Greedy algorithms

CMPSC 465 - Yana Safonova

Huffman coding

Constructing the prefix-free encoding tree

Idea: put more frequent symbols at smaller depth

Greedy approach: continually merge least frequent symbols/nodes until you have a full binary tree encoding all symbols

Example

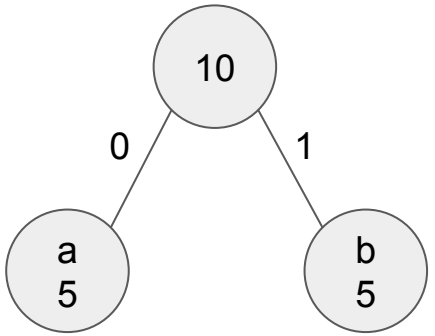
a: 5, b: 5, c: 7, d: 10, e: 15, f: 17

Example

a: 5, b: 5, c: 7, d: 10, e: 15, f: 17

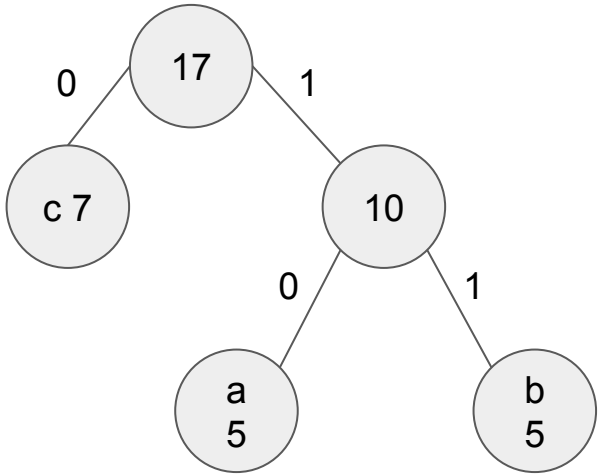
Example

a: 5, b: 5, c: 7, d: 10, e: 15, f: 17



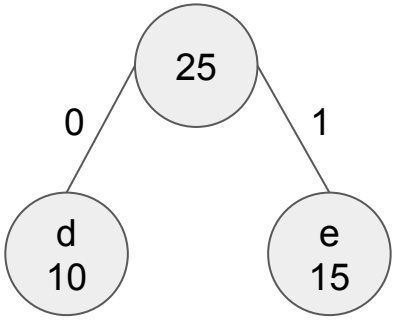
Example

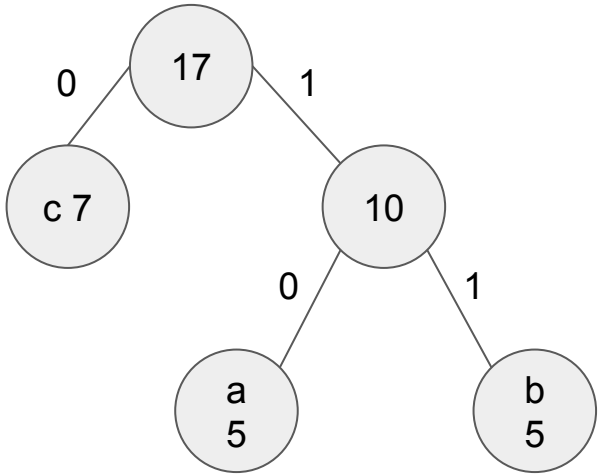
a: 5, b: 5, c: 7, d: 10, e: 15, f: 17



Example

