

Part I: Computer Science Essentials

Winfried Kurth

Organisational issues:

Course web page

http://www.uni-forst.gwdg.de/~wkurth/csm18_home.htm

(here you find also the slides and literature references)

Lectures and exercises on computer science are given in two blocks within a week:

Wednesday, April 25, 10:15 a.m. - 6 p.m.
(with a lunch break ca. 12 a.m. - 1 p.m.)

and Friday, April 27, 9:15 a.m. - 6 p.m.
(likewise with a lunch break).

The tutorial (not mandatory) starts on Thursday, April 26, 4:15 - 6 p.m., FSR 4.1.

Continuation of the course (linear algebra, calculus): on weekly basis, by A. Tavkhelizde and I. Kuzyakova

Written exam: on **Tuesday, 17 July, 2018**, 10:00-11:30 a.m.

Second exam date: *still to be fixed*

1. Introduction to computer science

Fundamental notions, systematical overview

What is "Computer Science" / "Informatics" ?

"Computer Science" – science about a tool?

better names would be: "*science of computing*"
or "*data processing science*"
(focuses on activity instead of tool)

"Informatics": continental-European for "computer science"

- French: "Informatique" (since 1960s)
- German: "Informatik"

Definition: "Science of the systematical processing of information, especially the automatic processing by use of digital computers".

Latin "informare":

to give structure to something; to educate; to picture

Information:

- independent fundamental entity of the world besides matter and energy
- depends on previous knowledge of the receiver of the information
- various approaches to quantify it
- we can consider information simply as "interpreted data".

Data: represented information
(e.g. text in a book, magnetic patterns on a
harddisk, ...)

But:

Hermeneutics – "the art of interpretation" – is *not*
part of informatics, despite its name. Social and
cultural aspects of information are largely ignored.

"Computer": comes from "to compute"
= "to calculate".

"*Algorithm*":

The word comes from the Persian textbook writer
Abu Ja'far Mohammed ibn Mûsâ al-Khowârizmî
(= "father of Ja'far Mohammed, son of Moses,
coming from Khowârizm" – a town in Usbekistan,
today called *Khiva*.)

Al-Khowârizmî lived in Bhagdad, in the "House of
Wisdom"

wrote book about calculation:

"*Kitab al jabr w'al-muqabala*" (= "rules of
reconstitution and reduction")

– here the word "algebra" comes from!

Modern meaning of "algorithm":

Finite set of rules which specify a sequence of operations in order to solve a certain problem, with the following properties:

1. **Termination:** An algorithm must come to an end after a finite number of steps.

2. **Definiteness:** Each step must be defined precisely.

3. **Input:** An algorithm *can* need input values (e.g. numbers).

4. **Output:** An algorithm *must* give one or more output values.

5. **Feasibility:** An algorithm must be feasible; e.g., no intermediate step must depend on the solution of some still unsolved mathematical problem.

(after Knuth 1973)

"*Programme*" (in American English: "program"):

Version of an algorithm which can be read, interpreted and carried out by a computer.

Programming languages were designed to write precise programmes (more precise than possible in our natural language!) suitable for computers.

Some notes concerning the history of *programming*:

Early phases of computer history: *Hardware* (= the machines) was in focus (reason for the name "computer science")

Later: *Software* (= programmes) increasingly important, increasingly expensive in comparison to hardware.

First "programmer": Was a woman (**Lady Ada Lovelace**, daughter of the poet Lord Byron): Developed programs for Babbage's (non-functional) "analytical engine"

An early concept for a programming notation was the "Plankalkül" (Zuse 1944), but it was not used in practice.

Programming these machines: Started with today so-called "**machine languages**" and "**assembler languages**" (both machine-specific).

Later: so-called "high-level languages"

- more abstraction
- better readability for humans
- trying to integrate traditional mathematical notations
- *platform-independent* (not specific to certain machine)

FORTRAN (1954), COBOL (1958), LISP (1960), Pascal (1971), C (1971), C++ (extension of C, 1992), Java (1995), XL (2008) ...

(later more about programming)

Subject areas of computer science

1960s/1970s: Development of specialized university curricula

Basis: Mathematics, electrical engineering; no interest in social or cultural conditions and consequences, or more specifically: in consequences for life at working place and leisure

Classical branches (from first recommendations for curricula in the 1960s): (a) **theoretical** informatics, (b) **technical** informatics, (c) **practical** informatics, (d) **applied** informatics

Theoretical informatics: mathematical basis: not general “theory” (which would include disciplines from the humanities and social sciences relevant to informatics), but specialized “mathematical base”. Example questions:

Which problems **can in principle be solved** by a machine?

How can **syntax and semantics** of programming languages be described?

Which kinds of **logic** can be used for automatic problem solving?

How do we measure **how complicated problems are**, for example with respect to time or memory requirements?

Which kinds of problems can be solved with which abstract models of computation?

How can be the **correctness** of a program be **proved** with mathematical exactness?

Technical informatics: focused on **hardware**. Example questions:

How can computational objects and operations be represented with **physical means**?

Which are the **basic parts** from which a computer should be built?

Which are the appropriate **architectural decisions** for a computer?

How can a processor be organized in order to execute a special kind of program especially quickly?

How is information **stored** for quick access with small cost?

Which are the technical conditions for building **networks** from separate computers?

How do we build computers which **survive some defects**?

Practical informatics: **non application specific programming**. Example questions:

Which are the standard problems occurring **in many application areas**, and how can they be solved?

Which **data structures** allow efficient solving of problems, and which algorithms are best used on these data structures?

What types of **programming languages** are best suited to different types of problems?

How must **service programs** be organized which provide the user with an easier to use view of the machine than the bare hardware would do?

How are high-level programs **translated** into a form which can be executed by the underlying hardware?

How does one design **user interfaces** for end users?

How does one organize the **development process** of large software systems? ("Software engineering")

Applied informatics: programming for **specific application fields**.
Example questions:

How are **graphical objects** represented in the computer, and how can they be visualized?

Which **numerical methods** exist to model states and processes happening in natural environments?

How should **data base systems** be structured to support the work processes in a company?

Which techniques exist to simulate the working of the **human mind** with computers?

What consequences has the use of computers for the **quality of life**, both in general and at the working place in particular?

Informatics in the social context:

What **ethical questions** arise from the use of computers, and how can they be answered?

(data security, privacy questions, computer viruses, hackers, violence-promoting games, software piracy, ownership of software and ideas, the open-source idea, use of information technology for warfare, for crime, for sexual exploitation, for terrorism...)

How does the use of computers influence our **way of thinking** (about the world, about humans, about the mind, about personal relationships of people...)?

How can computers, the Web and the "Web 2.0" (Facebook, Twitter, Wikipedia etc.) be used to improve education / autonomy of people / human rights / political participation... ?

2. Representation and measurement of information

In digital computers and media, all data are represented by combinations of only 2 elementary states: 0 and 1

(can be "charged" / "not charged", "on" / "off", "magnetized" / "not magnetized", "open" / "closed", "high current" / "low current", "plus" / "minus" etc.)

The smallest amount of information is thus the *bit* (binary digit). It expresses which of two alternatives is the case. The alternatives are often written 0 and 1, or (sometimes) 0 and L.

n bits: represent one out of 2^n alternatives.

Codes

To represent information in a computer, we must *encode* all with the two symbols 0 and 1 !

What is a *code* ?

Code (1): A mapping $f: A \rightarrow B$ from a set A of elements to be stored or transferred to a set B used for storage or transfer.

Code (2): The set B from definition (1).

Example:

A	B	C	J	K	L	S	T	U
D	E	F	M	N	O	V	W	X
G	H	I	P	Q	R	Y	Z	

$$A = \{A, B, C, \dots, Z\}$$

$$B = \{ _ , \sqcup , \sqcap , \dots , \sqsupset \}$$

MESSAGE \xrightarrow{f} $_ . \sqcup _ . _ . _ \sqsupset \sqcup$

digital (discrete) and **analogue** (continuous) codes

Analogue computers (representation of quantities with continuously changing quantities): have vanished

Example: Vinyl records (analogue) vs. compact disks (discrete)

Benefit of discrete data representations: avoiding **noise**

For digital computers, we need *binary* codes:
 B is a set of combinations of 0 and 1.

Examples:

For the primary **compass direction**: two bits necessary, and some convention which bit-pair represents which direction. Example code:

$$\{N, E, S, W\} \rightarrow \{0, 1\}^2, N \mapsto 00, E \mapsto 01, S \mapsto 10, W \mapsto 11$$

For **Boolean** values 'True' and 'False':

$$\{T, F\} \rightarrow \{0, 1\}, T \mapsto 1, F \mapsto 0$$

For **numbers** 0 to 9: Binary Coded Decimal (BCD, non-total code, i.e. some combinations are unused)

$$\begin{aligned} & \{0, 1, \dots, 9\} \rightarrow \{0, 1\}^4 \\ 0 & \mapsto 0000, 1 \mapsto 0001, 2 \mapsto 0010, 3 \mapsto 0011, \\ 4 & \mapsto 0100, 5 \mapsto 0101, 6 \mapsto 0110, 7 \mapsto 0111, \\ & 8 \mapsto 1000, 9 \mapsto 1001 \end{aligned}$$

Multiples of bits

Bits seldom occur as singles. Certain multiples of bits are used as *units for information (storage) capacity*.

