Greedy Algorithms

Mohammad Javad Dousti

Changelog

- Rev. 1
  - Used a more consistent notation (i.e., f_x and f_y) for frequency of x and y nodes. (slide 26)
- Rev. 2
  - If it *is* different … (slide 9)

Overview

- Introduction
- Fractional knapsack problem
- Activity-selection problem
- Huffman codes
- Sample problems

Introduction

- Greedy algorithms, like dynamic programming, used to solve optimization problems.
  - Dynamic programming is powerful but yet overkill for certain problems.
- Problems of interest must exhibit
  - Optimal substructure (like dynamic programming).
  - The **greedy-choice** property.
    - When we have a choice to make, make the one that looks best *right now*.
    - Make a **locally optimal choice** in hope of getting a **globally optimal solution**.

- Clearly, greedy algorithms don't always yield optimal solutions.
  - …but for many problems they do ☺

Greedy Algorithms

Fractional Knapsack Problem

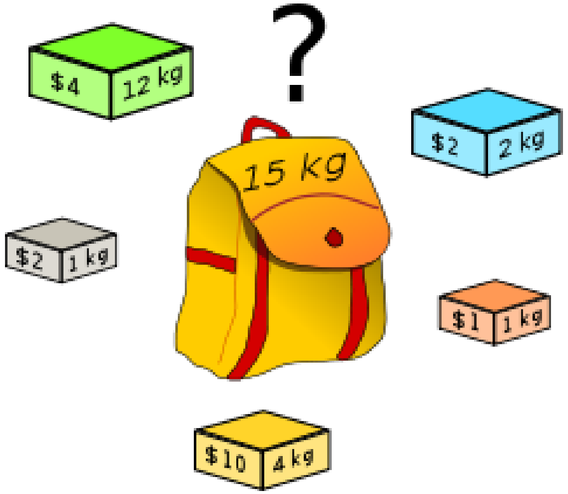
Fractional Knapsack Problem

- Given n objects, with weights $w_1, w_2, \ldots, w_n$ and values $v_1, v_2, \ldots, v_n$.

- **0-1 knapsack problem:** Objects can be taken as a whole.

$$\text{Maximize } \Sigma_{i=1}^{n} x_i v_i$$
$$\text{subject to } \Sigma_{i=1}^{n} x_i w_i \leq W \text{ and } x_i \in \{0, 1\}$$

  - Solution: Dynamic programming. We have seen it before.



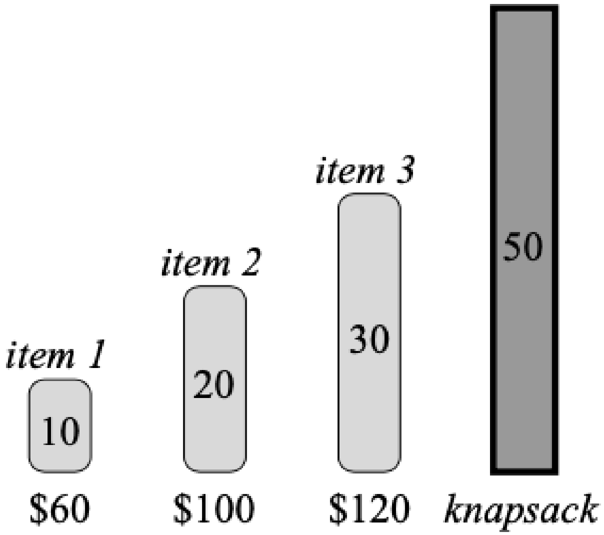
- **Fractional knapsack problem:** Objects can be taken partially.

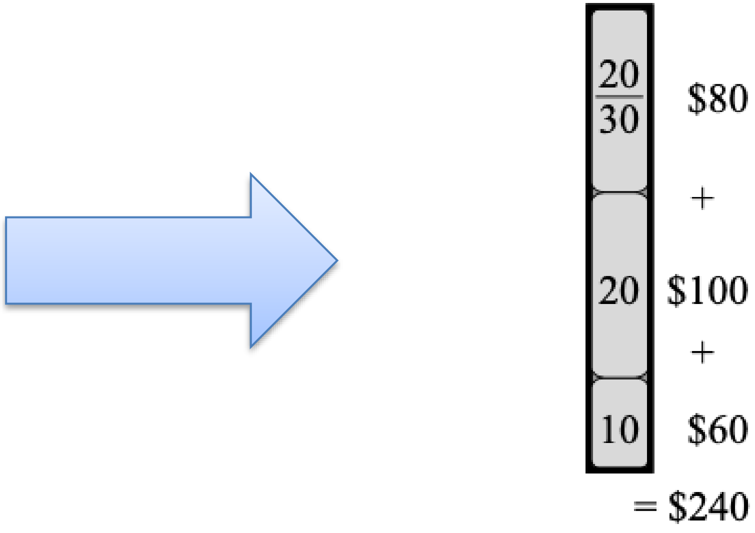
$$\text{Maximize } \Sigma_{i=1}^{n} x_i v_i$$
$$\text{subject to } \Sigma_{i=1}^{n} x_i w_i \leq W \text{ and } 0 \leq x_i \leq 1$$

