Chapter 4 : Trees

Omar Saadi
omar.saadi@um6p.ma

UM6P - College of Computing

1st year of the engineering degree of UM6P-CS
Academic year 2024-2025

Outline

1. Basic properties
2. Distance in trees and graphs
3. Binary trees
4. Huffman coding

More details on this chapter can be found in Sect. 2.1 and (last part of) Sect 2.3 of D. B. West's book.

Outline

1. Basic properties
2. Distance in trees and graphs
3. Binary trees
4. Huffman coding

Definition

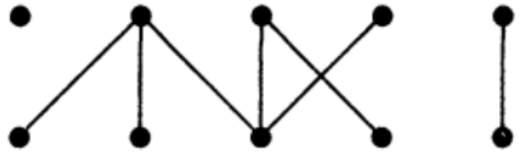
We say that a graph is **acyclic** if it doesn't contain any cycle.
A **forest** is an acyclic graph.
A **tree** is a connected acyclic graph.
A **leaf** (or **pendant vertex**) is a vertex of degree 1.

Definition

A **spanning subgraph** of G is a subgraph with vertex set $V(G)$.
A **spanning tree** is a spanning subgraph that is a tree.

Examples

- The following graph is a forest (no cycles).



- The following two graphs are trees (connected and no cycles). Their union is a forest.

- A tree is a connected forest, and every component of a forest is a tree.
- A graph with no cycles has no odd cycles ; hence trees and forests are bipartite.

Examples (Cont.)

- Paths are trees. A tree is a path if and only if its maximum degree is 2.
- A graph that is a tree has exactly one spanning tree ; the full graph itself.

Remark

A spanning subgraph of G doesn't need to be connected, and a connected subgraph of G doesn't need to be a spanning subgraph.
For example :

- If $n(G) \geq 2$, then the subgraph with vertex set $V(G)$ and edge set $\emptyset$ is spanning but not connected.
- If $n(G) \geq 3$, then a subgraph consisting of one edge and its endpoints is connected but not necessarily spanning.

Lemma

Every tree with at least two vertices has at least two leaves.
Deleting a leaf from an n-vertex tree produces a tree with $n-1$ vertices.

Proof : Consider the endpoints of a maximal path and use the connectivity and the acyclicity properties of the tree.

Remark

This lemma implies that every tree with more than one vertex arises from a smaller tree by adding a vertex of degree 1.
This remark is useful for induction reasoning when dealing with trees : growing an $n+1$-vertex tree from an arbitrary n-vertex tree by adding a new neighbor at an arbitrary old vertex generates all trees with $n+1$ vertices.

Characterizations of trees

Theorem

For an n-vertex graph G (with $n \geq 1$), the following assertions are equivalent (and characterize the trees with n vertices) :

1. *G is connected and has no cycles.*
2. *G is connected and has n - 1 edges.*
3. *G has $n-1$ edges and no cycles.*
4. *For all $u, v \in V(G)$, G has exactly one u, v-path.*

Proof : First prove the equivalence of $1, 2$, and 3 by proving that any two properties among the set {connected, acyclic, $n-1$ edges} imply the third one. Then prove that 1 and 4 are equivalent.

Corollary

1. Every edge of a tree is a cut-edge.
2. Adding one edge to a tree forms exactly one cycle.
3. Every connected graph contains a spanning tree.

Proof :

Proposition

If T, T' are spanning trees of a connected graph G and $e \in E(T) - E(T')$, then there is an edge $e' \in E(T') - E(T)$ such that $T - e + e'$ is a spanning tree of G. Their is also an edge $e'' \in E(T') - E(T)$ such that $T' + e - e''$ is a spanning tree of G.

Proof : Consider the two components of $T - e$ for the first result, and consider the unique cycle obtained by adding e to T' for the second result.

