Greedy Algorithms
Huffman Encoding

(Class 25)

From Book’s Page No 431

Huffman Encoding: Correctness

- Here correctness means:
  - Huffman encoding generates optimal solution.
  - And also generate unique codes solving prefix codes problem.

- Huffman algorithm uses a greedy approach to generate a prefix code T that minimizes the expected length $B(T)$ of the encoded string.

- In other words, Huffman algorithm generates an optimum prefix code.
- The question that remains is that: is the algorithm correct?
- Recall that the cost of any encoding tree T is:

$$\mathrm{B(T)} = \mathrm{n} \sum_{x \in \mathrm{C}} \mathrm{p(x)}.\, \mathrm{d_T}\,\mathrm{(x)}$$

- Our approach to prove the correctness of Huffman Encoding will be to show that:
  - Any tree that differs from the one constructed by Huffman algorithm can be converted into one that is equal to Huffman’s tree without increasing its costs (e.g., storage cost).

- Note that the binary tree constructed by Huffman algorithm is a full binary tree.

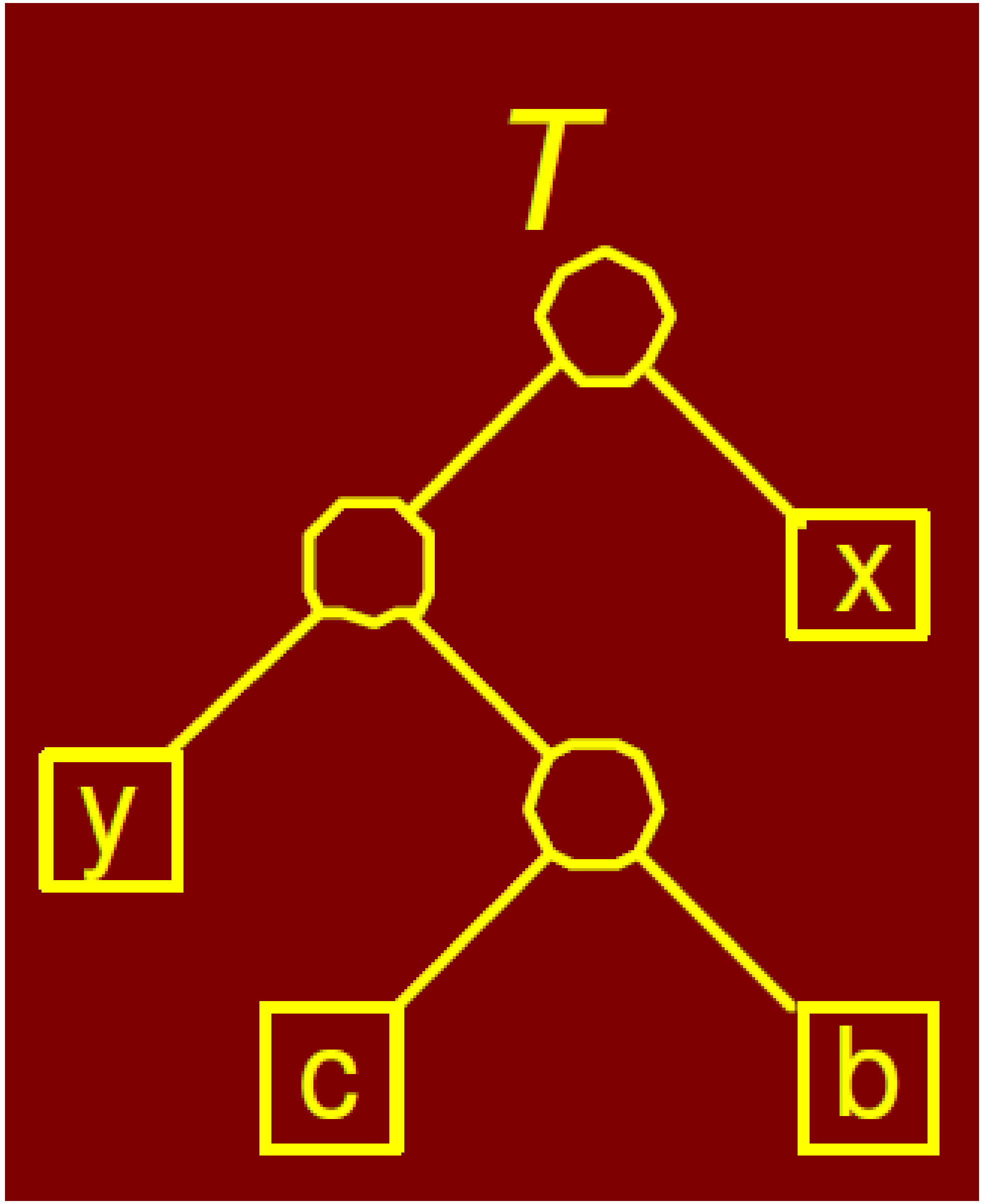
Claim:

- Consider two characters x and y with the smallest probabilities.
- Then there is optimal code tree in which these two characters are siblings at the maximum depth in the tree.