  - Solution: Greedy algorithm. Let's see how it works.

A Greedy Solution

- In every step, take as much as possible of any remaining object with the highest $\frac{v_i}{w_i}$.
- Greedy choice: Taking an object with the highest $\frac{v_i}{w_i}$.
- Runtime complexity: $O(n \log n)$
- Example:





But does it yield an optimum solution?

Proof of Optimality

- Assume object indices are sorted with decreasing order by $\frac{v_i}{w_i}$.
- Suppose by contradiction, $ALG=\{p_1, p_2, \ldots, p_n\}$ is a greedy algorithm which isn't optimal, whereas $OPT=\{q_1, q_2, \ldots, q_n\}$ is. In other words, $\sum_{i=1}^{n} p_i v_i < \sum_{i=1}^{n} q_i v_i$.
- Let i be the smallest index where $p_i \neq q_i$.
  - Greedy choice tell us than $p_i > q_i$.
  - By the optimality of *OPT*, there exists a j where $p_j < q_j$.
- Now consider another solution $OPT' = \{q_1', q_2', \ldots, q_n'\}$, where $q_k' = q_k$ for all $k \neq i,j$.
  - *OPT'* takes a little more of item i and a little less of item j compared to OPT:
    - $q_i' = qi + \varepsilon$ $(q_i' \leq 1)$ and $q_j' = q_j - \varepsilon\, w_i/w_j$. The total weight remains the same: $\sum_{i=1}^{n} q_i' w_i = \sum_{i=1}^{n} q_i w_i$.
    - Total value increases though: $\sum_{i=1}^{n} q_i' v_i = \sum_{i=1}^{n} q_i v_i + \varepsilon v_i - \varepsilon \frac{w_i}{w_j} v_j > \sum_{i=1}^{n} q_i v_i$.
- *OPT'* is a better solution than *OPT*, which is a contradiction. Hence, *ALG* is optimum.

A Way to Prove Optimality of Greedy Algorithms

- Use contradiction.
- Find the most similar solution to the greedy solution.
  - If it’s identical to the greedy solution, the problem is solved.
  - If it is different than the greedy solution, make another solution, which is not worse than the optimal solution but is more similar to the greedy solution.
    - Use this new solution arrive reach a contradiction.

Wrap Up

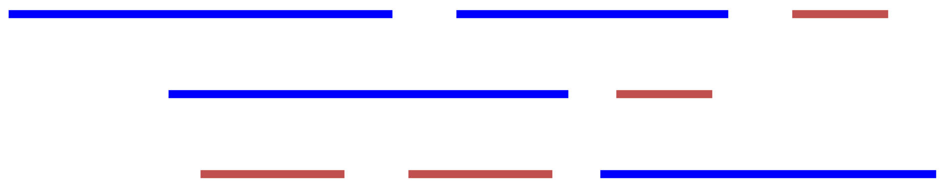
- Define the greedy choice.
- Prove that the greedy choice is optimal.
- Due to the optimal subproblem property, the remaining smaller problem is still similar to the original problem. Hence, using recursing and the greedy choice, one can solve the entire problem.
  - Usually, it’s easy (and recommended) to convert a recursive greedy algorithm to iterative.