Outline

1 Basic properties

2 Distance in trees and graphs

3 Binary trees

4 Huffman coding

Definition

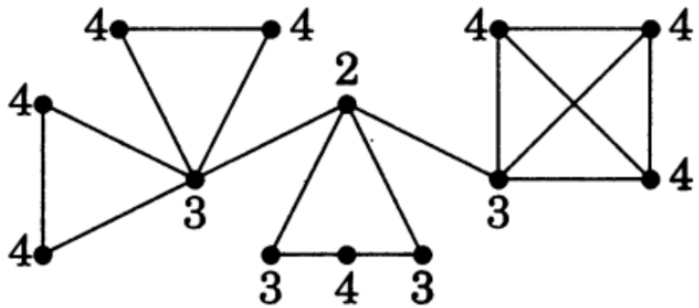
If G has a u, v-path, then the **distance** from u to v, written $d_G(u, v)$ or simply $d(u, v)$, is the least length of a u, v-path. If G has no such path, then $d(u, v) = \infty$.
The **diameter** $(\text{diam}(G))$ is $\max_{u,v \in V(G)} d(u, v)$.
The **eccentricity** of a vertex u, written $\epsilon(u)$, is $\max_{v \in V(G)} d(u, v)$.
The **radius** of a graph G, written $\text{rad}(G)$, is $\min_{u \in V(G)} \epsilon(u)$.

Remarks

- The diameter equals the maximum of the vertex eccentricities.
- In a disconnected graph, the diameter and radius (and every eccentricity) are infinite, because distance between vertices in different components is infinite.
- We use the word "diameter" due to its use in geometry, where it is the greatest distance between two elements of a set.

Examples

- The Petersen graph has diameter 2, since nonadjacent vertices have a common neighbor.
- The cycle C_n has diameter $\lfloor n/2 \rfloor$.
- In these two examples, every vertex has the same eccentricity, and $\text{diam}(G) = \text{rad}(G)$.
- Every path in a tree is the shortest (the only) path between its endpoints, so the diameter of a tree is the length of its longest path.
- In the graph below, each vertex is labeled with its eccentricity. The radius is 2, the diameter is 4, and the length of the longest path is 7.



Proposition

If G is a simple graph, then $\text{diam}(G) \geq 3 \Rightarrow \text{diam}(\overline{G}) \leq 3$.

Proof : Use the fact that if $\text{diam}(G) \geq 3$ then there exist nonadjacent vertices $u, v \in V(G)$ with no common neighbor.

Definition

The **center** of a graph G is the subgraph induced by the vertices of minimum eccentricity.

Remark : The center of a graph is the full graph if and only if the radius and diameter are equal.

Theorem (Jordan [1869])

The center of a tree is a vertex or an edge.

Proof : We use induction on the number of vertices in a tree T, where we obtain a smaller tree T' by deleting every leaf of T. Use the fact that $\forall u \in V(T'),\ \epsilon_{T'}(u) = \epsilon_T(u) - 1$.

Outline

1 Basic properties

2 Distance in trees and graphs

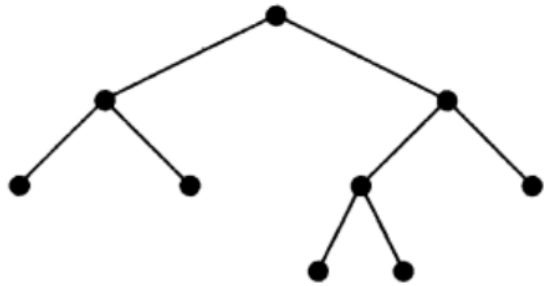
3 Binary trees

4 Huffman coding

Definition

A **rooted tree** is a tree with one vertex r chosen as **root**. For each vertex v, let $P(v)$ be the unique v, r-path.
The **parent** of v is its neighbor on $P(v)$; its children are its other neighbors.
Its **ancestors** are the vertices of $P(v) - v$. Its **descendants** are the vertices u such that $P(u)$ contains v.
The **leaves** are the vertices with no children.
A **rooted plane tree** or **planted tree** is a rooted tree with a left-to-right ordering specified for the children of each vertex.