1 Byte: 8 bits (can represent 1 of 2^8
= 256 alternatives).

Example: one of the integer numbers between
-128 and +127.

1 Halfbyte: 4 bits.

Typically, memory stores are built for *multiples of bytes*.

Prefixes: kilo, mega, giga, tera, peta, exa

- used in physics for the factors 10^3 , 10^6 , 10^9 , 10^{12} , 10^{15} , 10^{18}
- in computer science often used for the factors 2^{10} , 2^{20} , 2^{30} , 2^{40} , 2^{50} , 2^{60} , which are slightly larger

abbrev- viation	meaning	factor
KB	Kilobytes	$2^{10} = 1024$
MB	Megabytes	$2^{20} = 1,048,576$
GB	Gigabytes	$2^{30} = 1,073,741,824$
TB	Terabytes	$2^{40} = 1,099,511,627,776$
PB	Petabytes	$2^{50} = 1,125,899,906,842,624$
EB	Exabytes	$2^{60} = 1,152,921,504,606,846,976$

Representation of numbers in the computer

For positive integers, basically the *binary number system* is used.

Number systems

Different notations for integers.

Here we consider only *positional* number systems (the position of a digit determines its value).

Decimal number system: base 10; each digit represents a multiple of an exponent of 10. Digits 0..9.

$$\text{Example: } 123.456_{10} = 1 * 10^2 + 2 * 10^1 + 3 * 10^0 + 4 * 10^{-1} + 5 * 10^{-2} + 6 * 10^{-3}.$$

Binary number system: base 2. Only two digits: 0 and 1.

$$\text{Example: } 1101.01_2 = 1 * 2^3 + 1 * 2^2 + 0 * 2^1 + 1 * 2^0 + 0 * 2^{-1} + 1 * 2^{-2} = 13.25_{10}.$$

Hexadecimal system (better but unhistorical name: sedecimal number system): Base 16, digits 0..9,A..F. One digit for four bits. Examples: $A2.8_{16} = 162.5_{10}$, $FF_{16} = 255_{10}$.

The additional digits in the hexadecimal system:
 $A = 10, B = 11, C = 12, D = 13, E = 14, F = 15$.

Transformation from one number system to the other:

- Special case (easy): from binary to hexadecimal
Every 4 binary digits correspond directly to a hexadecimal digit

$$\text{Example: } \begin{array}{cccc} \underline{0000} & \underline{0010} & \underline{1100} & \underline{0110} \\ \rightarrow & 0 & 2 & C & 6 \end{array}$$

- from arbitrary system to decimal:
Horner scheme

Input: $z_{n-1} z_{n-2} \dots z_0$ to base b

start with $h_{n-1} = z_{n-1}$

calculate for $k = n-1, n-2, \dots, 1$:

$$h_{k-1} = h_k * b + z_{k-1}$$

Output: $z = h_0$

Example:

Input: binary number 1010 ($n = 4, b = 2$)

Start: $h_{n-1} = h_3 = z_3 = 1$

$k = n-1 = 3$: $h_2 = h_3 * 2 + z_2 = 1*2 + 0 = 2$

$k = 2$: $h_1 = h_2 * 2 + z_1 = 2*2 + 1 = 5$

$k = 1$: $h_0 = h_1 * 2 + z_0 = 2*5 + 0 = \mathbf{10} = z$

- from decimal to arbitrary:
Inverse Horner scheme

start with $h_0 = z$ (= input)

calculate for $k = 1, 2, 3, \dots$:

$$z_{k-1} = h_{k-1} \bmod b,$$

$$h_k = h_{k-1} \operatorname{div} b$$

(mod: rest when dividing by b , div: integral part from dividing by b)

Output: $z_{n-1} z_{n-2} \dots z_0$ to base b

Example:

Input: decimal number 34, transform in ternary system ($b = 3$)

Start: $h_0 = 34$

$$k = 1: \quad z_0 = h_0 \bmod 3 = 34 \bmod 3 = 1, \\ \quad \quad \quad h_1 = h_0 \operatorname{div} 3 = 34 \operatorname{div} 3 = 11$$

$$k = 2: \quad z_1 = h_1 \bmod 3 = 11 \bmod 3 = 2, \\ \quad \quad \quad h_2 = h_1 \operatorname{div} 3 = 11 \operatorname{div} 3 = 3$$

$$k = 3: \quad z_2 = h_2 \bmod 3 = 3 \bmod 3 = 0, \\ \quad \quad \quad h_3 = h_2 \operatorname{div} 3 = 3 \operatorname{div} 3 = 1,$$

$$k = 4: \quad z_3 = h_3 \bmod 3 = 1 \bmod 3 = 1, \\ \quad \quad \quad h_4 = h_3 \operatorname{div} 3 = 1 \operatorname{div} 3 = 0 \text{ (Stop)}$$

$\Rightarrow z = 1021$

Remark:

Arbitrary *real numbers* (i.e., non-integers) can also be represented using an arbitrary integer $b > 1$ as base.

Digits after the dot are interpreted as coefficients of b^{-n} ($n = 1, 2, 3, \dots$).

Example:

$$0.111_2 \text{ (base } b=2) = 1/2 + 1/4 + 1/8 = 7/8 = 0.875_{10}$$

Exercise

- (a) Calculate the decimal value of the binary number 1001111.
- (b) Calculate the hexadecimal representation of the decimal number 999.
- (c) What is the binary expansion of the value $1/3$?
(Hint: You can do "written division" analogously to the decimal case, but with doubling the remainder in every step instead of multiplying by 10.)

Special issues about representing numbers in the computer

Numbers are usually stored in sections of memory of *fixed size* (for reasons of organization of memory access in the computer).

Integer representation in finite cells ("words" with fixed length):

Computer memory: organized in **finite cells**. Typically: Multiples of a byte.

How to store numbers in a 4-byte cell? Some encoding necessary. 2^{32} different values can be represented.

Example: $0 \dots 2^{32} - 1$ can be represented as binary numbers.

Example including negative numbers: $-2^{31} \dots 2^{31} - 1$ can be represented as two's complements numbers.

Two's complement: Most used representation for integers from range $-2^{n-1} \dots 2^{n-1} - 1$ (with n -bit cell).

Non-negative numbers: Are represented simply as binary numbers. Using n bits, the highest bit is always 0.

Negative numbers: (a) Represent their absolute value as binary number, (b) then invert all bits (including the infinite number of leading zeros, resulting in an infinite number of leading ones), and (c) add a 1. The last n bits are the two's complement of the value to be represented.

Example for the "Two's complement":

8-bit two's complement representation of -77

1. Represent $+77$ as a binary number: 1001101
2. Invert all bits, including the leading 0s: $\dots 1110110010$
3. Add 1: $\dots 1110110011$
4. Use only the lowest (= rightmost) 8 bits: 10110011

Notice:

For 16-bit cells, the result would be 111111110110011 .

decimal system	8-bit two's complement
-128	1000 0000
-127	1000 0001
-126	1000 0010
...	...
-2	1111 1110
-1	1111 1111
0	0000 0000
1	0000 0001
...	...
126	0111 1110
127	0111 1111

Properties of the two's complement:

Code represents numbers $-2^{n-1} \dots 2^{n-1} - 1$.

High bit represents **sign**.

Minimal value represented by 1000..., maximal by 0111....

-1 represented by 111....

Floating-point representations

Built analogously to the "scientific representation" of numbers in the form $m * 10^e$

- but using the binary system:

Represent numbers in the form

$$s * m * 2^e$$

with sign s (+1 or -1), non-negative mantissa m , and integer exponent e .

Representation is **normalized** if $1 \leq m < 2$.

Finite number of bits for sign, mantissa and exponent; often used: 32 bits (single precision), 64 bits (double precision), 80 bits (extended precision)

Typical layout of 32-bit floating point number:

Bit 31: represents s (1: negative; 0: positive)

Bits 30..23 (8 bits): represent e : Binary representation of $e + 127$, which allows the values $-126 \dots 127$. Value 0 is used in representation of number 0 and of unnormalized numbers. Value 255_{10} used to represent infinity and other exceptional values.

Bits 22..0 (23 bits): represent m , by binary representation of the integer part of $m * 2^{23}$, without the leading 1.