Greedy Algorithms

Activity-Selection Problem

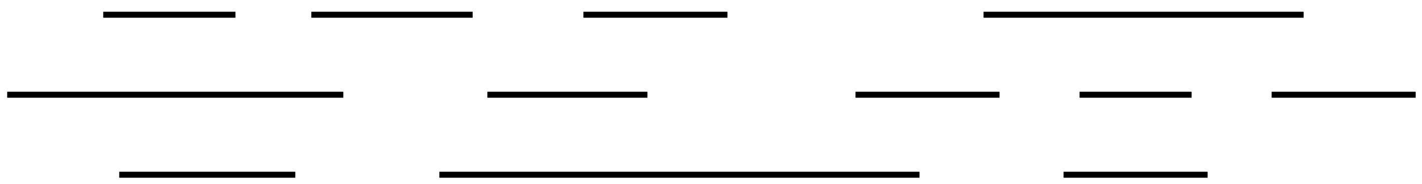
Activity-Selection Problem

- Given: n activities with start and finish times denoted by $[s_i, f_i]$.
- Problem: Find a maximum set of compatible activities. Activity i and j are compatible if they don't overlap, i.e., either $f_i < s_j$ or $f_j < s_i$.
- Example:

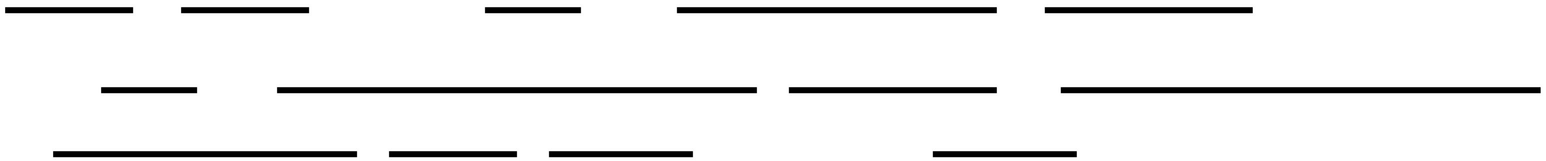


Greedy Choices: A Few Examples

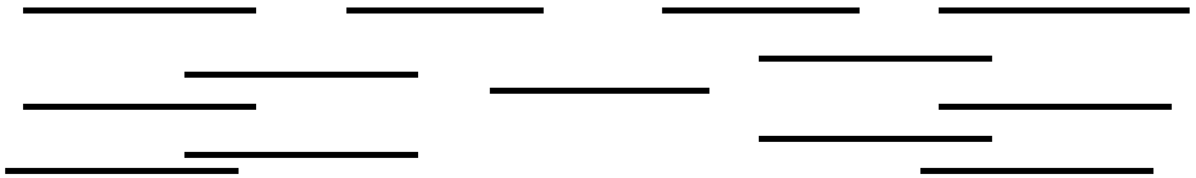
- ❑ Schedule earliest starting task



- ❑ Schedule shortest available task

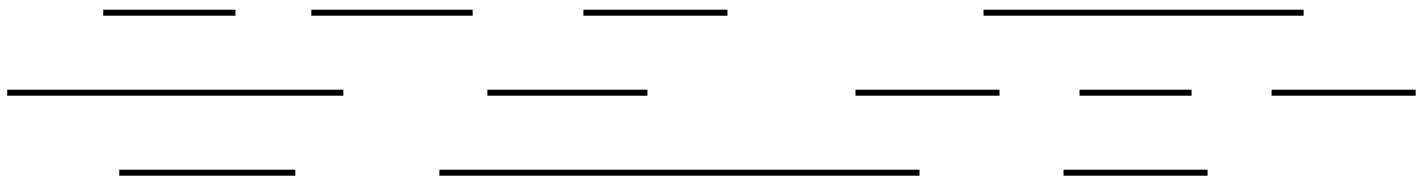


- ❑ Schedule task with fewest conflicting tasks

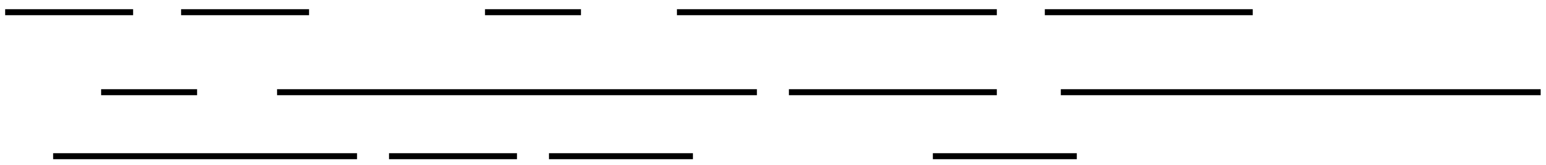


Greedy Choices: Earliest Finish Time

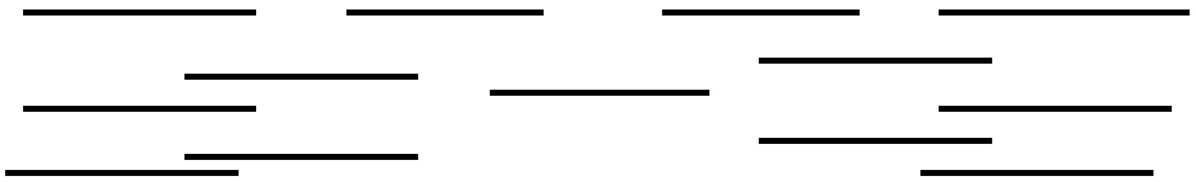
- Example 1



- Example 2



- Example 3



Greedy Algorithm

- Runtime complexity: $O(n)$
  - Considering the sort runtime complexity: $O(n \log n)$

```
ActivitySelector(s, f) {
    n = s.length
    A = {a₁}
    k = 1
    for m = 2 to n {
        if s[m] ≥ f[k] {
            A = A ∪ {aₘ}
            k = m
        }
    }
    return A
}
```

Proof of Optimality

- Suppose $A=\{a_1, a_2, \ldots, a_p\}$ be a greedy solution, $B=\{b_1, b_2, \ldots, b_q\}$ be the maximum-size subset of mutually compatible activities which , and $|A| < |B|$.
