Introduction

Large unstructured or semi-structured datasets require a high level of computational sophistication because operations that are easy at a small scale — such as moving data between machines or in and out of storage, visualizing the data, or displaying results —can all require substantial algorithmic ingenuity. As a data set becomes increasingly massive, it may be infeasible to gather it in one place and analyze it as a whole. Thus, there may be a need for algorithms that operate in a distributed fashion, analyzing subsets of the data and aggregating those results to understand the complete set. One aspect of this is the challenge of data assimilation, in which we wish to use new data to update model parameters without reanalyzing the entire data set. This is essential when new waves of data continue to arrive, or subsets are analyzed in isolation of one another, and one aims to improve the model and inferences in an adaptive fashion — for example, with streaming algorithms [NRC, 2013].

The White House Office of Science and Technology Policy (OSTP) — in concert with several Federal departments and agencies — created the “Big Data Research and Development Initiative” to:

- Advance state-of-the-art core technologies needed to collect, store, preserve, manage, analyze, and share huge quantities of data;
- Harness these technologies to accelerate the pace of discovery in science and engineering, strengthen national security, and transform teaching and learning;
- Expand the workforce needed to develop and use Big Data technologies.

By improving the ability to extract knowledge and insights from large and complex collections of digital data, the initiative promises to help solve some the Nation’s most pressing challenges [BIG DATA INITIATIVE, 2012].

For instance, as it is introduced in [Big data, 2012], the USA Department of Defense (DOD) is “placing a big bet on big data” investing $250 million annually (with $60 million available for new research projects) across the Military Departments in a series of programs that will:

- Harness and utilize massive data in new ways and bring together sensing, perception and decision support to make truly autonomous systems that can maneuver and make decisions on their own;
- Improve situational awareness to help war fighters and analysts and provide increased support to operations. The Department is seeking a 100-fold increase in the ability of analysts to extract information from texts in any language, and a similar increase in the number of objects, activities, and events that an analyst can observe.

The ***XDATA*** program seeks to develop computational techniques and software tools for analyzing large volumes of data, both semi-structured (e.g., tabular, relational, categorical, meta-data) and unstructured (e.g., text documents, message traffic). Central challenges to be addressed include:

- Developing scalable algorithms for processing imperfect data in distributed data stores;
- Creating effective human-computer interaction tools for facilitating rapidly customizable visual reasoning for diverse missions.

The program envisions open source software toolkits for flexible software development that enable processing of large volumes of data for use in targeted defense applications.

The *Cyber-infrastructure for a Billion Electronic Records (CI-BER)* of the USA National Archives & Records Administration (NARA) is a joint agency sponsored testbed notable for its application of a multi-agency sponsored cyber infrastructure and the National Archives' diverse 87+ million file collection of digital records and information now active at the Renaissance Computing Institute. This testbed will evaluate technologies and approaches to support sustainable access to ultra-large data collections.

At the end, in the USA National Science Foundation (NSF), the *Information Integration and Informatics* addresses the challenges and scalability problems involved in moving from traditional scientific research data to very large, heterogeneous data, such as the integration of new data types models and representations, as well as issues related to data path, information life cycle management, and new platforms [Big data, 2012].

Worldwide Big Data technology and services are expected to grow. The challenge is to strengthen Europe's position as provider of innovative multilingual products and services based on digital content and data, addressing well identified industry and consumer market needs. Research and Innovation activities in this challenge will provide professionals and citizens with new tools to model, analyze, and visualize vast amounts of data from which to extract more value, to make an intelligent use of data coming from different sources and to create, access, exploit, and re-use all forms of digital content in any language and with any device [HORIZON 2020, 2013].

In accordance with the actuality of these problems, this work is aimed to solve the problem of searching in big data structures by proposing a special kind of hashing, so-called "multi-layer hashing", i.e. by implementing recursively the same specialized hash function to build and resolve the collisions in hash tables. In other words, the main idea consists in using the specialized hashing function in depth till it is needed.

This approach is called "Natural Language Addressing" (NLA) [Ivanova et al, 2012a; Ivanova et al, 2013a; Ivanova et al, 2013d]. In this work we will concern:

- structured data (dictionaries, thesauruses, ontologies);
- semi-structured data (big RDF triple or quadruple datasets),

and will provide corresponded experiments and experimental practical implementation.

The idea of Natural Language Addressing (NLA)

➢ *Variety and orderliness*

The world around us can be described in one word as "Variety". It is difficult to agree that the world has not so needed orderliness, created over millennia, developed and maintained constantly as oasises of ***order*** in the core of the chaos... It is strange for our perception of the world as a four-dimensional existence. It is strange, because our mind builds a completely different picture of ordered spatiality and extensity.

The concept "order" has many meanings but here it is used in the sense of a condition of logical or comprehensible arrangement among the separate elements of a group [AHD, 2009]; a state in which all components or elements are arranged logically, comprehensibly, or naturally; sequence (*alphabetical order)* [Collins, 2003]; arrangement of thoughts, ideas, temporal events [WordNet, 2012].

One very important aspect of the order is that every entity of the ordered set has its own location in it. The names of these locations are called addresses.

The common sense meaning of the concept "***address***" is such as a description of the location of a person or organization, as written or printed on mail as directions for delivery [AHD, 2009]; the conventional form by which the location of a building is described [Collins, 2003]; a sign in front of a house or business carrying the conventional form by which its location is described; [WordNet, 2012].

