PSO 14

Huffman, LZW Compression, KD Trees

Question 1

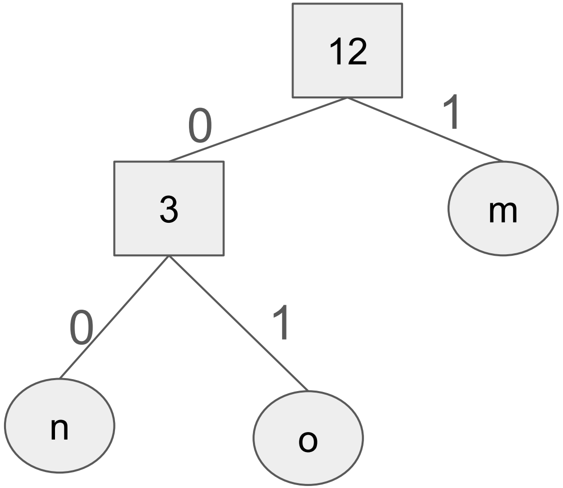
(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$

Huffman Idea: Compress the most frequent letters to be shortest, an example..



Inner-nodes: freqs
Leaves: letters

What is the most freq. letter? What's the encoding of 'o'? 'n'? 'm'?

Question 1

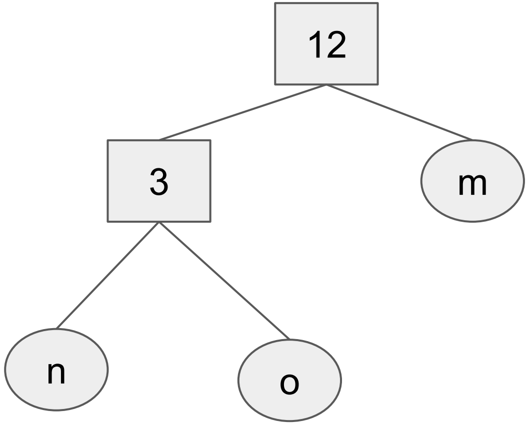
(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$

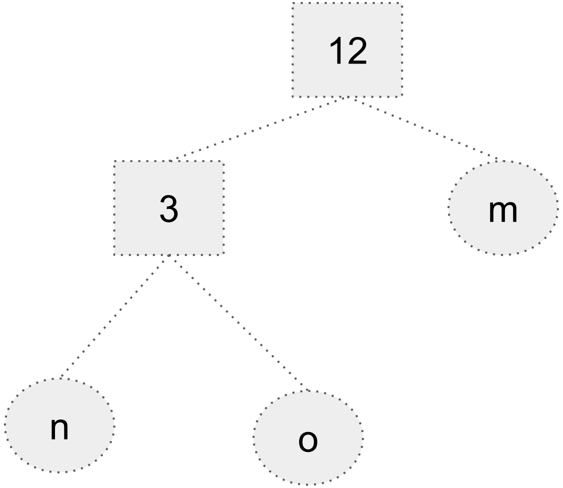
Huffman Idea: Compress the most frequent letters to be shortest



Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

Quick example: start off with freqs

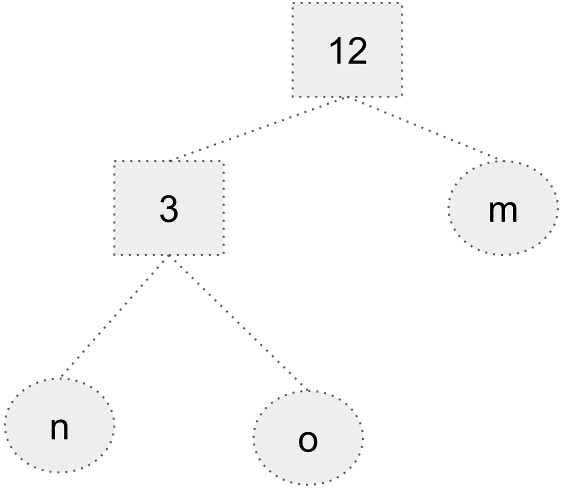


Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

Quick example

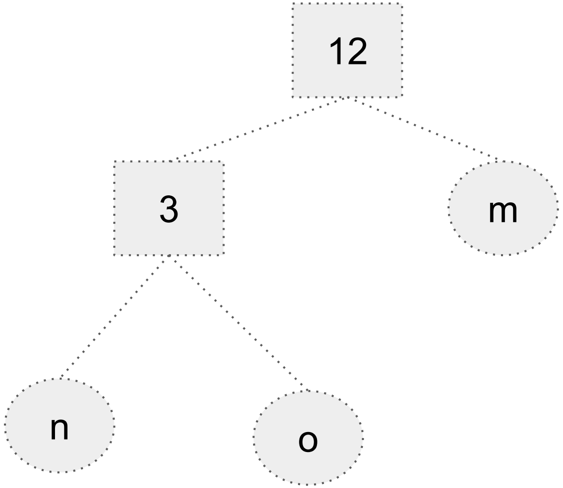


Steps:

1. **Add all letters to minHeap by their frequencies**
2. Pop off min, add to the tree *Bottom-up*
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

Quick example



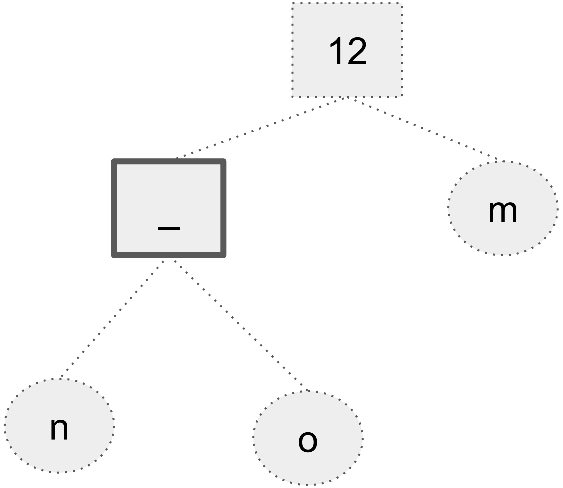
Steps:

1. **Add all letters to minHeap by their frequencies**
2. Pop off min, add to the tree *Bottom-up*
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

<u>Q</u>
(1,n)
(2,n)
(9,m)

Quick example: Step 2



Steps:

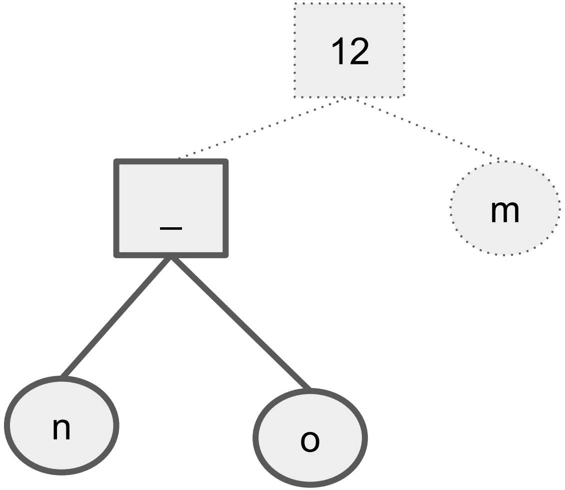
1. **Add all letters to minHeap by their frequencies**
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

Q
(1,n)
(2,n)
(9,m)

Step 2 in-depth:
2a. Initialize node curr

Quick example: Step 2



Steps:

1. **Add all letters to minHeap by their frequencies**
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

Step 2 in-depth:

2a. Initialize node curr

2b. Set children to be next two minHeap elts

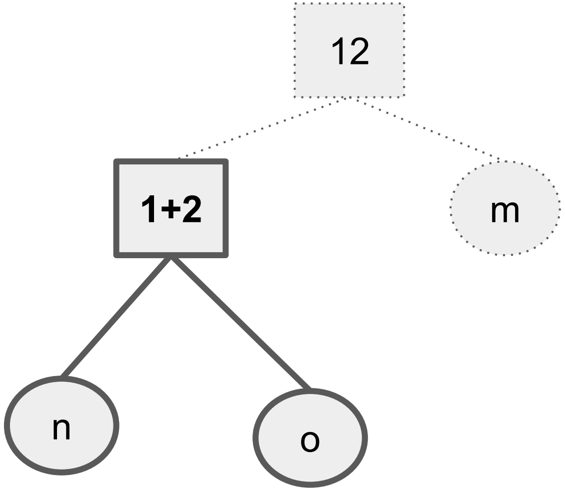
Q

~~(1,n)~~

~~(2,n)~~

(9,m)

Quick example: Step 2



Steps:

