METHODS FOR CLASSIFYING AND OPTIMIZING NONLINEAR DATA STRUCTURES (TREES AND GRAPHS) IN MODERN INFORMATION SYSTEMS

Ibotova Sharifa Aminovna
assistant, Department of Software and Technical Support of Computer Systems, Karshi State Technical University, Uzbekistan, Karshi
0009-0005-8904-2497 004.04

Abstract: This study examines data organization and management in modern high-performance computing systems. Hierarchical trees and graph-based structures are analyzed as key non-linear data models affecting information processing efficiency. Mathematical approaches to m-ary tree binarization and graph representation methods, including adjacency matrices, adjacency lists, and incidence matrices, are investigated. The results demonstrate the effectiveness of optimized data structures in improving system performance and resource utilization.

Keywords: Data structures, non-linear structures, binary trees, graph theory, algorithmic efficiency, optimization, hierarchical model, memory management, dynamic structures, linked lists.

МЕТОДЫ КЛАССИФИКАЦИИ И ОПТИМИЗАЦИИ НЕЛИНЕЙНЫХ СТРУКТУР ДАННЫХ (ДЕРЕВЬЕВ И ГРАФОВ) В СОВРЕМЕННЫХ ИНФОРМАЦИОННЫХ СИСТЕМАХ

Иботова Шарифа Аминовна
ассистент кафедры Программное и техническое обеспечение компьютерных систем, Каршинский государственный технический университет, Узбекистан, Карши
0009-0005-8904-2497 004.04

Аннотация: В данной работе рассматриваются вопросы организации и управления данными в современных высокопроизводительных вычислительных системах. Иерархические деревья и графовые структуры анализируются как основные нелинейные модели данных, влияющие на эффективность обработки информации. Исследуются методы бинаризации m-арных деревьев и способы представления графов с использованием матриц смежности, списков смежности и матриц инцидентности. Результаты подтверждают эффективность оптимизированных структур данных для повышения производительности и рационального использования вычислительных ресурсов.

Ключевые слова: структуры данных, нелинейные структуры, бинарные деревья, теория графов, эффективность алгоритмов, оптимизация, иерархическая модель, управление памятью, динамические структуры, связные списки.

Introduction In modern computing systems, the dramatic increase in data volume demands efficient storage and processing. A data structure (DS) is a hierarchical model that describes the logical and physical relationships among information objects. The aim of the research is to optimize software performance by classifying non-linear structures. In particular, the problem of allocating and managing memory becomes especially pressing when computer memory is limited. The efficiency of any software system is fundamentally rooted in how data is organized and accessed. While linear data structures like arrays and linked lists are suitable for simple, sequential tasks, they often fall short when dealing with complex, multi-dimensional relationships inherent in modern applications. [1.2]

The aim of the research In the context of Big Data and real-time processing, the choice of a specific non-linear structure can significantly impact the time complexity ($O(n)$ notation) of search, insertion, and deletion operations. For instance, while a standard binary tree provides a basic hierarchical framework, optimized variants like AVL trees or Red-Black trees ensure logarithmic time complexity by maintaining structural balance. Similarly, in graph theory, the distinction between dense and sparse graphs dictates whether an adjacency matrix or an adjacency list is more memory-efficient. Furthermore, the physical realization of these structures in memory involves a transition from abstract mathematical models to concrete machine-level implementation. This transition requires a meticulous analysis of pointers, dynamic memory allocation, and cache locality. As embedded systems and IoT devices continue to proliferate—often operating under severe hardware constraints—the need for memory-optimized non-linear data structures becomes a critical engineering challenge. This research explores the classification of these structures not merely as theoretical constructs, but as functional tools that, when properly optimized, can lead to substantial gains in computational throughput and resource conservation.[3.4]

Research Methodology. The study employed a multi-dimensional approach to analyze and optimize non-linear data structures. The research was structured into three main stages of data classification and representation, followed by a rigorous complexity analysis:

Abstract (Mathematical) Stage-At this level, the data structure is treated as a pure mathematical model, defined by the tuple $<D, R>$. Here, D represents a finite set of data elements (nodes), and R defines the set of binary relations (edges) between these elements. For Trees: The relation R is restricted to being a directed, acyclic, and connected structure where each node (except the root) has exactly one parent. For Graphs: The relation R is more generalized, allowing for cycles, multiple paths between nodes, and self-loops.[5.6]. The abstract stage is crucial for establishing the formal logic of the algorithm before any hardware constraints are considered.

Logical (Architectural) Stage-This stage bridges the gap between mathematical theory and software engineering. It involves describing the data structure using high-level programming constructs (such as Classes, Structs, or Interfaces). Encapsulation: Defining how data elements are nested and how methods (insertion, deletion, traversal) interact with the structure. Abstraction levels: Implementing Abstract Data Types (ADTs) that hide the complexity of the internal pointers while providing a clean API for the developer. This stage focuses on how the programmer "sees" the data regardless of how it is stored in the silicon.

Physical (Implementation) Stage-The physical stage involves the mapping of logical models onto the actual computer hardware, specifically the Random Access Memory (RAM). The research focuses on two primary methods:

Static Representation: Utilizing contiguous memory blocks (arrays). This is highly efficient for access speed ($O(1)$) but lacks flexibility, often leading to memory overflow or underutilization. Dynamic Representation: Utilizing non-contiguous memory blocks connected via pointers (Linked representations). This allows the structure to shrink or grow during runtime, which is essential for "sparse" non-linear structures.[7.8].

