Lecture 10: Huffman Coding

Encoding

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Encoding: Replace characters by corresponding codewords.

Q: How to design a code to minimize the length of the encoded message?

Ex: For a file with 100,000 characters with distribution in the table above, the fixed-length code requires

$$3 \cdot 100{,}000 = 300{,}000 \text{ bits}$$

The variable-length code requires

$$(45 \cdot 1 + 13 \cdot 3 + 12 \cdot 3 + 16 \cdot 3 + 9 \cdot 4 + 5 \cdot 4) \cdot 1000 = 224{,}000 \text{ bits}$$

Decoding

Decoding: Replace codewords by corresponding characters.

C_1 = {a = 00, b = 01, c = 10, d = 11}.
C_2 = {a = 0, b = 110, c = 10, d = 111}.
C_3 = {a = 1, b = 110, c = 10, d = 111}

A message is uniquely decodable if it can only be decoded in one way.

Ex:

- Relative to C_1, **010011** is uniquely decodable to **bad**.
- Relative to C_2, **1100111** is uniquely decodable to **bad**.
- But, relative to C 3 , **1101111** is not uniquely decipherable since it could have encoded to either **bad** or **acad**.

In fact, one can show that every message encoded using C_1 or C_2 is uniquely decodable.

- C_1: Because it is a fixed-length code.
- C_2: Because it is a prefix-free code.

Prefix Codes

Def: A code is called a prefix (free) code if no codeword is a prefix of another one.

Theorem: Every message encoded by a prefix free code is uniquely decodable.

Pf: Since no codeword is a prefix of any other, we can always find the first codeword in a message, peel it off, and continue decoding.

Ex: code: {**a = 0, b = 110, c = 10, d = 111**}.

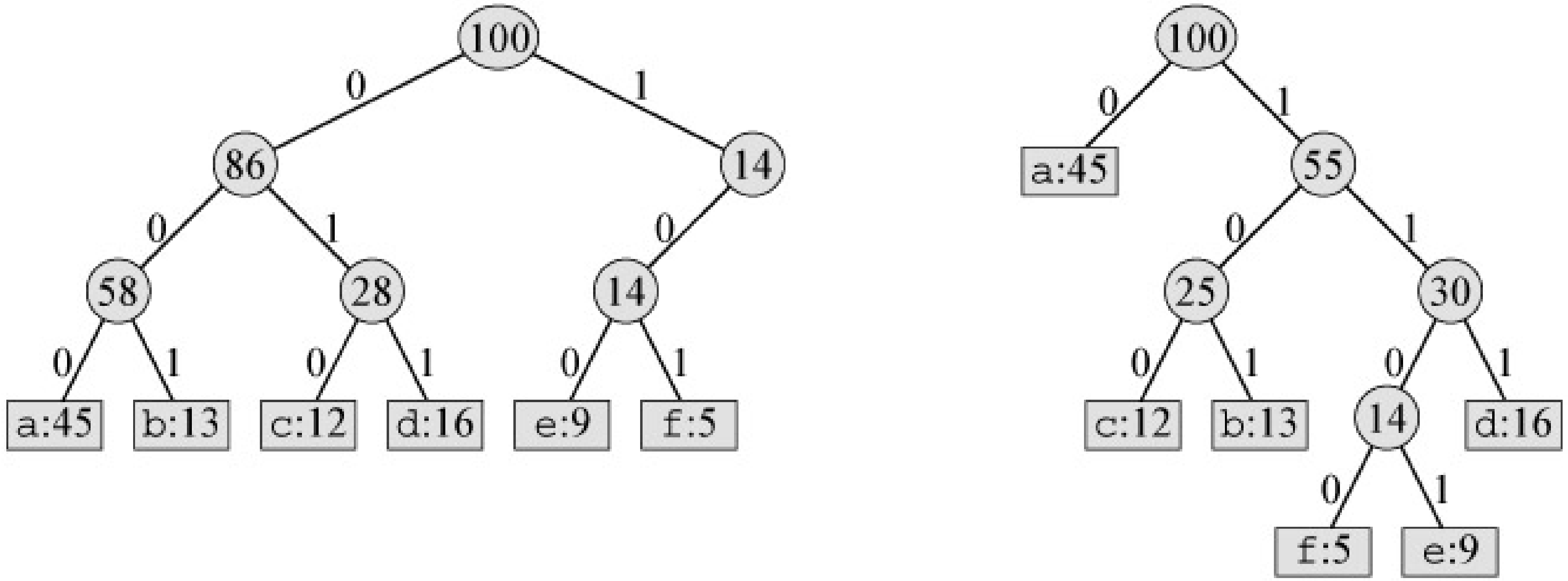
01101100 = 01101100 = abba

Note: There are other kinds of codes that are also uniquely decocable.