1. **Add all letters to minHeap by their frequencies**
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

Q

~~(1,n)~~

~~(2,n)~~

(9,m)

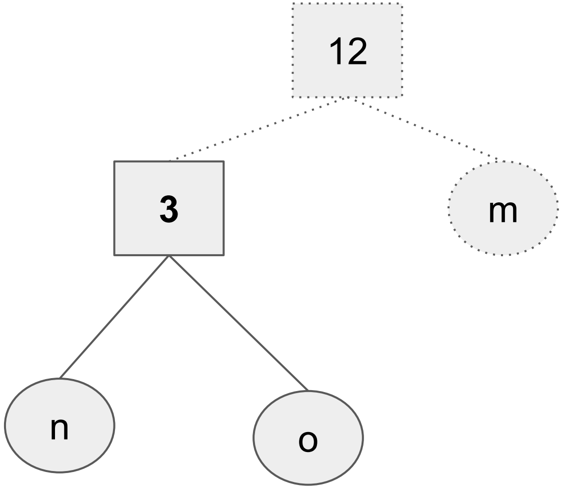
Step 2 in-depth:

2a. Initialize node curr

2b. Set children to be next two minHeap elts

2c. curr.freq = Add up freq of children

Quick example: Step 2



Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. **Put the current tree into minHeap with freq = tree size, repeat**

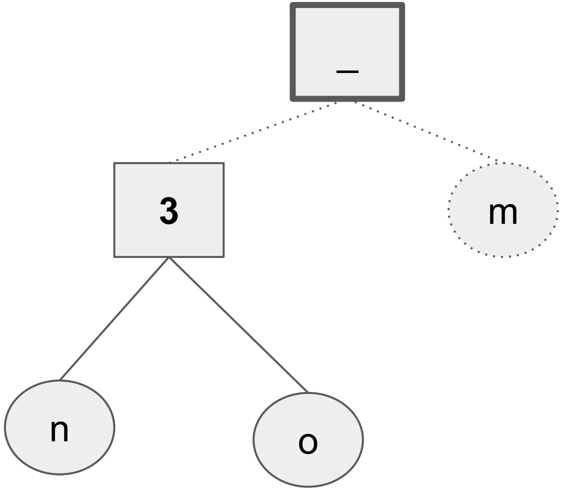
m	n	o
9	1	2

<u>Q</u>

(3,no)

(9,m)

Quick example: Step 2



Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

Q

(3,no)

(9,m)

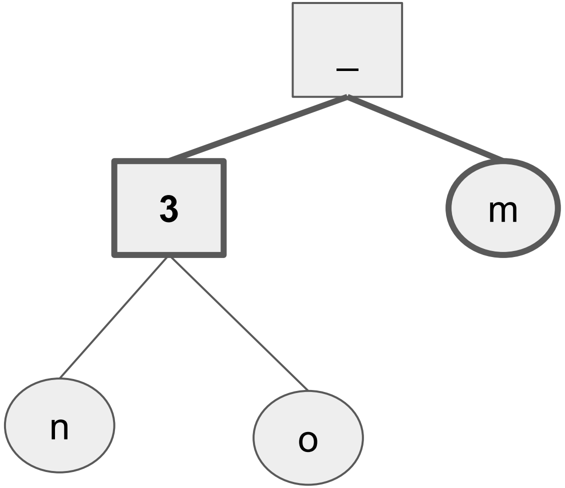
Step 2 in-depth:

2a. **Initialize node curr**

2b. Set children to be next two minHeap elts

2c. curr.freq = Add up freq of children

Quick example: Step 2



Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

Q

~~**(3,no)**~~

~~(9,m)~~

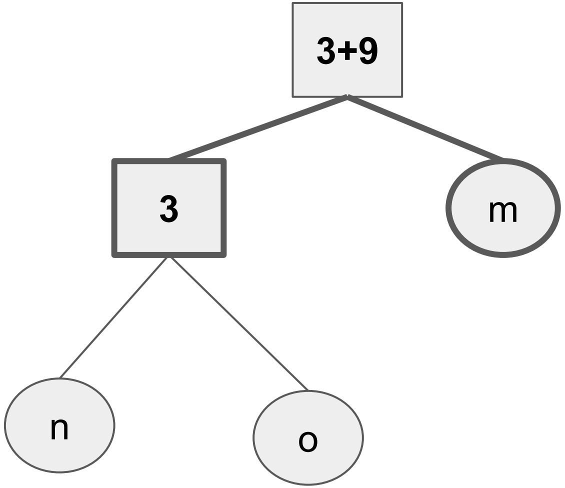
Step 2 in-depth:

2a. Initialize node curr

2b. **Set children to be next two minHeap elts**

2c. curr.freq = Add up freq of children

Quick example: Step 2



Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

m	n	o
9	1	2

Q

~~**(3,no)**~~

~~(9,m)~~

Step 2 in-depth:

2a. Initialize node curr

2b. Set children to be next two minHeap elts

2c. curr.freq = Add up freq of children

Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$

Q

(1,a)
(1,b)
(2,c)
(3,d)
(5,e)
(8,f)
(13,g)
(21,h)

Steps:

1. **Add all letters to minHeap by their frequencies**
2. Pop off min, add to the tree *Bottom-up*
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

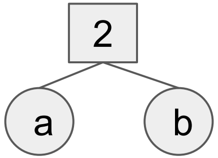
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



Q

~~(1,a)~~
~~(1,b)~~
(2,c)
(3,d)
(5,e)
(8,f)
(13,g)
(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

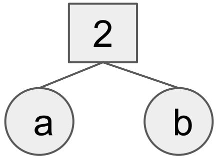
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



Q

(2,ab)
(2,c)
(3,d)
(5,e)
(8,f)
(13,g)
(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. **Put the current tree into minHeap with freq = tree size, repeat 2-3**

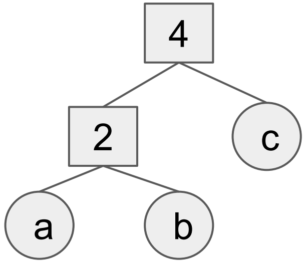
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



Q

~~(2,ab)~~

~~(2,c)~~

(3,d)

(5,e)

(8,f)

(13,g)

(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

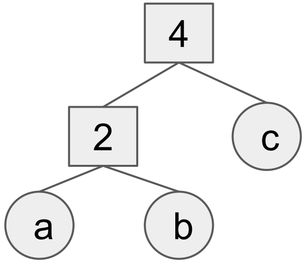
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



Q

(3,d)
(4,abc)
(5,e)
(8,f)
(13,g)
(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. **Put the current tree into minHeap with freq = tree size, repeat 2-3**

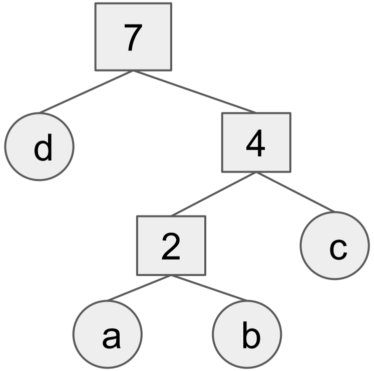
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



Q

~~(3,d)~~

~~(4,abc)~~

(5,e)

(8,f)

(13,g)

(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

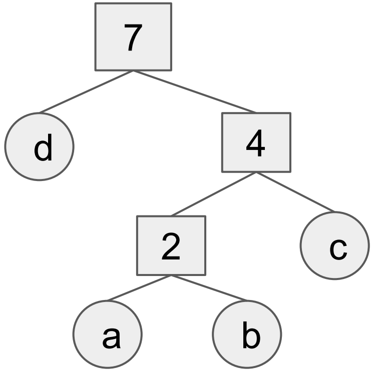
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



Q

(5,e)

(7,abcd)

(8,f)

(13,g)

(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. **Put the current tree into minHeap with freq = tree size, repeat 2-3**

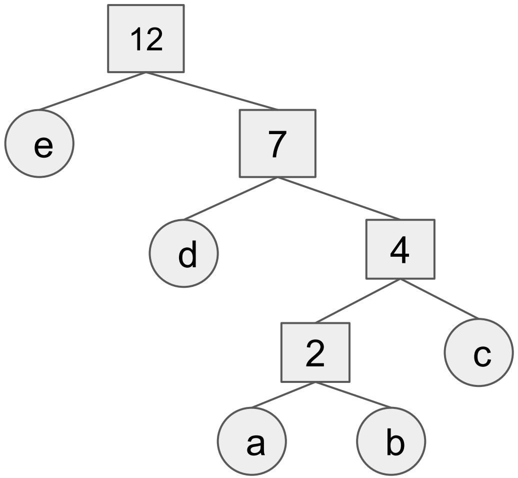
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



