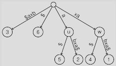
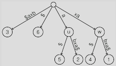


Datenstrukturen und Effiziente Algorithmen

Vorlesung *Datenstrukturen und Effiziente Algorithmen* im WS 18/19

Marc Hellmuth
Institut für Mathematik und Informatik
Universität Greifswald

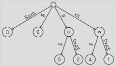


Greedy Heuristic

Greedy heuristic

> In a combinatorial optimization problem, try to find a (near) optimal solution, by making a sequence of choices such that each choice appears to be optimal at the time of choice.

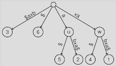
The greedy heuristic is usually efficient but does not always produce a correct or near optimal result.



Design Principles of a Greedy Algorithm

Design Principles

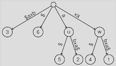
1. Choice and subproblem:
   Formulate the optimization problem as one in which a choice is made which leaves a subproblem to solve.
2. Greedy choice is safe:
   Prove that there is always an optimal solution to the original problem that makes the greedy choice.
3. Demonstrate optimal substructure:
   After choosing greedily, an optimal solution to the subproblem combined with the greedy choice yields an optimal solution to the original problem.



Minimum Spanning Tree

Definition 1 (Spanning Tree)

Let $G = (V, E)$ be a weighted, connected, undirected graph and $w(\{u, v\})$ be the weight of edge $\{u, v\}$.
A spanning tree of G is a subset $T \subset E$ such that (V, T) is a tree.

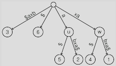


Minimum Spanning Tree

Definition 1 (Spanning Tree)

Let $G = (V, E)$ be a weighted, connected, undirected graph and $w(\{u, v\})$ be the weight of edge $\{u, v\}$.
A spanning tree of G is a subset $T \subset E$ such that (V, T) is a tree.
A minimum spanning tree (MST) is a spanning tree of minimum weight $w(T)$, where

$$w(T) := \sum_{e \in T} w(e).$$



Minimum Spanning Tree

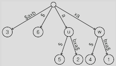
Definition 1 (Spanning Tree)

Let $G = (V, E)$ be a weighted, connected, undirected graph and $w(\{u, v\})$ be the weight of edge $\{u, v\}$.
A spanning tree of G is a subset $T \subset E$ such that (V, T) is a tree.
A minimum spanning tree (MST) is a spanning tree of minimum weight $w(T)$, where

$$w(T) := \sum_{e \in T} w(e).$$

MST problem

Find a minimum spanning tree for a given weighted, connected, undirected graph.



Kruskal's Algorithm

Kruskal's Algorithm

$T \leftarrow \{\}$

for each $v \in V$ **do**

    MAKE-SET(v)

sort the edges in E in nondecreasing order by weight

for each edge $\{u, v\} \in E$ in above order **do**

    if FIND-SET(u) $\neq$ FIND-SET(v) **then**

        $T \leftarrow T \cup \{\{u, v\}\}$

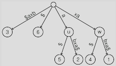
        UNION(u, v)

Kruskal's Algorithm

Kruskal's Algorithm

$T \leftarrow \{\}$
for each $v \in V$ **do**
    MAKE-SET(v)
sort the edges in E in nondecreasing order by weight
for each edge $\{u, v\} \in E$ in above order **do**
    if FIND-SET(u) $\neq$ FIND-SET(v) **then**
        $T \leftarrow T \cup \{\{u, v\}\}$
        UNION(u, v)

Running time



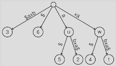
Kruskal's Algorithm

Kruskal's Algorithm

$T \leftarrow \{\}$
for each $v \in V$ **do**
    MAKE-SET(v)
sort the edges in E in nondecreasing order by weight
for each edge $\{u, v\} \in E$ in above order **do**
    if FIND-SET(u) $\neq$ FIND-SET(v) **then**
        $T \leftarrow T \cup \{\{u, v\}\}$
        UNION(u, v)

Running time

Observation: $|V| = O(|E|)$ as G is connected.



