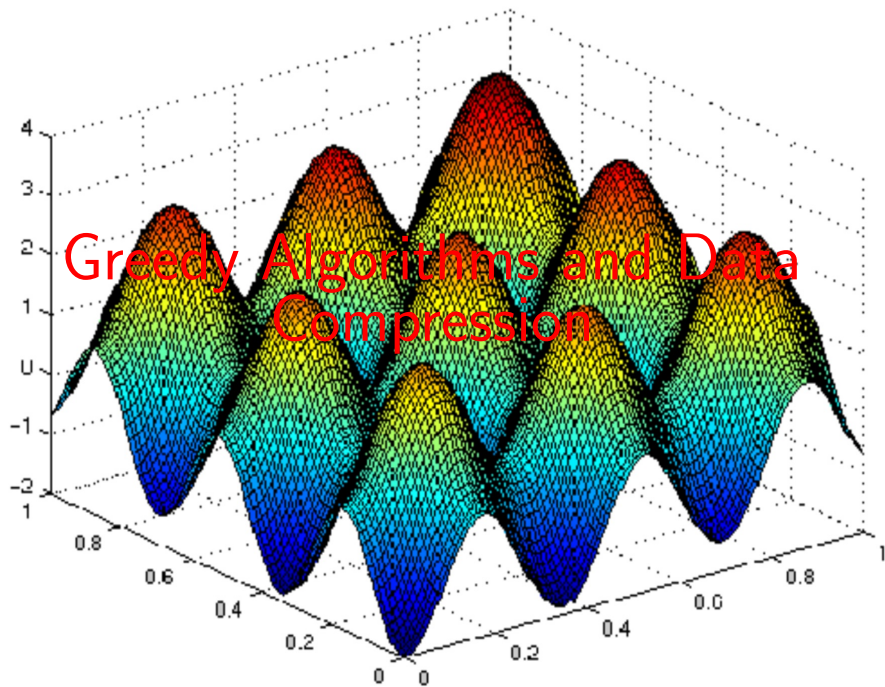
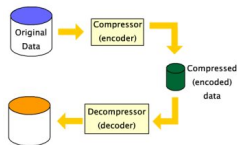


Greedy Algorithms and Data Compression



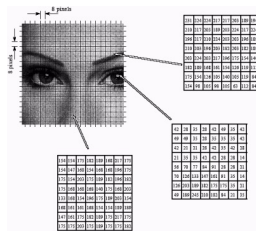
DATA COMPRESSION: Huffman's algorithm

Given as input a text \mathcal{T} over a finite alphabet Σ . We want to represent \mathcal{T} with as few bits as possible.



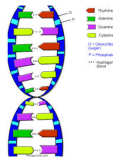
The goal of data compression is to reduce the time to transmit large files, and to reduce the space to store them.

If we are using variable-length encoding we need a system easy to encode and decode.



Example.

AAACAGTTGCAT ... GGTCCCTAGG
130.000.000



- ▶ *Fixed-length encoding*: $A = 00$, $C = 01$, $G = 10$ and $T = 11$. Needs 260Mbytes to store.
- ▶ *Variable-length encoding*: If A appears 7×10^8 times, C appears 3×10^6 times, G 2×10^8 and T 37×10^7 , better to assign a shorter string to A and longer to C

Prefix property

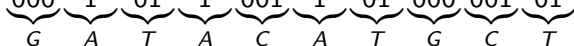
Given a set of symbols Σ , a **prefix code**, is $\phi : \Sigma \rightarrow \{0, 1\}^+$ (symbols to chain of bits) where for distinct $x, y \in \Sigma$, $\phi(x)$ is not a prefix of $\phi(y)$.

If $\phi(A) = 1$ and $\phi(C) = 101$ then ϕ is **no** prefix code.

$\phi(A) = 1, \phi(T) = 01, \phi(G) = 000, \phi(C) = 001$ is prefix code.

Prefix codes easy to decode (left-to-right):

000101100110100000101

000	1	01	1	001	1	01	000	001	01
									
G	A	T	A	C	A	T	G	C	T

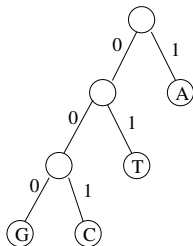
Prefix tree

We can identify an encoding with prefix property with a labeled binary tree.

A **prefix tree** T is a binary tree with the following properties:

- ▶ One leaf for symbol,
- ▶ Left edge labeled 0 and right edge labeled 1,
- ▶ Labels on the path from the root to a leaf specify the code for that leaf.