Theorem (proof omitted): The best prefix code can achieve the optimal data compression among any code that is uniquely decodable.

Q: How to find the prefix code that results in the smallest encoded message for a given file?

Correspondence between Binary Trees and Prefix Codes



(a) (b)

Left edge is labeled 0; right edge is labeled 1.

The binary string on a path from the root to a leaf is the codeword associated with the character at the leaf.

The depth of a leaf is equal to the length of the codeword.

Problem Restated

Problem definition: Given an alphabet A of n characters $a_1, \dots, a_n$ with weights $f(a_1), \dots, f(a_n)$, find a binary tree T with n leaves labeled $a_1, \dots, a_n$ such that

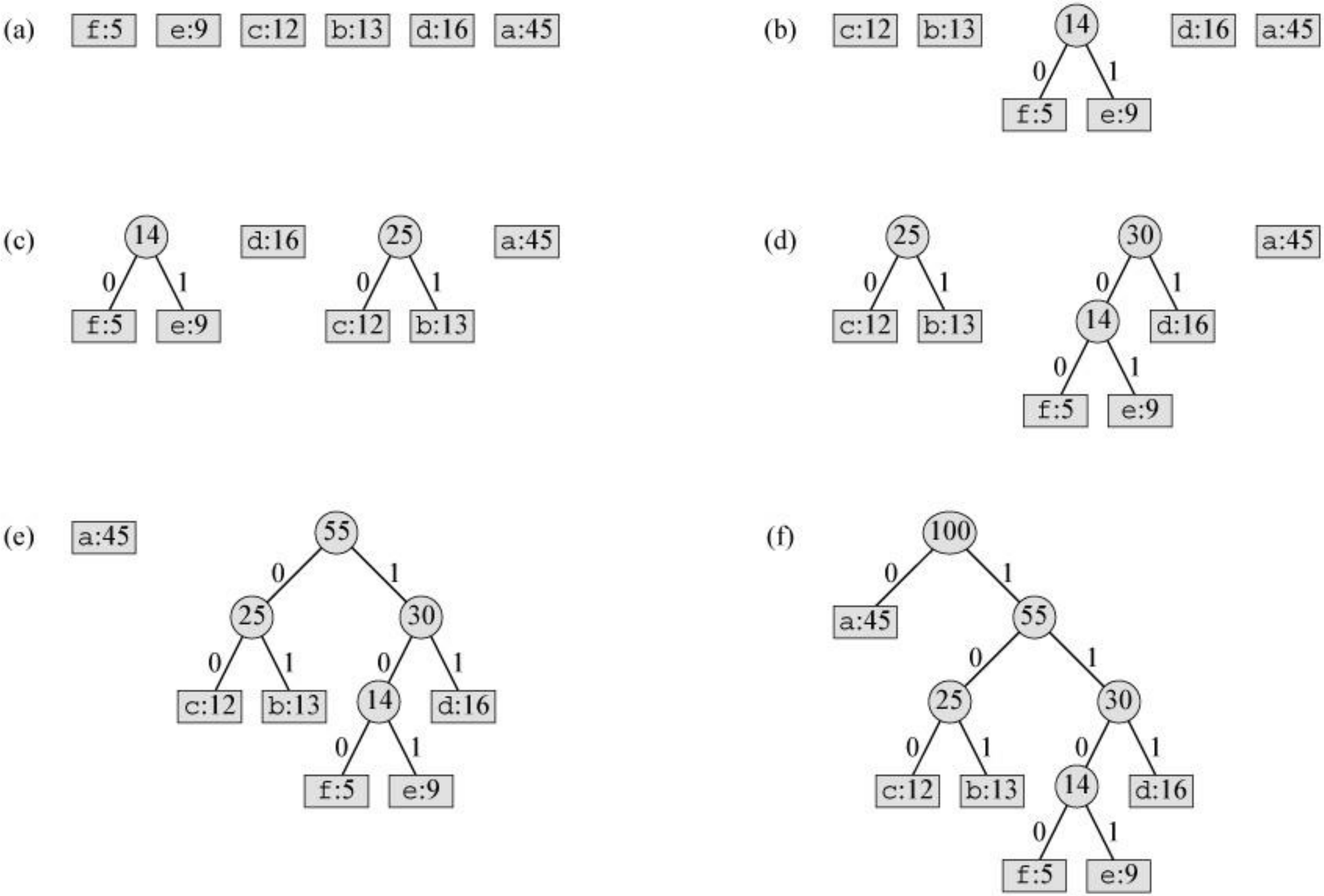
$$B(T) = \sum_{i=1}^{n} f(a_i) d(a_i)$$

is minimized, where $d(a_i)$ is the depth of a_i.