Q

~~(5,e)~~

~~(7,abcd)~~

(8,f)

(13,g)

(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

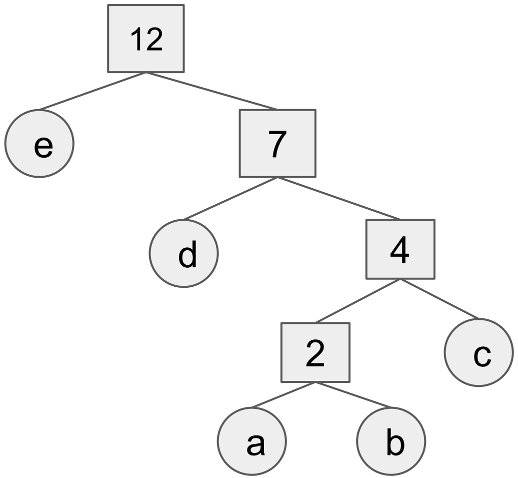
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



Q

(8,f)

(12,abcde)

(13,g)

(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. **Put the current tree into minHeap with freq = tree size, repeat 2-3**

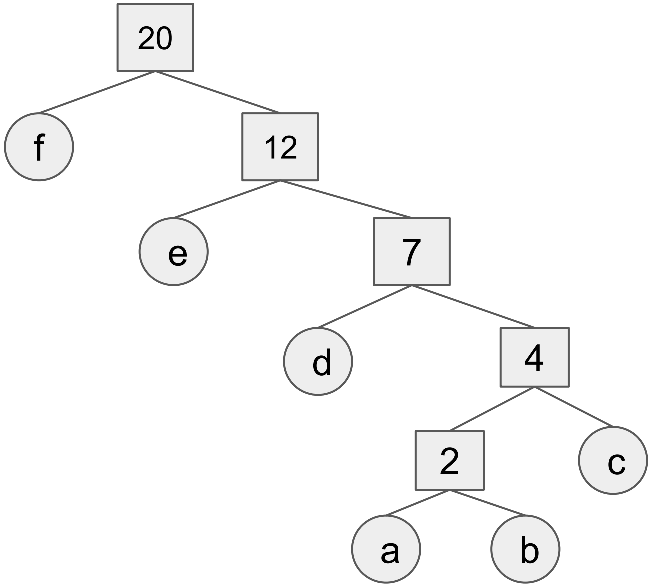
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



Q

~~(8,f)~~

~~(12,abcde)~~

(13,g)

(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

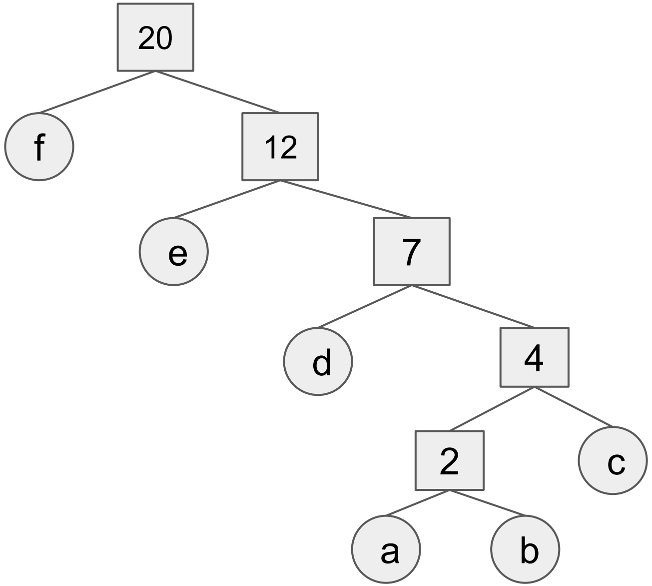
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$$a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$$



<u>Q</u>
(13,g)
(20,abcdef)
(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. **Put the current tree into minHeap with freq = tree size, repeat 2-3**

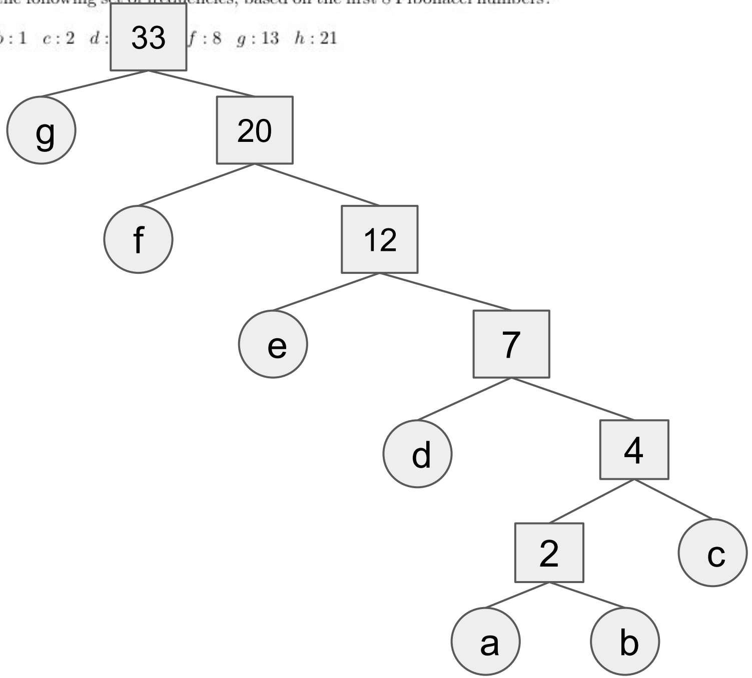
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$a:1\quad b:1\quad c:2\quad d:$ $f:8\quad g:13\quad h:21$



Q

~~(13,g)~~

~~(20,abcdef)~~

(21,h)

Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

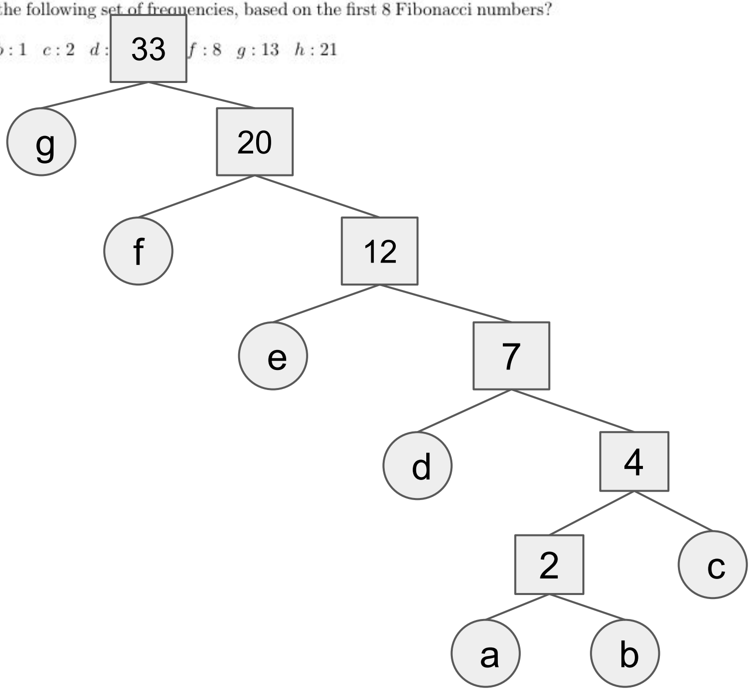
Question 1

(Huffman codes)

Recall that Huffman coding encodes high-frequency character with short codewords such that no codeword is a prefix for some other codeword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$a:1 \quad b:1 \quad c:2 \quad d:$ $\quad f:8 \quad g:13 \quad h:21$



Q

(21,h)

(33,abcdefg)

Steps:

1. Add all letters to minHeap by their frequencies
2. Pop off min, add to the tree *Bottom-up*
3. **Put the current tree into minHeap with freq = tree size, repeat 2-3**