A prefix tree for $\Sigma = \{A, T, G, C\}$ is



The encoding length

Given a text S on Σ , with $|S| = n$, and a prefix code ϕ , which is the length $B(S)$ of the encoded text?

$\forall x \in \Sigma$, define the **frequency** of x as

$$f(x) = \frac{\text{number occurrences of } x \in S}{n}$$

Notice: $\sum_{x \in \Sigma} f(x) = 1$.

The encoding length of S is

$$B(S) = \sum_{x \in \Sigma} n f(x) |\phi(x)| = n \underbrace{\sum_{x \in \Sigma} f(x) |\phi(x)|}_{\alpha(S)}.$$

$\alpha(S) = \sum_{x \in \Sigma} f(x) |\phi(x)|$ is the **average number of bits** per symbol.

The encoding length

In terms of the prefix tree of ϕ , given x and $f(x)$, the length of the codeword $|\phi(x)|$ is the depth of x in T , let us denote it by $d_x(T)$.

Let $\alpha(T) = \sum_{x \in \Sigma} f(x) d_x(T)$.

Example.

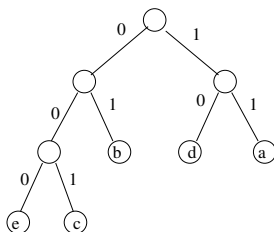
Let $\Sigma = \{a, b, c, d, e\}$ and let S be a text over Σ .

Let $f(a) = .32, f(b) = .25, f(c) = .20, f(d) = .18, f(e) = .05$

If we use a fixed length code we need $\lceil \lg 5 \rceil = 3$ bits.

Consider the prefix-code ϕ_1 :

$\phi_1(a) = 11, \phi_1(b) = 01, \phi_1(c) = 001, \phi_1(d) = 10, \phi_1(e) = 000$



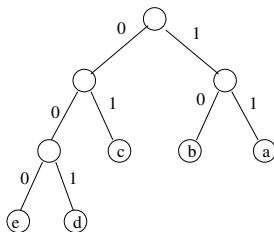
$$\alpha = .32 \cdot 2 + .25 \cdot 2 + .20 \cdot 3 + .18 \cdot 2 + .05 \cdot 3 = 2.25$$

In average, ϕ_1 reduces the bits per symbol over the fixed-length code from 3 to 2.25, about 25%

Is that the maximum reduction?

Consider the prefix-code ϕ_2 :

$$\phi_2(a) = 11, \phi_2(b) = 10, \phi_2(c) = 01, \phi_2(d) = 001, \phi_2(e) = 000$$



$$\alpha = .32 \cdot 2 + .25 \cdot 2 + .20 \cdot 2 + .18 \cdot 3 + .05 \cdot 3 = 2.23$$

is that the best? (the maximal compression)

Optimal prefix code.

Given a text, an **optimal prefix code** is a prefix code that minimizes the total number of bits needed to encode the text.

Note that an optimal encoding minimizes α .

Intuitively, in the prefix tree of an optimal prefix code, symbols with high frequencies should have small depth and symbols with low frequency should have large depth.

We will seek to construct an optimal prefix tree.

A property of optimal prefix trees.

A binary tree T is **full** if every interior node has two sons.

Lemma

The binary prefix tree corresponding to an optimal prefix code is full.

Proof.

Let T be the prefix tree of an optimal code, and suppose it contains a u with a son v .

If u is the root, construct T' by deleting u and using v com root. T' will yield a code with less bits to code the symbols.

Contradiction to optimality of T .

If u is not the root, let w be the father of u . Construct T' by deleting u and connecting directly v to w . Again this decreases the number of bits, contradiction to optimality of T . □

Greedy approach: Huffman code

Greedy approach due to David Huffman
(1925-99) in 1952, while he was a PhD student
at MIT



Wish to produce a labeled binary full tree, in which the leaves are as close to the root as possible. Moreover symbols with low frequency will be placed deeper than the symbol with high frequency.

Greedy approach: Huffman code

- ▶ Given S assume we computed $f(x)$ for every $x \in \Sigma$
- ▶ Sort the symbols by increasing f . Keep the dynamic sorted list in a priority queue Q .
- ▶ Construct a tree in bottom-up fashion, take two first elements of Q join them by a new *virtual node* with f the sum of the f 's of its sons, and place the new node in Q .
- ▶ When Q is empty, the resulting tree will be prefix tree of an optimal prefix code.

Huffman Coding: Construction of the tree.

Huffman Σ, S

Given Σ and S {compute the frequencies $\{f\}$ }

Construct priority queue Q of leaves for Σ , ordered by increasing f

while $Q.size() > 1$ **do**

 create a new node z

$x = \text{Extract-Min}(Q)$

$y = \text{Extract-Min}(Q)$

 make x, y the sons of z

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

 Insert $(Q, z, f(z))$

end while

$\Phi = \text{Extract-Min}(Q)$

If Q is implemented by a Heap, the algorithm has a complexity $O(n \lg n)$.