We will use the concept **"address"** in the sense accepted in the Computer Science: the code that identifies where a piece of information is stored [WordNet, 2012]; a name or number used in information storage or retrieval that is assigned to a specific memory location; the memory location identified by this name or number; a name or a sequence of characters that designates an e-mail account or a specific site on the Internet or other network [AHD, 2009].

It is important to take in account that the memory address may be of two kinds [Stably, 1970]:

- Physical location in any device (hard disk, main memory, flash memory);
- Logical (relative) location in a file given as an offset from the beginning of the file, i.e. the position of a byte in the file. In other words, it is the sequential number of the pointed byte in the file, starting from zero.

In this research we will use concept "*memory address*" only in the second sense, i.e. as ***offset in a file stored somewhere in the computer accessible local or global network environment.***

➢ *Name – Address - Route*

In January 1978, John F. Shoch from "Xerox Palo Alto Research Center" had written a very interesting note [Shoch, 1978a]. Later in the same year he had published this note in the paper [Shoch, 1978b]. This classical paper became as a mile stone in the further research concerning the naming, addressing and routing at the first place with its "extremely general definition" [Shoch, 1978a]:

*The "**name**" of a resource indicates "what" we seek,*
*an "**address**" indicates "where" it is, and*
*a "**route**" tell us "how to get there".*

This definition gives us a quick and intuitive understanding of the fundamental concepts of naming. Informally, a name is a string of symbols that identifies an object, thus both a human readable text-string and a binary number can be a name. Ideally, all objects would be named and handled in a uniform manner [Jording & Andreasen, 1994].

Shoch gave "some further detail to flesh this out" [Shoch, 1978a]:

I. A "*name*" is a symbol - usually a human-readable string - identifying some resource, or set of resources. The name (what we seek) needs to be bound to the address (where it is).

II. An "*address*", however, is the data structure whose format can be recognized by all elements in the domain, and which defines the fundamental addressable object. The address (where something is) needs to be bound to the route (how to get there).

III. A "*route*" is the specific information needed to forward a piece of information to its specified address.

Thus, a "name" may be used to derive an "address", which may then be used to derive a "route".

There is an interesting similarity between this structure and mechanisms used in programming languages (where one must bind a value to a variable), or in operating systems (where one must link a particular piece of code into a module) [Shoch, 1978a].

Establishing and supporting the correspondence between logical and physical addresses is duty of the operating systems or, in general, of all service functions of the local or global networks. This correspondence is transparent for the end user programs which request the logical address and operating environment is responsible to locate and access concrete physical location.

Special kind of files are so called "*main memory mapped files*" which are accessible as files but during their processing are stored in the main computer memory and only updated their blocks are written on the external memory devices. Such kind of processing of files is useful for speeding the work of programs.

The concept "(logical) address" is closely connected with the term "***information model***" [Ivanova, 2013].

➢ *Information models*

We continuously build information models of the world and of ourselves in this world. The need of coordinating our actions with others humans or intelligent devices requires constant information exchange (interaction), the basis of which are the information models.

In the Computer Science, the term "***information model***" is popular.

"An information model is a representation of concepts, relationships, constraints, rules, and operations to specify data semantics for a chosen domain of discourse. The advantage of using an information model is that it can provide sharable, stable, and organized structure of information requirements for the domain context. An information modeling language is a formal syntax that allows users to capture data semantics and constraints" [Lee, 1999].

In other words, the modeled objects are information structures and the relations between them. The "***computer information models***" concern logical organization of storing the information and operations with it.

It is wrong to believe that the information models are a phenomenon only of humans. But only for humans there exist letters and accordingly ***textual information models*** (see for instance [Čech, 2012]).

➢ *Addressing in the textual information models*

The simplest textual information model is a linear structure of text elements – letters, words, sentences or more complicated structures like tags in XML.

Some models have continuous internal structure which may be divided on substructures, etc. For instance, the Brookshear's "Overview of the Computer Science" is such model. It is represented by a book with chapters [Brookshear, 2012]. It is a complex information model because contains non-textual elements: graphics and pictures.