Example: representing $+26.625$ as a 32-bit normalized floating point number: $26.625_{10} = 11010.101_2$. Normalizing yields $1.1010'1010_2 * 2^4$. 32-bit floating point number ($s=0, e=131_{10}$):

0'10000011'101010100000000000000000

Digital representation of text

based on representation of *letters*

- depending on the alphabet: certain number of bits necessary
- for 26 letters: at least 5 bits necessary ($2^4 = 16 < 26$, $2^5 = 32 > 26$)
- but also encoding of digits, special signs, upper- and lower-case letters... desirable

traditional 7-bit code:

ASCII (= *American Standard Code for Information Interchange*)

ISO-646 norm

later extended to 8-bit code

examples: 00110000 = hex 30 = 48_{10} = digit 0

00110001 = hex 31 = 49_{10} = digit 1

...

00111010 = hex 3A = 58_{10} = ':'

...

01000001 = hex 41 = 65_{10} = 'A'

01000010 = hex 42 = 66_{10} = 'B'

...

01100001 = hex 61 = 97_{10} = 'a'

...

ASCII Table:

Non-printable characters						Printable characters					
Dez	Okt	Hex	Char	Code	Remark	Dez	Okt	Hex	Char	Remark	
0	000	0x00	Ctrl-@	NUL	Null prompt	32	040	0x20		blank	
1	001	0x01	Ctrl-A	SOH	Start of heading	33	041	0x21	!	exclamation mark	
2	002	0x02	Ctrl-B	STX	Start of text	34	042	0x22	"	quotation mark	
3	003	0x03	Ctrl-C	ETX	End of Text	35	043	0x23	#		
4	004	0x04	Ctrl-D	EOT	End of transmission	36	044	0x24	\$	Dollar character	
5	005	0x05	Ctrl-E	ENQ	Enquiry	37	045	0x25	%		
6	006	0x06	Ctrl-F	ACK	Acknowledge	38	046	0x26	&		

7	007	0x07	Ctrl-G	BEL	Bell
8	010	0x08	Ctrl-H	BS	Backspace
9	011	0x09	Ctrl-I	HT	Horizontal tab
10	012	0x0A	Ctrl-J	LF	Line feed
11	013	0x0B	Ctrl-K	VT	Vertical tab
12	014	0x0C	Ctrl-L	FF	Form feed
				NP	New page
13	015	0x0D	Ctrl-M	CR	Carriage return
14	016	0x0E	Ctrl-N	SO	Shift out
15	017	0x0F	Ctrl-O	SI	Shift in
16	020	0x10	Ctrl-P	DLE	Data link escape
17	021	0x11	Ctrl-Q	DC1	X-ON
18	022	0x12	Ctrl-R	DC2	
19	023	0x13	Ctrl-S	DC3	X-Off
20	024	0x14	Ctrl-T	DC4	
21	025	0x15	Ctrl-U	NAK	No acknowledge
22	026	0x16	Ctrl-V	SYN	Synchronous idle
23	027	0x17	Ctrl-W	ETB	End transmission blocks
24	030	0x18	Ctrl-X	CAN	Cancel
25	031	0x19	Ctrl-Y	EM	End of medium
26	032	0x1A	Ctrl-Z	SUB	Substitute
27	033	0x1B	Ctrl-[ESC	Escape
28	034	0x1C	Ctrl-\	FS	File separator
29	035	0x1D	Ctrl-]	GS	Group separator
30	036	0x1E	Ctrl-^	RS	Record separator
31	027	0x1F	Ctrl- <u>_</u>	US	Unit separator
127	0177	0x7F		DEL	Delete or rubout

39	047	0x27	'	apostroph
40	050	0x28	(
41	051	0x29)	
42	052	0x2A	*	asterisk
43	053	0x2B	+	plus sign
44	054	0x2C	,	comma
45	055	0x2D	-	minus sign
46	056	0x2E	.	dot
47	057	0x2F	/	slash
48	060	0x30	0	
49	061	0x31	1	
50	062	0x32	2	
51	063	0x33	3	
52	064	0x34	4	
53	065	0x35	5	
54	066	0x36	6	
55	067	0x37	7	
56	070	0x38	8	
57	071	0x39	9	
58	072	0x3A	:	colon
59	073	0x3B	;	semicolon
60	074	0x3C	<	less than
61	075	0x3D	=	euquality character
62	076	0x3E	>	greater than
63	077	0x3F	?	interrogation mark
64	0100	0x40	@	at
65	0101	0x41	A	
66	0102	0x42	B	
67	0103	0x43	C	
68	0104	0x44	D	
69	0105	0x45	E	
70	0106	0x46	F	
71	0107	0x47	G	
72	0110	0x48	H	
73	0111	0x49	I	
74	0112	0x4A	J	
75	0113	0x4B	K	
76	0114	0x4C	L	
77	0115	0x4D	M	
78	0116	0x4E	N	
79	0117	0x4F	O	
80	0120	0x50	P	
81	0121	0x51	Q	
82	0122	0x52	R	
83	0123	0x53	S	
84	0124	0x54	T	
85	0125	0x55	U	
86	0126	0x56	V	
87	0127	0x57	W	
88	0130	0x58	X	
89	0131	0x59	Y	

	90	0132	0x5A	Z	
	91	0133	0x5B	[
	92	0134	0x5C	\	backslash
	93	0135	0x5D]	
	94	0136	0x5E	^	caret
	95	0137	0x5F	_	low line
	96	0140	0x60	`	back quote
	97	0141	0x61	a	
	98	0142	0x62	b	
	99	0143	0x63	c	
	100	0144	0x64	d	
	101	0145	0x65	e	
	102	0146	0x66	f	
	103	0147	0x67	g	
	104	0150	0x68	h	
	105	0151	0x69	i	
	106	0152	0x6A	j	
	107	0153	0x6B	k	
	108	0154	0x6C	l	
	109	0155	0x6D	m	
	110	0156	0x6E	n	
	111	0157	0x6F	o	
	112	0160	0x70	p	
	113	0161	0x71	q	
	114	0162	0x72	r	
	115	0163	0x73	s	
	116	0164	0x74	t	
	117	0165	0x75	u	
	118	0166	0x76	v	
	119	0167	0x77	w	
	120	0170	0x78	x	
	121	0171	0x79	y	
	122	0172	0x7A	z	
	123	0173	0x7B	{	
	124	0174	0x7C		
	125	0175	0x7D	}	
	126	0176	0x7E	~	

ASCII not sufficient for alphabets of the non-Anglo-american world (not even for European alphabets with ä, ö, ü, ß, é, ø, ñ, å...)

Unicode:

2 byte (= 16 bit) code for multilingual text processing
- can represent 65536 characters

amongst them: 27786 Chinese-Japanese-Korean
characters

11172 Hangul characters (Korean)

ancient Nordic runes

Tibetan characters

Cherokee characters ...

complete list see <http://www.unicode.org/charts/>

Unicode "Escape sequence" (to utilise it in the programming language Java):

e.g., `\u0041` = 'A' (0041 = hexadecimal representation)

Some characters occur more frequently in texts than others:

better use *variable-length* code

UTF-8: Universal Transformation Format

Characters encoded with variable number of bytes

⇒ for texts with many ASCII characters (like on many web pages) shorter as Unicode

Strings (or *words*): sequences of characters

encoded by sequences of the corresponding code words

Digital representation of pictures

Gray levels: encode each gray level by a number from a fixed interval (e.g. 0, 1, ..., 255: 8-bit representation – 0 = black, 255 = white)

Colours:

several colour models possible

the most frequently used one:

RGB model

(red / green / blue: primary colours for additive colour composition)

Each colour from a certain range ("gamut") can be mixed from these primary colours

examples with 8-bit intensities:

black	(0, 0, 0)
white	(255, 255, 255)
medium gray	(127, 127, 127)
red	(255, 0, 0)
green	(0, 255, 0)
blue	(0, 0, 255)
light blue	(127, 127, 255)
yellow	(255, 255, 0)

Pictures:

typically represented as raster images – rectangular array (matrix) of *pixels*, each pixel represented by its 3 colour values.

Representation of text documents (book pages, web pages...)

Level of representation is important.

- (1) Is there text on the page? – One bit.
- (2) What is the text on the page? –
Representation of letter sequence (e.g., string of ASCII characters).
- (3) What is the exact layout of the text on the page? –
"formatted text"
 - use special characters for formatting, or
 - represent the page by a rasterized black-and-white image.

Text documents with graphical elements:

- represent all as a single raster image, or
- use combined representation: several data files, one for the text, the other for the pictorial parts
→ HTML web pages are built like this

file <name>.html or <name>.htm contains text,
layout information and links to other pages
files <name>.gif or <name>.jpg or <name>.png
contain images

Messages and redundancy

Message: A finite sequence of letters, used to transfer some information via encoding/transfer/decoding

Signal: The physical representation of the message (examples: as voltage pattern or light pattern)

Redundancy: Part of a message which is not necessary for the transferred information (later explained more exactly)

Error correction by **redundant** codes: Natural languages allow to detect many errors.

Example in informatics: **Parity bits**. Even parity: 9 bits per byte. 9th bit makes number of one-bits even. Allows detection of single-bit errors. Computer memory sometimes uses 9 bits per byte for this purpose.