Greedy idea:

- Pick two characters x, y from A with the smallest weights
- Create a subtree that has these two characters as leaves.
- Label the root of this subtree as z.
- Set frequency $f(z) \leftarrow f(x) + f(y)$.
- Remove x, y from A and add z to A.
- Repeat the above procedure (called a merge), until only one character is left.

Example


(a)
f:5
e:9
c:12
b:13
d:16
a:45
(b)
c:12
b:13
14
0
1
f:5
e:9
d:16
a:45
(c)
14
0
1
f:5
e:9
d:16
25
0
1
c:12
b:13
a:45
(d)
25
0
1
c:12
b:13
30
0
1
14
d:16
0
1
f:5
e:9
a:45
(e)
a:45
55
0
1
25
30
0
1
c:12
b:13
0
1
14
d:16
0
1
f:5
e:9
(f)
100
0
1
a:45
55
0
1
25
30
0
1
c:12
b:13
0
1
14
d:16
0
1
f:5
e:9

The Algorithm

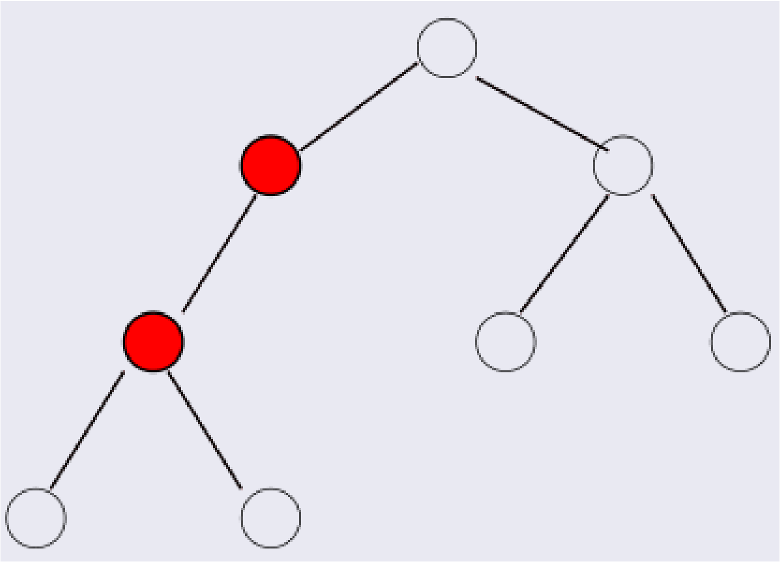
```
Huffman(A):
create a min-priority queue Q on A, with weight as key
for i ← 1 to n − 1
    allocate a new node z
    x ← Extract-Min(Q)
    y ← Extract-Min(Q)
    z.left ← x
    z.right ← y
    z.weight ← x.weight + y.weight
    Insert(Q,z)
return Extract-Min(Q) // return the root of the tree
```

Running time: $O(n \log n)$

Huffman Coding: Correctness

Lemma 1: An optimal prefix code tree must be "full", i.e., every internal node has exactly two children.

Pf: If some internal node had only one child,



then we could simply get rid of this node and replace it with its child. This would decrease the total cost of the encoding.