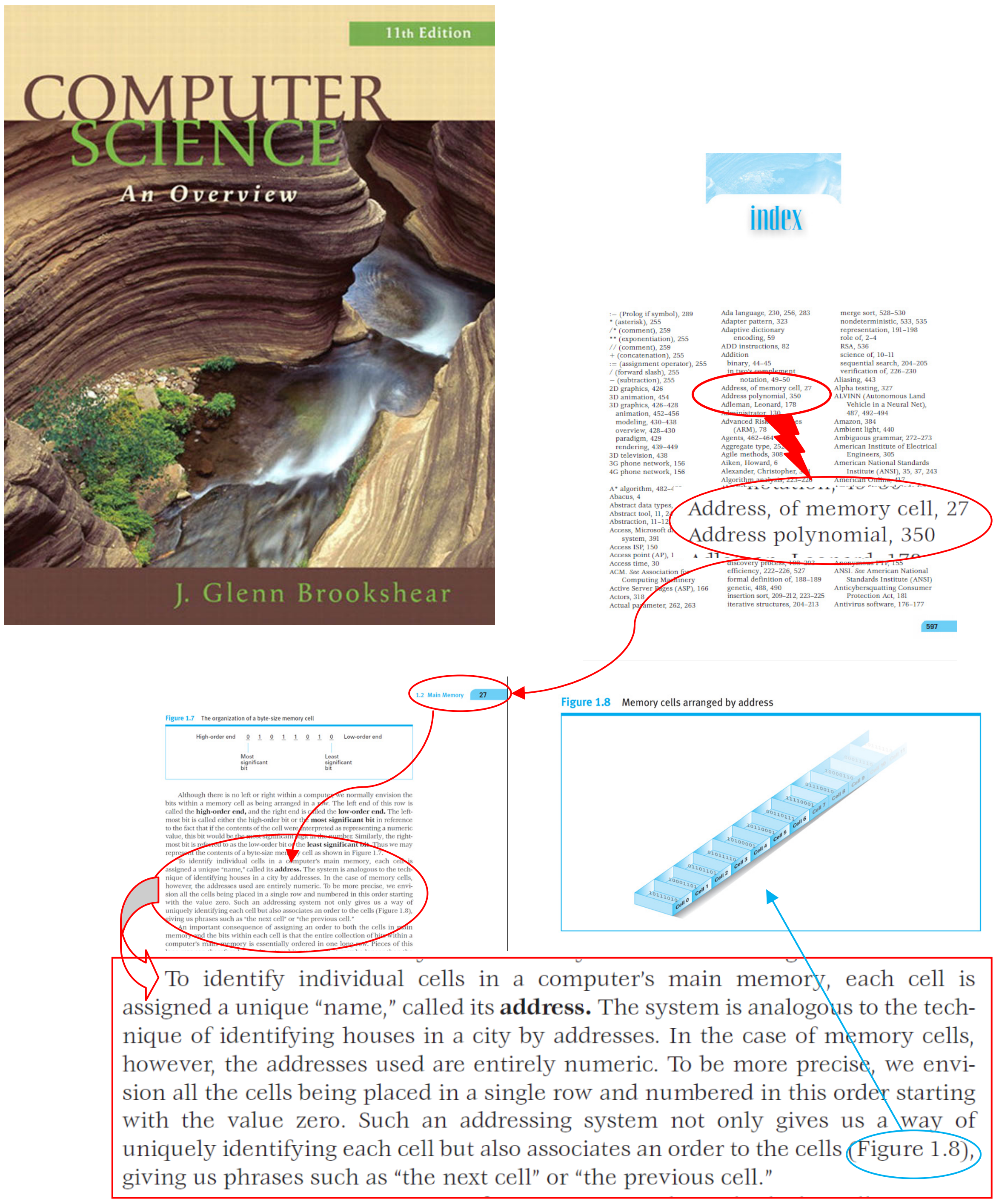
The nonlinear information models may be represented by graphs of interconnected textual elements. An example of such model is graphical representation of any ontology. Other examples are relational structures usually represented by sets of tables.

When the definitions are placed randomly in a book, for the sake of convenience at the end of book is located an index with main concepts and numbers of the pages where the concepts are defined. One needs to follow simple algorithm to find a definition. This is illustrated at Figure 1 for the concept "address, of memory cell".

The important elements of the textual models may be defined by corresponded definitions located in different places of the text. If the concepts together with theirs definitions are ordered alphabetically, like in a dictionary (Figure 2), going through the text one may found the needed concept and its definition.

In other words, irrespective of the type of the textual information model, every text element has its own location in the text and, respectively, its own relative address in the text document (page, paragraph, number of word, etc.) or file (relative offset from the first position in the file). Some of the elements may be so important to be pointed by their relative positions in an *index*.

Index is a sequential arrangement of material, especially in alphabetical or numerical order, which serves to guide, point out or otherwise facilitate reference, especially: a more or less detailed alphabetized list of names, places, subjects, etc, treated in the text of a printed work. It usually appears at the end of the book and identifies page numbers on which information about each subject appears [AHD, 2009; Collins, 2003].



:– (Prolog if symbol), 289
* (asterisk), 255
/* (comment), 259
** (exponentiation), 255
// (comment), 259
+ (concatenation), 255
:= (assignment operator), 255
/ (forward slash), 255
– (subtraction), 255
2D graphics, 426
3D animation, 454
3D graphics, 426–428
animation, 452–456
modeling, 430–438
overview, 428–430
paradigm, 429
rendering, 439–449
3D television, 438
3G phone network, 156
4G phone network, 156

A* algorithm, 482–
Abacus, 4
Abstract data types,
Abstract tool, 11, 2
Abstraction, 11–12
Access, Microsoft d
system, 391
Access ISP, 150
Access point (AP), 1
Access time, 30
ACM. *See* Association for Computing Machinery
Active Server Pages (ASP), 166
Actors, 318
Actual parameter, 262, 263

Ada language, 230, 256, 283
Adapter pattern, 323
Adaptive dictionary encoding, 59
ADD instructions, 82
Addition
binary, 44–45
in two's complement notation, 49–50
Address, of memory cell, 27
Address polynomial, 350
Adleman, Leonard, 178
Administrator, 130
Advanced Risk es (ARM), 78
Agents, 462–464
Aggregate type, 252
Agile methods, 308
Aiken, Howard, 6
Alexander, Christopher,
Algorithm analysis, 223–226

Address, of memory cell, 27
Address polynomial, 350

discovery process, 198–202
efficiency, 222–226, 527
formal definition of, 188–189
genetic, 488, 490
insertion sort, 209–212, 223–225
iterative structures, 204–213

merge sort, 528–530
nondeterministic, 533, 535
representation, 191–198
role of, 2–4
RSA, 536
science of, 10–11
sequential search, 204–205
verification of, 226–230
Aliasing, 443
Alpha testing, 327
ALVINN (Autonomous Land Vehicle in a Neural Net), 487, 492–494
Amazon, 384
Ambient light, 440
Ambiguous grammar, 272–273
American Institute of Electrical Engineers, 305
American National Standards Institute (ANSI), 35, 37, 243
American Online, 417
Anonymous FTP, 155
ANSI. *See* American National Standards Institute (ANSI)
Anticybersquatting Consumer Protection Act, 181
Antivirus software, 176–177