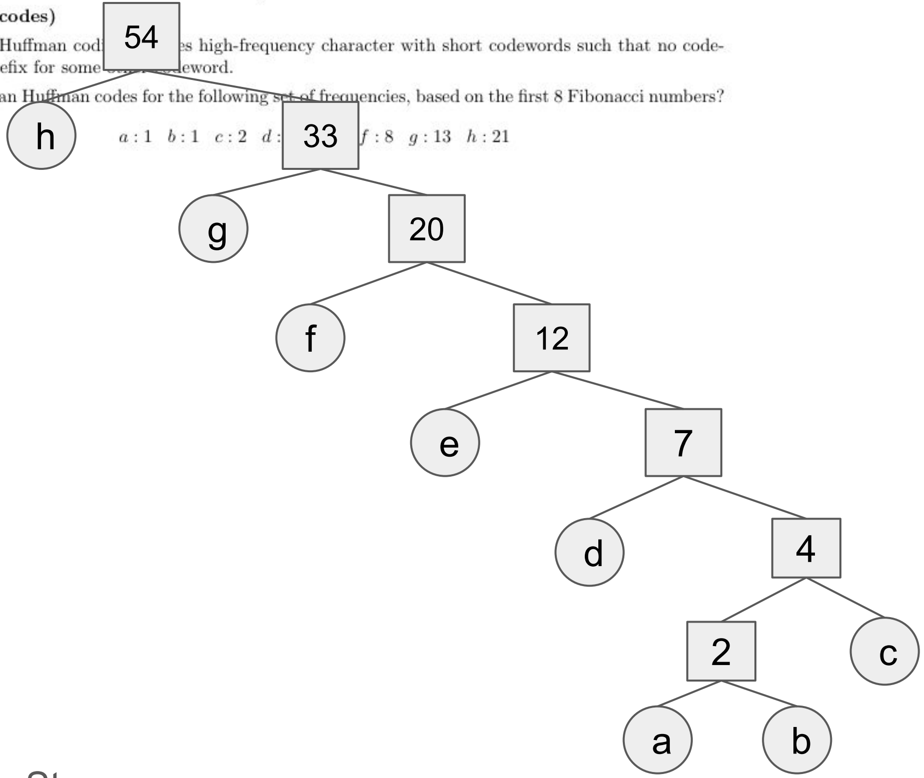
Question 1

(Huffman codes)

Recall that Huffman cod es high-frequency character with short codewords such that no code-word is a prefix for some eword.

(1) What is an Huffman codes for the following set of frequencies, based on the first 8 Fibonacci numbers?

$a:1 \quad b:1 \quad c:2 \quad d: \quad f:8 \quad g:13 \quad h:21$



Q

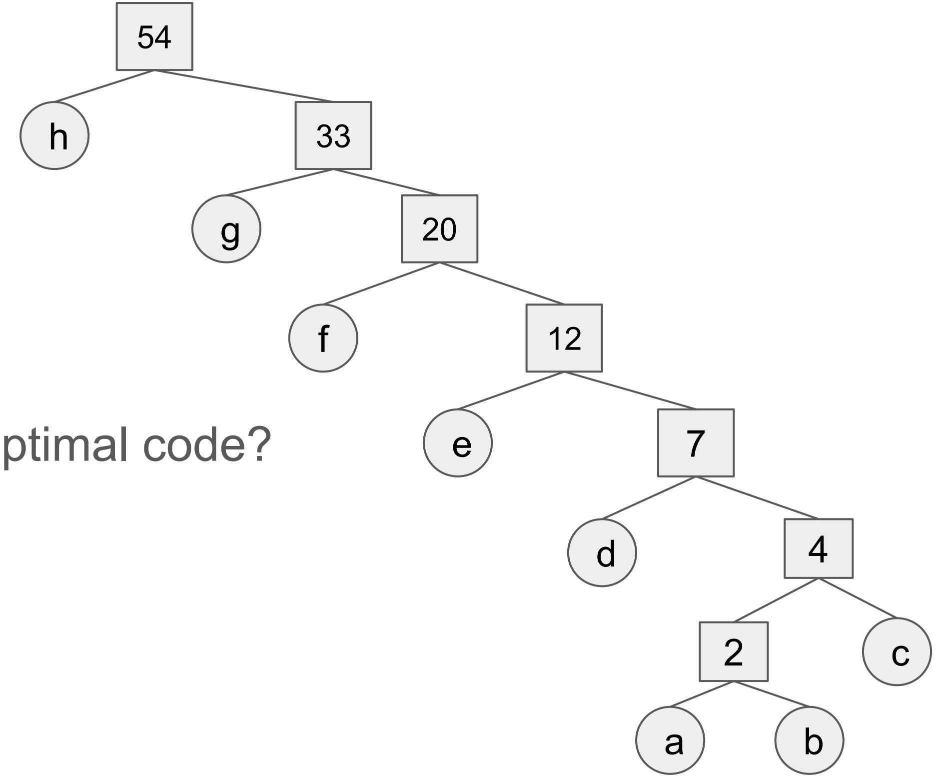
~~(21,h)~~

~~(33,abcdefg)~~

Steps:

1. Add all letters to minHeap by their frequencies
2. **Pop off min, add to the tree *Bottom-up***
3. Put the current tree into minHeap with freq = tree size, repeat 2-3

(2) A code is called **optimal** if it can be represented by a full binary tree, in which all of the nodes have either 0 or 2 children. Is the optimal code unique?



Can I get another optimal code?

Question 2

(LZW Compression)

Consider the alphabet $\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}$ and the input string

$$S = \mathtt{aabcaaccc}.$$

1. Apply LZW with the initial dictionary

$$1 \mapsto \mathtt{a}, \quad 2 \mapsto \mathtt{b}, \quad 3 \mapsto \mathtt{c}.$$

   Write down:

   - the sequence of output codes;
   - all new dictionary entries created during encoding.

LZW - Encoding

- Initialize i=0
- Repeat while i < n
  - Find largest j such that T[i,i+j] is in dictionary, but T[i,i+j+1] is not
  - Suppose that T[i,i+j] is kth item in the dictionary
  - Append (k, T[i+j+1]) to the output file
  - If (i+j+1< n) then Add T[i,i+j+1] to D
  - Set i=i+j+2

LZW-Encoding

- Initialize i=0
- Repeat while i < n
  - Find largest j such that T[i,i+j] is in dictionary, but T[i,i+j+1] is not
  - Suppose that T[i,i+j] is kth item in the dictionary
  - Append (k, T[i+j+1]) to the output file
  - If (i+j+1< n) then Add T[i,i+j+1] to D
  - Set i=i+j+2

$\{\overset{1}{a}, \overset{2}{b}, \overset{3}{c}$

$$S = \underset{i=0}{\underline{a}}abcaaccc.$$

LZW - Encoding

- Initialize i=0
- Repeat while i < n
  - Find largest j such that T[i,i+j] is in dictionary, but T[i,i+j+1] is not
  - Suppose that T[i,i+j] is kth item in the dictionary
  - Append (k, T[i+j+1]) to the output file
  - If (i+j+1< n) then Add T[i,i+j+1] to D
  - Set i=i+j+2

$$\{\overset{1}{a}, \overset{2}{b}, \overset{3}{c}$$

$$S = \underset{i=0}{\text{a}}\text{abcaaccc}.$$

j = 1 since S[0,1] = a in dict

but S[0,2] = aa not in dict.

LZW - Encoding

$D = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}$

- Initialize i=0
- Repeat while i < n
  - Find largest j such that T[i,i+j] is in dictionary, but T[i,i+j+1] is not
  - Suppose that T[i,i+j] is kth item in the dictionary
  - Append (k, T[i+j+1]) to the output file
  - If (i+j+1< n) then Add T[i,i+j+1] to D
  - Set i=i+j+2

$S = \underset{i=0}{\underline{a}}abcaaccc.$

$j = 1$ since $S[0,1] = a$ in dict
but $S[0,2] = aa$ not in dict.

Codes: (1, a)

LZW - Encoding

- Initialize i=0
- Repeat while i < n
  - Find largest j such that T[i,i+j] is in dictionary, but T[i,i+j+1] is not
  - Suppose that T[i,i+j] is kth item in the dictionary
  - Append (k, T[i+j+1]) to the output file
  - If (i+j+1< n) then Add T[i,i+j+1] to D
  - Set i=i+j+2

$D = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}$

$S = \underset{i=0}{a}abcaaccc.$

$j = 1$ since $S[0,1] = a$ in dict
but $S[0,2] = aa$ not in dict.

Codes: (1, a)

LZW - Encoding

- Initialize i=0
- Repeat while i < n
  - Find largest j such that T[i,i+j] is in dictionary, but T[i,i+j+1] is not
  - Suppose that T[i,i+j] is kth item in the dictionary
  - Append (k, T[i+j+1]) to the output file
  - If (i+j+1< n) then Add T[i,i+j+1] to D
  - Set i=i+j+2

$D = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}$

$$S = \text{aabcaaccc}.$$

j = 1 since S[0,1] = a in dict
but S[0,2] = aa not in dict.

