step 3: Prove Greedy-Choice Property



$$
\begin{aligned}
B(T') - B(T'') &= \textstyle\sum_{s\in S} \mathrm{freq}(s) d_{T'}(s) - \sum_{s\in S} \mathrm{freq}(s) d_{T''}(s) \\
&= \mathrm{freq}(y) d_{T'}(y) + \mathrm{freq}(b) d_{T'}(b) - \mathrm{freq}(y) d_{T''}(y) - \mathrm{freq}(b) d_{T''}(b) \\
&= \mathrm{freq}(y) d_{T'}(y) + \mathrm{freq}(b) d_{T'}(b) - \mathrm{freq}(y) d_{T'}(b) - \mathrm{freq}(b) d_{T'}(y) \\
&= (\mathrm{freq}(b) - \mathrm{freq}(y))(d_{T'}(b) - d_{T'}(y)) \geq 0 \quad \because \mathrm{freq}(y) \leq \mathrm{freq}(b)
\end{aligned}
$$

- Because T' is OPT, T'' must be another optimal solution.

Practice: prove the optimal tree must be a full tree

Correctness and Optimality

- Theorem: Huffman algorithm generates an optimal prefix code
- Proof
  - Use induction to prove: Huffman codes are optimal for n symbols
    - $n = 2$, trivial
    - For a set S with $n + 1$ symbols,
      1. Based on the greedy choice property, two symbols with minimum frequencies are siblings in T
      2. Construct T' by replacing these two symbols x and y with z s.t. $S' = (S \backslash \{x, y\}) \cup \{z\}$ and $\mathrm{freq}(z) = \mathrm{freq}(x) + \mathrm{freq}(y)$
      3. Assume T' is the optimal tree for n symbols by inductive hypothesis
      4. Based on the optimal substructure property, we know that when T' is optimal, T is optimal too (case $n + 1$ holds)

This induction proof framework can be applied to prove its ***optimality*** using the **optimal substructure** and the **greedy choice property**.

Pseudo Code

Prefix Code Problem
Input: n positive integers $w_1, w_2, \dots, w_n$ indicating word frequency
Output: a binary tree of n leaves with minimal cost

```
Huffman(S)
  n = |S|
  Q = Build-Priority-Queue(S)
  for i = 1 to n - 1
    allocate a new node z
    z.left = x = Extract-Min(Q)
    z.right = y = Extract-Min(Q)
    freq(z) = freq(x) + freq(y)
    Insert(Q, z)
    Delete(Q, x)
    Delete(Q, y)
  return Extract-Min(Q) // return the prefix tree
```

$$T(n) = \Theta(n \log n)$$

Drawbacks of Huffman Codes

- Huffman's algorithm is optimal for a symbol-by-symbol coding with a known input probability distribution
- Huffman's algorithm is sub-optimal when
  - blending among symbols is allowed
  - the probability distribution is unknown
  - symbols are not independent

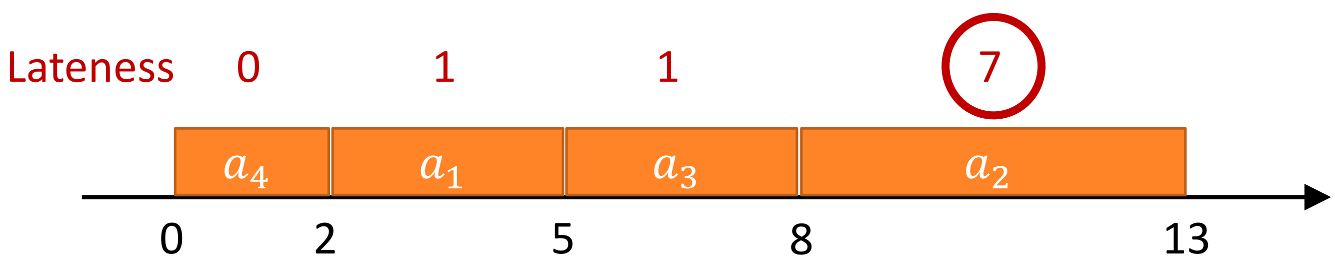
Scheduling to Minimize Lateness

Scheduling to Minimize Lateness

- Input: a finite set $S = \{a_1, a_2, \ldots, a_n\}$ of n tasks, their processing time $t_1, t_2, \ldots, t_n$, and integer deadlines $d_1, d_2, \ldots, d_n$