  - Among all optimum solutions, B is chosen such that it has the most common intervals with A.
  - Both A and B activities are sorted in increasing order of finish time.
- a_i and b_i are the first different activities in A and B.
- One can create B' as follows: $B'=B-\{b_i\}\cup\{a_i\}$.
  - B' is still a set of compatible activities because all activities would start after b_i finishes which happens after a_i finishes.
  - $|B'|=|B|$
- This process can continue until B becomes identical to A, i.e., $|A| = |B|$, which is a contradiction. Hence, the greedy solution is optimum.

Huffman Codes

Some slides are courtesy of Dr. Mahini.

Lossless Data Compression (1)

File

ABAA
CDAA
DDCC

12 characters

Encoding

A = 01010000
B = 01010001
C = 01110011
D = 01011111

Stored File

01010000 01010001 01010000 01010000
01110011 01011111 01010000 01010000
01011111 01011111 01110011 01110011

96 bits

Lossless Data Compression (2)

A new compression with 23 bits

File

ABAA
CDAA
DDCC

12 characters

Encoding

A = 0
B = 110
C = 10
D = 111

Stored File

0 110 0 0
10 111 0 0
111 111 10 10

23 bits

What is lossless data compression?

A = 00
B = 000
C = 100
D = 111

Stored File

000000

File

AAA

File

BB

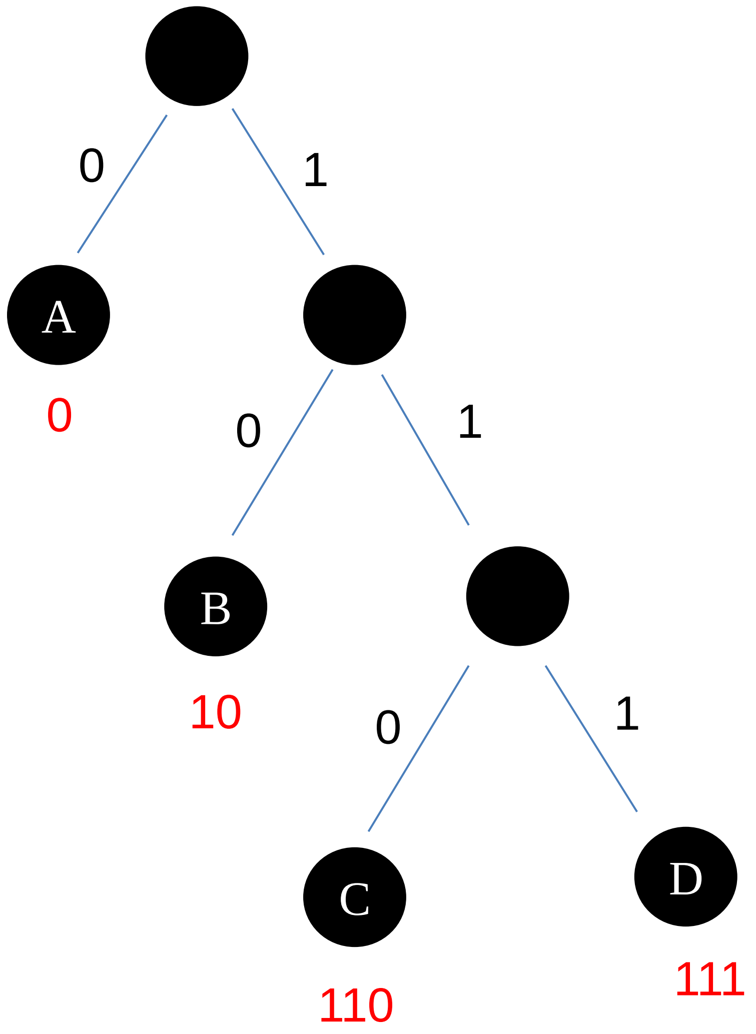
No code should be the prefix of the other one