Example :



Definition

A **binary tree** is a rooted plane tree where each vertex has at most two children, and each child of a vertex is designated as its left child or right child.
The subtrees rooted at the children of the root are the **left subtree** and the **right subtree** of the tree.
A k-**ary tree** allows each vertex up to k children.

Remark

- Binary trees permit storage of data for quick access.
- We store each item at a leaf and access it by following the path from the root.
- We encode the path by recording 0 when we move to a left child and 1 when we move to a right child.
- For each leaf, the search time is the length of its code word.

Outline

1 Basic properties

2 Distance in trees and graphs

3 Binary trees

4 Huffman coding

In a text file, each character is stored as a binary list.
The classical way to encode characters is to use the ASCII code. It encodes each character using 7 bits. This allows to encode 128 characters that appear in text files. The ASCII table is the following :

Control Characters				Graphic Symbols											
Name	Dec	Binary	Hex	Symbol	Dec	Binary	Hex	Symbol	Dec	Binary	Hex	Symbol	Dec	Binary	Hex
NUL	0	0000000	00	space	32	0100000	20	@	64	1000000	40	'	96	1100000	60
SOH	1	0000001	01	!	33	0100001	21	A	65	1000001	41	a	97	1100001	61
STX	2	0000010	02	"	34	0100010	22	B	66	1000010	42	b	98	1100010	62
ETX	3	0000011	03	#	35	0100011	23	C	67	1000011	43	c	99	1100011	63
EOT	4	0000100	04	$	36	0100100	24	D	68	1000100	44	d	100	1100100	64
ENQ	5	0000101	05	%	37	0100101	25	E	69	1000101	45	e	101	1100101	65
ACK	6	0000110	06	&	38	0100110	26	F	70	1000110	46	f	102	1100110	66
BEL	7	0000111	07	'	39	0100111	27	G	71	1000111	47	g	103	1100111	67
BS	8	0001000	08	(	40	0101000	28	H	72	1001000	48	h	104	1101000	68
HT	9	0001001	09	)	41	0101001	29	I	73	1001001	49	i	105	1101001	69
LF	10	0001010	0A	*	42	0101010	2A	J	74	1001010	4A	j	106	1101010	6A
VT	11	0001011	0B	+	43	0101011	2B	K	75	1001011	4B	k	107	1101011	6B
FF	12	0001100	0C	,	44	0101100	2C	L	76	1001100	4C	l	108	1101100	6C
CR	13	0001101	0D	−	45	0101101	2D	M	77	1001101	4D	m	109	1101101	6D
SO	14	0001110	0E	.	46	0101110	2E	N	78	1001110	4E	n	110	1101110	6E
SI	15	0001111	0F	/	47	0101111	2F	O	79	1001111	4F	o	111	1101111	6F
DLE	16	0010000	10	0	48	0110000	30	P	80	1010000	50	p	112	1110000	70
DC1	17	0010001	11	1	49	0110001	31	Q	81	1010001	51	q	113	1110001	71
DC2	18	0010010	12	2	50	0110010	32	R	82	1010010	52	r	114	1110010	72
DC3	19	0010011	13	3	51	0110011	33	S	83	1010011	53	s	115	1110011	73
DC4	20	0010100	14	4	52	0110100	34	T	84	1010100	54	t	116	1110100	74
NAK	21	0010101	15	5	53	0110101	35	U	85	1010101	55	u	117	1110101	75
SYN	22	0010110	16	6	54	0110110	36	V	86	1010110	56	v	118	1110110	76
ETB	23	0010111	17	7	55	0110111	37	W	87	1010111	57	w	119	1110111	77
CAN	24	0011000	18	8	56	0111000	38	X	88	1011000	58	x	120	1111000	78
EM	25	0011001	19	9	57	0111001	39	Y	89	1011001	59	y	121	1111001	79
SUB	26	0011010	1A	:	58	0111010	3A	Z	90	1011010	5A	z	122	1111010	7A
ESC	27	0011011	1B	;	59	0111011	3B	[	91	1011011	5B	{	123	1111011	7B
FS	28	0011100	1C	<	60	0111100	3C	\	92	1011100	5C	|	124	1111100	7C
GS	29	0011101	1D	=	61	0111101	3D	]	93	1011101	5D	}	125	1111101	7D
RS	30	0011110	1E	>	62	0111110	3E	^	94	1011110	5E	~	126	1111110	7E
US	31	0011111	1F	?	63	0111111	3F	_	95	1011111	5F	Del	127	1111111	7F