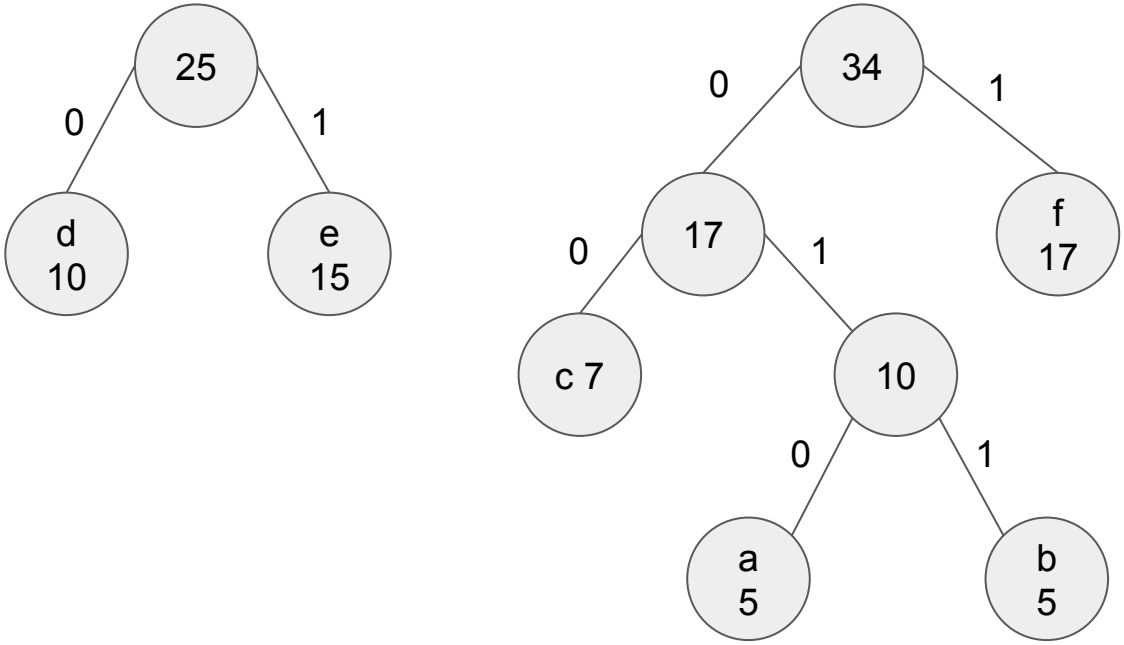
a: 5, b: 5, c: 7, d: 10, e: 15, f: 17





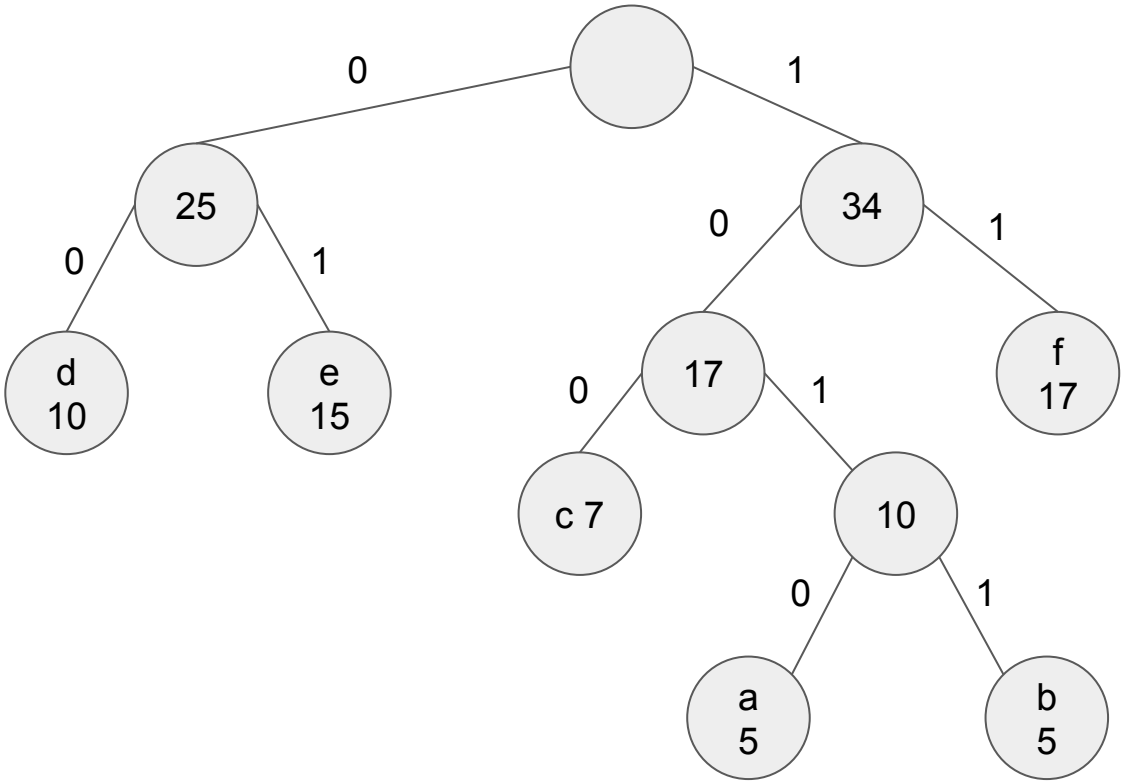
Example

a: 5, b: 5, c: 7, d: 10, e: 15, f: 17



Example

a: 5, b: 5, c: 7, d: 10, e: 15, f: 17

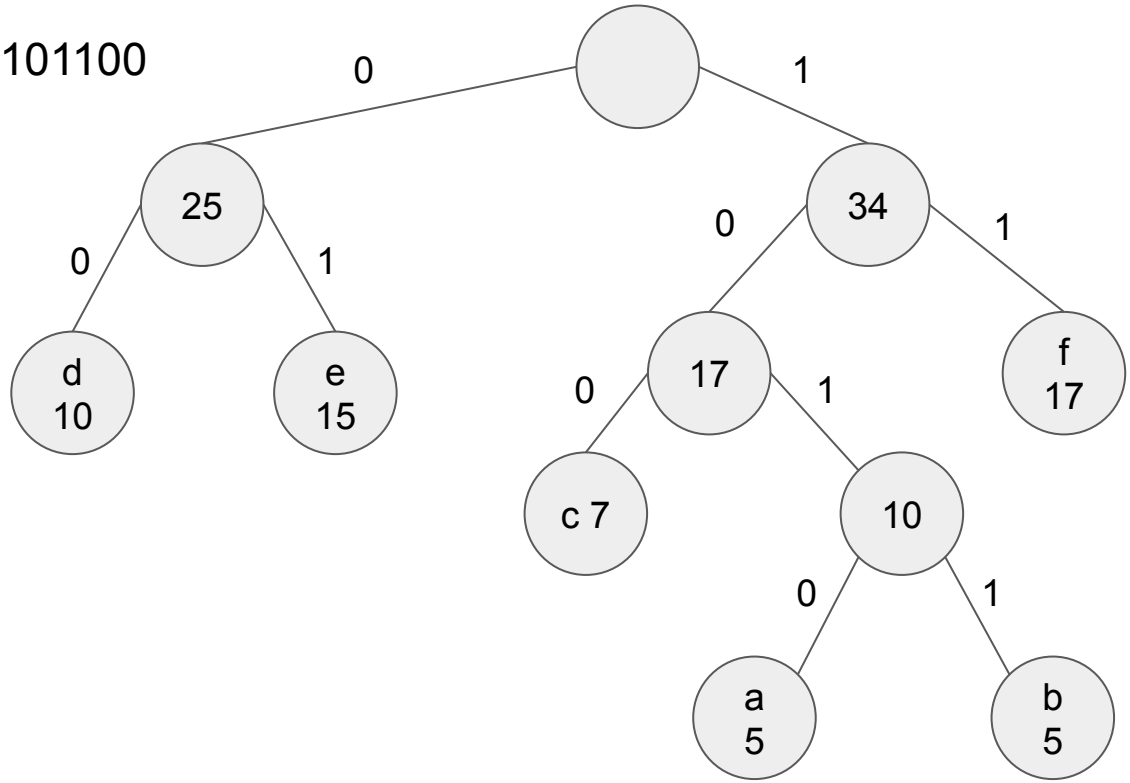


Example

a: 5, b: 5, c: 7, d: 10, e: 15, f: 17

Questions:

- The total cost
- Code for b, e?
- Decode for 001101100



Proof of optimality - claim

Proof sketch

- Greedy choice property:

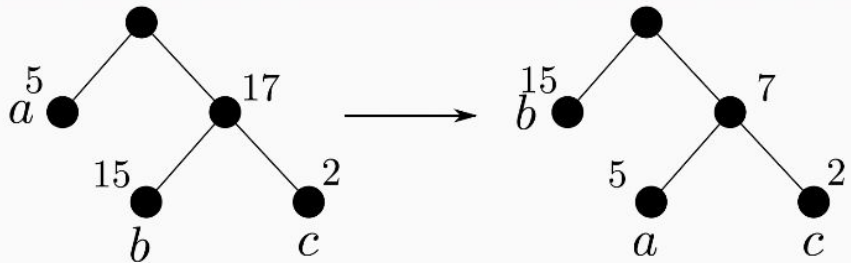
Claim

Every optimal solution has two lowest frequent symbols as leaves connected to an internal node of greatest depth

Proof. (exchange argument).

Suppose we have a tree T with two lowest frequent symbols not as deep as possible. Then at least one has a smaller depth. Switch it with one of the deepest nodes that is more frequent.

This improves the encoding length. Thus T is not optimal $\square$



Proof of optimality - by induction

- Alphabet Γ, f(a) are counts of the symbols
- For N = 2, the proof is trivial

Proof of optimality - by induction

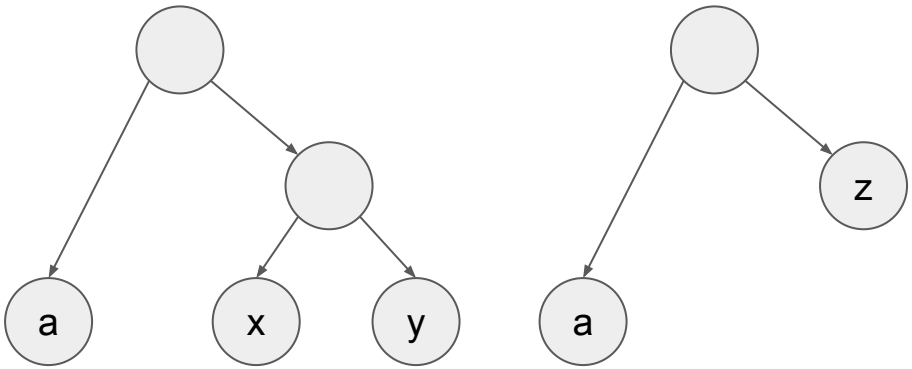
- Alphabet Γ, f(a) are counts of the symbols
- For N = 2, the proof is trivial
- **Induction hypothesis:** The algorithm produces an optimal tree for alphabets with $n-1 \geq 2$ symbols

Proof of optimality - by induction

- Alphabet Γ, f(a) are counts of the symbols
- For N = 2, the proof is trivial
- **Induction hypothesis:** The algorithm produces an optimal tree for alphabets with $n-1 \geq 2$ symbols
- Γ has N symbols,
  - H is a tree produced by Huffman algorithm
  - x and y have lowest counts
- **According to the claim:** symbols x and y are siblings in the tree