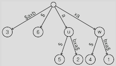
Kruskal's Algorithm

Kruskal's Algorithm

$T \leftarrow \{\}$
for each $v \in V$ **do**
    MAKE-SET(v)
sort the edges in E in nondecreasing order by weight
for each edge $\{u, v\} \in E$ in above order **do**
    if FIND-SET(u) $\neq$ FIND-SET(v) **then**
        $T \leftarrow T \cup \{\{u, v\}\}$
        UNION(u, v)

Running time

Observation: $|V| = O(|E|)$ as G is connected.
All MAKE-SET, FIND-SET, UNION operations together:
$O((|E| + |V|) \cdot \alpha(|V|)) = O(|E| \cdot \alpha(|V|))$



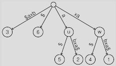
Kruskal's Algorithm

Kruskal's Algorithm

1: $T \leftarrow \{\}$
2: **for** each $v \in V$ **do**
3:     MAKE-SET(v)
4: sort the edges in E in nondecreasing order by weight
5: **for** each edge $\{u, v\} \in E$ in above order **do**
6:     **if** FIND-SET(u) $\neq$ FIND-SET(v) **then**
7:         $T \leftarrow T \cup \{\{u, v\}\}$
8:         UNION(u, v)

Running time

Observation: $|V| = O(|E|)$ as G is connected.
All MAKE-SET, FIND-SET, UNION operations together:
$O((|E| + |V|) \cdot \alpha(|V|)) = O(|E| \cdot \alpha(|V|))$
Line 4: $O(|E| \cdot \log |E|) = O(|E| \cdot \log |V|)$



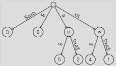
Kruskal's Algorithm

Kruskal's Algorithm

1: $T \leftarrow \{\}$
2: **for** each $v \in V$ **do**
3:     MAKE-SET(v)
4: sort the edges in E in nondecreasing order by weight
5: **for** each edge $\{u, v\} \in E$ in above order **do**
6:     **if** FIND-SET(u) $\neq$ FIND-SET(v) **then**
7:         $T \leftarrow T \cup \{\{u, v\}\}$
8:         UNION(u, v)

Running time

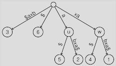
Observation: $|V| = O(|E|)$ as G is connected.
All MAKE-SET, FIND-SET, UNION operations together:
$O((|E| + |V|) \cdot \alpha(|V|)) = O(|E| \cdot \alpha(|V|))$
Line 4: $O(|E| \cdot \log |E|) = O(|E| \cdot \log |V|)$
Total time $O(|E| \log |V|)$
Here: α is the slightly superlinear function defined in section about *disjoint sets*.



Kruskal's Algorithm

Kruskal's algorithm is greedy

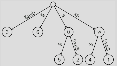
- Subproblems:
  For a given T that is a subset of a MST, find a MST containing T.



Kruskal's Algorithm

Kruskal's algorithm is greedy

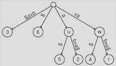
- Subproblems:
  For a given T that is a subset of a MST, find a MST containing T.
- Greedy choice:
  Choose the lightest edge e from $E \setminus T$ such that T remains acyclic.



Kruskal's Algorithm

Kruskal's algorithm is greedy

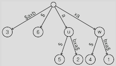
- **Subproblems:**
  For a given T that is a subset of a MST, find a MST containing T.
- **Greedy choice:**
  Choose the lightest edge e from $E \setminus T$ such that T remains acyclic.
- **Greedy choice is save:**
  Needs to be proven: $T \cup \{e\}$ is a subset of a MST



Kruskal's Algorithm

Kruskal's algorithm is greedy

- Subproblems:
  For a given T that is a subset of a MST, find a MST containing T.
- Greedy choice:
  Choose the lightest edge e from $E \setminus T$ such that T remains acyclic.
- Greedy choice is save:
  Needs to be proven: $T \cup \{e\}$ is a subset of a MST
- Optimal substructure:
  Trivial: A MST containing $T \cup \{e\}$ is a MST containing T



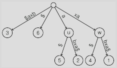
Prefix Codes

Binary Character Code

Let C be a finite set of objects.
A binary code or short code is an injective mapping

$$C \to \{0, 1\}^+.$$

Each object is represented by a unique binary string, its codeword.