Example

Consider the text: *for each rose, a rose is a rose, the rose*

with $\Sigma = \{\text{for/ each/ rose/ a/ is/ the/ ,/ } \}$

Frequencies:

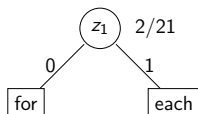
$f(\text{for}) = 1/21$, $f(\text{rose}) = 4/21$, $f(\text{is}) = 1/21$,

$f(\text{a}) = 2/21$, $f(\text{each}) = 1/21$, $f(,) = 2/21$,

$f(\text{the}) = 1/21$, $f(\text{b}) = 9/21$.

Priority Queue:

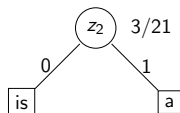
$Q = ((\text{for}:1/21), (\text{each}:1/21), (\text{is}:1/21), (\text{a}:2/21), (,:2/21), (\text{the}:2/21), (\text{rose}:4/21), (\text{b}:9/21))$



Then, $Q = ((\text{is}:1/21), (\text{a}:2/21), (,:2/21), (\text{the}:2/21), (z_1:2/21), (\text{rose}:4/21), (\text{b}:9/21))$

Example.

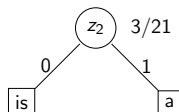
$$Q=((\text{is}:1/21), (\text{a}:2/21), (, :2/21), (\text{the}:2/21), (z_1:2/21), (\text{rose}:4/21), (\text{b}:9/21))$$



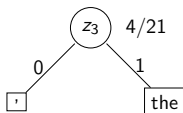
Then, $Q=((, :2/21), (\text{the}:2/21), (z_1:3/21), (z_2:3/21), (\text{rose}:4/21), (\text{b}:9/21))$

Example.

$Q=((\text{is}:1/21), (\text{a}:2/21), (, :2/21), (\text{the}:2/21), (z_1:2/21), (\text{rose}:4/21), (\text{b}:9/21))$



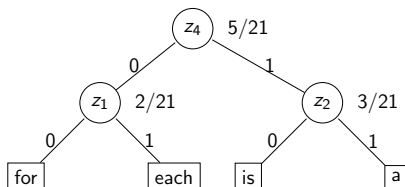
Then, $Q=((, :2/21), (\text{the}:2/21), (z_1:3/21), (z_2:3/21), (\text{rose}:4/21), (\text{b}:9/21))$



Then, $Q=((z_1:2/21), (z_2:3/21), (\text{rose}:4/21), (z_3:4/21), (\text{b}:9/21))$

Example.

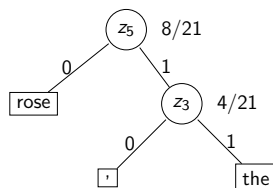
$$Q=((z_1:2/21), (z_2:3/21), (rose:4/21), (z_3:4/21), (b:9/21))$$



Then, $Q=((rose:4/21), (z_3:4/21), (z_4:5/21), (b:9/21))$

Example.

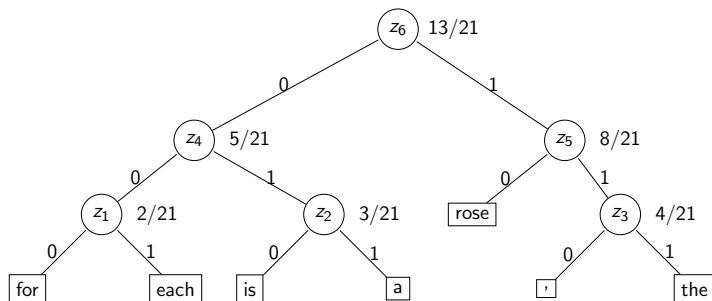
$$Q=((\text{rose}:4/21), (z_3:4/21), (z_4:5/21), (b:9/21))$$



Then, $Q=((z_4:5/21), (z_5:8/21), (b:9/21))$

Example.