Proof of optimality - by induction

- Alphabet Γ, f(a) are counts of the symbols
- For N = 2, the proof is trivial
- **Induction hypothesis:** The algorithm produces an optimal tree for alphabets with n−1 ≥ 2 symbols
- Γ has N symbols,
  - H is a tree produced by Huffman algorithm
  - x and y have lowest counts
- **According to the claim:** symbols x and y are siblings in the tree
- Let's modify Γ:
  - Γ' = Γ – {x, y} + {z} (n – 1 symbols)
  - f(z) = f(x) + f(y)
- Let's modify H:
  - Remove x and y, replace their parent with z:
  - H' = H – {x, y} + {z}



Proof of optimality - by induction

- Alphabet Γ, f(a) are counts of the symbols
- For N = 2, the proof is trivial
- **Induction hypothesis:** The algorithm produces an optimal tree for alphabets with $n-1 \geq 2$ symbols
- Γ has N symbols,
  - H is a tree produced by Huffman algorithm
  - x and y have lowest counts
- **According to the claim:** symbols x and y are siblings in the tree
- Let's modify Γ:
  - Γ' = Γ – {x, y} + {z} (n – 1 symbols)
  - f(z) = f(x) + f(y)
- Let's modify H:
  - Remove x and y, replace their parent with z:
  - H' = H – {x, y} + {z}

- cost(H) = cost(H') + f(x) + f(y)
- H' is a tree that can be produced by Huffman algorithm for Γ'
- **H' is optimal for Γ' according to the induction claim**

Proof of optimality - by induction

Previous steps:

- Let's modify Γ:
  - Γ' = Γ – {x, y} + {z}
  - f(z) = f(x) + f(y)
- Let's modify H:
  - Remove x and y, put z instead, so
  - H' = H – {x, y} + {z}
- cost(H) = cost(H') + f(x) + f(y)
- H' is optimal for Γ' according to the induction claim

New steps:

- Let's T be an optimal tree for Γ, f. x and y are siblings in the tree too
- T' can be computed in the same way as H'

- cost(T) = cost(T') + f(x) + f(y) ≥ cost(H') + f(x) + f(y) = cost(H)
- H is the an optimal tree too

Pseudocode

```
1  def HUFFMAN(f): // f : f[1], ..., f[n]; Γ has n symbols
2      T: empty tree;
3      H: priority queue ordered by f;
4      for i := 1 in n:
5          insert(H, i);
6      for k := n + 1 in 2n − 1:
7          i := extract_min(H);
8          j := extract_min(H);
9          Creat a node k in T with children i and j;
10         f[k] := f[i] + f[j];
11         insert(H, k);
```

Pseudocode

```
1  def HUFFMAN(f): // f : f[1], ..., f[n]; Γ has n symbols
2      T: empty tree;
3      H: priority queue ordered by f;
4      for i := 1 in n:
5          insert(H, i);
6      for k := n + 1 in 2n − 1:
7          i := extract_min(H);
8          j := extract_min(H);
9          Creat a node k in T with children i and j;
10         f[k] := f[i] + f[j];
11         insert(H, k);
```

Binary heap: insert $O(\log n)$, extract_min: $O(\log n)$

Lines 4-5: $O(n \log n)$; Lines 6-11: $O(n \log n)$

Total cost: $O(n \log n)$

Question

```
1  def HUFFMAN(f): // f : f[1], ..., f[n]; Γ has n symbols
2      T: empty tree;
3      H: priority queue ordered by f;
4      for i := 1 in n:
5          insert(H, i);
6      for k := n + 1 in 2n − 1:
7          i := extract_min(H);
8          j := extract_min(H);
9          Creat a node k in T with children i and j;
10         f[k] := f[i] + f[j];
11         insert(H, k);
```

Question: why $2n - 1$ in line 6?

Question

```
1  def HUFFMAN(f): // f : f[1], ..., f[n]; Γ has n symbols
2      T: empty tree;
3      H: priority queue ordered by f;
4      for i := 1 in n:
5          insert(H, i);
6      for k := n + 1 in 2n − 1:
7          i := extract_min(H);
8          j := extract_min(H);
9          Creat a node k in T with children i and j;
10         f[k] := f[i] + f[j];
11         insert(H, k);
```

Question: why $2n-1$ in line 6?