Proof:

- Let T be any optimal prefix code tree with two siblings b and c at the maximum depth of the tree.
- Assume that:

$$p(b) \leq p(c) \quad and \quad p(x) \leq p(y)$$


T
x
y
c
b

- Since x and y have the two smallest probabilities (we claimed this), it follows that:

$$p(x) \le p(b) \quad and \quad p(y) \le p(c)$$

- Since b and c are at the deepest level of the tree, we know that:

$$d(b) \ge d(x) \quad and \quad d(c) \ge d(y)$$

- Where d is the depth.

- Thus, we have:

$$p(b) - p(x) \geq 0$$

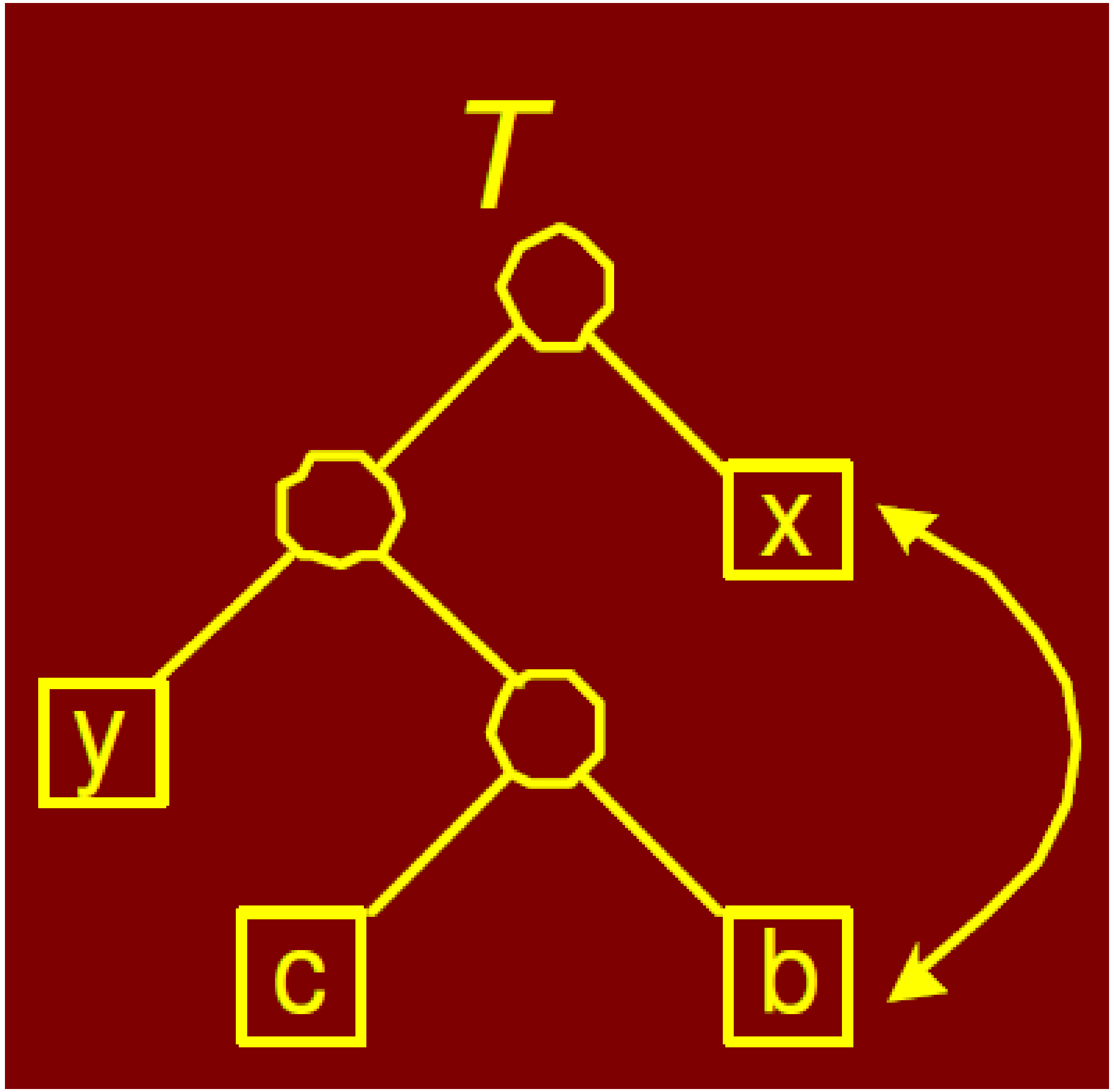