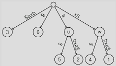
Prefix Codes

Binary Character Code

Let C be a finite set of objects.
A binary code or short code is an injective mapping

$$C \to \{0, 1\}^{+}.$$

Each object is represented by a unique binary string, its codeword.
If all codewords have the same length, then the code is said to be a fixed-length code, otherwise a variable-length code.



Prefix Codes

Binary Character Code

Let C be a finite set of objects.
A binary code or short code is an injective mapping

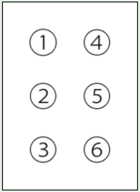
$$C \to \{0, 1\}^+.$$

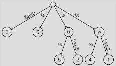
Each object is represented by a unique binary string, its codeword.
If all codewords have the same length, then the code is said to be a fixed-length code, otherwise a variable-length code.

Example 2 (Braille)

Braille is a fixed-length binary code.

object	codeword
A	100000
B	110000
C	100100
D	100110
...	...

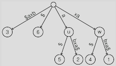




Prefix Codes

Prefix Code

A prefix code (*German*: präfixfreier Code) is a code in which no codeword is a prefix of another codeword.



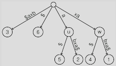
Prefix Codes

Prefix Code

A prefix code (*German*: präfixfreier Code) is a code in which no codeword is a prefix of another codeword.

Example 3

- phone numbers



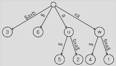
Prefix Codes

Prefix Code

A prefix code (*German*: präfixfreier Code) is a code in which no codeword is a prefix of another codeword.

Example 3

- phone numbers
- Morse code (ternary code: {short, long, pause})



Prefix Codes

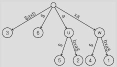
Prefix Code

A prefix code (*German*: präfixfreier Code) is a code in which no codeword is a prefix of another codeword.

Example 3

- phone numbers
- Morse code (ternary code: {short, long, pause})
- the code

A	0
B	10
C	110
D	111



Prefix Codes

Prefix Code

A prefix code (*German*: präfixfreier Code) is a code in which no codeword is a prefix of another codeword.

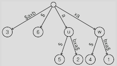
Example 3

- phone numbers
- Morse code (ternary code: {short, long, pause})
- the code

A	0
B	10
C	110
D	111

Decoding

For prefix codes a sequence of codewords can be decoded online:
100111 $\rightarrow$



Prefix Codes

Prefix Code

A prefix code (*German*: präfixfreier Code) is a code in which no codeword is a prefix of another codeword.

Example 3

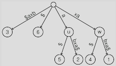
- phone numbers
- Morse code (ternary code: {short, long, pause})
- the code

A	0
B	10
C	110
D	111

Decoding

For prefix codes a sequence of codewords can be decoded online:

$100111 \to 10, 0, 111 \to$



Prefix Codes

Prefix Code

A prefix code (*German*: präfixfreier Code) is a code in which no codeword is a prefix of another codeword.

Example 3

- phone numbers
- Morse code (ternary code: {short, long, pause})
- the code

A	0
B	10
C	110
D	111

Decoding

For prefix codes a sequence of codewords can be decoded online:
$100111 \to 10, 0, 111 \to$ BAD

Huffman Codes

Lossless Data Compression with Prefix Codes

Consider a sequence s of objects and the corresponding sequence t of codewords.
Aim: Choose prefix code such that t has minimal length.

Huffman Codes

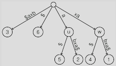
Lossless Data Compression with Prefix Codes

Consider a sequence s of objects and the corresponding sequence t of codewords.
Aim: Choose prefix code such that t has minimal length.