Figure 1.7 The organization of a byte-size memory cell

High-order end 0 1 0 1 1 0 1 0 Low-order end
Most significant bit
Least significant bit

Although there is no left or right within a computer, we normally envision the bits within a memory cell as being arranged in a row. The left end of this row is called the **high-order end,** and the right end is called the **low-order end.** The leftmost bit is called either the high-order bit or the **most significant bit** in reference to the fact that if the contents of the cell were interpreted as representing a numeric value, this bit would be the most significant digit in the number. Similarly, the rightmost bit is referred to as the low-order bit or the **least significant bit.** Thus we may represent the contents of a byte-size memory cell as shown in Figure 1.7.

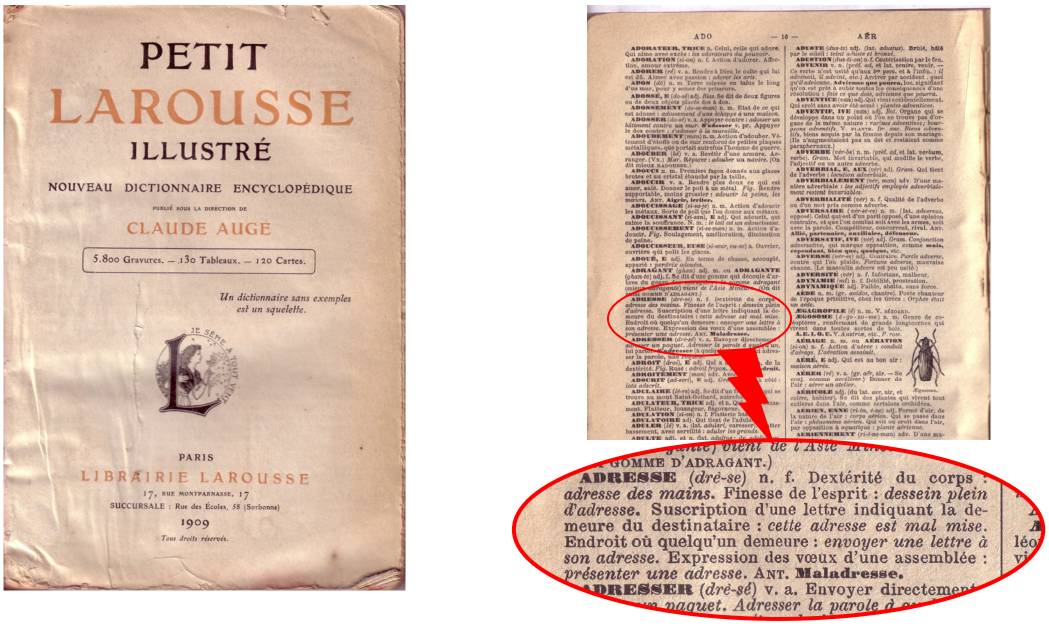
To identify individual cells in a computer's main memory, each cell is assigned a unique "name," called its **address.** The system is analogous to the technique of identifying houses in a city by addresses. In the case of memory cells, however, the addresses used are entirely numeric. To be more precise, we envision all the cells being placed in a single row and numbered in this order starting with the value zero. Such an addressing system not only gives us a way of uniquely identifying each cell but also associates an order to the cells (Figure 1.8), giving us phrases such as "the next cell" or "the previous cell."

An important consequence of assigning an order to both the cells in main memory and the bits within each cell is that the entire collection of bits within a computer's main memory is essentially ordered in one long row. Pieces of this

To identify individual cells in a computer's main memory, each cell is assigned a unique "name," called its **address.** The system is analogous to the technique of identifying houses in a city by addresses. In the case of memory cells, however, the addresses used are entirely numeric. To be more precise, we envision all the cells being placed in a single row and numbered in this order starting with the value zero. Such an addressing system not only gives us a way of uniquely identifying each cell but also associates an order to the cells (Figure 1.8), giving us phrases such as "the next cell" or "the previous cell."

Figure 1. Addressing by indexing [Brookshear, 2012]

Sets of concepts and their definitions, organized in dictionaries, are ordered alphabetically and this way location of every concept may be found easily.



PETIT
LAROUSSE
ILLUSTRÉ
NOUVEAU DICTIONNAIRE ENCYCLOPÉDIQUE
PUBLIÉ SOUS LA DIRECTION DE
CLAUDE AUGÉ
5.800 Gravures. — 130 Tableaux. — 120 Cartes.
Un dictionnaire sans exemples est un squelette.
JE SÈME À TOUT VENT
PARIS
LIBRAIRIE LAROUSSE
17, RUE MONTPARNASSE, 17
SUCCURSALE : Rue des Écoles, 58 (Sorbonne)
1909
Tous droits réservés.