- And

$$d(b) - d(x) \geq 0$$

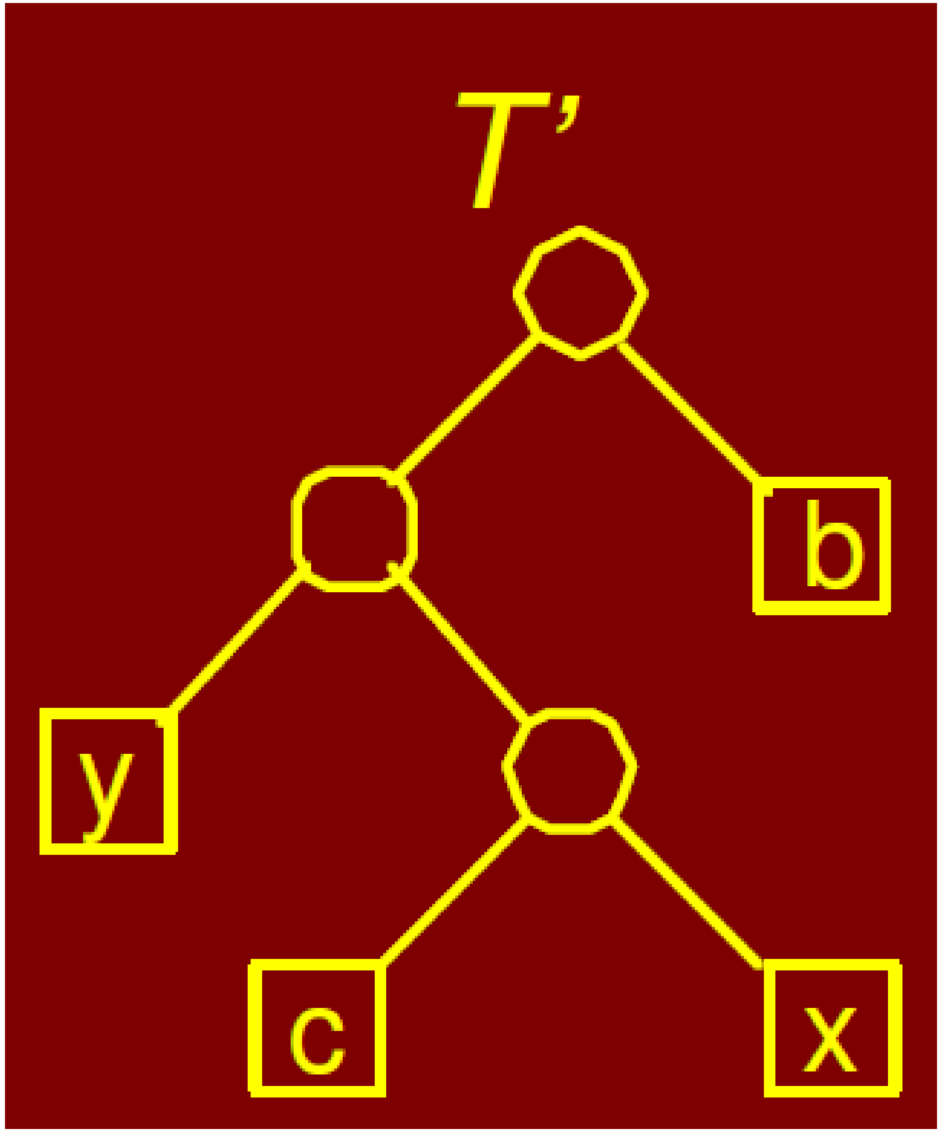
- Hence their product is non-negative. That is,

$$\big(p(b) - p(x)\big).\big(d(b) - d(x)\big) \geq 0$$

- Now swap the positions of x and b in the tree:



- This results in a new tree T':



- Let's see how the cost changes.
- The cost of T' is:

$$\mathrm{B}(T') = \mathrm{B}(\mathrm{T}) - \mathrm{p}(x)\mathrm{d}(x) + \mathrm{p}(x)\mathrm{d}(\mathrm{b}) - \mathrm{p}(\mathrm{b})\mathrm{d}(\mathrm{b}) + \mathrm{p}(\mathrm{b})\mathrm{d}(x)$$