Example 4 (Protein sequences)

A protein sequence s of length 1000 consists of the following objects ($C = \{\text{amino acids}\}, |C| = 20$)).
A fixed-length code would require $\lceil \log_2 20 \rceil = 5$ bits per codeword and therefore 5000 bits for coding the whole sequence.

amino acid	1 letter	freq
Alanine	A	60
Arginine	R	67
Asparagine	N	37
Aspartic acid	D	53
Cysteine	C	17
Glutamic acid	E	60
Glutamine	Q	48
Glycine	G	76
Histidine	H	20
Isoleucine	I	39
Leucine	L	78
Lysine	K	60
Methionine	M	25
Phenylalanine	F	44
Proline	P	61
Serine	S	87
Threonine	T	51
Tryptophan	W	7
Tyrosine	Y	26
Valine	V	84



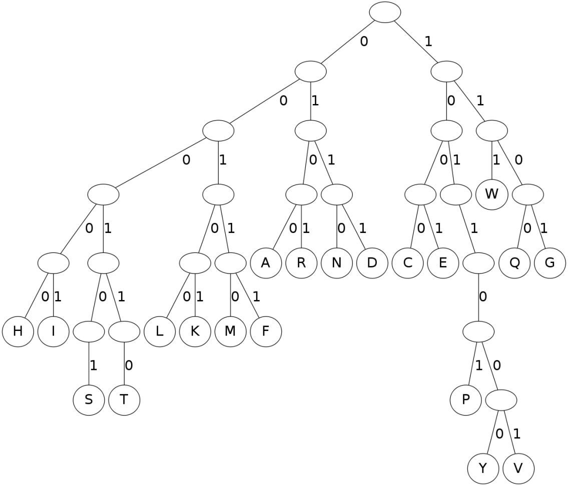
Huffman Codes

Tree representing a prefix code

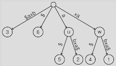
- a binary prefix code can be represented by a binary tree T with edge labels in $\{0, 1\}$
- the edges from an internal node to its sons have different labels
- decoding a sequence of codewords can be done efficiently parsing T root to leaf for each object

Huffman Codes

Example 5 (A tree representing a prefix code)



A has codeword 0100
Y has codeword 1011000
etc.

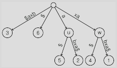


Huffman Codes

Cost of a tree

Let T be a binary tree representing a prefix code for objects in the set C. For $c \in C$

- let $c.freq > 0$ be the frequency of character c
- let $d_T(c)$ be the depth of the leaf representing the codeword for c (= number of bits in the codeword).



Huffman Codes

Cost of a tree

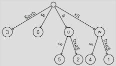
Let T be a binary tree representing a prefix code for objects in the set C. For $c \in C$

- let $c.freq > 0$ be the frequency of character c
- let $d_T(c)$ be the depth of the leaf representing the codeword for c (= number of bits in the codeword).

Define

$$B(T) := \sum_{c \in C} c.freq \cdot d_T(c)$$

to be the cost of tree T.
$B(T)$ is the total number of bits required when coding all objects using the code represented by T.



Huffman Codes

Cost of a tree

Let T be a binary tree representing a prefix code for objects in the set C. For $c \in C$

- let $c.freq > 0$ be the frequency of character c
- let $d_T(c)$ be the depth of the leaf representing the codeword for c (= number of bits in the codeword).

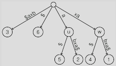
Define

$$B(T) := \sum_{c \in C} c.freq \cdot d_T(c)$$

to be the cost of tree T.
$B(T)$ is the total number of bits required when coding all objects using the code represented by T.

Example 6

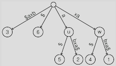
For above tree and frequencies we get the costs $B(T) = 60 \cdot 4 + 67 \cdot 4 + \cdots + 84 \cdot 7 = 4947$.



Huffman Codes

Observation

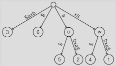
Every optimal tree (with minimal costs) is a full binary tree. (Why?)



Huffman Codes

Observation

Every optimal tree (with minimal costs) is a full binary tree. (Why?)
Let $n = |C|$ be the number of objects. The number of internal nodes of a full binary tree T with n leaves is $n - 1$.



Huffman Codes

Observation

Every optimal tree (with minimal costs) is a full binary tree. (Why?)
Let $n = |C|$ be the number of objects. The number of internal nodes of a full binary tree T with n leaves is $n - 1$.