P[1] + a = aa

Codes: (1, a)

LZW - Encoding

- Initialize i=0
- Repeat while i < n
  - Find largest j such that T[i,i+j] is in dictionary, but T[i,i+j+1] is not
  - Suppose that T[i,i+j] is kth item in the dictionary
  - Append (k, T[i+j+1]) to the output file
  - If (i+j+1< n) then Add T[i,i+j+1] to D
  - Set i=i+j+2

$$D = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}$$

$$S = \text{aabcaaccc}.$$

$i=2$

$P[1]+a = aa$

Codes: $(1, a)$

LZW-Encoding

- Initialize i=0
- Repeat while i < n
  - Find largest j such that T[i,i+j] is in dictionary, but T[i,i+j+1] is not
  - Suppose that T[i,i+j] is kth item in the dictionary
  - Append (k, T[i+j+1]) to the output file
  - If (i+j+1< n) then Add T[i,i+j+1] to D
  - Set i=i+j+2

$D = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}$

$$S = \text{aabcaaccc}.$$

$i = 2$

$j = 1$ since $S[2,3] = b \in D$

but $S[2,4] = bc \notin D$

$D[1] + a = aa$

Codes: $(1, a)$

LZW-Encoding

- Initialize i=0
- Repeat while i < n
  - Find largest j such that T[i,i+j] is in dictionary, but T[i,i+j+1] is not
  - Suppose that T[i,i+j] is kth item in the dictionary
  - Append (k, T[i+j+1]) to the output file
  - If (i+j+1< n) then Add T[i,i+j+1] to D
  - Set i=i+j+2

$D = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}$

$S = \text{aabcaaccc}.$

$i=2$

$j=1$ since $S[2,3]=b \in D$

but $S[2,4]=bc \notin D$

$D[1]+a = aa$

Codes: (1, a) , (2, c)

LZW-Encoding

- Initialize i=0
- Repeat while i < n
  - Find largest j such that T[i,i+j] is in dictionary, but T[i,i+j+1] is not
  - Suppose that T[i,i+j] is kth item in the dictionary
  - Append (k, T[i+j+1]) to the output file
  - If (i+j+1< n) then Add T[i,i+j+1] to D
  - Set i=i+j+2

$D = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}$

$S = \text{aabcaaccc.}$

$i = 2$

$j = 1$ since $S[2,3] = b \in D$

but $S[2,4] = bc \notin D$

$D[1] + a = aa$

$D[2] + c = bc$

Codes: $(1, a)$, $(2, c)$

LZW - Encoding

- Initialize i=0
- Repeat while i < n
  - Find largest j such that T[i,i+j] is in dictionary, but T[i,i+j+1] is not
  - Suppose that T[i,i+j] is kth item in the dictionary
  - Append (k, T[i+j+1]) to the output file
  - If (i+j+1< n) then Add T[i,i+j+1] to D
  - Set i=i+j+2

$$D=\{\overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}\}$$

$$S = \text{aabcaaccc}.$$

D[1]+a = aa

D[2]+c = bc

Codes: (1, a) , (2, c)

LZW - Encoding

- Initialize i=0
- Repeat while i < n
  - Find largest j such that T[i,i+j] is in dictionary, but T[i,i+j+1] is not
  - Suppose that T[i,i+j] is kth item in the dictionary
  - Append (k, T[i+j+1]) to the output file
  - If (i+j+1< n) then Add T[i,i+j+1] to D
  - Set i=i+j+2

$$D = \{\overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}$$

$$S = \text{aabcaaccc.}$$

i=4

D[1]+a = aa

D[2]+c = bc

Codes: (1, a), (2, c)

LZW - Encoding

- Initialize i=0
- Repeat while i < n
  - Find largest j such that T[i,i+j] is in dictionary, but T[i,i+j+1] is not
  - Suppose that T[i,i+j] is kth item in the dictionary
  - Append (k, T[i+j+1]) to the output file
  - If (i+j+1< n) then Add T[i,i+j+1] to D
  - Set i=i+j+2

$$D = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}$$

$$S = aabc\underline{a}accc.$$

$i=4$

$j=2$ since $S[4,6] = aa \in D$

but $S[4,7] = aac \notin D$

$D[1]+a = aa$ $\quad$ $D[2]+c = bc$

Codes: $(1, a)$, $(2, c)$

LZW - Encoding

- Initialize i=0
- Repeat while i < n
  - Find largest j such that T[i,i+j] is in dictionary, but T[i,i+j+1] is not
  - Suppose that T[i,i+j] is kth item in the dictionary
  - Append (k, T[i+j+1]) to the output file
  - If (i+j+1< n) then Add T[i,i+j+1] to D
  - Set i=i+j+2

$D = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}, \overset{6}{aac}$

$$S = \text{aabcaaccc.}$$

i=4

J = 2 since S[4,6] = aa ∈ D

but S[4,7] = aac ∉ D

D[1]+a = aa

D[2]+c = bc

Codes: (1, a), (2, c), (4, c)

Skipping steps...

LZW - Encoding

- Initialize i=0
- Repeat while i < n
  - Find largest j such that T[i,i+j] is in dictionary, but T[i,i+j+1] is not
  - Suppose that T[i,i+j] is kth item in the dictionary
  - Append (k, T[i+j+1]) to the output file
  - If (i+j+1< n) then Add T[i,i+j+1] to D
  - Set i=i+j+2

$D = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}, \overset{6}{aac}$

$S = \text{aabcaaccc}.$

i=4

J= 2 since $S[4,6] = aa \in D$

but $S[4,7] = aac \notin D$

$D[1]+a = aa$ $D[2]+c = bc$

Codes: (1, a), (2, c), (4, c)

Skipping steps...

LZW - Encoding

- Initialize i=0
- Repeat while i < n
  - Find largest j such that T[i,i+j] is in dictionary, but T[i,i+j+1] is not
  - Suppose that T[i,i+j] is kth item in the dictionary
  - Append (k, T[i+j+1]) to the output file
  - If (i+j+1< n) then Add T[i,i+j+1] to D
  - Set i=i+j+2

$$D = \{\overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}, \overset{6}{aac}$$

$$S = \text{aabcaaccc}.$$

$D[1]+a = aa$ $\quad$ $D[2]+c = bc$ $\quad$ $D[4]+c = aac$

Codes: (1, a), (2, c), (4, c)

In this last step, what do we add?

LZW - Encoding

- Initialize i=0
- Repeat while i < n
  - Find largest j such that T[i,i+j] is in dictionary, but T[i,i+j+1] is not
  - Suppose that T[i,i+j] is kth item in the dictionary
  - Append (k, T[i+j+1]) to the output file
  - If (i+j+1< n) then Add T[i,i+j+1] to D
  - Set i=i+j+2

$$D = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}, \overset{6}{aac}, \overset{7}{cc} \}$$

$$S = \text{aabcaaccc}.$$

D[1]+a = aa   D[2]+c = bc   D[4]+c = aac

Codes: (1, a), (2, c), (4, c), (3, c)

In this last step, what do we add?

Output : (1, a), (2, c), (4, c), (3, c)

Goal : Decode to $S = \text{aabcaaccc}.$

IDEA : Recreate the final dictionary $D = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}, \overset{6}{aac}, \overset{7}{cc} \}$

from initial $D_0 = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c} \}$

Goal: Decode to $S = \text{aabcaaccc}.$

IDEA: Recreate the final dictionary $D = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}, \overset{6}{aac}, \overset{7}{cc} \}$

from initial $D_0 = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c} \}$

$\underbrace{(1, a)}_{D[1]+a=aa}, \underbrace{(2, c)}_{D[2]+c=bc}, \underbrace{(4, c)}_{D[4]+c=aac}, \underline{\underbrace{(3, c)}_{D[3]+c=cc}}$

Start left to right...

Goal: Decode to $S = \text{aabcaaccc}.$

IDEA: Recreate the final dictionary $D = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}, \overset{6}{aac}, \overset{7}{cc} \}$

from initial $D_0 = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c} \}$

$\overset{D[1]+a = aa}{(1, a)}, \overset{D[2]+c = bc}{(2, c)}, \overset{D[4]+c = aac}{(4, c)}, \overset{D[3]+c = cc}{(3, c)}$

Start left to right...

We know $(1, a)$ gives us aa.

Goal: Decode to $S = \text{aabcaaccc}.$

IDEA: Recreate the final dictionary $D = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}, \overset{6}{aac}, \overset{7}{cc} \}$

from initial $D_0 = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa} \}$

$D[1]+a = aa$ $\quad$ $D[2]+c = bc$ $\quad$ $D[4]+c = aac$ $\quad$ $D[3]+c = cc$

$(1, a)$, $(2, c)$, $(4, c)$, $(3, c)$

Start left to right...

We know $(1, a)$ gives us aa.