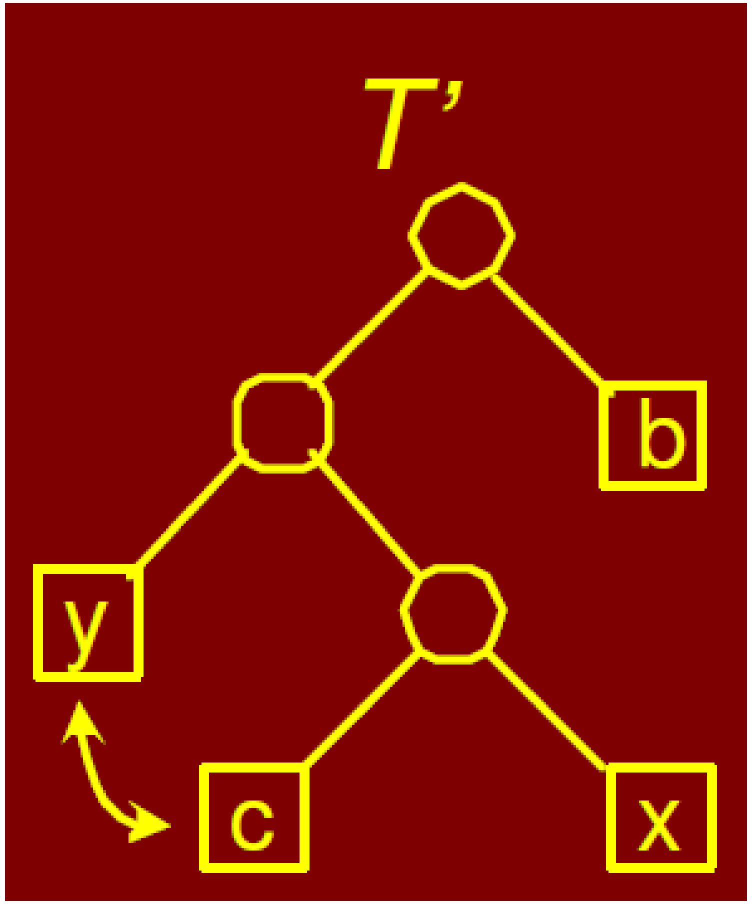
$$\mathrm{B}(T') = \mathrm{B}(\mathrm{T}) + \mathrm{p}(x)[\mathrm{d}(\mathrm{b}) - \mathrm{d}(x)] - \mathrm{p}(\mathrm{b})[\mathrm{d}(\mathrm{b}) - \mathrm{d}(x)]$$

$$\mathrm{B}(T') = \mathrm{B}(\mathrm{T}) - (\mathrm{p}(\mathrm{b}) - \mathrm{p}(\mathrm{x}))(\mathrm{d}(\mathrm{b}) - \mathrm{d}(\mathrm{x}))$$

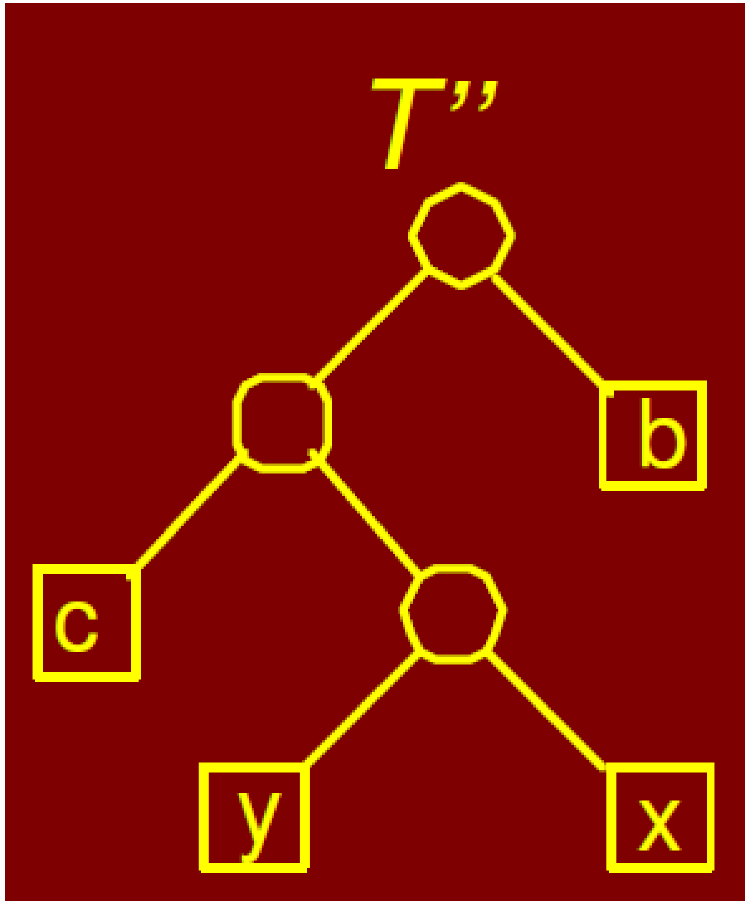
$$\mathrm{B}(T') \leq \mathrm{B}(T) \quad \text{because} \quad (p(\mathrm{b}) - p(\mathrm{x}))(\mathrm{d}(\mathrm{b}) - \mathrm{d}(\mathrm{x})) \geq 0$$