ADO — 16 — AÉR

GOMME D'ADRAGANT.)
ADRESSE (*drè-se*) n. f. Dextérité du corps : *adresse des mains.* Finesse de l'esprit : *dessein plein d'adresse.* Suscription d'une lettre indiquant la demeure du destinataire : *cette adresse est mal mise.* Endroit où quelqu'un demeure : *envoyer une lettre à son adresse.* Expression des vœux d'une assemblée : *présenter une adresse.* ANT. **Maladresse.**
ADRESSER (*drè-sé*) v. a. Envoyer directement
paquet. Adresser la parole

Figure 2. Addressing by natural language order [Auge, 1909]

➢ *Computer indexes*

The text information models may be stored as files in the (internal or external) computer memory. Locating the concepts and definitions may be done by:

– Direct scanning the files;
– Indexing and based on it search of the pointer to the address (number of the first byte) of the text element (record in the file).

Scanning the files is convenient only for small volumes of concepts and definitions. Some rationalization is possible using some algorithms like binary search.

Indexing is creating tables (indexes) that point to the location of folders, files and records. Depending on the purpose, indexing identifies the location of resources based on file names, key data fields in a database record, text within a file or unique attributes in a graphics or video file [PC mag, 2013].

In database design, an index is a list or a reference table of keys (or keywords), each of which identifies a unique record or document and is used to locate a particular element within a data array or table. Indices make it faster to find specific records and to sort records by the index field - that is, the field used to identify each record [Webopedia, 2013; AHD, 2009; Collins, 2003].

For large volumes of concepts, the indexes became too large and additional, secondary indexing is needed. Such multi-level index structures are well-known B-trees of Rudolf Bayer [Bayer, 1971] as well as B^+ trees [Knuth, 1997] (Figure 3).

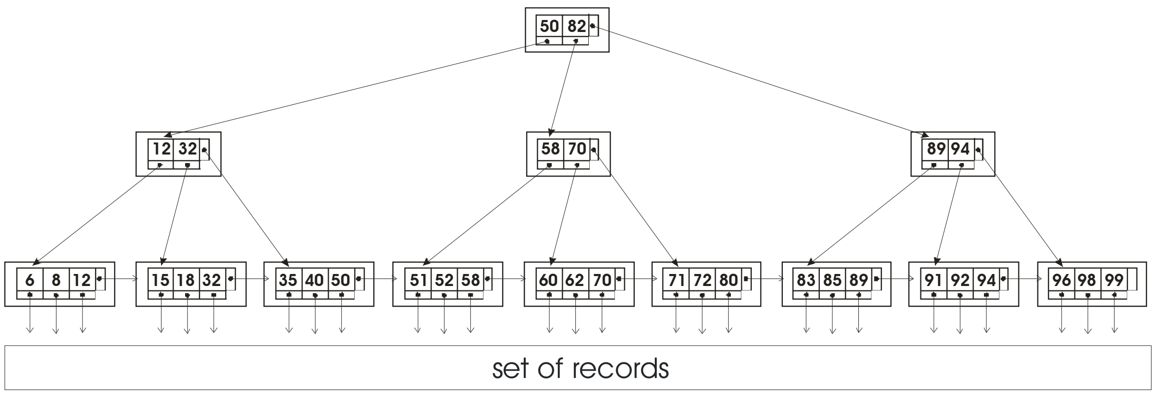


Figure 3. B-tree

Let repeat, *the main idea of indexing is to facilitate the search by search in the (multi-level) index* and after that to ensure the *direct access to the address given by the pointer*. The address is relative offset of the first byte of the record from the beginning (first byte) of the file. The value of the offset is just what the pointer consists.

In other words, the goal of data indexing is to ease the search of and access to data at any given time. This is done by creating a data structure called index and providing faster access to the data. Accessing data is determined by the physical storage device being used. Indexing could potentially provide large increases in performance for large-scale analysis of unstructured data. Additionally the implementation of the chosen index must be suitable in terms of index construction time and storage utilization [Faye et al, 2012].

Indexing needs resources: memory for storing additional information and time for processing, which may be quite a long, especially for updating of the indexes when new elements are added or some old ones are removed.

➢ *Naming the addresses*

Basic element of an index is couple: (name, address).

For instance such couples on Figure 1 are:

(“Address, of memory cell”, 27) (“Address polynomial”, 350)

In different sources the “name” is called “key”, “concept”, etc. The address usually is given by any “number”, “pointer”, “offset”, “location”, etc.

There are two interpretations of the couple (concept, address):

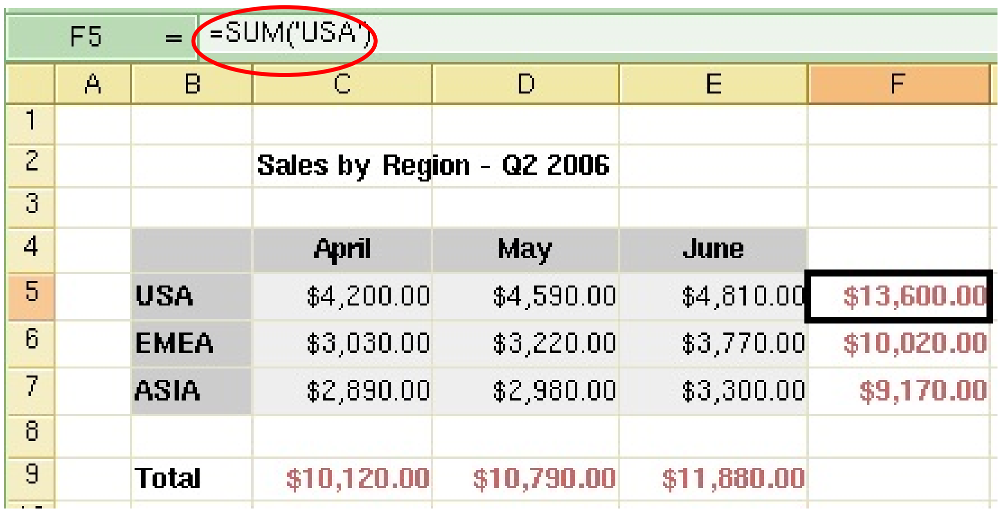
1) The address is *a connection of the concept with its definition in the text*, i.e. practically we have triple: ***(name, address, definition)***.