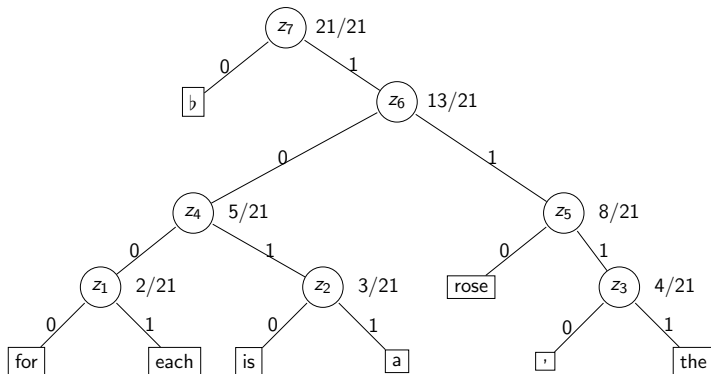
$$Q=((z_4:5/21), (z_5:8/21), (b:9/21))$$



Then, $Q=((b:9/21), (z_6:13/21))$

Example.

$$Q=((b:9/21),(z_6:13/21))$$



Then, $Q=((z_7:21/21))$

for each rose, a rose is a rose, the rose is Huffman coded as

10000100101101110010110110010100101101101110011110110

Example

Therefore *for each rose, a rose is a rose, the rose* is Huffman codified as

10000100101101110010110110010100101101101110011110110

which has encoded length 53, and $\alpha = 53/21 = 2.523\dots$

With a fix size code, we need 4 bits per symbol \Rightarrow 84 bits instead of the 53 we use with Huffman's.

The solution is not unique!

Why does the Huffman's algorithm produce an optimal prefix code?

Correctness.

Theorem (Greedy property)

Let Σ be an alphabet, and let x, y be two symbols with the lowest frequency. Then, there is an optimal prefix code Φ in which the code for x and y have the same length and they differ only in the last bit.

Proof.

Assume that T is optimal but that x and y have not the same code length. In T there must be two symbols a and b siblings at max. depth. Assume $f(a) \leq f(b)$ and $f(x) \leq f(y)$, otherwise sort them accordingly.

We construct T' by exchanging x with a and y with b . As $f(x) \leq f(a)$ and $f(y) \leq f(b)$ then $B(T') \leq B(T)$. So T' is optimal and verifies the property. □

Theorem (Optimal substructure)

Assume T' is an optimal prefix tree for $(\Sigma - \{x, y\}) \cup \{z\}$ where x, y are two symbols with the lowest frequencies, and z has frequency $f(x) + f(y)$. The T obtained from T' by making x and y children of z is an optimal prefix tree for Σ .

Proof.

Let T_0 be any prefix tree for Σ . Must show $B(T) \leq B(T_0)$.

We only need to consider T_0 where x and y are siblings, their parent has frequency $f(x) + f(y)$.

Let T'_0 be obtained by removing x, y from T_0 . As T'_0 is a prefix tree for $(\Sigma - \{x, y\}) \cup \{z\}$, then $B(T'_0) \geq B(T')$.

Comparing T_0 with T'_0 we get,

$B(T'_0) + f(x) + f(y) = B(T_0)$ and $B(T') + f(x) + f(y) = B(T)$,

Putting together the three identities, we get $B(T) \leq B(T_0)$. \square

Optimality of Huffman

Huffman is optimal under assumptions:

- ▶ The compression is **lossless**, i.e. *uncompressing the compressed file yield the original file.*
- ▶ We must know the alphabet beforehand (characters, words, etc.),
- ▶ We must pre-compute the frequencies of symbols, i.e. read the data twice, **which make it very slow for many real applications.**
- ▶ A good source for extensions of Huffman encoding compression is the Wikipedia article on it:
https://en.wikipedia.org/wiki/Huffman_coding.

Greedy and approximation algorithms

- ▶ Many times the Greedy strategy yields a **feasible solution** with value which is **near** to the optimum solution.
- ▶ In many practical cases, when finding the global optimum is hard, the greedy may yield a *good enough* feasible solution:
An approximation to the optimal solution.
- ▶ An **approximation algorithm** for the problem always computes a close valid output. Heuristics also could yield good solutions, but they do not have a theoretical guarantee of closeness.
- ▶ Greedy is one of the algorithmic techniques used to design approximations algorithms.

Greedy and approximation algorithms

- ▶ For any optimization problem, let $c(*)$ be the value of the optimization function, let \mathcal{A} be an algorithm, that for each input x produces a valid solution $\mathcal{A}(x)$ to x . Let $\text{opt}(x)$ be the cost of an optimal solution to x .
- ▶ We want to design a **fast** algorithm that produce solutions **close** to the optimal.
- ▶ For a NP-hard problem, we don't know if it has polynomial time algorithms, we want to design algorithms that are fast (polynomial) and that outputs **approximate**.