Lossless Data Compression: Problem

Input: a file with n characters such that the frequency of character i is f_i

Goal: Assign code c_i with length of h_i to character i to minimize $\sum h_i \times f_i$ (length of the stored file) such that no code is the prefix of the other one.

You can model "prefix codes" with a binary tree

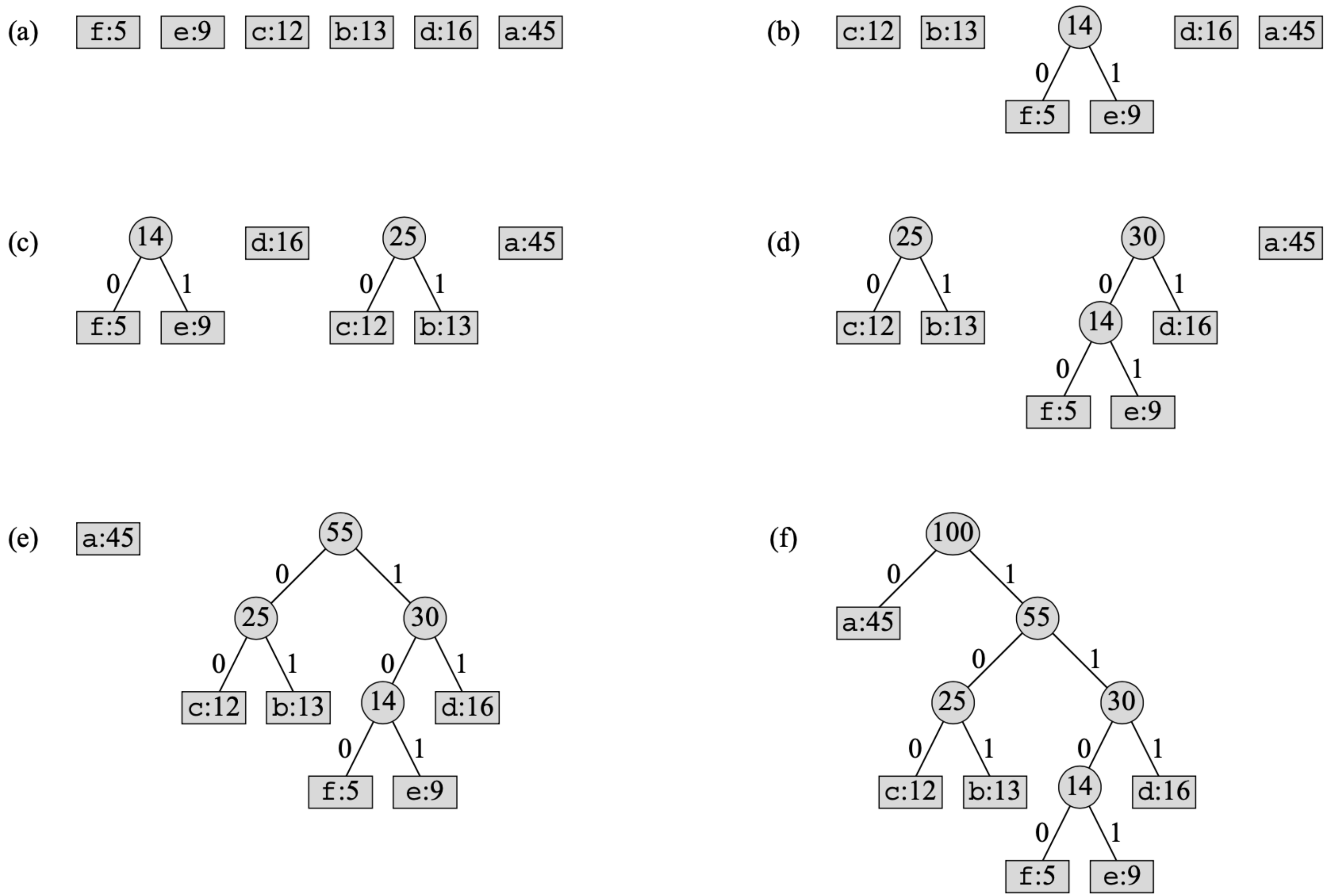


Huffman Codes: A Greedy Algorithm

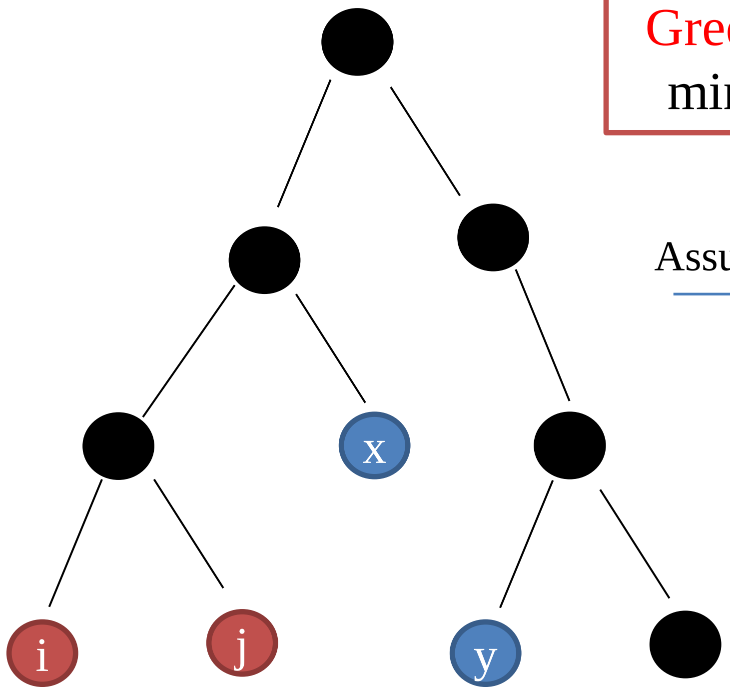
- Runtime complexity: $O(n \log n)$