Bottom-up strategy

Consider the following generic leaf-to-root strategy for construcing a full binary tree T.

```
n ← |C|
create a leaf node for every object in C
for i = 1..n − 1 do
    pick two nodes x and y from C
    create a new internal node z[i] with edges to x and y labeled 0 and 1, respectively
    z[i].freq ← x.freq + y.freq
    remove x and y from C and add z[i] to C
return z[n − 1] as root of the tree
```

Huffman Codes

Observations

1. above pseudocode is generic, it leaves open the choice of nodes x, y in line 4

Huffman Codes

Observations

1. above pseudocode is generic, it leaves open the choice of nodes x, y in line 4
2. every full binary tree T can be constructed with this procedure

Huffman Codes

Observations

1. above pseudocode is generic, it leaves open the choice of nodes x, y in line 4
2. every full binary tree T can be constructed with this procedure
3. $z[i].freq$ is the sum of the frequencies of all objects in the subtree rooted at $z[i]$

Huffman Codes

Observations

1. above pseudocode is generic, it leaves open the choice of nodes x, y in line 4
2. every full binary tree T can be constructed with this procedure
3. $z[i].freq$ is the sum of the frequencies of all objects in the subtree rooted at $z[i]$
4. $$\sum_{i=1}^{n-1} z[i].freq = B(T)$$

Huffman Codes

Observations

1. above pseudocode is generic, it leaves open the choice of nodes x, y in line 4
2. every full binary tree T can be constructed with this procedure
3. $z[i].freq$ is the sum of the frequencies of all objects in the subtree rooted at $z[i]$
4. $$\sum_{i=1}^{n-1} z[i].freq = B(T)$$

Proof.

$$\sum_{i=1}^{n-1} z[i].freq = \sum_{i=1}^{n-1} \sum_{\substack{c \in C \\ c \text{ descendant of } z[i]}} c.freq$$

Huffman Codes

Observations

1. above pseudocode is generic, it leaves open the choice of nodes x, y in line 4
2. every full binary tree T can be constructed with this procedure
3. $z[i].freq$ is the sum of the frequencies of all objects in the subtree rooted at $z[i]$
4. $$\sum_{i=1}^{n-1} z[i].freq = B(T)$$

Proof.

$$\sum_{i=1}^{n-1} z[i].freq = \sum_{i=1}^{n-1} \sum_{\substack{c \in C \\ c \text{ descendant of } z[i]}} c.freq = \sum_{c \in C} \sum_{\substack{1 \leq i < n \\ z[i] \text{ ancestor of } c}} c.freq$$

Huffman Codes

Observations

1. above pseudocode is generic, it leaves open the choice of nodes x, y in line 4
2. every full binary tree T can be constructed with this procedure
3. $z[i].freq$ is the sum of the frequencies of all objects in the subtree rooted at $z[i]$
4. $$\sum_{i=1}^{n-1} z[i].freq = B(T)$$

Proof.

$$\sum_{i=1}^{n-1} z[i].freq = \sum_{i=1}^{n-1} \sum_{\substack{c \in C \\ c \text{ descendant of } z[i]}} c.freq = \sum_{c \in C} \sum_{\substack{1 \leq i < n \\ z[i] \text{ ancestor of } c}} c.freq$$

$$= \sum_{c \in C} d_T(c) \cdot c.freq$$

Huffman Codes

Observations

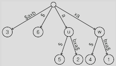
1. above pseudocode is generic, it leaves open the choice of nodes x, y in line 4
2. every full binary tree T can be constructed with this procedure
3. $z[i].\mathit{freq}$ is the sum of the frequencies of all objects in the subtree rooted at $z[i]$
4. $$\sum_{i=1}^{n-1} z[i].\mathit{freq} = B(T)$$

Proof.

$$\sum_{i=1}^{n-1} z[i].\mathit{freq} = \sum_{i=1}^{n-1} \sum_{\substack{c \in C \\ c \text{ descendant of } z[i]}} c.\mathit{freq} = \sum_{c \in C} \sum_{\substack{1 \leq i < n \\ z[i] \text{ ancestor of } c}} c.\mathit{freq}$$

$$= \sum_{c \in C} d_T(c) \cdot c.\mathit{freq} = B(T)$$

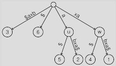
□



Huffman Codes

Greedy choice of internal nodes

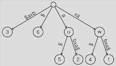
Want to minimize $B(T) = \sum_{i=1}^{n-1} z[i].freq$.