Huffman Coding: Correctness

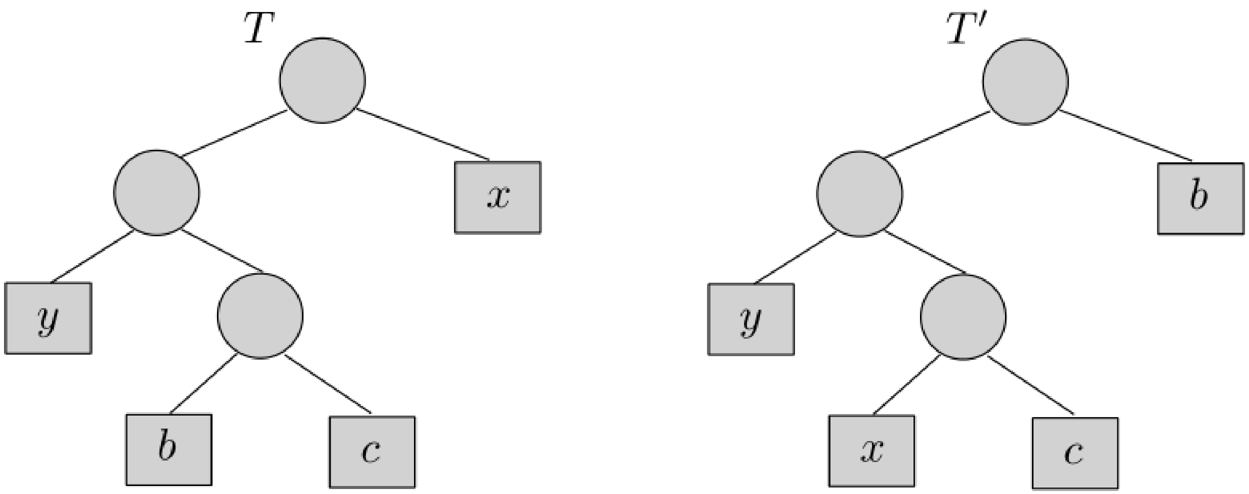
Observation: Moving a small-frequency character downward in T doesn't make it worse.

Lemma 2: Let T be prefix code tree and T' be another obtained from T by swapping two leaf nodes x and b. If,

$$f(x) \le f(b), d(x) \le d(b)$$

then,

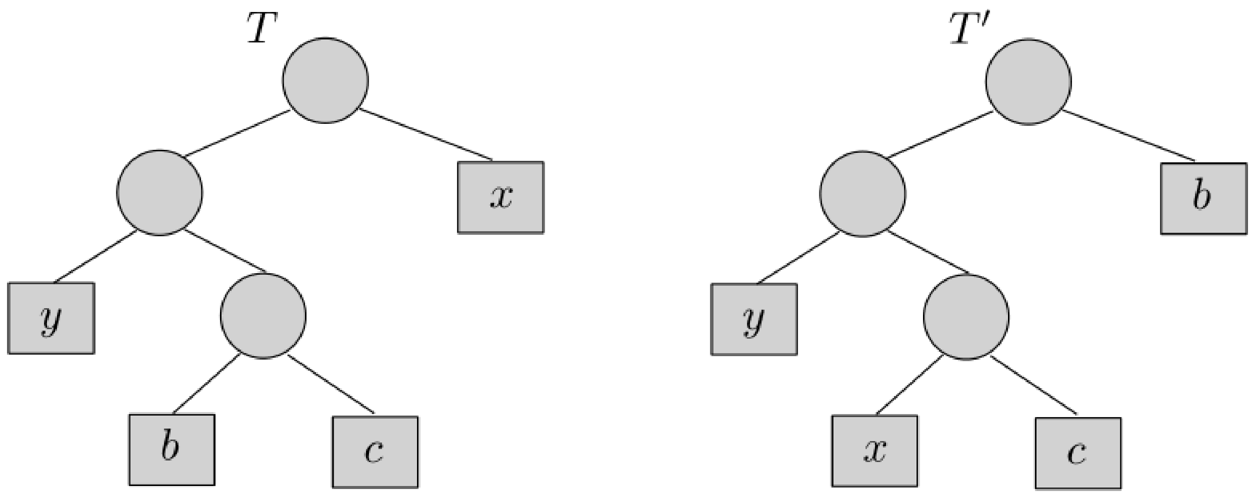
$$B(T') \le B(T).$$



Huffman Coding: Correctness

Pf:

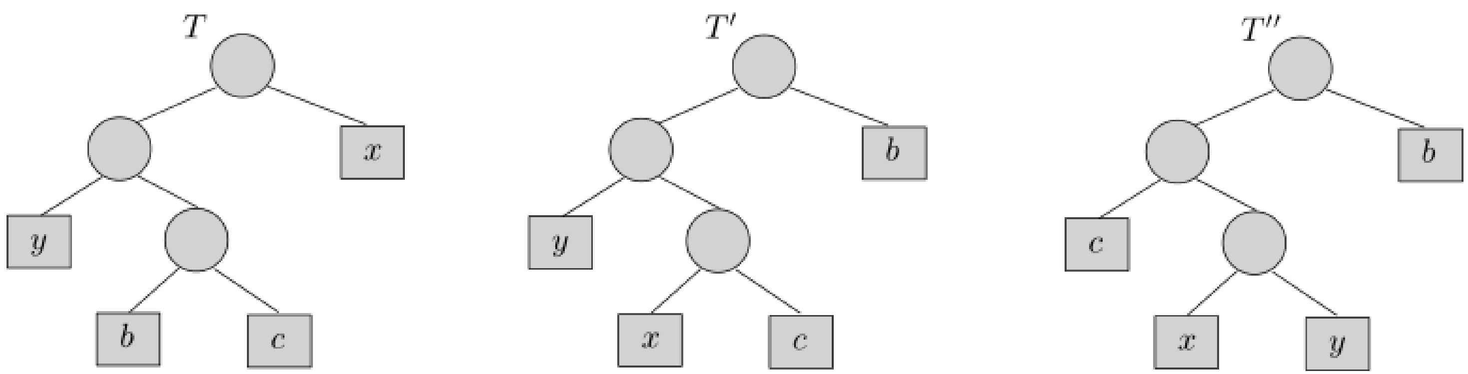
$$
\begin{aligned}
B(T') &= B(T) - f(x)d(x) - f(b)d(b) + f(x)d(b) + f(b)d(x) \\
&= B(T) + \underbrace{(f(x) - f(b))}_{\leq 0}\underbrace{(d(b) - d(x))}_{\geq 0} \\
&\leq B(T).
\end{aligned}
$$



Huffman Coding: Correctness

Lemma 3: Consider the two characters x and y with the smallest frequencies. There is an optimal code tree in which these two letters are sibling leaves at the deepest level of the tree.

Pf: Let T be any optimal prefix code tree, b and c be two siblings at the deepest level of the tree (must exist because T is full).



Assume without loss of generality that $f(x) \leq f(b)$ and $f(y) \leq f(c)$

- (If necessary) swap x with b and swap y with c.
- Proved due to Lemma 2.

Huffman Coding: Correctness

Lemma 4: Let T be a prefix code tree and x and y are two sibling leaves. Let T' be obtained from T by removing x and y, naming the parent z, and setting $f(z) = f(x) + f(y)$. Then

$$B(T) \;=\; B(T') + f(x) + f(y).$$

Pf: $B(T) = B(T') - f(z)d(z) + f(x)(d(z) + 1) + f(y)(d(z) + 1)$

$$= B(T') - \big(f(x) + f(y)\big)d(z) + \big(f(x) + f(y)\big)(d(z) + 1)$$
$$= B(T') + f(x) + f\ (y).$$

Huffman Coding: Correctness

Theorem: The Huffman tree is optimal.

Pf: (By induction on n, the number of characters)

- Base case $n = 2$: Tree with two leaves. Obviously optimal.
- Induction hypothesis: Huffman's algorithm produces optimal tree in the case of $n - 1$ characters.
- Induction step: Consider the case of n characters:
  - Let H be the tree produced by the Huffman's algorithm.
  - Need to show: H is optimal.
- Due to the way Huffman's algorithm works,
  - There are two characters x and y with the smallest frequencies that are sibling leaves in H.
- Let H' be obtained from H by removing x and y, naming the parent z, and setting $f(z) = f(x) + f(y)$
- Alphabet for H: A; alphabet for H' : $A' = A - \{x, y\} \cup \{z\}$
- By Lemma 4, $B(H) = B(H') + f(x) + f(y)$.

Huffman Coding: Correctness

- H' is the tree produced by Huffman's algorithm for A'
- By the induction hypothesis, H' is optimal for A'.

- By Lemma 3, there exists an optimal tree T where x and y are sibling leaves.
- Let T' be obtained from T by removing x and y, naming the parent z, and setting $f(z) = f(x) + f\ (y)$.
- T' is a prefix code tree for alphabet A'.
- By Lemma 4, $B(T) = B(T') + f(x) + f(y)$.
- Hence

$$\begin{aligned} B(H) &= B(H') + f(x) + f(y) \\ &\leq B(T') + f(x) + f(y) \qquad (H' \text{ is optimal for } A') \\ &= B(T). \end{aligned}$$

- Therefore, H must be optimal. Proved.