2) The concept is a name of a computer main memory address and may be used for user friendly style of programming and the third part (value) may be variable, i.e. practically we have triple: ***(name, address, value)***.

In the very beginning, replacing the address by name was used in the programming languages for pointing the addresses by names of identificators, like in Algol 60 [Naur, 1963]. (In this case, the address is real memory location but not offset in a file in the memory. We will not discuss all kinds of addressing in the computer memory used in processor’s registers. In this text, the address is relative offset from the first byte of a file, sometimes given together with information for file location (path to the) file.)

Later, the same idea was used in the Web navigation systems. Web navigation is mostly based on Uniform Resource Locators (URLs). URLs can be hard to remember and change constantly. For instance, in the International Human-Friendly Web Navigation System, the RealNames' Internet Keywords offer an alternative Web addressing scheme using natural language, replacing unfriendly URLs like http://www.fordvehicles.com/vehiclehome.asp?vid=12 with common names such as "Ford Mustang". Building a fully international system that provides a human-friendly naming infrastructure for the whole Web is a challenging task. By leveraging Unicode to represent names it is possible to build a global naming engine that, coupled with knowledge of local customs simplifies Web navigation through the use of natural language keywords [Arrouse, 1999].

Some of the electronic spreadsheets have possibility to point a group of cells and/or rows with any name and further to use this name in functions and other operations assuming all cells and/or rows named by this name [Zoho sheet, 2012] (Figure 4).



F5 = =SUM('USA')

	A	B	C	D	E	F
1						
2			Sales by Region - Q2 2006			
3						
4			April	May	June	
5		USA	$4,200.00	$4,590.00	$4,810.00	$13,600.00
6		EMEA	$3,030.00	$3,220.00	$3,770.00	$10,020.00
7		ASIA	$2,890.00	$2,980.00	$3,300.00	$9,170.00
8						
9		Total	$10,120.00	$10,790.00	$11,880.00	

Figure 4. Natural Language Addressing in a spreadsheet

For instance, Zoho Sheet can recognize and correlate names used in formulas with cells/cell ranges automatically. You have to just give the row/column header of a table as arguments to functions and Zoho Sheet will auto-recognize the cell range associated with the name. It is very convenient to quickly type in the formulas with these names instead of worrying about keying in the proper cell range.

Consider the sheet on Figure 4, available at http://zohosheet.com/public.do?fid=25835.

Look at the formulas in the cells F5:F7 and C9:E9. The formula =SUM (USA) will automatically add the cell values in the row with the header ‘USA’. Earlier you had to use =SUM (C5:E5). Now the row header can directly be used. You do not even need to name/label the cell ranges. You can even copy and paste these formulas to adjacent rows or columns and they will automatically be adjusted relatively. In this case, copying the F5 cell and pasting it to F6, will result in the formula =SUM (EMEA) in F6. (Here the concept “address” is used as cell co-ordinates in the table

(C5, C9, E5, F6, etc.) but not as offset in a file.)

The approach of replacing cell addresses with names in Zoho Sheet was called “***Natural Language Addressing***”.

➢ *Using encoding of the name both as address and as route*

In this research we follow a proposition of Krassimir Markov to use the computer encoding of name (concept) letters as logical address of connected to it information. This way no indexes are needed and high speed direct access to the text elements is available. It is similar to the natural order addressing in a dictionary shown at Figure 2 where no explicit index is used but the concept by itself locates the definition. Our approach is similar to one in the Zoho Sheet, too.

Because of this we will use the same term: “***Natural Language Addressing***”.

Shoch's definition [Shoch, 1978a] failed to capture that addresses are names too and names must eventually be mapped to routes [Jording & Andreasen, 1994]. In this sense, the idea of Natural Language Addressing (NLA) is to use encoding of the name *both as relative address and as route in a multi-dimensional information space* and this way to speed the access to stored information.

For instance, let have the next definition:

*“**London:** The capital city of England and the United Kingdom, and the largest city, urban zone and metropolitan area in the United Kingdom, and the European Union by most measures”.*

In the computer memory, for example, it may be stored in a file at relative address “00084920” and the index couple is: (“London”, “00084920”)

At the memory address “00084920” the main text, “*The capital ... measures*.” will be stored.

To read/write the main text, firstly we need to find name “*London*” in the index and after that to access memory address “00084920” to read/write the definition.

If we assume that name “London” in the computer memory is encoded by six numbers (letter codes), for instance by using ASCII encoding system London is encoded as (76, 111, 110, 100, 111, 110), than we may use these codes for direct address to memory, i.e.

(“London”, “76, 111, 110, 100, 111, 110”)

Above we have written two times the same name and this is truth. Because of this we may omit this couple and index, and read/write directly to the address “76, 111, 110, 100, 111, 110”.

For human this address will be shown as “London”, but for the computer it will be “76, 111, 110, 100, 111, 110”.

Now, what we need is a tool for storing and accessing information using Natural Language Addressing. At first glance, such tool may be the hash tables.