+ we added aa to the dictionary after

Goal: Decode to $S = \text{aabcaaccc}.$

IDEA: Recreate the final dictionary $D = \{\overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}, \overset{6}{aac}, \overset{7}{cc}\}$

from initial $D_0 = \{\overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}\}$

$D[1]+a = aa$ $\quad$ $D[2]+c = bc$ $\quad$ $D[4]+c = aac$ $\quad$ $D[3]+c = cc$ |

$(1, a), \underline{(2, c)}, (4, c), \underline{(3, c)}$

nextup,

We know $(2,c)$ gives us bc

+ we added bc to the dictionary after

Goal: Decode to $S = \text{aabcaaccc}.$

IDEA: Recreate the final dictionary $D = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}, \overset{6}{aac}, \overset{7}{cc} \}$

from initial $D_0 = \{ \overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc} \}$

$\underbrace{(1, a)}_{}$, $(2, c)$, $(4, c)$, $(3, c)$

$(1,a) = D[1]+a = aa$, $(2,c) = D[2]+c = bc$, $(4,c) = D[4]+c = aac$, $(3,c) = D[3]+c = cc$

(why?)

nextup,

We know $(4, c)$ gives us aac

+ we added aac to the dictionary after

Goal: Decode to $S = \text{aabcaaccc}.$

IDEA: Recreate the final dictionary $D = \{\overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}, \overset{6}{aac}, \overset{7}{cc}\}$

from initial $D_0 = \{\overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}, \overset{6}{aac}\}$

$D[1]+a = aa$ $\quad D[2]+c = bc$ $\quad D[4]+c = aac$ $\quad D[3]+c = cc$

$(1, a), (2, c), (4, c), (3, c)$

nextup,

We know $(4, c)$ gives us aac

+ we added aac to the dictionary after

(why?) we recreated $D_0[4] = aa$

Goal: Decode to $S = \text{aabcaaccc}.$

IDEA: Recreate the final dictionary $D = \{\overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}, \overset{6}{aac}, \overset{7}{cc}\}$

from initial $D_0 = \{\overset{1}{a}, \overset{2}{b}, \overset{3}{c}, \overset{4}{aa}, \overset{5}{bc}, \overset{6}{aac}, \overset{7}{cc}\}$

$D[1]+a = aa$, $D[2]+c = bc$, $D[4]+c = aac$, $D[3]+c = cc$

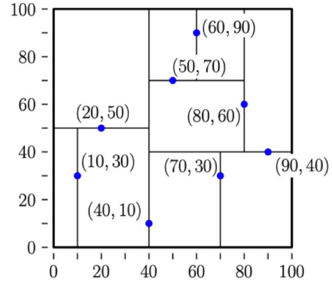
$(1, a)$, $(2, c)$, $(4, c)$, $(3, c)$

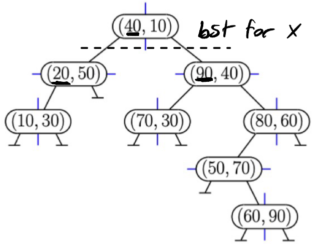
nextup,

We know $(3, c)$ gives us cc

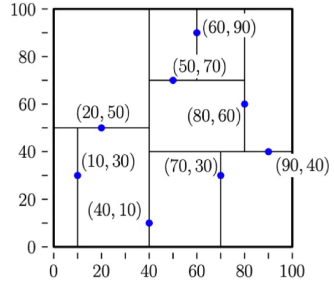
+ we added cc to the dictionary after

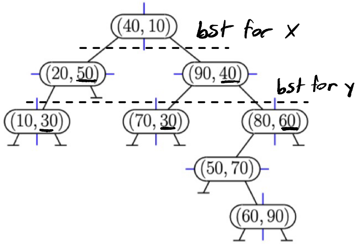
(2) **(kd-trees)** Consider the kd-tree shown below. We assume it's a standard kd-tree where the cutting dimensions alternates between x and y with each level. Show the final tree after the operation **insert((70,50))**.



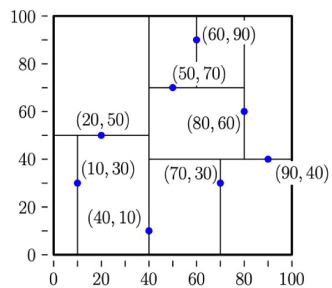


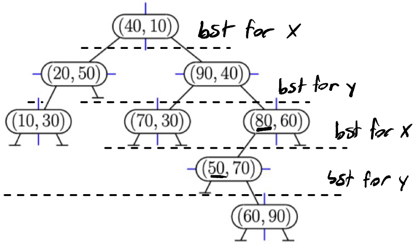
(2) **(kd-trees)** Consider the kd-tree shown below. We assume it's a standard kd-tree where the cutting dimensions alternates between x and y with each level. Show the final tree after the operation **insert((70,50))**.





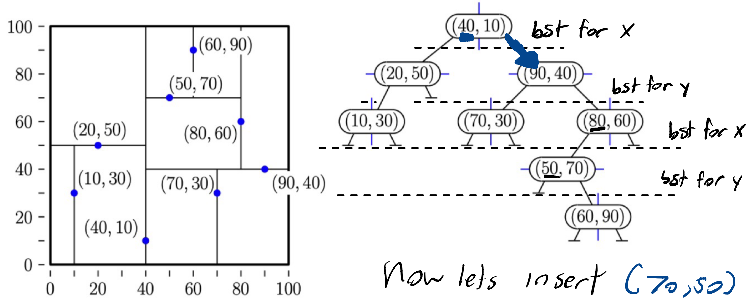
(2) **(kd-trees)** Consider the kd-tree shown below. We assume it's a standard kd-tree where the cutting dimensions alternates between x and y with each level. Show the final tree after the operation **insert((70,50))**.





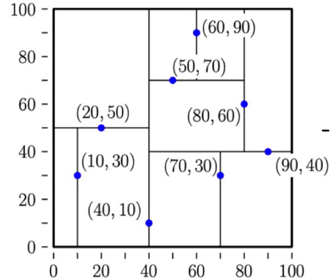
Now lets insert (70,50)

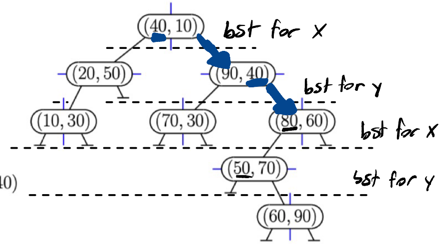
(2) **(kd-trees)** Consider the kd-tree shown below. We assume it's a standard kd-tree where the cutting dimensions alternates between x and y with each level. Show the final tree after the operation **insert((70,50))**.



Now lets insert (70,50)

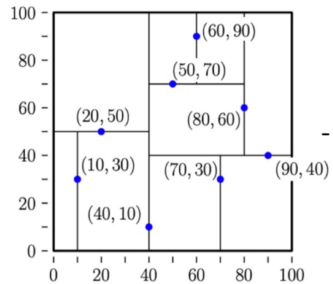
(2) **(kd-trees)** Consider the kd-tree shown below. We assume it's a standard kd-tree where the cutting dimensions alternates between x and y with each level. Show the final tree after the operation **insert((70,50))**.

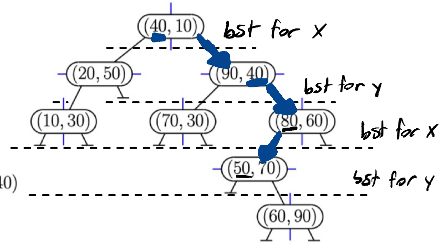




Now lets insert (70,50)

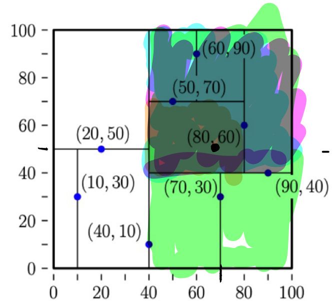
(2) **(kd-trees)** Consider the kd-tree shown below. We assume it's a standard kd-tree where the cutting dimensions alternates between x and y with each level. Show the final tree after the operation **insert((70,50))**.

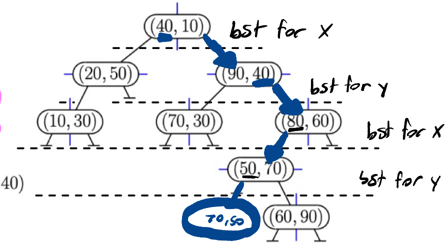




Now lets insert (70,50)

(2) **(kd-trees)** Consider the kd-tree shown below. We assume it's a standard kd-tree where the cutting dimensions alternates between x and y with each level. Show the final tree after the operation **insert((70,50))**.