Approximation algorithm: Formal definition

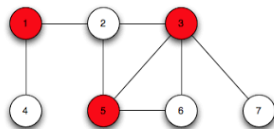
- ▶ For a given optimization problem, let \mathcal{A} be an algorithm, that for each input x produces a valid solution with cost $\mathcal{A}(x)$ to x . Let $\text{opt}(x)$ be the cost of an optimal solution to x .
- ▶ For $r > 1$, \mathcal{A} is an **r -approximation algorithm** if, for any input x :

$$\frac{1}{r} \leq \frac{\mathcal{A}(x)}{\text{opt}(x)} \leq r.$$

- ▶ r is called the **approximation ratio**.
- ▶ Given an optimization problem, for any input x :
 - ▶ in a MAX problem, we require $\mathcal{A}(x) \leq \text{opt}(x) \leq r\mathcal{A}(x)$.
 - ▶ in a MIN problem, we require $\text{opt}(x) \leq \mathcal{A}(x) \leq r\text{opt}(x)$.

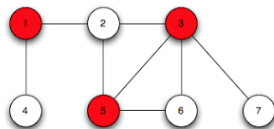
An easy example: VERTEX COVER problem

Recall the problem of Vertex cover: Given a graph $G = (V, E)$ with $|V| = n, |E| = m$ find the minimum set of vertices $S \subseteq V$ such that it covers every edge of G .



An easy example: VERTEX COVER problem

Recall the problem of Vertex cover: Given a graph $G = (V, E)$ with $|V| = n, |E| = m$ find the minimum set of vertices $S \subseteq V$ such that it covers every edge of G .



GreedyVC for I : $G = (V, E)$

$E' = E, S = \emptyset,$

while $E' \neq \emptyset$ **do**

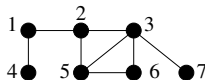
 Pick $e \in E'$, say $e = (u, v)$

$S = S \cup \{u, v\},$

$E' = E' - \{(u, v) \cup \{\text{edges incident to } u, v\}\}$

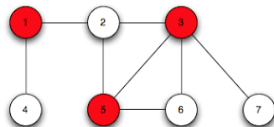
end while

return S .



An easy example: VERTEX COVER problem

Given a graph $G = (V, E)$ with $|V| = n, |E| = m$ find the minimum set of vertices $S \subseteq V$ such that it covers every edge of G .



GreedyVC $G = (V, E)$

$E' = E, S = \emptyset,$

while $E' \neq \emptyset$ **do**

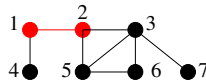
 Pick $e \in E'$, say $e = (u, v)$

$S = S \cup \{u, v\},$

$E' = E' - \{(u, v) \cup \{\text{edges incident to } u, v\}\}$

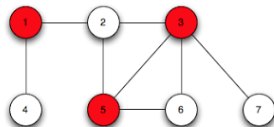
end while

return $S.$



An easy example: VERTEX COVER problem

Given a graph $G = (V, E)$ with $|V| = n, |E| = m$ find the minimum set of vertices $S \subseteq V$ such that it covers every edge of G .



GreedyVC $G = (V, E)$

$E' = E, S = \emptyset,$

while $E' \neq \emptyset$ **do**

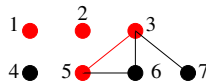
 Pick $e \in E'$, say $e = (u, v)$

$S = S \cup \{u, v\},$

$E' = E' - \{(u, v) \cup \{\text{edges incident to } u, v\}\}$

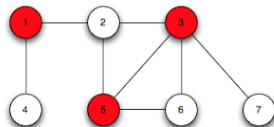
end while

return $S.$



An easy example: VERTEX COVER problem

Given a graph $G = (V, E)$ with $|V| = n, |E| = m$ find the minimum set of vertices $S \subseteq V$ such that it covers every edge of G



GreedyVC $G = (V, E)$

$E' = E, S = \emptyset,$

while $E' \neq \emptyset$ **do**

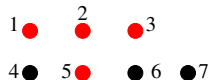
 Pick $e \in E'$, say $e = (u, v)$

$S = S \cup \{u, v\},$

$E' = E' - \{(u, v) \cup \{\text{edges incident to } u, v\}\}$

end while

return $S.$



An easy example: Vertex cover

Theorem GreedyVC runs in $O(m + n)$ steps. Moreover, if S is solution computed on input G , $|S| \leq 2\text{opt}(G)$.

Proof.

- ▶ The edges selected among by **GreedyVC** do not share any vertex.
- ▶ Therefore, an optimal solution must have at least one of the two endpoints of each edge while **GreedyVC** takes both.
- ▶ So, $|S| \leq 2\text{opt}(G)$.

The decision problem for Vertex Cover is NP-complete. Moreover, unless $P=NP$, vertex cover can't be approximated within a factor $r \leq 1.36$