➢ *Hashing and natural language addressing*

The array “76, 111, 110, 100, 111, 110” may be assumed as an offset, i.e. as number “076111110100111110”. This causes two main problems:

- We need a hypothetic file with unlimited length;

- The offset points to only one byte but the definition is 170 bytes long and will occupy the next addresses.

A possible solution is using hash tables.

Hash tables are used in a wide variety of applications. In networking systems, they are used for a number of purposes, including: load balancing, intrusion detection, TCP/IP state management, and IP address lookups. Hash tables are often attractive since sparse tables result in constant-time, O(1), query, insert and delete operations. However, as the table occupancy, or load, increases, collisions will occur which in turn places greater pressure on the collision resolution policy and often dramatically increases the cost of the primitive operations. In fact, as the load increases, the average query time increases steadily and very large worst case query times become more likely [Kumar & Crowley, 2005].

This means that we could not use encoding of names as keys for hash tables or direct offsets. We need a special organization of file internal structure and function that will transform the name in a unique location in the file where the definition will be stored without collisions with other texts.

This problem is already solved at the level of file system – every file has its own name and file system converts it (using file allocation table – FAT) in an address of the file's first block on the hard disk. This is convenient for information which is relatively long – whole documents, images, music files, etc. because every file occupies at least one cluster (2KB, 4KB or more hard disk space). We have to extend this idea to be used into the file. For this purpose we have to establish special kind of file internal organization with additional specialized indexing.

The idea presented in this work differs from the hashing by two characteristics:

- The function which juxtapose the letters to integer numbers is one-one mapping and this way no collisions exist;
- This mapping (hash function) may be used recursively for every symbol of a string to build hierarchical multi-layer set of hash tables and this way to speed the access to information.

For instance, the array "76, 111, 110, 100, 111, 110" may be assumed as a route to (co-ordinates of) a point in a multi-dimensional (in this case: six-dimensional) information space and the definition may be stored in this point.

In other words, our function may be used recursively for every symbol and this way we will create hierarchical multi-layer set of tables. For the case of word "London" we will have six layers.

The natural language does not contain words only of six letters long. The length of the words is variable and in addition there exist names as phrases like "Address polynomial" above. The set of all natural words and phrases defines a multi-dimensional logical address space with variable dimensions and unlimited size.

What we need are:

- A special algorithm which converts such multi-dimensional addresses in concrete routes to linear (relative) locations in the files (on the hard disk, for example);
- A program tool which will realize this algorithm.

A solution of this problem is presented in this monograph.

Brief overview of the content

Chapter 1. *Firstly in this chapter, we will remember the needed basic mathematical concepts. Special attention will be paid to the Names Sets – mathematical structure which is used further for building models needed for our research. We will use strong hierarchies of named sets to create a specialized mathematical model for new kind of organization of information bases called "Multi-Domain Information Model" (MDIM). The "information spaces" defined in the model are kind of strong hierarchies of enumerations (named sets).*

At the end, we will remember the main features of hashing and types of hash tables as well as the idea of "Dynamic perfect hashing" and "Trie", especially – the "Burst trie". Hash tables and tries give very good starting point. The main problem is that they are designed as structures in the main memory which has limited size, especially in small desktop and laptop computers.

Chapter 2 *presents state of the art in the storing models.*

This chapter is aimed to introduce the main data structures and storing technologies which we will use to compare our results. Mainly they are graph data models as well as Resource Description Framework (RDF) storage and retrieval technologies.

Firstly we shortly define concepts of storage model and data model.

Mapping of the data models to storage models is based on program tools called "access methods". Their main characteristics will be outlined.

During the eighties of the last century, the total growing of the research and developments in the computers' field, especially in image processing, data mining and mobile support cause impetuous progress of establishing convenient "spatial information structures" and "spatial-temporal information structures" and corresponding access methods. Important cases of spatial representation of information are Graph models. Because of this, Graph models and databases will be discussed more deeply and examples of different graph database models will be presented. The need to manage information with graph-like nature especially in RDF-databases has reestablished the relevance of this area.

In accordance with this, the analyses of RDF databases as well as of the storage and retrieval technologies for RDF structures will be in the center of our attention. Storing models for several popular ontologies and summary of main types of storing models for ontologies and, in particular, RDF data, will be discussed.

Our attention will be paid to addressing and naming (labeling) in graphs with regards to introducing the Natural Language Addressing (NL-addressing) in graphs. A sample graph will be analyzed to find its proper representation.

Taking in account the interrelations between nodes and edges, we will see that a "multi-layer" representation is possible and the identifiers of nodes and edges can be avoided. As result of the analysis of the example, the advantages and disadvantages of the multi-layer representation of graphs will be outlined.

For practical implementation of NLA we need a proper model for database organization and corresponded specialized tools. To achieve such possibilities, we will use "Multi-Domain Information Model" (MDIM) and corresponded to it software tools to realize dynamic perfect hashing and burst tries as external memory structures.

***Chapter 3** introduces an Access method based on NL-addressing.*

This chapter is aimed to introduce a new access method based on the idea of NL-addressing.

For practical implementation of NLA we need a proper model for database organization and corresponded specialized tools. Hash tables and tries give very good starting point. The main problem is that they are designed as structures in the main memory which has limited size, especially in small desktop and laptop computers. Because of this we need analogous disk oriented database organization.