Huffman Codes

Greedy choice of internal nodes

Want to minimize $B(T) = \sum_{i=1}^{n-1} z[i].freq$.
Will minimize $z[i].freq$ in every step i, independent of considering future possible choices $z[j], j > i$.



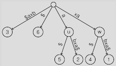
Huffman Codes

Greedy choice of internal nodes

Want to minimize $B(T) = \sum_{i=1}^{n-1} z[i].freq$.
Will minimize $z[i].freq$ in every step i, independent of considering future possible choices $z[j], j > i$.

HUFFMAN(C)

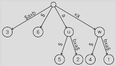
```
n ← |C|
construct a min-priority queue Q with elements C and
frequencies as keys
for i = 1..n − 1 do
    create a new internal node z[i]
    z[i].left ← x ← EXTRACT-MIN(Q)
    z[i].right ← y ← EXTRACT-MIN(Q)
    label edges from z[i] to x and y with 0 and 1,
respectively
    z[i].freq ← x.freq + y.freq
    INSERT(Q, z[i])
return z[n − 1] as root of the tree
```



Huffman Codes

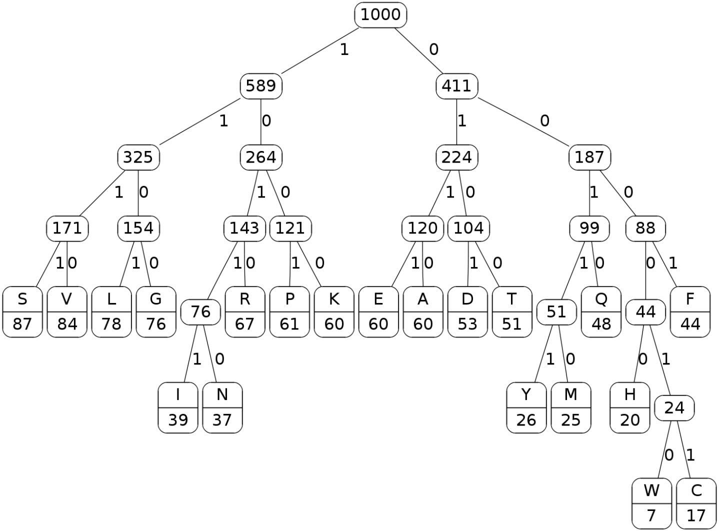
Example 7 (Huffman's algorithm)

$z[1].left = W$	$z[1].right = C$	$z[1].freq = 24$
$z[2].left = H$	$z[2].right = z[1]$	$z[2].freq = 44$
$z[3].left = M$	$z[3].right = Y$	$z[3].freq = 51$
$z[4].left = N$	$z[4].right = I$	$z[4].freq = 76$
$z[5].left = z[2]$	$z[5].right = F$	$z[5].freq = 88$
$z[6].left = Q$	$z[6].right = z[3]$	$z[6].freq = 99$
$z[7].left = T$	$z[7].right = D$	$z[7].freq = 104$
$z[8].left = A$	$z[8].right = E$	$z[8].freq = 120$
$z[9].left = K$	$z[9].right = P$	$z[9].freq = 121$
$z[10].left = R$	$z[10].right = z[4]$	$z[10].freq = 143$
$z[11].left = G$	$z[11].right = L$	$z[11].freq = 154$
$z[12].left = V$	$z[12].right = S$	$z[12].freq = 171$
$z[13].left = z[5]$	$z[13].right = z[6]$	$z[13].freq = 187$
$z[14].left = z[7]$	$z[14].right = z[8]$	$z[14].freq = 224$
$z[15].left = z[9]$	$z[15].right = z[10]$	$z[15].freq = 264$
$z[16].left = z[11]$	$z[16].right = z[12]$	$z[16].freq = 325$
$z[17].left = z[13]$	$z[17].right = z[14]$	$z[17].freq = 411$
$z[18].left = z[15]$	$z[18].right = z[16]$	$z[18].freq = 589$
$z[19].left = z[17]$	$z[19].right = z[18]$	$z[19].freq = 1000$