- Note that the heap doesn't correspond to the Huffman tree.

```
HuffmanCode(C) {
    n = |C|
    Q = C   // Q is a binary min-heap
    for i = 1 to n - 1 {
        allocate a new node z
        z.left = x = Extract-Min(Q)
        z.right = y = Extract-Min(Q)
        z.freq = x.freq + y.freq
        Insert(Q, z)
    }
    return Extract-Min(Q)
}
```

Huffman Codes: Example

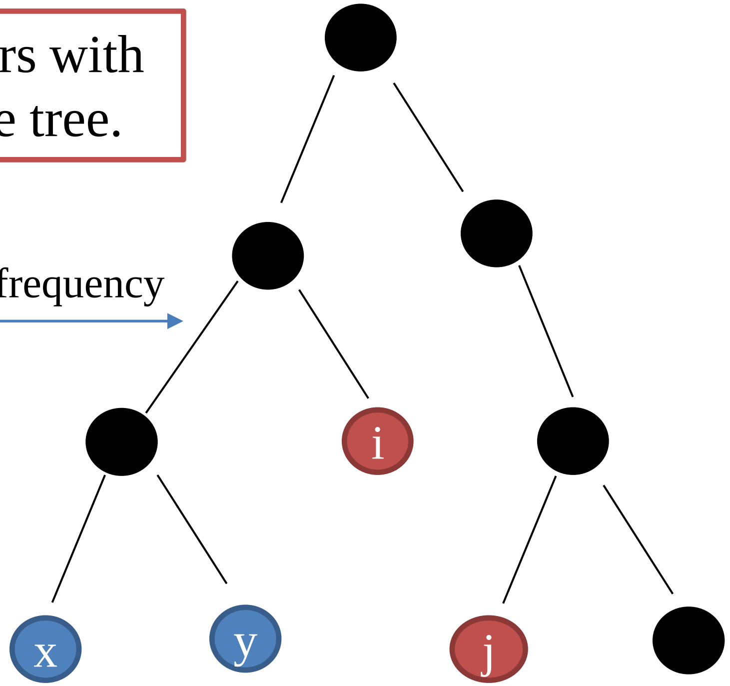


Greedy Choice



Greedy choice: The two characters with min frequency are siblings in the tree.