Other example: ISBN code (International Standard Book Number) – last character is a parity character

Entropy and quantification of information

Shannon's information theory:

Information as a measurable, statistical property of signals

How can we measure information and redundancy of characters in a message?

Assumption: N -character alphabet $\{ x_1, x_2, \dots, x_N \}$

Number of bits per character:

$$H_0 = \log_2 N$$

(Remember: $\log_2 N = (\log N)/(\log 2)$)

Information content of a single character x_i : $\log_2 \frac{1}{p(x_i)}$

Here, $p(x_i)$ is the probability of x_i .

Entropy = average value of information content of all characters

$$= H = \sum_{k=1}^N p(x_k) * \log_2 \frac{1}{p(x_k)}$$

Binary encoding needs at least, on average, H bits per character.

Redundancy: $R = H_0 - H$.

Example: Four-letter alphabet $\{a, b, c, d\}$

Probabilities: $p_a = 0.5, p_b = 0.25, p_c = 0.125, p_d = 0.125$

Thus:

$H_0 = 2$ bits per character encodable

Entropy: $0.5 * 1 + 0.25 * 2 + 0.125 * 3 + 0.125 * 3 = 1.75$ bits per character encoded

Redundancy: 0.25 bits per character

Examples:

– $a \mapsto 00, b \mapsto 01, c \mapsto 10, d \mapsto 11$: on average 2 bits per character

– $a \mapsto 0, b \mapsto 10, c \mapsto 110, d \mapsto 111$: on average 1.75 bits per character (optimal, no redundancy)

Exercises

1. How much digital storage capacity would be necessary for the content of a library with 100,000 books, if we assume that each book has 200 pages, each page has 50 lines and each line has 80 characters, and if a 1-byte ASCII code is used for the characters?

2. Below you find a part of an 8-bit ASCII code table from a web page. What character string is encoded by the bit string given in binary representation as

0100'0011'0100'1000'0100'1001'0100'0101'0100'0110 ?

(The apostrophs are only used for better overview.)

Part of 8-bit ASCII code table:

decimal	hexadecimal	character
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
...

3. Find the 8-bit two's complement representation of the negative integer -84 .

4. For a binary representation of genetic information, the four bases from DNA, adenine (A), thymine (T), guanine (G) and cytosine (C), are simply encoded by the 8-bit ASCII codes of their first letters (A = 01000001, etc.).

(a) What is the redundancy of this code, if we assume that the four bases occur with equal frequencies? (In the calculation of H_0 , consider all possible 8-bit blocks as symbols of the alphabet.

(b) Give a non-redundant binary code for A, T, G, C.

3. Databases and Geographical Information Systems (GIS)

Databases

Motivation:

Computers are often used

- for dealing with **large amounts of data**
- and in situations where **data integrity** is important for the survival of an organization.

Examples:

- Banking
- e-commerce (commercial transactions via WWW – e.g., **amazon.com** or **ebay.com**)
- meteorological measurements
- booking systems (trains, airlines...)
- telecommunication (phone numbers, fax numbers, mobile phone data...)

Main problems:

- How can large amounts of data be organized so that they can be accessed quickly?
- How can data be organized so that hardware and software failures do not lead to a disaster?
- How can data be changed by several agents in parallel without interference?

Today these problems are being dealt with on the conceptual basis of **relational database management systems** (RDBMS), typically using some dialect of **SQL** (structured query language) as notation for definition and manipulation of data.

In these slides: Only very basic concepts are discussed.

Introduction using an example

Simplistic example: public library. Data organized in **tables**.

- table “Users” with columns `UserID`, `Name`, `Address`, `BirthDate`
- table “Books” with columns `BookID`, `Title`, `Author`, `Keywords`
- table “BorrowedBooks” with columns `UserID`, `BookID`, `BorrowedSince`, `BorrowedUntil`

Principles of database tables

- Relational databases hold the data in (typically several) **tables**.
- Each row represents one **record**.
- The number and meanings of the **columns** of a table is (more or less) **fixed**.
- The number of **rows** of a table is **variable**.

"Entity relationship model":

- Each table describes one kind of **entities** or a **relation** (typically between several entities)
- a model of a certain part of reality based on the concepts of entities and their relationships is called an entity-relationship model.

In our example:

tables "Books", "Users" represent entities,
table "BorrowedBooks" represents a relation between these entities.

Attributes, key candidates and keys

Columns in a table are called **attributes**. Some attributes or attribute combinations **characterize** entities. Such attributes or attribute combinations are **key candidates**. One of the key candidates is designated as **primary key**. The primary key of an entity is used in order to refer to it from other entities or from relations.

In our example, `UserID` is used as primary key in the “Users” table, and `BookID` is used as primary key in the “Books” table. These attributes are used in “BorrowedBooks” in order to refer to the related entities.

Data definition and data manipulation with SQL

Two kinds of languages for working with relational data bases are distinguished:

data definition language (DDL)

data manipulation language (DML)

DDL and DML are today typically combined in dialects of SQL (structured query language) and supported by producers of database management systems. The different dialects are based on similar principles. We will give examples.

Data definition consists in the definition of the structure of tables and their interrelations.

During data definition, it must be defined for each table:

- which **attributes** it contains,
- how each attribute is to be **represented** (a data type must be chosen),
- which attributes form the **primary key** of the table, and
- which attributes refer as **keys** to other tables.

A notation which allows to define tables in this way is called a **data definition language** (DDL).

Data manipulation consists in adding, changing and deleting table rows and in the selection of data from the data base.

A DDL only allows to describe the structure of a data base, not to change its content in any way.

A notation which allows to manipulate tables is called a **data manipulation language** (DML).

Data definition

The “Users” table from the public library example could be defined like this:

```
CREATE TABLE Users (  
  UserID      INT(10) NOT NULL,  
  Name        CHAR(100),  
  Address     CHAR(100),  
  Birthdate   DATE,  
  PRIMARY KEY (UserID)  
)
```

This instruction creates a table names “Users” with the four already described columns. `UserID` is represented a ten-digit decimal number, `Name` and `Address` are represented as 100 characters, `Birthdate` as a date, and `UserID` is the primary key of the table.

For `UserID`, a value must be given for each row in the table – for the other three columns, a standard value (`NULL`) might be used in order to designate that the value of the attribute is not known.

The table “Books” might be defined similarly, only the attribute `Keywords` presents problems. Which amount of memory should we reserve for the keywords of a book if we do not want to restrict the number of keywords beforehand?

One solution consists in the definition of an extra table “Keywords”:

```
CREATE TABLE Keywords (  
  BookID  INT(10),  
  Keyword CHAR(100)  
)
```

Key words have a maximal length of 100 characters, but the number of key words which can be given for a book is not restricted, since the same book can occur any number of times in the table.

The “Books” table could be declared like this:

```
CREATE TABLE Books (  
    BookID INT(10) NOT NULL,  
    Title CHAR(100),  
    Author CHAR(100),  
    PRIMARY KEY (BookID)  
)
```

The table representing currently borrowed books might be declared like this:

```
CREATE TABLE BorrowedBooks (  
    UserID INT(10),  
    BookID INT(10),  
    BorrowedSince DATE,  
    BorrowedUntil DATE  
)
```

Data manipulation

The following operations can be used to manipulate the data in the tables:

- The **SELECT** command selects information from the data base.
- The **INSERT** command inserts rows into a table.
- The **UPDATE** command changes the content of existing rows in a table.
- The **DELETE** command removes rows from a table.

SELECT

The list of overdue books can be determined as follows:

```
SELECT b.BookID, b.Author, b.Title, l.BorrowedSince
FROM Books AS b, BorrowedBooks AS l
WHERE      b.BookID = l.BookID
          AND l.BorrowedUntil < TODAY
```

This statement is also called a **query** (the data base system is queried for some data).

This query returns a **table with four columns**. Each row represents an overdue book; the first column contains the book id, the second the author, the third the book title, and the last column the date when the book was borrowed.

A query has the following form:

- After the keyword **FROM**, the tables are listed from which data is to be collected. We use all combinations of rows from “Books” and “BorrowedBooks”, and we abbreviate “Books” as “b” and “BorrowedBooks” as “l” elsewhere in the query.
- The **WHERE** keyword defines a filter: only those combination of rows from the **FROM** clause are kept which fulfill the condition given behind the **WHERE**: The book ids of the two entries must match, and the date until which the book must be given back must lie in the past.
- The **SELECT** keyword introduces a list of expressions which are evaluated for each row combination filtered out by the **WHERE**. In the example, these are simply some of the attributes.

INSERT

When a book is borrowed, a row has to be added to table `BorrowedBooks`. The following instruction adds a row with `UserID 1053465`, `43565` as `BookID`, `TODAY` as `BorrowedSince` and `TODAY+14` as `BorrowedUntil`. The order of the arguments is the same as the order of the columns in the table declaration.

```
INSERT INTO BorrowedBooks
VALUES (1053465, 43565, TODAY, TODAY+14)
```

The general form is the following: After the keywords `INSERT INTO` and the name of the table, the keyword `VALUES` starts a list of values representing the row to be inserted.

