Huffman

Huffman-code is een binaire boom toegepast

Huffman-code

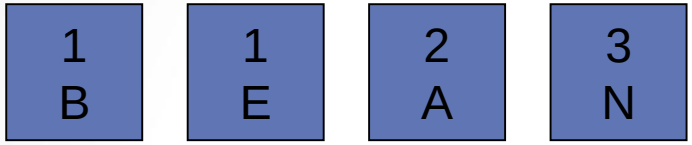
- Bedoeld om teksten te comprimeren.
- Niet alle letters een even lange (binaire) code
  - letter die vaak voorkomt een korte code
  - letter die zelden voorkomt de langere code
- Dus moet de frequentie van alle letters in de code worden bepaald.

Voorbeeld

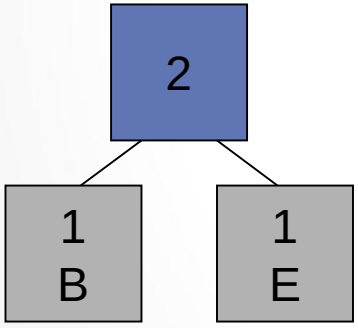
- Code : “bananen”
  - b : 1 keer
  - e : 1 keer
  - a : 2 keer
  - n : 3 keer
- “b” en “e” krijgen een lange code
- “a” een kortere
- “n” de kortste

Bepaling code

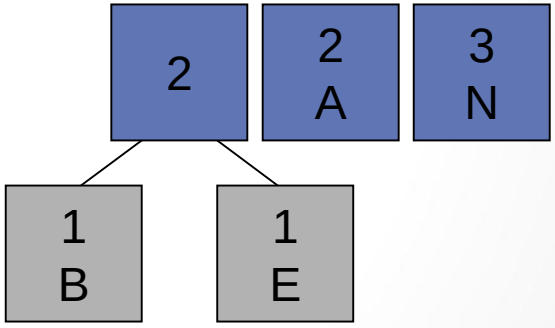
- De letters op frequentie sorteren: (Array van Knopen)



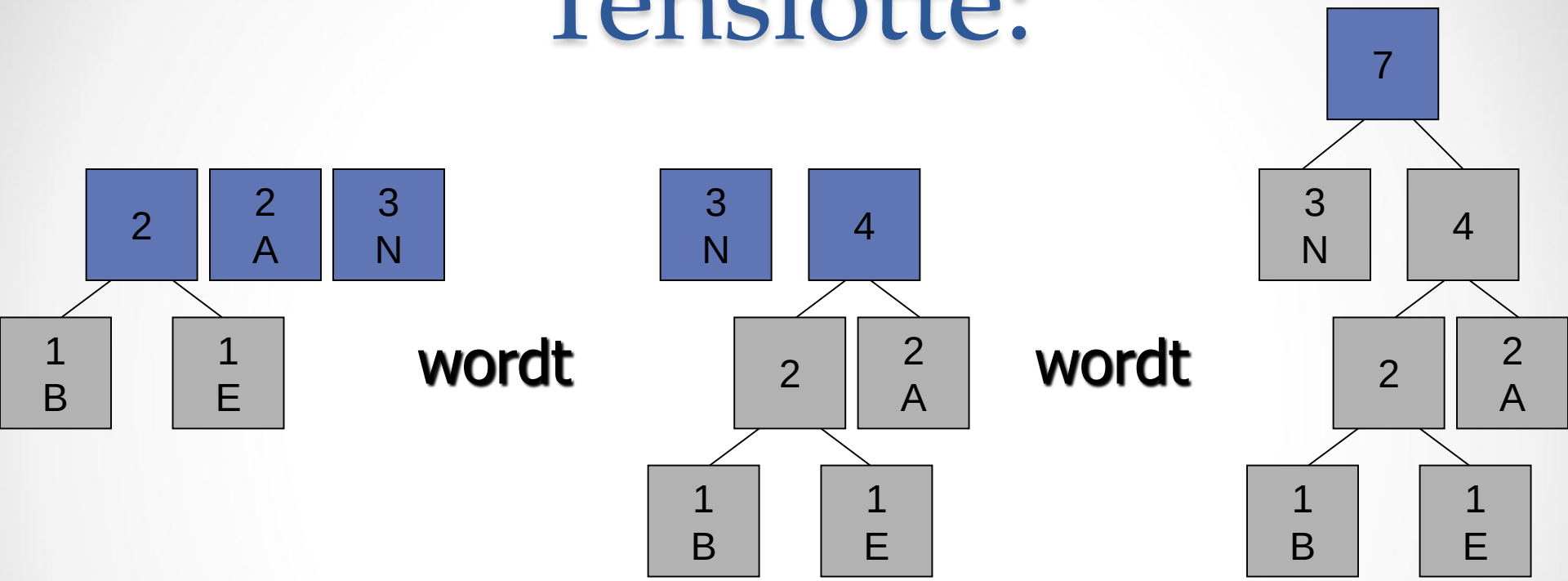
- Van links af in een boom zetten: B en E vervangen door 2