Assume *x* and *y* are characters with min frequency



$$h_i = h_j = h$$
$$h_X = h'$$
$$h_Y = h''$$

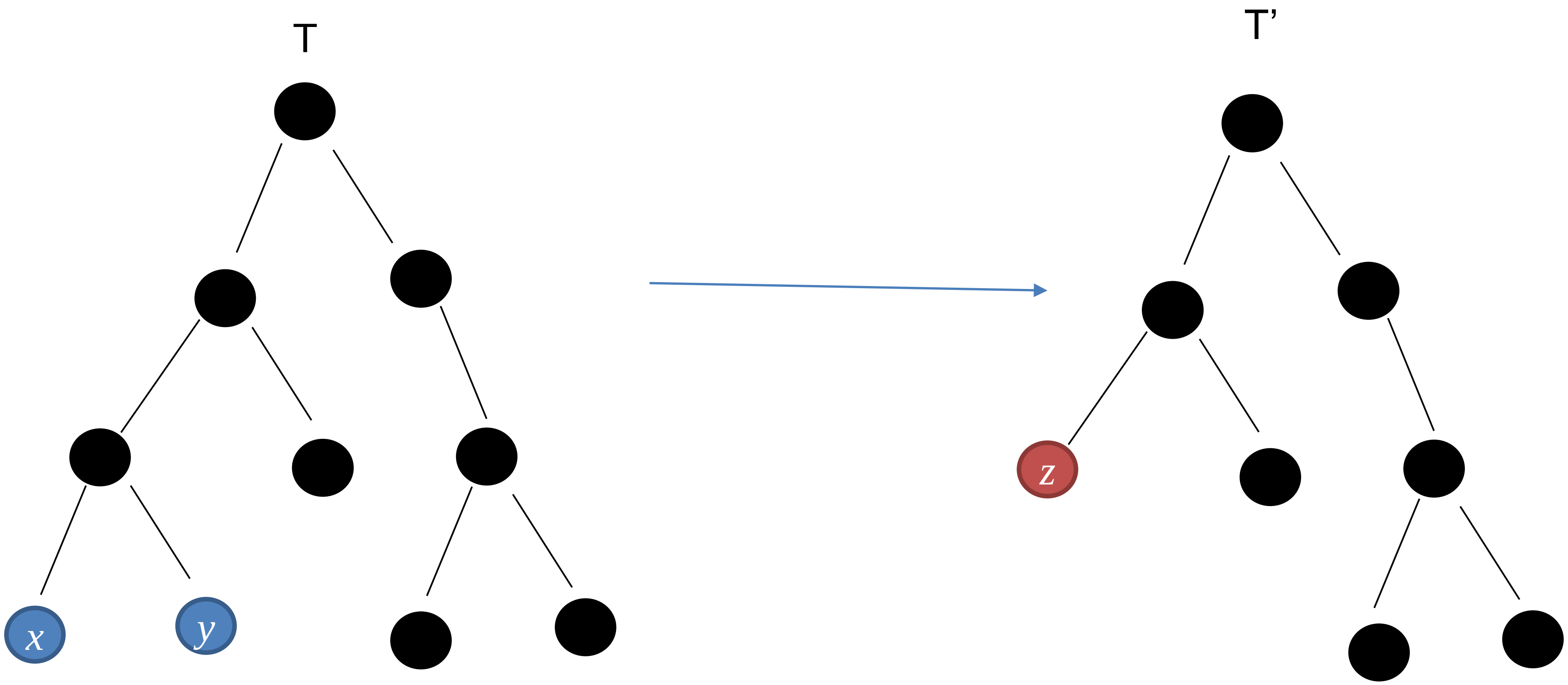
$$f_X \times h_X + f_y \times h_y + f_i \times h_i + f_j \times h_j$$

$$\boxed{f_X \times h' + f_y \times h'' + f_i \times h + f_j \times h} \geq \boxed{f_X \times h + f_y \times h + f_i \times h' + f_j \times h''}$$

Exchanging the lowest nodes (i.e., *i* and *j*) with *x* and *y* does not increase the cost.

Optimal Subproblem Property

- Do we have the same sub-problem?



$$\boxed{Cost(T) - Cost(T') = f_x h + f_y h - f_*(h-1) = (f_x + f_y)h - (f_x + f_y)(h-1) = f_x + f_y}$$