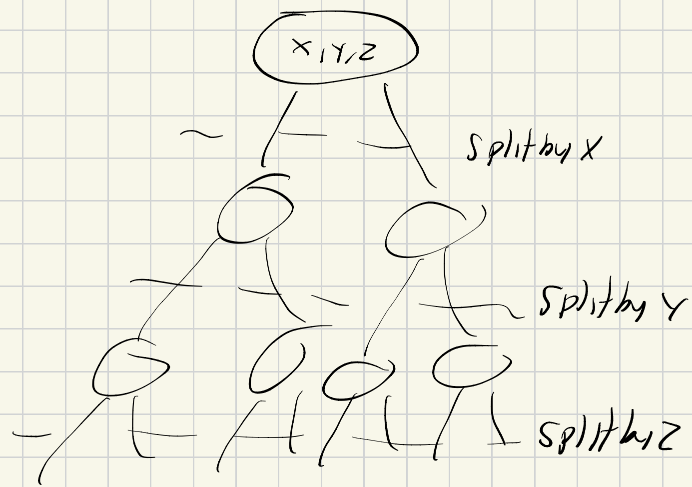




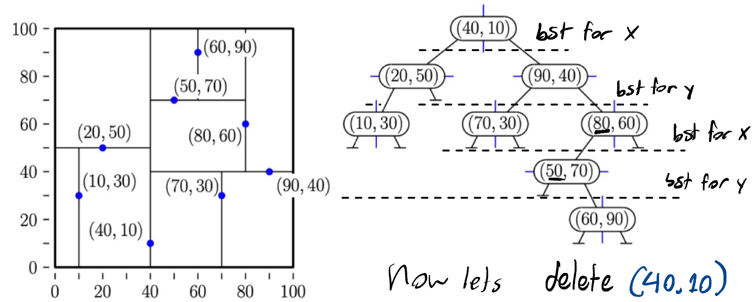
Now lets insert (70,50)

2d tree

3d tree



(2) **(kd-trees)** Consider the kd-tree shown below. We assume it's a standard kd-tree where the cutting dimensions alternates between x and y with each level. Show the final tree after the operation **insert((70,50))**.



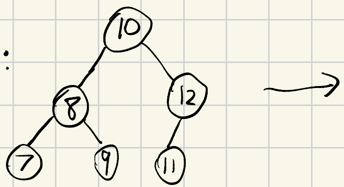
Now lets delete (40.10)

Recall deletion in normal BST

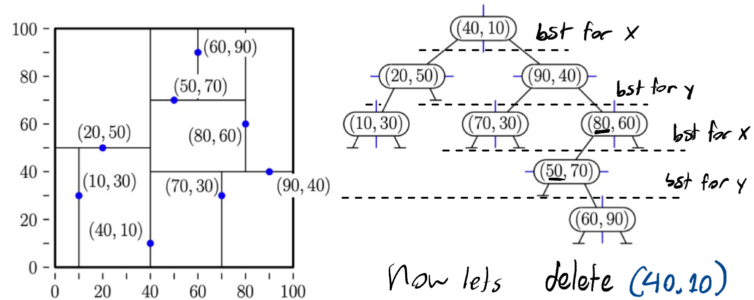
- Find Predecessor/Successor leaf to replace

ex:

delete(10):



(2) **(kd-trees)** Consider the kd-tree shown below. We assume it's a standard kd-tree where the cutting dimensions alternates between x and y with each level. Show the final tree after the operation **insert((70,50))**.



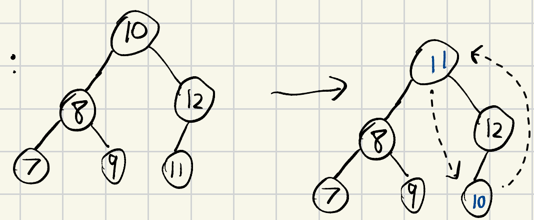
Now lets delete (40.10)

Recall deletion in normal BST

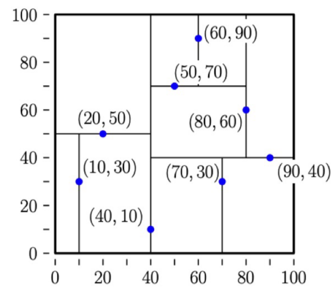
- Find Predecessor/Successor leaf to replace

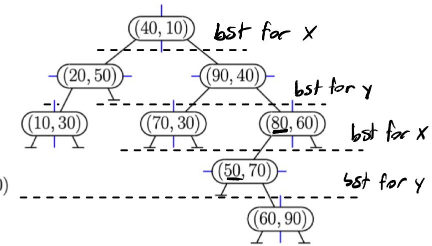
ex:

delete(10):



(2) **(kd-trees)** Consider the kd-tree shown below. We assume it's a standard kd-tree where the cutting dimensions alternates between x and y with each level. Show the final tree after the operation **insert((70,50))**.





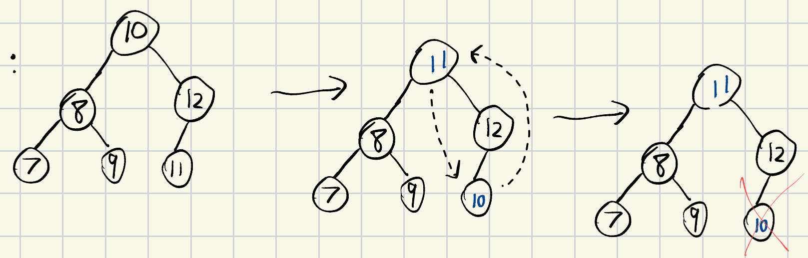
Now lets delete (40,10)

Recall deletion in normal BST

- Find Predecessor/Successor leaf to replace

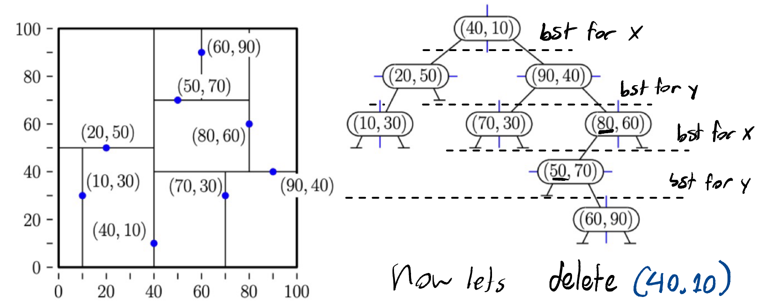
ex:

delete(10):



(If not a leaf continue recursively)

(2) **(kd-trees)** Consider the kd-tree shown below. We assume it's a standard kd-tree where the cutting dimensions alternates between x and y with each level. Show the final tree after the operation **insert((70,50))**.



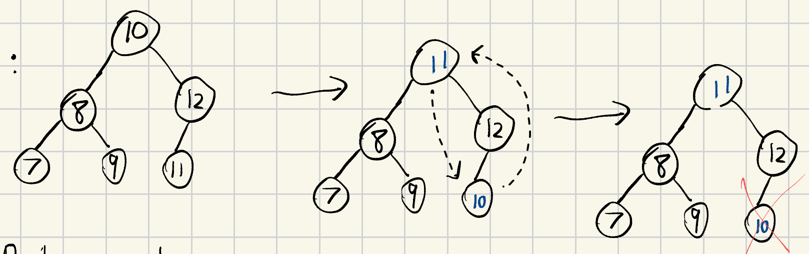
Now lets delete (40.10)

Recall deletion in normal BST

- Find Predecessor/Successor leaf to replace

ex:

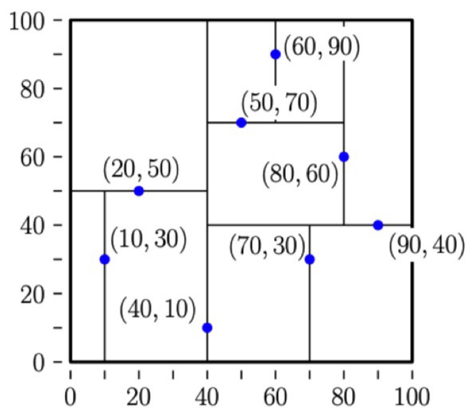
delete(10):

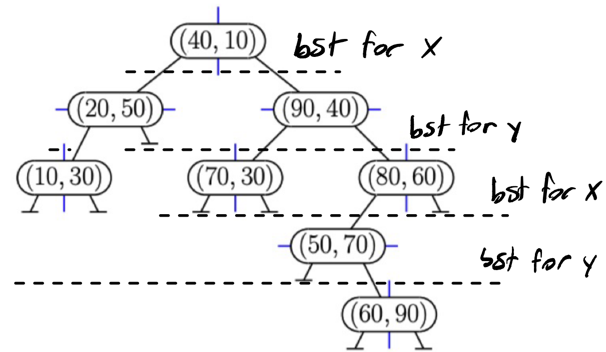


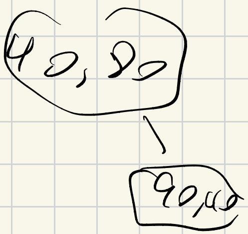
Deletion in KD-trees is the same.

except you choose successor/predecessor based on dimension

(2) **(kd-trees)** Consider the kd-tree shown below. We assume it's a standard kd-tree where the cutting dimensions alternates between x and y with each level. Show the final tree after the operation **insert((70,50))**.