Job	**1**	**2**	**3**	**4**
Processing Time (t_i)	3	5	3	2
Deadline (d_i)	4	6	7	8

- Output: a schedule that minimizes the maximum lateness



Scheduling to Minimize Lateness

Scheduling to Minimize Lateness Problem
Input: n tasks with their processing time $t_1, t_2, \dots, t_n$, and deadlines $d_1, d_2, \dots, d_n$
Output: the schedule that minimizes the maximum lateness

- Let a schedule H contains $s(H,j)$ and $f(H,j)$ as the start time and finish time of job j
  - $f(H,j) - s(H,j) = t_j$
  - Lateness of job j in H is $L(H,j) = \max\{0, f(H,j) - d_j\}$
- The goal is to minimize $\max_j L(H,j) = \max_j\{0, f(H,j) - d_j\}$

Possible Greedy Choices

Scheduling to Minimize Lateness Problem
Input: n tasks with their processing time $t_1, t_2, \dots, t_n$, and deadlines $d_1, d_2, \dots, d_n$
Output: the schedule that minimizes the maximum lateness

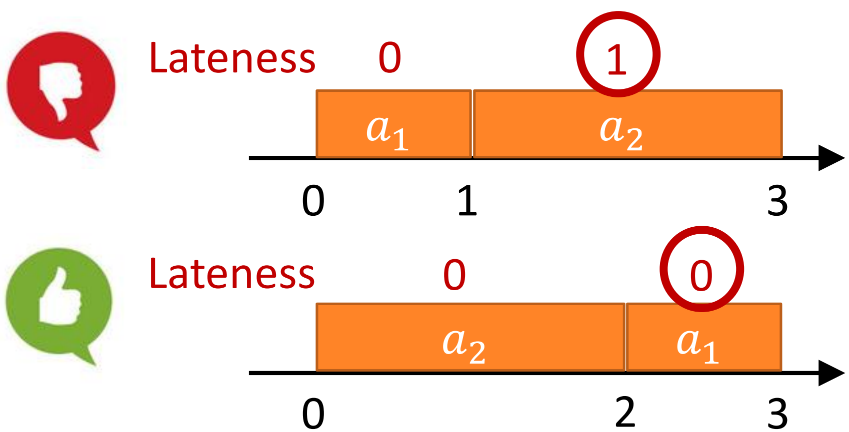
- Greedy idea
  - Shortest-processing-time-first w/o idle time?
  - Earliest-deadline-first w/o idle time?

Practice: prove that any schedule w/ idle is not optimal

Possible Greedy Choices

Scheduling to Minimize Lateness Problem
Input: n tasks with their processing time $t_1, t_2, \dots, t_n$, and deadlines $d_1, d_2, \dots, d_n$
Output: the schedule that minimizes the maximum lateness

- Idea
  - **Shortest-processing-time-first w/o idle time?**



Job	1	2
Processing Time (t_i)	1	2
Deadline (d_i)	10	2

Possible Greedy Choices

Scheduling to Minimize Lateness Problem
Input: n tasks with their processing time $t_1, t_2, \ldots, t_n$, and deadlines $d_1, d_2, \ldots, d_n$
Output: the schedule that minimizes the maximum lateness

- Idea
  - **Earliest-deadline-first w/o idle time?**
- Greedy algorithm

```
Min-Lateness(n, t[], d[])
  sort tasks by deadlines s.t. d[1]≤d[2]≤ ...≤d[n]
  ct = 0 // current time
  for j = 1 to n
    assign job j to interval (ct, ct + t[j])
    s[j] = ct
    f[j] = s[j] + t[j]
    ct = ct + t[j]
  return s[], f[]
```

$$T(n) = \Theta(n \log n)$$

Prove Correctness
– Greedy-Choice Property

Scheduling to Minimize Lateness Problem
Input: n tasks with their processing time $t_1, t_2, \dots, t_n$, and deadlines $d_1, d_2, \dots, d_n$
Output: the schedule that minimizes the maximum lateness