UPDATE

In order to lengthen the borrowing time of the book with id `43565` by a week, the following command could be executed:

```
UPDATE BorrowedBooks
SET BorrowedUntil = BorrowedUntil + 7
WHERE BookID = 43565
```

After `UPDATE`, the name of the table to be changed is given. The `WHERE` predicate defines which rows are affected by the change, and after `SET` it is defined which columns in the rows to be changed are updated, and to which value.

DELETE

When a book is brought back by a user, its entry has to be taken out of the “BorrowedBooks” table:

```
DELETE FROM BorrowedBooks
WHERE BookID = 43565
```

Further elements of the SQL language

Above we have only seen the most elementary SQL language elements. Many SQL dialects present many more features.

Examples:

- **Integrity constraints** can be used in order to define *conditions on the content* of a database which shall never be violated during manipulations.
- **Foreign key relations** are used in order to make explicit that values in a column are keys of some other table. They are a special case of integrity constraints.
- **Index declarations** are used in order to accelerate searching in tables.
- **Stored procedures** are used in order to store instructions which are to be executed by the database.
- Further **table operations**: *set union, set difference, set intersection, grouping of results, sorting of results.*
- **Views** allow to shield the users of a database from the internal representation of the data.

- **Database administration** consists in deciding how tables etc. are represented and which users get which kind of access to the database.
- **Invariants** and **triggers** are language elements which ensure the fulfillment of integrity constraints independently of the application programme.
- **Transactions** are language elements which ensure that a sequence of changes is either executed *completely* or *not at all*, even in the case of hardware or software failures.

Conceptual database design

The **conceptual design** of a relational database often proceeds according to the following steps:

- First the **entities** relevant in the application area are collected and their types are determined. (Example: books, users)
- Then the relevant **relationships** between entities are determined. (Example: BorrowedBooks)
- For each entity type and each relationship type, the **attributes** and their data types are determined.
- Finally, **integrity conditions** for the database are specified. (Example: BorrowedUntil must not be earlier than Borrowed-Since)

On the basis of this design it is decided how entities, relationships, attributes are represented in a specific database management system.

Architecture of database applications

Database applications often use a *three-layer* architecture:

- A **DBMS** operates as the kernel of the system. It ensures data persistency, data integrity etc.
- An **application layer** provides application-specific functionality. In our example, it would provide the functions “borrow a book”, “lengthen borrowing time”, “register new user” etc.
- A **presentation layer** defines the user interface, which today is often graphical, and not seldom with an alternative using the WWW.

These three components might run as **three different programs** on different computers: A **web-browser** runs the presentation layer, the web-server dispatches the user input to an **application program**, and the application program accesses a **relational database** on a dedicated database server.

Exercise

(a) Change the query from the library example in the script so that in addition to the overdue book, the result also contains the name of the person who has borrowed the book.

(b) Write a query which determines all books which have to be given back in the next two weeks.

(Assume that from addition of a date and a number, a date results which lies the given number of days after the given date.)

Geographical Information Systems

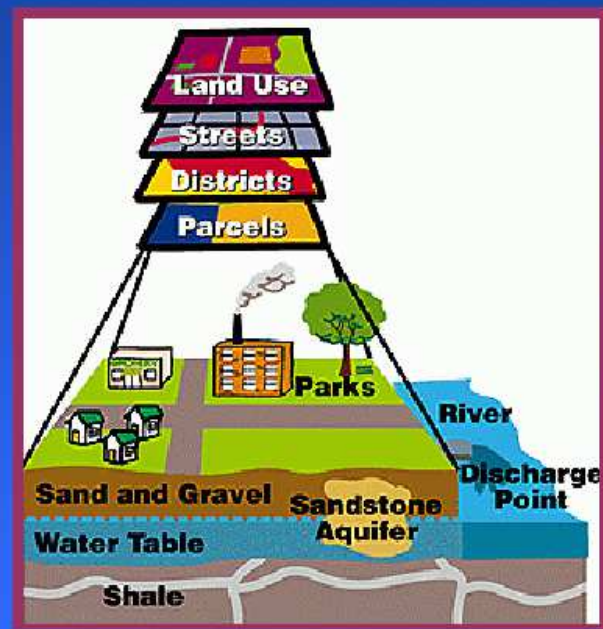
What is a Geographical Information System (GIS)?

- Software, hardware and data to help manipulate, analyse and present information that is tied to *spatial locations* (usually geographical locations).

Estimates are that 80 % of all data stored worldwide has a *spatial* component (Source: www.gis.com).

A GIS contains a classical database, but extends its functionality by methods adapted to spatial information.

Particularly, a GIS provides...



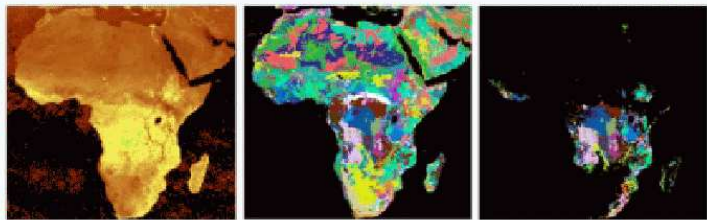
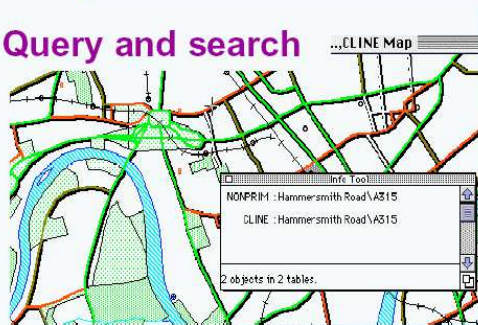
- **A method to visualize, manipulate, analyze, and display spatial data**
- **"Smart Maps" linking a database to the map**

- special forms of *query*, designed to extract information with spatial properties from a database (e.g., taking neighbourhoods into account)

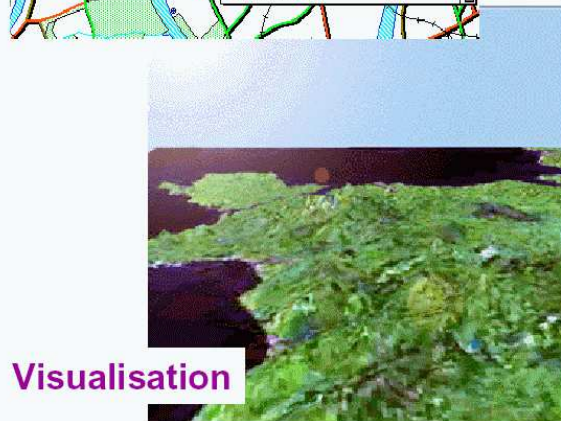
- special forms of *data analysis* (e.g., geostatistics)
- special forms of *integrity checking* adapted to spatial data.

What can a GIS do?