To achieve such possibilities, we decided to use "Multi-Domain Information Model" (MDIM) and corresponded to it software tools. MDIM and its realizations are not ready to support NL-addressing. We will upgrade them for ensuring the features of NL-addressing via new access method called NL-ArM.

The program realization of NL-ArM, based on specialized hash functions and two main functions for supporting the NL-addressing, access will be outlined. In addition, several operations aimed to serve the work with thesauruses and ontologies as well as work with graphs, will be presented.

***Chapter 4** is aimed to outline two basic experiments.*

In this chapter we will present two types "clear" experiments: with a text file and a relational database. The reason is that they are wide used for storing of semi-structured data.

***Chapter 5** contains description of experiments for NL-storing of small datasets.*

In this chapter we will present several experiments aimed to show the possibilities of NL-addressing to be used for NL-storing of small size datasets which contain up to one hundred thousands of instances.

The goal of the experiments with different small datasets, like dictionary, thesaurus or ontology, is to discover regularities in the NL-addressing realization. More concretely, two regularities of time for storing by using NL-addressing will be examined:

— *It depends on number of elements in the instances;*

— *It not depends on number of instances in datasets.*

This chapter starts with introduction of the idea of knowledge representation. Further in the chapter three experiments with small size datasets are outlined: for NL-storing of dictionaries, thesauruses, and ontologies. Presentation of every experiment starts with introductory part aimed to give working definition and to outline state of the art in storing concrete structures.

We start with analyzing the easiest one: NL-storing dictionaries. After that, NL-storing of thesauruses will be analyzed. An experiment with WordNet thesaurus and program WordArM based on NL-addressing will be discussed.

At the end, a special attention will be given to NL-storing ontologies. This part of the chapter begins with introducing the basic ontological structures as well as the corresponded operations and tools for operating with ontologies. Further, NL-storing models for ontologies will be discussed and experiments with OntoArM program for storing ontologies based on NL-addressing will be outlined.

***Chapter 6** grounds on analyzing experiments for NL-storing middle-size and large RDF-datasets.*

In this chapter we will present results from series of experiments which are needed to estimate the storing time of NL-addressing for middle-size and large RDF-datasets.

The experiments for NL-storing of middle-size and large RDF-datasets are aimed to estimate possible further development of NL-ArM. We assume that its "software growth" will be done in the same grade as one of the known systems like Virtuoso, Jena, and Sesame. We will analyze what will be the place of NL-ArM in this environment. Our hypothesis is that NL-addressing will have good performance.

Chapter will start with describing the experimental storing models and algorithm used in this research. Further an estimation of experimental systems will be provided to make different configurations comparable. Special proportionality constants for hardware and software will be proposed. Using proportionality constants, experiments with middle-size and large datasets became comparable.

Experiments will be provided with both real and artificial datasets. Experimental results will be systematized in corresponded tables. For easy reading visualization by histograms will be given.

***Chapter** 7 contains analysis of experiments.*

In this chapter we will analyze experiments presented in previous chapters 4, 5, and 6, which contain respectively results from (1) basic experiments; (2) experiments with structured datasets; (3) experiments with semi-structured datasets. Special attention will be paid to analyzing of storing times of NL-ArM access method and its possibilities for multi-processing.

*In **Chapter 8** practical aspects will be discussed.*

Some practical aspects of implementation and using of NL-addressing will be discussed in this chapter.

NL-addressing is approach for building a kind of so called "post-relational databases". In accordance with this the transition to non-relational data models will be outlined.

The implementation has to be done following corresponded methodologies for building and using of ontologies. Such known methodology will be discussed in the chapter. It is called "METHONTOLOGY" and guides in how to carry out the whole ontology development through the specification, the conceptualization, the formalization, the implementation and the maintenance of the ontology.

Special case is creating of ontologies of text documents which are based on domain ontologies. It consists of Document annotation and Ontology population which we will illustrate following the OntoPop platform [Amardeilh, 2006].

The software realized in this research was practically tested as a part of an instrumental system for automated construction of ontologies "ICON" ("Instrumental Complex for Ontology designatioN") which is under development in the Institute of Cybernetics "V.M.Glushkov" of NAS of Ukraine.

In this chapter we briefly will present ICON and its structure. Attention will be paid to the storing of internal information resources of ICON realized on the base of NL-addressing and experimental programs WordArM and OntoArM.

***Conclusion** contains a short presentation of the next steps. Special attention is done on the area of so called "Big Data" and possible implementation of NLA for processing of large semi-structured data sets. A brief outline of the main achievements of this work is given.*

***Appendix A** outlines the program realizations of tools for storing information using NL-addressing: WordArM, OntoArM, and RDFArM for storing thesauruses, small ontologies, and large RDF triple datasets. Some illustrative tables and figures from experiments as well as other supporting information are given.*

***Appendix B** contains short information about tools analyzed in the monograph. Main attention will be paid to Protégé 4.2 and SPARQL. Storage characteristics of analyzed RDF triple stores will be presented shortly in two groups: (1) DBMS based approaches, (2) Multiple indexing frameworks.*

The monograph was written by:

- Krassimir Markov: Introduction, Ch. 2, 4. 7, Coclusion, and Appendix B2-B3 (125 pp.);

- Krassimira Ivanova: Ch. 1, 3, 5, 6, 8.1-8.3, and Appendix A1-A5 (153 pp.);

- Vitalii Velychko: Chapter 8.4, Appendix A6 - A8, and Appendix B1 (27 pp.)

- Koen Vanhoof and Juan Castellanos: consulting and editing.