Answer: if a full binary tree has n leaves, then it has $2n-1$ total nodes

Sum of the geometric series 1 + 2 + 4 + … + n for the tree where each level has 2^k vertices

Any full binary tree proof by induction

Data encoding and compressing

Encoding: rewriting a string in a new alphabet

Compressing: representing information in a compact way

Data encoding and compressing

Encoding: rewriting a string in a new alphabet

ASCII/8859-1 Text

A	0100 0001
S	0101 0011
C	0100 0011
I	0100 1001
I	0100 1001
/	0010 1111
8	0011 1000
8	0011 1000
5	0011 0101
9	0011 1001
-	0010 1101
1	0011 0001
	0010 0000
t	0111 0100
e	0110 0101
x	0111 1000
t	0111 0100

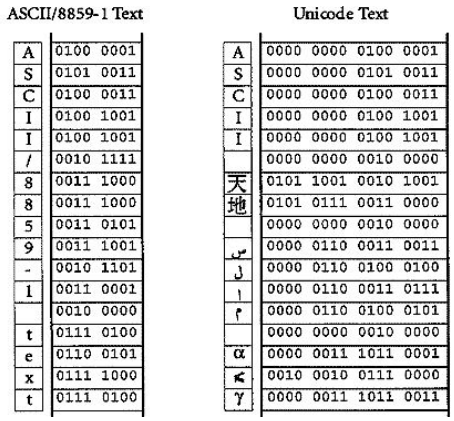
Unicode Text

A	0000 0000 0100 0001
S	0000 0000 0101 0011
C	0000 0000 0100 0011
I	0000 0000 0100 1001
I	0000 0000 0100 1001
	0000 0000 0010 0000
天	0101 1001 0010 1001
地	0101 0111 0011 0000
	0000 0000 0010 0000
ي	0000 0110 0011 0011
ل	0000 0110 0100 0100
ا	0000 0110 0011 0111
م	0000 0110 0100 0101
	0000 0000 0010 0000
α	0000 0011 1011 0001
κ	0010 0010 0111 0000
γ	0000 0011 1011 0011

Compressing: representing information in a compact way

Data encoding and compressing

Encoding: rewriting a string in a new alphabet



ASCII/8859-1 Text	
A	0100 0001
S	0101 0011
C	0100 0011
I	0100 1001
I	0100 1001
/	0010 1111
8	0011 1000
8	0011 1000
5	0011 0101
9	0011 1001
-	0010 1101
1	0011 0001
	0010 0000
t	0111 0100
e	0110 0101
x	0111 1000
t	0111 0100

Unicode Text	
A	0000 0000 0100 0001
S	0000 0000 0101 0011
C	0000 0000 0100 0011
I	0000 0000 0100 1001
I	0000 0000 0100 1001
	0000 0000 0010 0000
天	0101 1001 0010 1001
地	0101 0111 0011 0000
	0000 0000 0010 0000
ي	0000 0110 0011 0011
ل	0000 0110 0100 0100
ا	0000 0110 0011 0111
م	0000 0110 0100 0101
	0000 0000 0010 0000
α	0000 0011 1011 0001
κ	0010 0010 0111 0000
γ	0000 0011 1011 0011

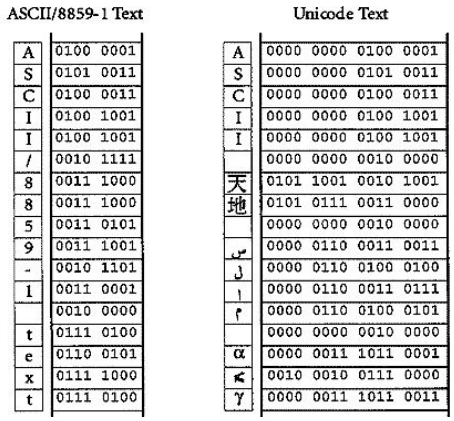
Huffman algorithm represents both!