Prefix-free coding

Objective : Given a text file, we want to encode each character appearing in that text using a binary list with the goal of minimizing the total number of bits needed to store this text file.

The length of code words **may vary** ; so we need a way to recognize the end of the current word.

Idea : If no code word is an initial portion of another, we can easily recover our characters. We can do this by going through the list of bits (from left to right), and at each time that a sublist of bits correspond to a code of a character we make the transformation "bits to character", and start again for the remaining bits.

Definition : We call such coding a **prefix-free coding**.

Example : If we have only three characters a, b, c with the following prefix-free coding $a \leftrightarrow 1$, $b \leftrightarrow 01$, $c \leftrightarrow 00$, then we can get the following encodings : $aabacbc \leftrightarrow 11011000100$ and $ccbaac \leftrightarrow 0000011100$.

Remark

Under the prefix-free condition, the binary code words correspond to the leaves of a binary tree using the left/right encoding described above.

Input : The text file uses n characters $c_1, \cdots, c_n$ with probabilities of occurrence $p_1, \cdots, p_n$ respectively.
Notice that p_i is the number of occurrences of c_i divided by the total number N of characters in the text.
Goal : Construct the binary tree such that the bit lengths $l_1, \cdots, l_n$ assigned to the leaves $c_1, \cdots, c_n$ are such that :

$$\text{The expected length of a code word } \sum_{i \in [n]} p_i l_i \text{ is minimal.}$$

Notice that we are minimizing indeed the total length of the encoding $N \times \sum_{i \in [n]} p_i l_i$.

Huffman's Algorithm

Input : Weights (frequencies or probabilities) $p_1, \cdots, p_n$.

Output : Prefix-free code (equivalently, a binary tree).

Idea : Infrequent items should have longer codes ; put infrequent items deeper by combining them into parent vertices.

Initial case : When $n = 2$, the optimal length is one, with 0 and 1 being the codes assigned to the two items (the tree has a root and two leaves).

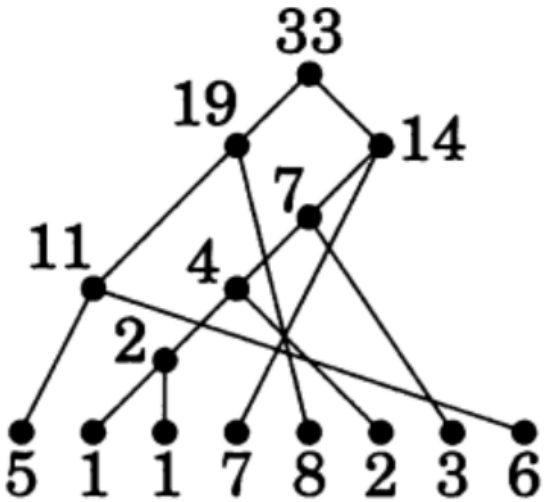
Recursion : When $n > 2$, replace the two items with the smallest probabilities p, p' with a single item q of weight $p + p'$. Treat the smaller set as a problem with $n - 1$ items.
After solving it, give children with weights p, p' to the resulting leaf with weight q. Equivalently, replace the code computed for the combined item q with its extensions by 1 and 0, assigned to the items that were replaced.