- Thus, the cost does not increase, implying that T' is an optimal tree.

- Now, by switching y with c we get the tree T''.
- Using a similar argument, we can show that T'' is also optimal.



$\Longrightarrow$



- The final tree T'' satisfies the claim we made earlier, i.e., consider two characters x and y with the smallest probabilities.
- Then there is optimal code tree in which these two characters are siblings at the maximum depth in the tree.
- The claim we just proved asserts that the first step of Huffman algorithm is the proper one to perform (the greedy step).

- The complete proof of correctness for Huffman algorithm follows by induction on n.
- **Claim:** Huffman algorithm produces the optimal prefix code tree.
- **Proof:** The proof is by induction on n, the number of characters.
- For the basis case, $n = 1$, the tree consists of a single leaf node, which is obviously optimal.
- We want to show it is true with exactly n characters.

- Suppose we have exactly n characters.
- The previous claim states that two characters x and y with the lowest probability will be siblings at the lowest level of the tree.
- Remove x and y and replace them with a new character z whose probability is:

$$p(z) = p(x) + p(y)$$

- Thus $n - 1$ characters remain.

- Consider any prefix code tree T made with this new set of $n-1$ characters.
- We can convert T into prefix code tree T' for the original set of n characters by replacing z with nodes x and y.
- This is essentially undoing the operation where x and y were removed and replaced by z.

- The cost of the new tree T' is:

$$\begin{aligned}
\mathrm{B}(\mathrm{T}') \quad &= \mathrm{B}(\mathrm{T}) - \mathrm{p}(z)\mathrm{d}(z) + \mathrm{p}(x)[\mathrm{d}(z) + 1] + \mathrm{p}(y)[\mathrm{d}(z) + 1] \\
&= \mathrm{B}(\mathrm{T}) - (\mathrm{p}(x) + \mathrm{p}(y))\mathrm{d}(z) + (\mathrm{p}(x) + \mathrm{p}(y))[\mathrm{d}(z) + 1] \\
&= \mathrm{B}(\mathrm{T}) + (\mathrm{p}(x) + \mathrm{p}(y))[\mathrm{d}(z) + 1 - \mathrm{d}(z)] \\
\mathrm{B}(\mathrm{T}') &= \mathrm{B}(\mathrm{T}) + \mathrm{p}(x) + \mathrm{p}(y)
\end{aligned}$$

- The cost changes but the change depends in no way on the structure of the tree T (T is for n− 1 characters).
- Therefore, to minimize the cost of the final tree T', we need to build the tree T on $n-1$ characters optimally.
- By induction, this is exactly what Huffman algorithm does.
- Thus, the final tree is optimal.

Activity Selection Problem

- The activity scheduling is a simple scheduling problem for which the greedy algorithm approach provides an optimal solution.
- We are given a set $S = \{a_1, a_2, \ldots, a_n\}$ of n activities that are to be scheduled to use some resource.
- Each activity a_i must be started at a given start time s_i and ends at a given finish time f_i.

- An example is that a number of lectures are to be given in a single lecture hall.
- The start and end times have been set up in advance.
- The lectures are to be scheduled.
- There is only one resource (e.g., lecture hall).

- Some start and finish times may overlap.
- Therefore, not all requests can be honored.
- We say that two activities a_i and a_j are non-interfering if their start-finish intervals do not overlap.
- For example, $(s_i, f_i) \cap (s_j, f_j) = \emptyset$.

- The activity selection problem is to select a maximum-size set of mutually non-interfering activities for use of the resource.
- So how do we schedule the largest number of activities on the resource?
- Intuitively, we do not like long activities because they occupy the resource and keep us from honoring other requests.

- This suggests the greedy strategy:
  - Repeatedly select the activity with the smallest duration $(f_i - s_i)$ and schedule it, provided that it does not interfere with any previously scheduled activities.

- Unfortunately, this turns out to be non-optimal.
- In the next class, we will see the optimal way to solve this problem.