Huffman Codes

Example 8 (The tree from Huffman's algorithm)



$$B(T) = z[1].freq + \cdots + z[19].freq = 4195$$

Huffman's Algorithm

Running Time

If the min-priority queue is implemented with a heap, then line 2 takes time $O(n)$ and each of the $n - 1$ iterations of lines 4-9 take time $O(\log n)$ totaling to a running time of $O(n \log n)$.

Huffman's Algorithm

Running Time

If the min-priority queue is implemented with a heap, then line 2 takes time $O(n)$ and each of the $n-1$ iterations of lines 4-9 take time $O(\log n)$ totaling to a running time of $O(n \log n)$.

We will now prove the correctness of HUFFMAN using three lemmas.

Lemma 9

Let T be any tree and x and y be two different objects, such that

$$x.freq \leq y.freq \qquad \text{and} \qquad d_T(x) \leq d_T(y).$$

Let T' be the tree obtained from T by exchanging leaves x and y. Then $B(T') \leq B(T)$.

Proof.

(chalk board) □

Huffman's Algorithm

Running Time

If the min-priority queue is implemented with a heap, then line 2 takes time $O(n)$ and each of the $n-1$ iterations of lines 4-9 take time $O(\log n)$ totaling to a running time of $O(n \log n)$.

We will now prove the correctness of HUFFMAN using three lemmas.

Lemma 9

Let T be any tree and x and y be two different objects, such that

$$x.freq \leq y.freq \qquad \textit{and} \qquad d_T(x) \leq d_T(y).$$

Let T' be the tree obtained from T by exchanging leaves x and y. Then $B(T') \leq B(T)$.

Proof.

(chalk board) □

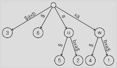
Lemma 10

Let x and y be two objects with lowest frequencies. Then there exists an optimal prefix code in which the codewords for x and y have the same length and differ only in the last bit.

Proof.

(chalk board) □

Marc Hellmuth



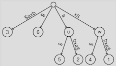
Huffman Codes

Lemma 11

Let C be a set of objects and x and y be two characters with lowest frequencies.
Let $C' = C \setminus \{x, y\} \cup \{z\}$ for a new object z with $z.freq = x.freq + y.freq$.
Let T' be an optimal tree for C'.
Then the tree T obtained from T' by replacing the leaf node for z with an internal node having x and y as children, is optimal for C.

Proof.

(chalk board) □



Huffman Codes

Lemma 11

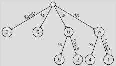
Let C be a set of objects and x and y be two characters with lowest frequencies.
Let $C' = C \setminus \{x, y\} \cup \{z\}$ for a new object z with $z.freq = x.freq + y.freq$.
Let T' be an optimal tree for C'.
Then the tree T obtained from T' by replacing the leaf node for z with an internal node having x and y as children, is optimal for C.

Proof.

(chalk board) □

Theorem 12

Procedure HUFFMAN *produces an optimal prefix code.*



Huffman Codes

Lemma 11

Let C be a set of objects and x and y be two characters with lowest frequencies.
Let $C' = C \setminus \{x, y\} \cup \{z\}$ for a new object z with $z.freq = x.freq + y.freq$.
Let T' be an optimal tree for C'.
Then the tree T obtained from T' by replacing the leaf node for z with an internal node having x and y as children, is optimal for C.

Proof.

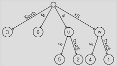
(chalk board) □

Theorem 12

Procedure HUFFMAN *produces an optimal prefix code.*

Proof.