- Greedy choice: first select the task with the earliest deadline
- Proof via contradiction
  - Assume that there is no OPT including this greedy choice
    - If OPT processes a_1 as the i-th task (a_k), we can switch a_k and a_1 into OPT'
  - The maximum lateness must be equal or lower → $L(\mathrm{OPT}') \leq L(\mathrm{OPT})$

exchange argument

Prove Correctness
– Greedy-Choice Property

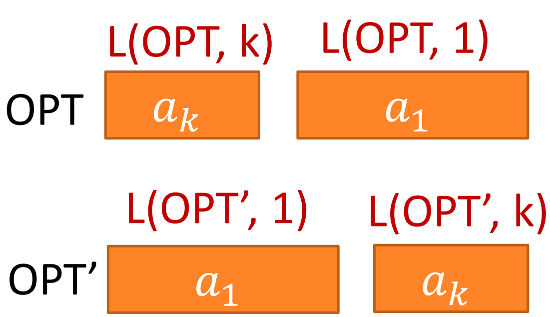
Scheduling to Minimize Lateness Problem
Input: n tasks with their processing time $t_1, t_2, \ldots, t_n$, and deadlines $d_1, d_2, \ldots, d_n$
Output: the schedule that minimizes the maximum lateness

- $L(\text{OPT'}) \le L(\text{OPT})$

$$\iff \max(L(\text{OPT'}, 1), L(\text{OPT'}, k)) \le \max(L(\text{OPT}, k), L(\text{OPT}, 1))$$

$$\iff \max(L(\text{OPT'}, 1), L(\text{OPT'}, k)) \le L(\text{OPT}, 1)$$

$$\iff L(\text{OPT'}, k) \le L(\text{OPT}, 1) \because L(\text{OPT'}, 1) \le L(\text{OPT}, 1)$$



<u>If a_k is not late in OPT':</u>

$$L(\text{OPT'}, k) = 0$$

<u>If a_k is late in OPT':</u>

$$\begin{aligned} L(\text{OPT'}, k) &= f(\text{OPT'}, k) - d_k \\ &= f(\text{OPT}, 1) - d_k \\ &\le f(\text{OPT}, 1) - d_1 \\ &= L(\text{OPT}, 1) \end{aligned}$$

Generalization of this property?

Prove Correctness
– No Inversions

Scheduling to Minimize Lateness Problem
Input: n tasks with their processing time $t_1, t_2, \dots, t_n$, and deadlines $d_1, d_2, \dots, d_n$
Output: the schedule that minimizes the maximum lateness

- There is an optimal scheduling w/o *inversions* given $d_1 \le d_2 \le \cdots \le d_n$
  - a_i and a_j are *inverted* if $d_i < d_j$ but a_j is scheduled before a_i
- Proof via contradiction
  - Assume that OPT has a_i and a_j that are inverted
  - Let OPT' = OPT but a_i and a_j are swapped
  - OPT' is equal or better than OPT → $L(\mathrm{OPT}') \le L(\mathrm{OPT})$

Prove Correctness
– No Inversions

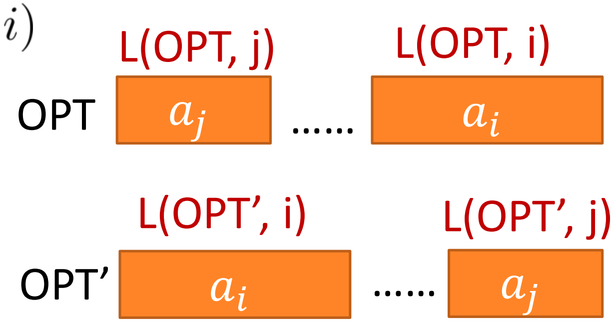
Scheduling to Minimize Lateness Problem
Input: n tasks with their processing time $t_1, t_2, \ldots, t_n$, and deadlines $d_1, d_2, \ldots, d_n$
Output: the schedule that minimizes the maximum lateness

- $L(\text{OPT'}) \leq L(\text{OPT})$

$$\iff \max(L(\text{OPT'}, i), L(\text{OPT'}, j)) \leq \max(L(\text{OPT}, j), L(\text{OPT}, i))$$

$$\iff \max(L(\text{OPT'}, i), L(\text{OPT'}, j)) \leq L(\text{OPT}, i) \because d_i < d_j$$

$$\iff L(\text{OPT'}, j) \leq L(\text{OPT}, i) \because L(\text{OPT'}, i) \leq L(\text{OPT}, i)$$



If a_j is not late in OPT’:

$$L(\text{OPT'}, j) = 0$$

If a_j is late in OPT’:

$$\begin{aligned} L(\text{OPT'}, j) &= f(\text{OPT'}, j) - d_j \\ &= f(\text{OPT}, i) - d_j \\ &\leq f(\text{OPT}, i) - d_i \\ &= L(\text{OPT}, i) \end{aligned}$$