We krijgen dan (na sorteren)



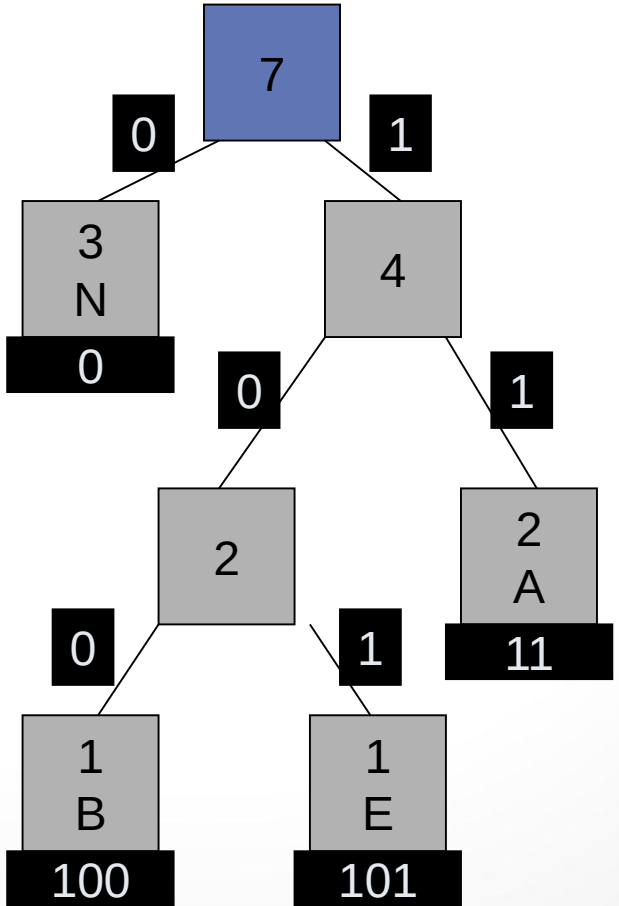
Tenslotte:



Nu nog binaire waarden geven

Post-order door de boom

- elke linker-tak (lijntje) krijgt een 0
- elke rechtertak (lijntje) krijgt een 1

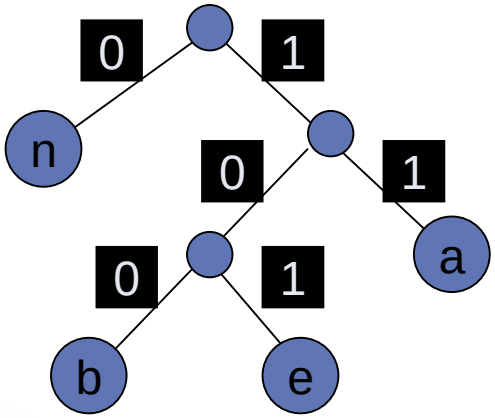


Coderen bericht

- We hadden gevonden
  - n=0
  - a=11
  - b=100
  - e=101
- bananen wordt dus 100 11 0 11 0 101 0