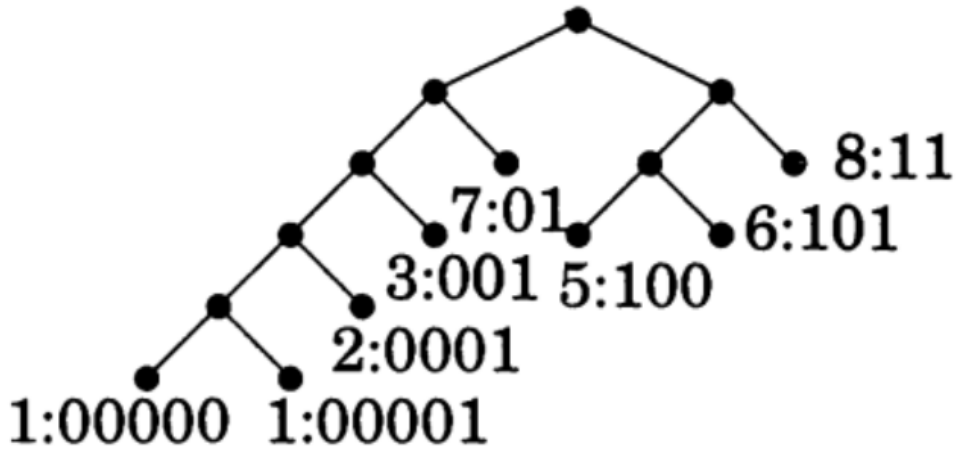
Notice that each vertex that is not a leaf has exactly two children.

Example

Consider eight items with frequencies $5, 1, 1, 7, 8, 2, 3, 6$. Huffman's algorithm will output the tree below in the left, working from the bottom up. The values of the vertices are the (combined) frequencies.

We obtain code words as shown in the tree below in the right.





In their original order, the items have code words 100, 00000, 00001, 01, 11, 0001, 001, and 101. The expected length of a code word is $L = \sum_i p_i l_i = 90/33$.

Optimality of Huffman coding

Theorem

Given a probability distribution $\{p_1, \cdots, p_n\}$ on n items, Huffman's algorithm produces the prefix-free code with minimum expected length.

Proof

We use induction on n. Basis step : $n = 2$. The algorithm encodes each item as a single bit, which is optimal.
Induction step : $n > 2$. Suppose that the algorithm computes the optimal code when given a distribution for $n - 1$ items.
With n items, an optimal binary tree T must assign the items with probabilities $p_1 \geq \cdots \geq p_n$ to leaves in increasing order of depth, because otherwise we can exchange two items and get a strictly better encoding. Therefore the two items with the smallest probabilities are assigned to leaves of greatest depth.
Since permuting items at a given depth doesn't change the expected length, we may assume that **items n and $n-1$ appear as siblings at greatest depth** in T.

Proof (Cont.)

Let T' be the tree obtained from T by deleting these leaves, and let $\{q_1, \cdots, q_{n-1}\}$ be the probability distribution obtained by replacing $\{p_{n-1}, p_n\}$ by $q_{n-1} = p_{n-1} + p_n$, and $\forall i \in [n-2], q_i = p_i$. The tree T' yields a code for $\{q_i\}$. Notice that, if k is the depth of the leaf assigned of weight q_{n-1} in T', we have

$$\begin{aligned} \text{length}(T) = \sum_{i \in [n]} p_i l_i = \sum_{i \in [n-2]} p_i l_i + (k+1)(p_{n-1} + p_n) = & \sum_{i \in [n-1]} q_i l_i + q_{n-1} \\ = & \text{length}(T') + q_{n-1} \ . \end{aligned}$$

So it suffices that T' is optimal to get that T is optimal because they have the same length up to the fixed constant q_{n-1}.
By the induction hypothesis, an optimal choice for T' is obtained by applying Huffman's algorithm to $\{q_i\}$. Since the replacement of $\{p_{n-1}, p_n\}$ by q_{n-1} is exactly the first step of Huffman's Algorithm for $\{p_i\}$, we conclude that Huffman's algorithm generates the optimal tree T for $\{p_i\}$.