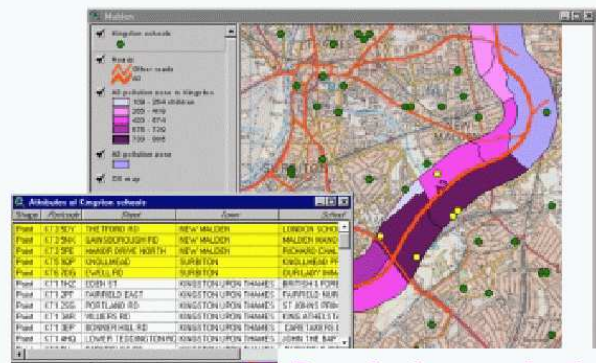
Query and search



Overlay analysis



Visualisation

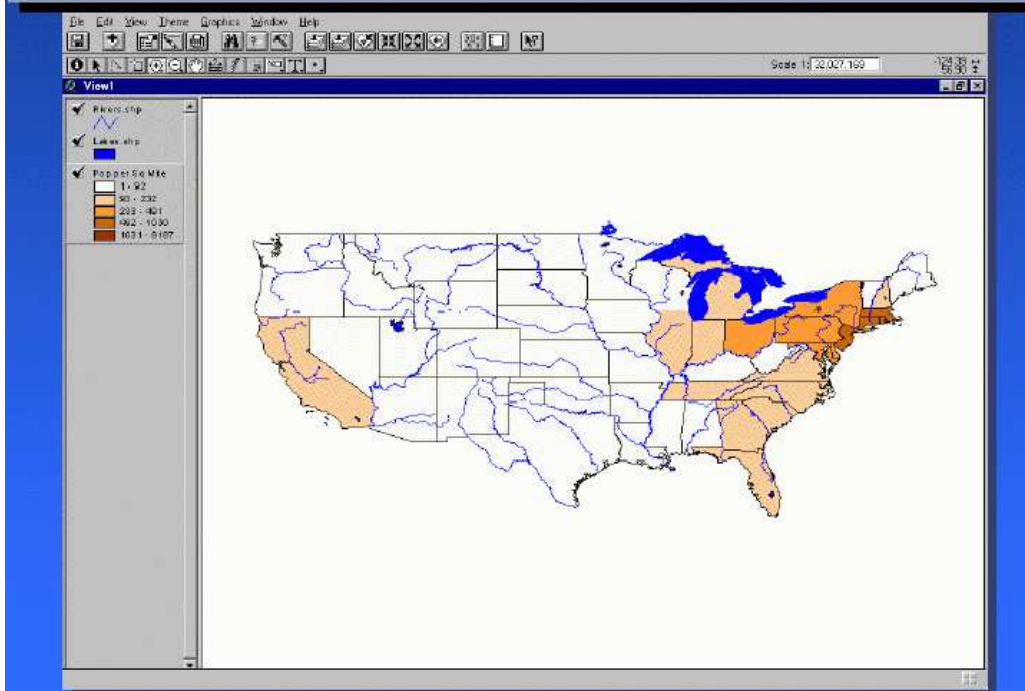


Proximity analysis

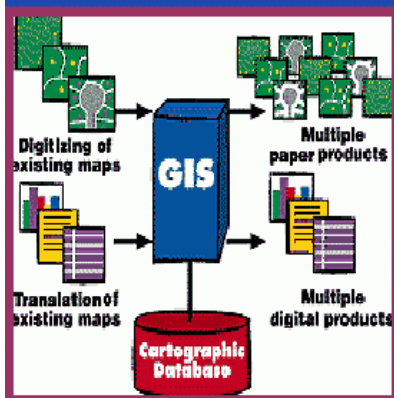
Database "Not Easy to Interpret"

ST	Abundance of States	State name	State	Year	Population	Area	Population	Area	Population	Area	Population	Area	Population	Area	Population	Area	Population	Area
Poland	9220.870	Poland	PL	1990	32000000	312700	32000000	312700	32000000	312700	32000000	312700	32000000	312700	32000000	312700	32000000	312700
Poland	14728.638	Montana	MT	1990	2000000	370000	2000000	370000	2000000	370000	2000000	370000	2000000	370000	2000000	370000	2000000	370000
Poland	5718.184	Alabama	AL	1990	3000000	130000	3000000	130000	3000000	130000	3000000	130000	3000000	130000	3000000	130000	3000000	130000
Poland	70810.152	North Dakota	ND	1990	600000	140000	600000	140000	600000	140000	600000	140000	600000	140000	600000	140000	600000	140000
Poland	77133.024	South Dakota	SD	1990	800000	190000	800000	190000	800000	190000	800000	190000	800000	190000	800000	190000	800000	190000
Poland	9710.448	Virginia	VA	1990	4000000	110000	4000000	110000	4000000	110000	4000000	110000	4000000	110000	4000000	110000	4000000	110000
Poland	50480.006	Wisconsin	WI	1990	5000000	140000	5000000	140000	5000000	140000	5000000	140000	5000000	140000	5000000	140000	5000000	140000
Poland	83303.986	Utah	UT	1990	2000000	220000	2000000	220000	2000000	220000	2000000	220000	2000000	220000	2000000	220000	2000000	220000
Poland	36124.232	Vermont	VT	1990	500000	90000	500000	90000	500000	90000	500000	90000	500000	90000	500000	90000	500000	90000
Poland	8811.450	Washington	WA	1990	4000000	170000	4000000	170000	4000000	170000	4000000	170000	4000000	170000	4000000	170000	4000000	170000
Poland	58107.801	Oregon	OR	1990	3000000	240000	3000000	240000	3000000	240000	3000000	240000	3000000	240000	3000000	240000	3000000	240000
Poland	3200.914	New Hampshire	NH	1990	1000000	90000	1000000	90000	1000000	90000	1000000	90000	1000000	90000	1000000	90000	1000000	90000
Poland	96207.220	Iowa	IA	1990	2500000	140000	2500000	140000	2500000	140000	2500000	140000	2500000	140000	2500000	140000	2500000	140000
Poland	31724.86	Massachusetts	MA	1990	6000000	90000	6000000	90000	6000000	90000	6000000	90000	6000000	90000	6000000	90000	6000000	90000
Poland	72233.337	Nebraska	NE	1990	3000000	130000	3000000	130000	3000000	130000	3000000	130000	3000000	130000	3000000	130000	3000000	130000
Poland	48603.079	New York	NY	1990	18000000	140000	18000000	140000	18000000	140000	18000000	140000	18000000	140000	18000000	140000	18000000	140000
Poland	65593.283	Pennsylvania	PA	1990	12000000	110000	12000000	110000	12000000	110000	12000000	110000	12000000	110000	12000000	110000	12000000	110000
Poland	43934.34	Connecticut	CT	1990	3000000	30000	3000000	30000	3000000	30000	3000000	30000	3000000	30000	3000000	30000	3000000	30000
Poland	7694.650	Florida	FL	1990	12000000	170000	12000000	170000	12000000	170000	12000000	170000	12000000	170000	12000000	170000	12000000	170000
Poland	7821.283	Arkansas	AR	1990	2000000	130000	2000000	130000	2000000	130000	2000000	130000	2000000	130000	2000000	130000	2000000	130000
Poland	8210.510	Indiana	IN	1990	5000000	90000	5000000	90000	5000000	90000	5000000	90000	5000000	90000	5000000	90000	5000000	90000
Poland	15067.293	Colorado	CO	1990	3000000	260000	3000000	260000	3000000	260000	3000000	260000	3000000	260000	3000000	260000	3000000	260000
Poland	84651.85	Idaho	ID	1990	1000000	140000	1000000	140000	1000000	140000	1000000	140000	1000000	140000	1000000	140000	1000000	140000
Poland	197714.10	Kentucky	KY	1990	3000000	100000	3000000	100000	3000000	100000	3000000	100000	3000000	100000	3000000	100000	3000000	100000
Poland	41316.62	Ohio	OH	1990	11000000	110000	11000000	110000	11000000	110000	11000000	110000	11000000	110000	11000000	110000	11000000	110000
Poland	9257.934	Illinois	IL	1990	12000000	140000	12000000	140000	12000000	140000	12000000	140000	12000000	140000	12000000	140000	12000000	140000
Poland	36.003	Distric of Columbia	DC	1990	500000	600	500000	600	500000	600	500000	600	500000	600	500000	600	500000	600
Poland	204.806	Delaware	DE	1990	700000	3000	700000	3000	700000	3000	700000	3000	700000	3000	700000	3000	700000	3000
Poland	24240.212	West Virginia	WV	1990	4000000	60000	4000000	60000	4000000	60000	4000000	60000	4000000	60000	4000000	60000	4000000	60000
Poland	3203.792	Michigan	MI	1990	9000000	140000	9000000	140000	9000000	140000	9000000	140000	9000000	140000	9000000	140000	9000000	140000
Poland	146101.181	California	CA	1990	28000000	160000	28000000	160000	28000000	160000	28000000	160000	28000000	160000	28000000	160000	28000000	160000
Poland	43181.777	Kansas	KS	1990	3000000	180000	3000000	180000	3000000	180000	3000000	180000	3000000	180000	3000000	180000	3000000	180000
Poland	32114.84	Tennessee	TN	1990	4000000	130000	4000000	130000	4000000	130000	4000000	130000	4000000	130000	4000000	130000	4000000	130000
Poland	88161.104	Yupia	YS	1990	5000	1000	5000	1000	5000	1000	5000	1000	5000	1000	5000	1000	5000	1000
Poland	58811.824	Missouri	MO	1990	3000000	160000	3000000	160000	3000000	160000	3000000	160000	3000000	160000	3000000	160000	3000000	160000
Poland	11707.552	Alabama	AL	1990	3000000	130000	3000000	130000	3000000	130000	3000000	130000	3000000	130000	3000000	130000	3000000	130000
Poland	7082.336	Delaware	DE	1990	700000	3000	700000	3000	700000	3000	700000	3000	700000	3000	700000	3000	700000	3000
Poland	48648.813	North Carolina	NC	1990	7000000	110000	7000000	110000	7000000	110000	7000000	110000	7000000	110000	7000000	110000	7000000	110000

Visualization "Worth a Thousand Words"