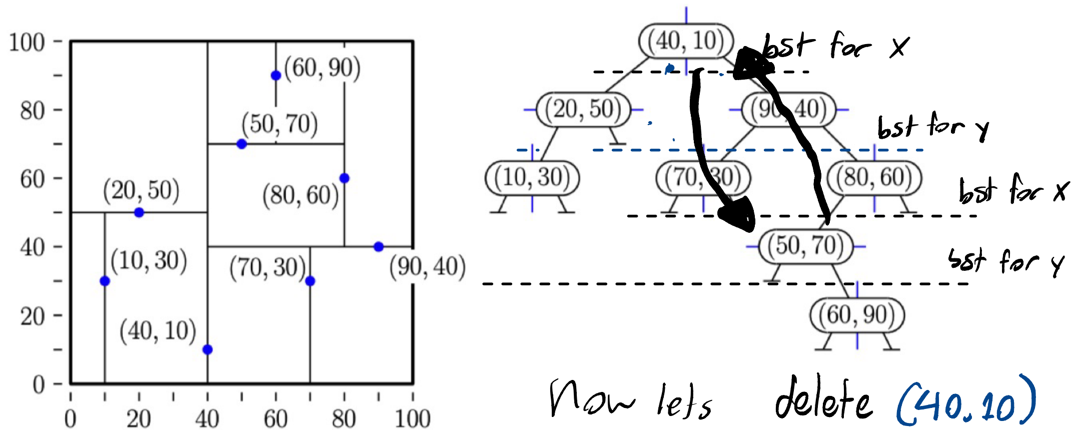




Now lets delete (40, 10)

1) Find successor for $\underline{X=40}$

(2) **(kd-trees)** Consider the kd-tree shown below. We assume it's a standard kd-tree where the cutting dimensions alternates between x and y with each level. Show the final tree after the operation **insert((70,50))**.

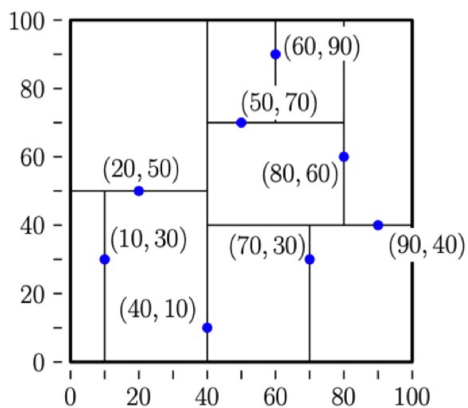


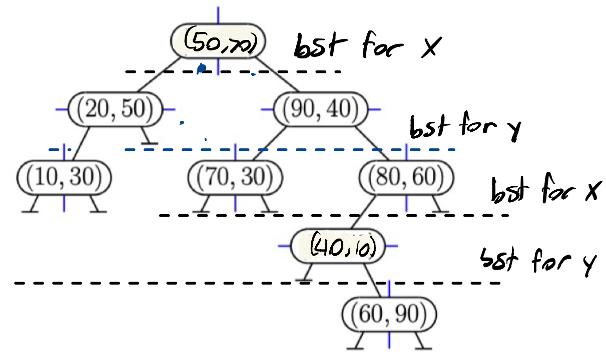
Now lets delete (40.10)

2) Swap with (40,10)

[Think about why this preserves k-d order]

(2) **(kd-trees)** Consider the kd-tree shown below. We assume it's a standard kd-tree where the cutting dimensions alternates between x and y with each level. Show the final tree after the operation **insert((70,50))**.



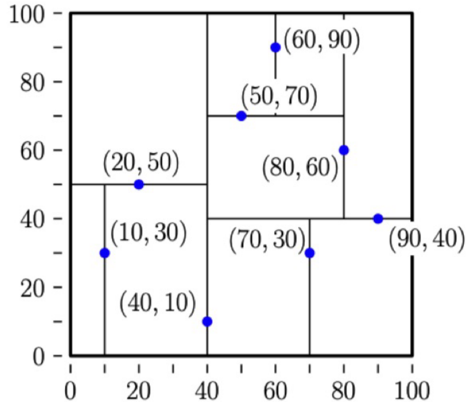


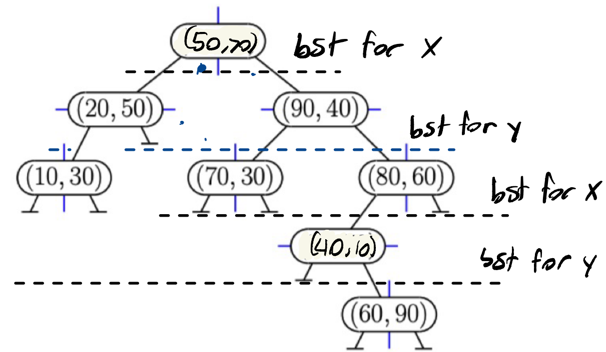
Now lets delete (40.10)

2) Swap with (40,10)

[Think about why this preserves k-d order]

(2) **(kd-trees)** Consider the kd-tree shown below. We assume it's a standard kd-tree where the cutting dimensions alternates between x and y with each level. Show the final tree after the operation **insert((70,50))**.

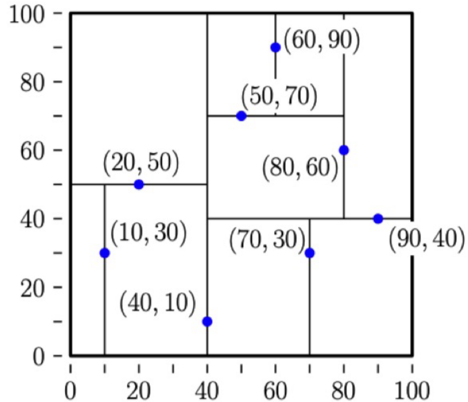


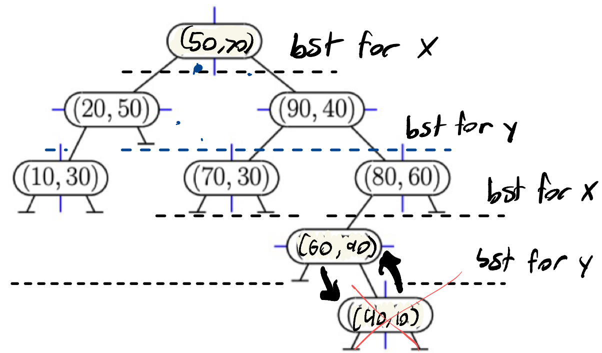


Now lets delete (40.10)

2.2) Not at leaf yet, find successor in y = 10 and swap.

(2) **(kd-trees)** Consider the kd-tree shown below. We assume it's a standard kd-tree where the cutting dimensions alternates between x and y with each level. Show the final tree after the operation **insert((70,50))**.

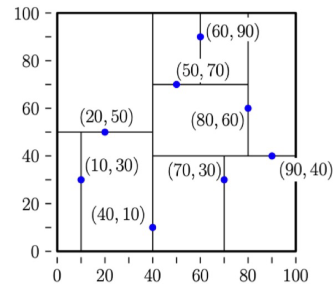


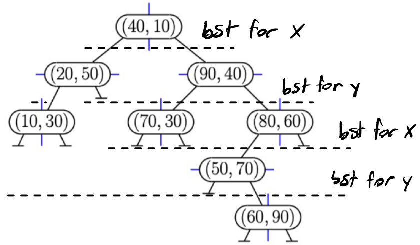


Now lets delete (40.10)

2.2) Not at leaf yet, find successor in y = 10 and swap.

(2) **(kd-trees)** Consider the kd-tree shown below. We assume it's a standard kd-tree where the cutting dimensions alternates between x and y with each level. Show the final tree after the operation **insert((70,50))**.





Now lets delete (80,60).

Left as Exercise

NOTE: If no successor, find a predecessor instead and vice/versa.