Induction on iteration *i* of HUFFMAN using lemma 11. □

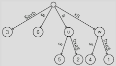


A matroid is a tuple $(R, \mathbb{F})$ such that

- **M1** $\mathbb{F} \neq \emptyset$ is a collection of subsets of the set R, i.e., $\mathbb{F} \subseteq \mathbb{P}(R)$. *(Elements in $\mathbb{F}$ are called independent)*
- **M2** *Closed w.r.t. Inclusion:* $Y \in \mathbb{F}, X \subseteq Y \Rightarrow X \in \mathbb{F}$
- **M3** *Exchange Property:* For all $X, Y \in \mathbb{F}$ and $|Y| > |X| \Rightarrow$ exists $y \in Y \setminus X$ such that $X \cup \{y\} \in \mathbb{F}$.

If $(R, \mathbb{F})$ satisfies (M1) and (M2) but not necessarily (M3), then $(R, \mathbb{F})$ is called independent system.

Many optimization problems can be formulated as independent system, where R is ground set of elements that can be chosen (eg. edges in the MST-problem) and $\mathbb{F}$ is a set of subsets of feasible solutions (eg. all spanning forests in a graph).



A matroid is a tuple $(R, \mathbb{F})$ such that

M1 $\mathbb{F} \neq \emptyset$ is a collection of subsets of the set R, i.e., $\mathbb{F} \subseteq \mathbb{P}(R)$. *(Elements in $\mathbb{F}$ are called independent)*

M2 *Closed w.r.t. Inclusion:* $Y \in \mathbb{F}$, $X \subseteq Y \Rightarrow X \in \mathbb{F}$

M3 *Exchange Property:* For all $X, Y \in \mathbb{F}$ and $|Y| > |X| \Rightarrow$ exists $y \in Y \setminus X$ such that $X \cup \{y\} \in \mathbb{F}$.

Lemma 13

If $(R, \mathbb{F})$ is an independent system, then the following conditions are equivalent:

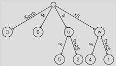
M3 *For all $X, Y \in \mathbb{F}$ and $|Y| > |X| \Rightarrow$ exists $y \in Y \setminus X$ such that $X \cup \{y\} \in \mathbb{F}$.*

M3' *For all $X, Y \in \mathbb{F}$ and $|Y| = |X| + 1 \Rightarrow$ exists $y \in Y \setminus X$ such that $X \cup \{y\} \in \mathbb{F}$.*

M3" *All maximal independent subsets of E have the same cardinality.*

Proof.

chalkboard. □



Bases of an independent system $(R, \mathbb{F})$ are all maximal elements of $\mathbb{F}$.

Lemma 14

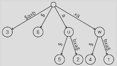
The basis elements of a matroid have always the same size.

Proof.

Let X, Y be bases of $\mathbb{F}$ such that $|Y| > |X|$

$\overset{(M3)}{\Rightarrow} \exists y \in Y \setminus X$ such that $X \cup \{y\} \in \mathbb{F}$

$\Rightarrow X$ is not maximal and thus no basis; a contradiction □



MAX-GREEDY($(R, \mathbb{F}), w : R \to \mathbb{R}^+$)

sort elements in R such that $w(e_1) \geq w(e_2) \geq \cdots \geq w(e_m)$
$F \leftarrow \emptyset$
for $i = 1..m$ **do**
  if $F \cup \{e_i\} \in \mathbb{F}$ **then**
    $F \leftarrow F \cup \{e_i\}$
return F

Runtime: If $f(m)$ denotes the runtime to check if $F \cup \{e_i\} \in \mathbb{F}$, we have total-runtime $O(m\log(m) + mf(m))$.

Theorem 15

Let $(R, \mathbb{F})$ be an independent system. Then, $(R, \mathbb{F})$ is a matroid if and only if MAX-GREEDY *returns a maximum-weighted element in $\mathbb{F}$ for all weighting functions $w : R \to \mathbb{R}^+$.*

Proof.

chalkboard. □