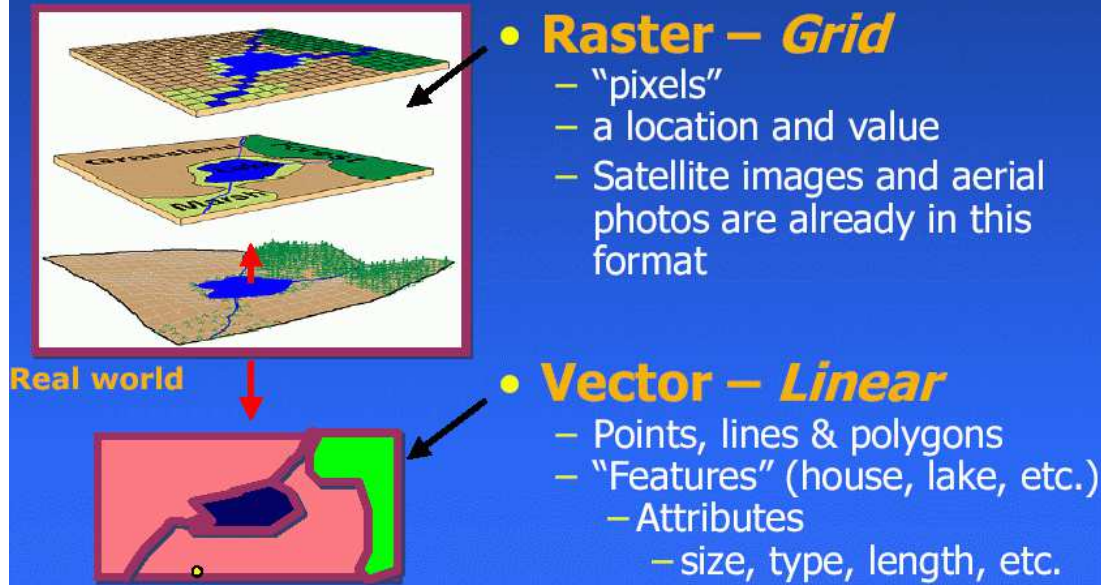


Data For GIS Applications



- **Digitized and Scanned Maps**
 - purchased, donated, free (Internet)
 - created by user
- **Data Bases** – Tables of data
- **GPS** – Global Positioning System
 - accurate locations
- **Field Sampling of Attributes**
- **Remote Sensing & Aerial Photography**

Two Ways to Input and Visualize Data The World in GIS



The vector representation is more appropriate for senseful queries (and is more exact)
– basis for relational database representation of geographical data

Typical *entities* of a GIS:

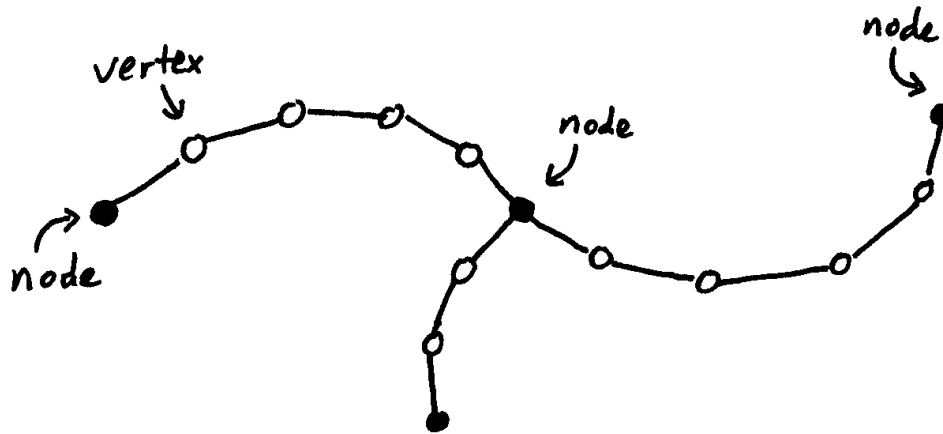
- *Points*
- *Tics* (= special points for which the exact real-world coordinates are known, used to fit a digital map into a global coordinate system)
- *Lines*, also called *arcs* (more precisely: Multilines, i.e. consisting of several linear segments)



- *Polygons* (closed multilines, possibly with additional attributes)
- *Annotations* (text objects associated with points).

The endpoints of a line (and possible branching points) are called *nodes*.

Intermediate points (without branching) are called *vertices*.



Tables in the underlying relational database:

- Tic table
- boundary table (represents the spatial extent of the map – a surrounding rectangle)
- arc attribute table (AAT)
- polygon attribute table (PAT).

E.g., a *polygon* is represented as a line in the PAT, with attributes:

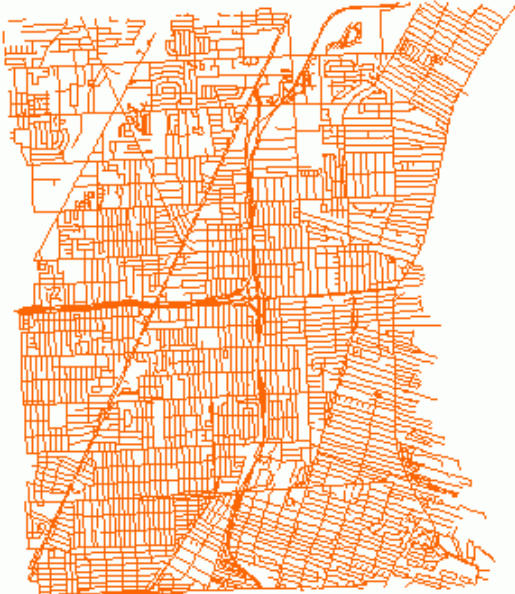

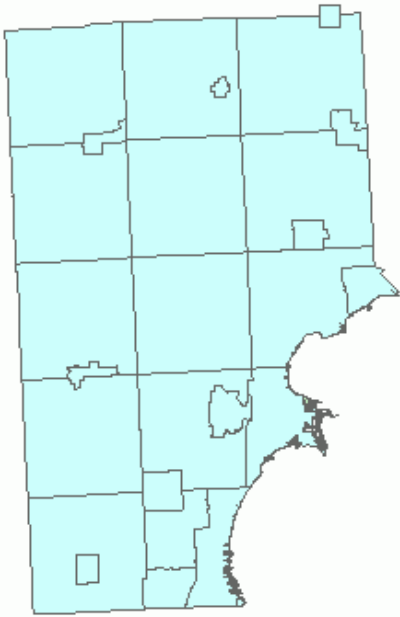
polygon ID, nodes, arcs, a label point (in the interior), further attributes (e.g., area, slope, population density...).

Details differ between different GIS.

Usually, a GIS does not only contain information for a single map of a region, but *several sorts of information for the same region*:

each sort of information is represented in an extra *coverage* (also called *layer*, *cover* or *theme*).

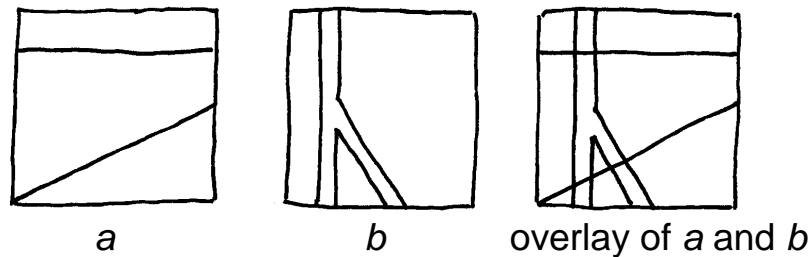
Example: Different coverages of a town area

<p>Road Centerline/Address (Geographic Township) - source: 2000 Digital Orthophotography</p> <ul style="list-style-type: none">• Road Arcs• Address Ranges• Road Names 	<p>Hydrology (Geographic Township) - source: 2000 Digital Orthophotography</p> <ul style="list-style-type: none">• Water feature arcs (rivers, streams, drains)• Water feature polygons (lakes, ponds, retention basins)• Names of County maintained drains 
<p>Municipal Boundaries (County) - source: 2000 Digital Orthophotography</p> <ul style="list-style-type: none">• Community polygons• Community name 	<p>(source: http://macombcountymi.gov/gis/gis_coverage_samples.htm)</p>

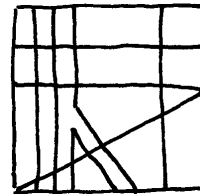
How to combine several coverages?

- *Overlay operation*

From two geometries, the GIS calculates the coarsest common geometry:



Attention: The following geometry



would also

be a common geometry of *a* and *b*, but not the coarsest one!