Optimal Solution = Greedy Choice + Subproblem Solution

The earliest-deadline-first greedy algorithm is optimal

Task-Scheduling

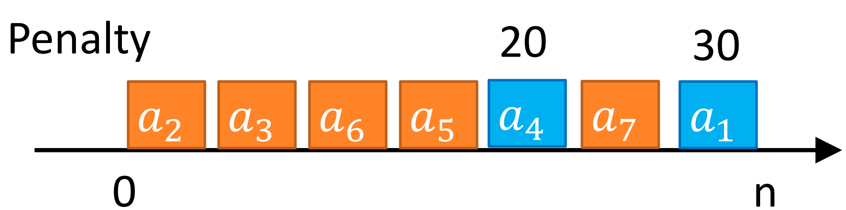
Textbook Chapter 16.5 – A task-scheduling problem as a matroid

Task-Scheduling Problem

- Input: a finite set $S = \{a_1, a_2, \ldots, a_n\}$ of n ***unit-time tasks***, their corresponding integer deadlines $d_1, d_2, \ldots, d_n$ $(1 \leq d_i \leq n)$, and nonnegative penalties $w_1, w_2, \ldots, w_n$ if a_i is not finished by time d_i

Job	1	2	3	4	5	6
Deadline (d_i)	1	2	3	4	4	6
Penalty (w_i)	30	60	50	20	70	10

- Output: a schedule that minimizes the total penalty

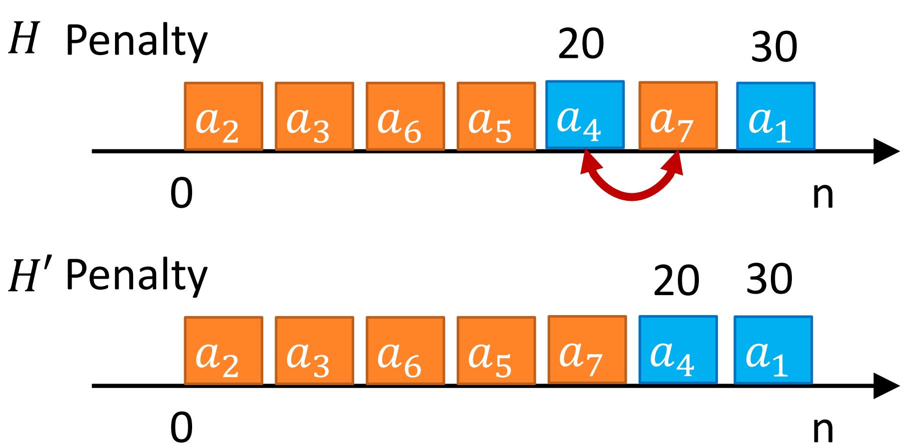


Task-Scheduling Problem

Task-Scheduling Problem
Input: n tasks with their deadlines $d_1, d_2, \dots, d_n$ and penalties $w_1, w_2, \dots, w_n$
Output: the schedule that minimizes the total penalty

- Let a schedule H is the OPT
  - A task a_i is <u>late</u> in H if $f(H, i) > d_j$
  - A task a_i is <u>early</u> in H if $f(H, i) \le d_j$
  - We can have an **early-first** schedule H' with the same total penalty (OPT)

Task	1	2	3	4	5	6	7
d_i	1	2	3	4	4	4	6
w_i	30	60	40	20	50	70	10



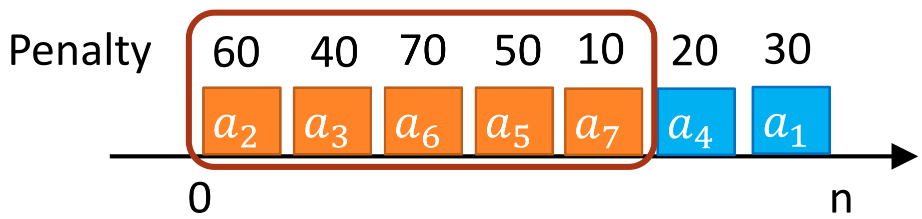
If the late task proceeds the early task, switching them makes the early one earlier and late one still late

Possible Greedy Choices

Task-Scheduling Problem
Input: n tasks with their deadlines $d_1, d_2, \ldots, d_n$ and penalties $w_1, w_2, \ldots, w_n$
Output: the schedule that minimizes the total penalty