Some ZIP-like decompressing tools

Compressing: representing information in a compact way

Data encoding and compressing

Encoding: rewriting a string in a new alphabet



ASCII/8859-1 Text	
A	0100 0001
S	0101 0011
C	0100 0011
I	0100 1001
I	0100 1001
/	0010 1111
8	0011 1000
8	0011 1000
5	0011 0101
9	0011 1001
-	0010 1101
1	0011 0001
	0010 0000
t	0111 0100
e	0110 0101
x	0111 1000
t	0111 0100

Unicode Text	
A	0000 0000 0100 0001
S	0000 0000 0101 0011
C	0000 0000 0100 0011
I	0000 0000 0100 1001
I	0000 0000 0100 1001
	0000 0000 0010 0000
天	0101 1001 0010 1001
地	0101 0111 0011 0000
	0000 0000 0010 0000
ي	0000 0110 0011 0011
ل	0000 0110 0100 0100
ا	0000 0110 0011 0111
م	0000 0110 0100 0101
	0000 0000 0010 0000
α	0000 0011 1011 0001
κ	0010 0010 0111 0000
γ	0000 0011 1011 0011

Huffman algorithm represents both!

Some ZIP-like decompressing tools

Compressing: representing information in a compact way

- Huffman coding is not an always optimal compressing tool
- Previously, we proved that Huffman coding provides an optimal solution. Where is the contradiction?

Compressing

Huffman encoding compresses the data in a very specific way:

- each symbol represented by its own binary code

Compressing

Huffman encoding compresses the data in a very specific way:

- each symbol represented by its own binary code

S = aaaaaaaabbbbbbccccaaaaaaa (f(a) = 15, f(b) = 6, f(c) = 4)

a = 0, b = 10, c = 11

Cost(S) = 15 * 1 + 6 * 2 + 4 * 2 = 35

Another compression: a8b6c4a7

Compressing

Huffman encoding compresses the data in a very specific way:

- each symbol represented by its own binary code

S = abcabcabcabcabcabcabcabcabc (f(a) = 9, f(b) = 9, f(c) = 9)

a = 0, b = 10, c = 11

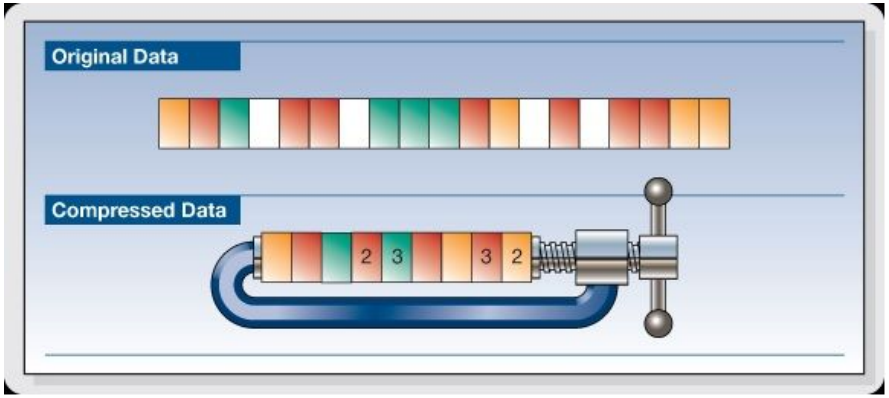
Cost(S) = 9 * 1 + 9 * 2 + 9 * 2 = 45

Another compression: (abc)9

Compressing

Compression tools find redundancy in the data and represent it in a compact way

Sometimes representation is not alphabet-based (no codes for each symbol)



- General
- Audio
- Image
- Video
- Genetic information: large strings over a small alphabet {A, C, G, T}