Using overlay, a GIS can give answer to questions like this:

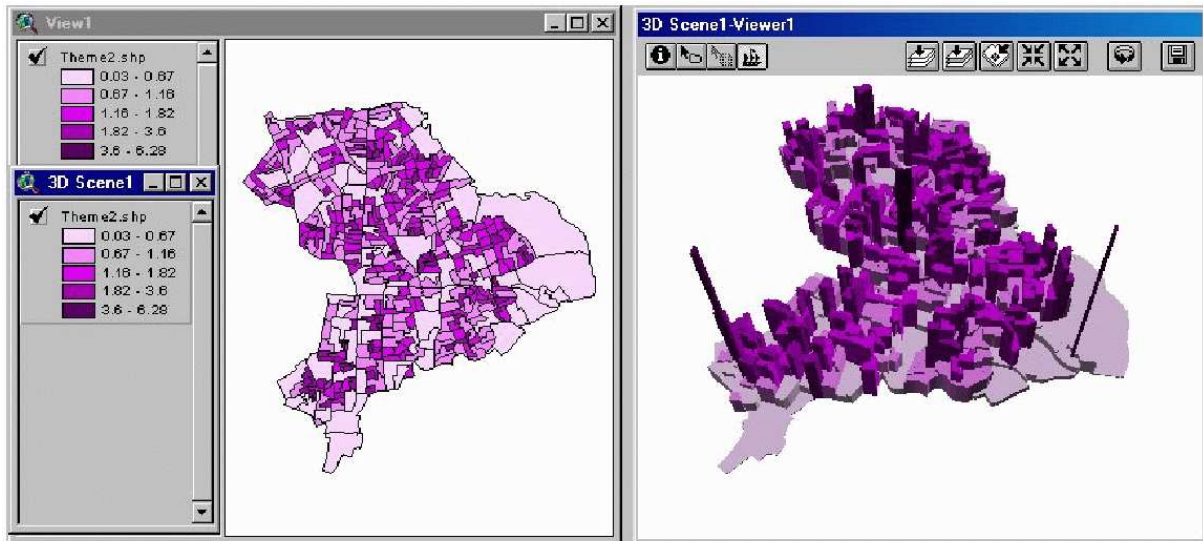
"What forest areas of district *x* are within 100 m distance to a road, are stocked with conifers and have a slope < 5 degrees?" (e.g., for a chalking action)

Layers used for this task:

- landuse map (→ forests)
- political district map (→ district *x*)
- road map (→ 100 m neighbourhood to a road)
- forest type map (→ stocked with conifers)
- digital elevation model (→ slope < 5 degrees)

Selection of polygons of the overlay using an "and" operation

Further functionality of GIS: 3D visualization



Representing Attribute Data in 3-D: Population Density in Small Census Areas in the London Borough of Hackney

Widely used GIS products:

- ESRI ArcGIS (licenced commercial software)
- QuantumGIS (free and open source, <http://www.qgis.org>)

The Web can be seen as a sort of database – but very different from relational databases:

- highly distributed, decentralized;
- based on the hypertext model instead of the entity-relationship model;
- with only very weak standards to restrict form and content of the pages;
- very large
- without a universal query language.

(*Search engines* try to compensate the last item; see below.)

History of the WWW:

- Idea of hypertext: Vannevar Bush 1945
- Origin of WWW: a project at CERN (Geneva) in 1989
- *Tim Berners-Lee* and *Robert Cailliau*
- their system: ENQUIRE, realized core ideas of the Web in order to enable access to library information that was scattered on several different computers at CERN
- proposal for the WWW: published by Berners-Lee on November 12, 1990
- *first web page* on November 13 on a NeXT workstation
- Christmas 1990: Berners-Lee built the first web browser and the first web server
- August 6, 1991: summary of the WWW project posted in a newsgroup in the internet
- April 30, 1993: CERN announced that the WWW would be free to anyone
- 1993: Browser Mosaic (forerunner of Internet Explorer or Firefox) starts to popularize the WWW

The three core standards of the Web:

- Uniform Resource Locator (URL): specifies how each page of information is given a unique address at which it can be found (e.g., `http://en.wikipedia.org/wiki/World_Wide_Web`)
- Hypertext Transfer Protocol (HTTP): specifies how the browser and server send the information to each other
- Hypertext Markup Language (HTML): a webpage description language used to encode the information so that it can be displayed on a variety of devices and under different operating systems.

Later extensions:

- Cascading Style Sheets (CSS): define the appearance of elements of a web page, separating appearance and content
- XML: more general language than HTML, designed to enable a better separation of appearance and content; also applicable to other sorts of information
- ECMAScript (also called JavaScript or JScript): a programming language with commands for the browser, enables embedding of programmes (scripts) into web pages. Thus web pages can be changed dynamically.
- Hypertext Transfer Protocol Secure (HTTPS): Extension of HTTP where the protocol SSL is evoked to encrypt the complete data transfer
- Java applets (small programmes) can be embedded in web pages and run on the computer of the Web user

The *World Wide Web Consortium (W3C)* develops and maintains some of these standards (HTML, CSS) in order to enable computers to effectively store and communicate different kinds of information.

Problems with the Web:

- *highly decentralized*, no control of the content

→ there is a lot of false and misleading information, hate campaigns, promotion of sexual exploitation, of terrorism and of other crimes...

- *highly dynamic: Web pages change all the time!*
Links point to nowhere when the target page was removed...

→ when you give a Web address in the References section of a scientific paper or in your thesis, you should add the *date* when you visited that page!

Archive of (a part of) the Web:

<http://archive.org/web/>

→ lost Web references can (in some cases) be reconstructed if the date is known

- *highly chaotic*: no global index or table of content is available; search for a certain content is complicated and time consuming

→ development of specialized search engines, the most well-known one: *Google* (<http://www.google.de>)

How does a search engine work?

- First component: a web crawler, visiting all accessible web pages worldwide, one after the other, following the hyperlinks

but: when you look for a certain keyword, this process would take much too long!

→

- second component: a large database, containing keywords and web addresses where these keywords were already found

the web crawler is working in the background and does only actualize the database

when you invoke Google, you search in Google's database, not in the Web!

→ not all Web pages can be found, because not all are in the database

Usually, you get many, many, many Web pages containing a given keyword (often millions...)

→

first remedy: make more intelligent queries

e.g., combining several keywords by "and", or looking for phrases instead of keywords (use quotation marks)

– Google provides such facilities under "extended search"

still there are often too many results

→ prioritisation of the found web pages necessary

- third component of the search engine (and best capital of the Google company): a *ranking algorithm* for search results

Basic principles of Google ranking of web pages

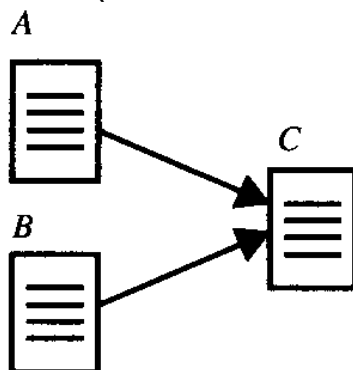
(Attention: the exact algorithm is changing continuously and is not published)

"Importance" of a web page:
recursively defined, using the hyperlink structure of the Web

*The importance of a page is the larger,
the more important pages refer to it!*

More precisely:

Let $FLinks(A)$ be the set of all outgoing links (forward links) of a page A and $BLinks(A)$ the set of all incoming links (backward links) of A



$$\begin{aligned}FLinks(A) &= \{C\} \\FLinks(B) &= \{C\} \\BLinks(C) &= \{A, B\}\end{aligned}$$

- A has high page rank if the sum of the page ranks of its incoming links is high,
- a page B distributes its importance in equal parts to all pages which are referred by it:

$$PageRank(A) = \frac{1}{c} \sum_{B \in BLinks(A)} \frac{PageRank(B)}{|FLinks(B)|}.$$

(c = normalisation factor)

Iterative determination of the page rank:

- initially, an arbitrary mapping of values to all web pages is done (typically, the *constant value* 1 is used),
- *iterate the calculation* using the above formula for all pages, until the values remain stable,
- they *converge* against the Eigenvectors of the adjacency matrix of the graph consisting of the web pages (nodes) and their links (edges).
(Adjacency matrix: $a_{ij} = 1$ iff nodes i and j are connected by an edge.)

Additionally, the Google page rank utilizes:

- *proximity* of the given key words to each other (in the text),
- the *anchor texts* of the links: these are the texts which can be clicked upon. A page A gets higher importance when the anchor texts of links referring to A contain the keywords, too.

The underlying technology of the WWW:
the ***Internet*** (short for "Interconnected Networks")

predecessor (end of the 1960s): ARPANET (U.S. military project)

was later used to connect universities and research labs

Internet today: A worldwide network of computer networks

- Computers in this network communicate using the standardized *TCP/IP protocol* (Transmission Control Protocol / Internet Protocol: Rules governing the communication)
- *Transmission of the information in small portions*
- For identification, each computer in the net has a unique number, the *IP address*
- to get identifiers which can better be memorized: *Domain Name System* (DNS)
 - system of (textual) names, association between names and IP addresses
- hierarchy: Domains, subdomains, sub-subdomains..., e.g.,
www.uni-forst.gwdg.de
(from right to left!)

- *Top-level domains*: Country abbreviations and some others ("generics"): .de, .fr, .eu, .com, .edu, .gov ...
- Lowest level: host name of a single computer (here: www, Web server of the forestry faculty)
- domain name corresponds to IP address
- transformation of domain names into IP addresses and vice versa: Task of special computers, so-called *nameservers*
- this transformation takes place any time when you click on a hyperlink on a web page!
- each nameserver is responsible for a certain part of the hierarchical name space.