- Rethink the problem: *"maximize the total penalty for the set of early tasks"*

Task	1	2	3	4	5	6	7
d_i	1	2	3	4	4	4	6
w_i	30	60	40	20	50	70	10

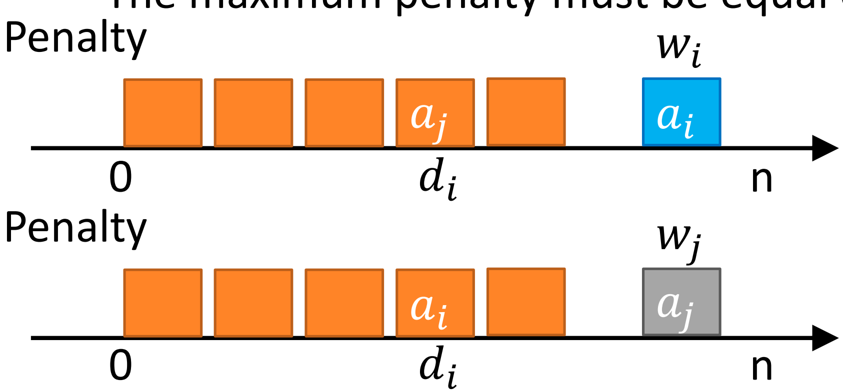


- Greedy idea
  - Largest-penalty-first w/o idle time?
  - Earliest-deadline-first w/o idle time?

Prove Correctness

Task-Scheduling Problem
Input: n tasks with their deadlines $d_1, d_2, \dots, d_n$ and penalties $w_1, w_2, \dots, w_n$
Output: the schedule that minimizes the total penalty

- Greedy choice: select the **largest-penalty** task into the early set if *feasible*
- Proof via contradiction
  - Assume that there is no OPT including this greedy choice
    - If OPT processes a_i after d_i, we can switch a_j and a_i into OPT'
  - The maximum penalty must be equal or lower, because $w_i \geq w_j$



$w_i \geq w_k$ for all a_k in the early set

Prove Correctness

Task-Scheduling Problem
Input: n tasks with their deadlines $d_1, d_2, \ldots, d_n$ and penalties $w_1, w_2, \ldots, w_n$
Output: the schedule that minimizes the total penalty

- Greedy algorithm

```
Task-Scheduling(n, d[], w[])
  sort tasks by penalties s.t. w[1] ≥ w[2] ≥ … ≥ w[n]
  for i = 1 to n
    find the latest available index j <= d[i]
    if j > 0
      A = A ∪ {i}
      mark index j unavailable
return A // the set of early tasks
```

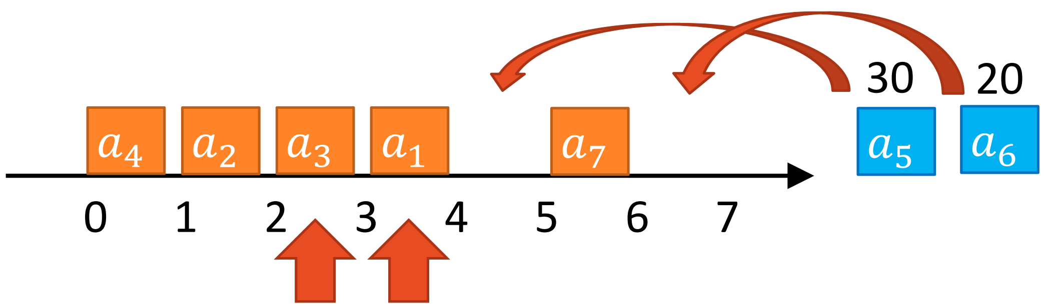
$$T(n) = O(n^2)$$

Can it be better?

Practice: reduce the time for finding the latest available index

Example Illustration

Job	1	2	3	4	5	6	7
Deadline (d_i)	4	2	4	3	1	4	6
Penalty (w_i)	70	60	50	40	30	20	10



Total penalty = 30 + 20 = 50

Practice: how about the greedy algorithm using "earliest-deadline-first"

Concluding Remarks

- "Greedy": always makes the choice that looks best at the moment in the hope that this choice will lead to a globally optimal solution
- When to use greedy
  - Whether the problem has optimal substructure
  - Whether we can make a greedy choice and remain only one subproblem
  - Common for optimization problem

Optimal Solution = Greedy Choice + Subproblem Solution

- Prove for correctness
  - Optimal substructure
  - Greedy choice property

Question?

Important announcement will be sent to @ntu.edu.tw mailbox
& post to the course website

Course Website: http://ada17.csie.org

Email: ada-ta@csie.ntu.edu.tw