Terug lezen

- Bericht 1001101101010
- Net zo lang met een cijfer de boom in
  (o→ naar links, 1→ naar rechts)
  totdat je op een blad bent aangeland



Huffman codering

- Je zult de gebruikte binairy tree moeten meezenden met het bericht
- De boodschap moet 2 keer worden gelezen.
  - t.b.v. frequentie tellen
  - boom maken

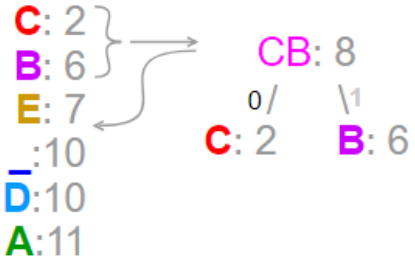
```
public class CharCount {
        public int count;
        public char character;
        public string binaireWaarde="";
        public CharCount(char c) {
                character=c;
                count=0;
        }
}
```

```
public class Knoop {
        public CharCount userObject;
        public Knoop rechts,links;
        public Knoop(CharCount o){
                userObject=o;
        }
        …
}
```

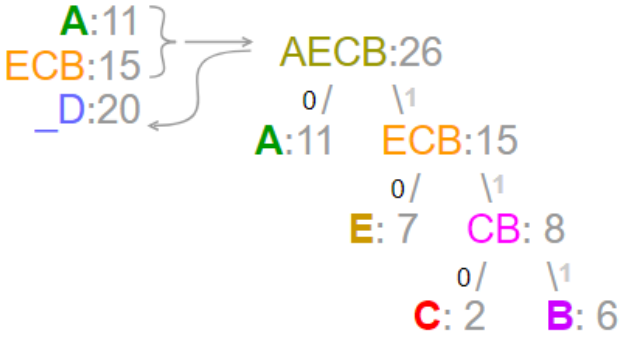
```
IDictionaryEnumerator etr = tree.GetEnumerator();
while (etr.MoveNext())
{
   CharCount ccnt = (CharCount)etr.Value;
}
```

1. "A_DEAD_DAD_CEDED_A_BAD_BABE_A_BEADED_ABACA_BED"

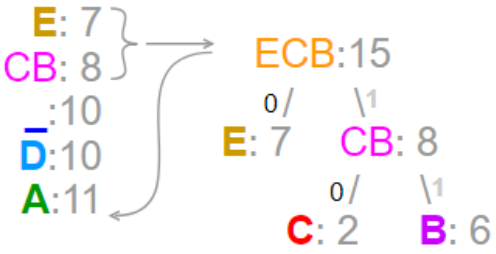
2.



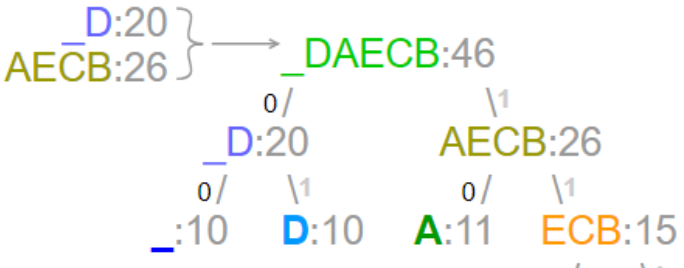
5.



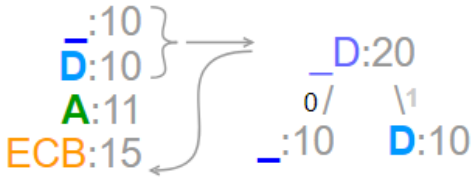
3.



6.



4.



7.

8. "1000011101001000110010011101100111001001000111110010011111011111100010001111110100111001001011111011101000111111001"

- https://en.wikipedia.org/wiki/Huffman_coding