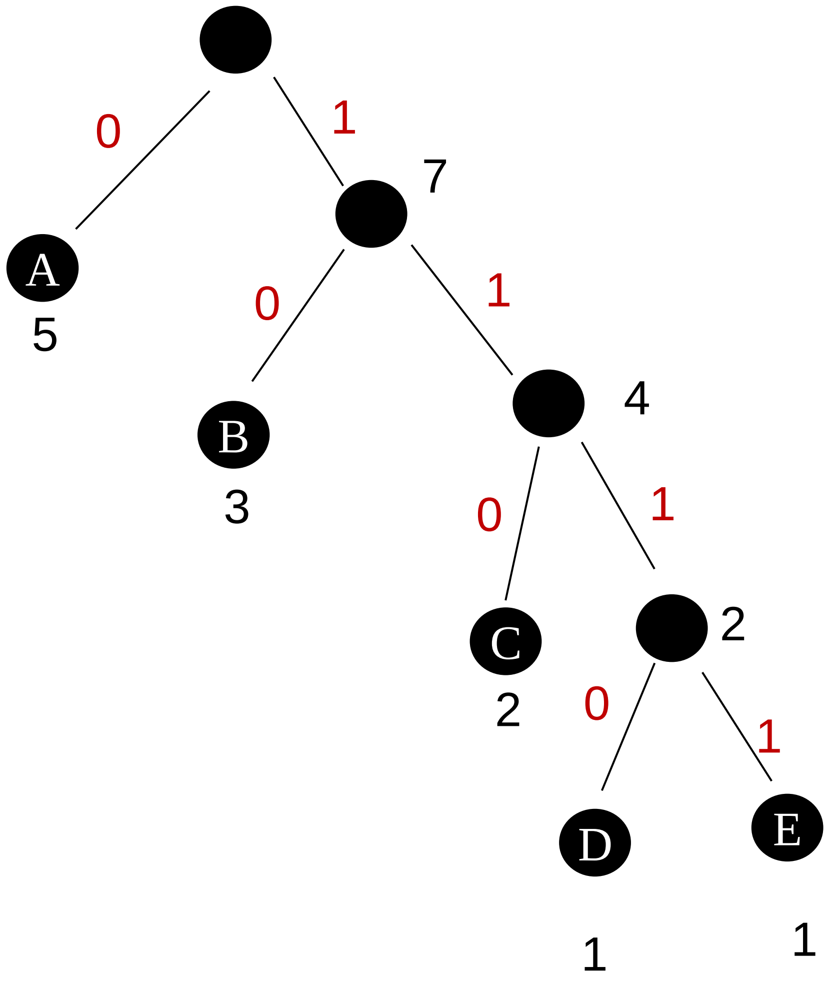
Constant

Optimal Subproblem Property (cont'd)

- *T* can be created from an optimal prefix tree *T'* by replacing a leaf node *z* with *x* and *y*.
- Suppose *T* does not represent an optimal prefix code, while *T''* does, i.e., $Cost(T'') < Cost(T)$.
  - We can assume than *x* and *y* are sibling leaf nodes in *T''*. Why?
    - Due to greedy choice property.
- Now suppose *T'''* is created from *T''* by merging *x* and *y* into *z* leaf node:
  - $Cost(T''') = Cost(T'') - f_x - f_y < Cost(T) - f_x - f_y = Cost(T')$
  - This is a contradiction as we assumed *T'* is an optimal tree. Hence, *T* represents an optimal prefix code.

Another Example

Character	Frequency	Code
A	5	0
B	3	10
C	2	110
D	1	1110
E	1	1111



Sample Problems

True or False?

- A greedy algorithm always makes the choice that looks best at the moment.
- Problems solved using dynamic programming cannot be solved thru greedy algorithms.
- If a problem can be solved using both the greedy method and dynamic programming, greedy will always give you a lower time complexity.

Fill in the Blanks

- A ________________ algorithm may produce an optimal solution when ________________ algorithm cannot, because ________________ algorithm will exclude some possible solutions while ________________ algorithm will examine all possible solutions.

  - Use either "greedy" or "dynamic programming".

Coin/Money Change Problem: Greedy Algorithm

- The objective of the *Coin Problem* is to come up with the minimum number of coins to pay X cents.
  - Consider the greedy approach to solving the coin problem for US coins (25¢, 10¢, 5¢, and 1¢):
    - We start with the largest coin (25¢ coin) and use it as many times as possible, then use as many 10¢ coins as possible on the remainder, then 5¢, then 1¢.
    - Prove or disprove that “this greedy algorithm always gives an optimal solution”, i.e. gives the minimum number of coins to pay X cents.
  - Now suppose that 5¢ coins are not allowed, only 1¢, 10¢, and 25¢.
    - Prove or disprove that “the corresponding greedy approach (25¢ then 10¢ then 1¢) always gives an optimal solution”.

Gradient Descent

- An approach used to find the local minimum of a function.