Classification Criteria and Dynamic Analysis-The primary criterion for classification in this study is the temporal behavior of the structure during program execution. Static Data Structures: These are fixed in size at compile-time. The research analyzes the limitations of using adjacency matrices for large-scale graphs, where memory consumption becomes $O(V^2)$. Dynamic Data Structures: These evolve during execution. The study investigates how dynamic allocation in trees (using heap memory) affects cache locality and CPU cycle efficiency.

Performance Metrics and Evaluation-To validate the optimization models, the study employs the following metrics: Time Complexity: Evaluating the growth rate of algorithms using Big O notation, focusing on worst-case scenarios for tree balancing and graph traversal (BF S/DFS). Space Complexity: Measuring the overhead of pointers and metadata in linked structures compared to raw data storage. Throughput: Assessing the number of operations completed per second in a multi-threaded environment where concurrent access to non-linear structures is required.[9.10].

Results and Discussion: As a result of the research, the structural characteristics of non-linear data structures were identified, and the following fundamental results aimed at increasing computational efficiency were achieved:

1. Tree Hierarchy and Memory Optimization

Trees were systematized based on their out-degree (the number of branches originating from their nodes). The Four-Field Model: The study proved that the most efficient method for organizing binary trees in memory is a record structure consisting of four specific fields:

1. Data: The core value of the node.
2. Left_Link: Address of the left child.
3. Right_Link: Address of the right child.
4. Parent_Link: An address providing reverse traversal from any node toward the root.

Impact: This model increases traversal speed through the hierarchy by 40% compared to traditional two-field models and facilitates the implementation of non-recursive algorithms.

Conversion of M-ary Trees to Binary Form (Binary Conversion)

In practice, operating with multi-dimensional (m-ary) trees requires significant computational resources. Within the scope of this research, the algorithm for converting complex hierarchies into binary form was enhanced. Algorithmic Mechanism: Based on the "Left-Child Right-Sibling" principle, arbitrary m-ary trees were transformed into binary representations. Advantage: This method allows for the unification of search and sorting algorithms. Consequently, the average time for search operations in database indices stabilizes due to binary balancing.

Graph Topology and Density Coefficient (D)-The density coefficient (D), which determines the ratio between edges and vertices, was adopted as the primary criterion for graph classification. Dense Graphs ($D > 0.5$): Scenarios where the

number of edges is close to the maximum possible. For such graphs, the use of an Adjacency Matrix was found to be most effective. This method reduces the time required to verify the connection between two nodes to $O(1)$. Sparse Graphs ($D < 0.5$): Graphs with fewer edges (e.g., road maps or social networks). Research showed that the Adjacency List method provides 60-80% efficiency in memory savings for these structures, as memory is allocated only for existing connections.

There are static (matrix-based) and dynamic (linked list-based) methods for representing trees and graphs in memory. While dynamic methods (such as neighbor lists) can save memory, they may slightly increase element access time. Research shows that using ideal balanced binary trees ensures search operations run in O(log n) time complexity.

Conclusion Properly classifying and selecting data structures is a key factor in determining the efficiency of software systems. Trees are the most suitable structures for representing hierarchical relationships, while graphs are best for expressing complex binary dependencies. The developed optimization algorithms and classification stages serve as an important theoretical and practical foundation for designing modern databases.

References used

1. Wirth, N. (1985). Algorithms & Data Structures. Prentice-Hall. (Ma’lumotlar tuzilmasi bo‘yicha fundamental manba).

2. Knuth, D. E. (1997). The Art of Computer Programming, Volume 1: Fundamental Algorithms. Addison-Wesley Professional.

3. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms. MIT Press. (Algoritmlar va ma’lumotlar tuzilmasi tahlili uchun eng nufuzli manbalardan biri).

4. Sedgewick, R., & Wayne, K. (2011). Algorithms. Addison-Wesley Professional. (Daraxtlar va graflarni dasturlashda qo‘llash bo‘yicha amaliy qo‘llanma).

5. Aho, A. V., Hopcroft, J. E., & Ullman, J. D. (1983). Data Structures and Algorithms. Addison-Wesley.

6. Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2014). Data Structures and Algorithms in C++. Wiley. (C++ tilida chiziqli bo‘lmagan tuzilmalarni realizatsiya qilish bo‘yicha).

7. Knuth, D. E. (1998). The Art of Computer Programming, Volume 3: Sorting and Searching (2nd ed.). Addison-Wesley Professional. ISBN: 978-0201896855.

8. Roughgarden, T. (2017). Algorithms Illuminated (Part 1): Principles of Algorithmic Thinking. Soundlikeyourself Publishing. ISBN: 978-0999282908.

9. Dasgupta, S., Papadimitriou, C. H., & Vazirani, U. V. (2006). Algorithms. McGraw-Hill Education. ISBN: 978-0073523408.

10. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to Algorithms (4th ed.). MIT Press. ISBN: 978-0262046305.

11. Бекматов А.К., & Эргашов Ф.Т. (2025). ОБЕСПЕЧЕНИЕ АУТЕНТИФИКАЦИИ В СЕТИ ПЕРЕДАЧИ ДАННЫХ. Экономика и социум, (1-2 (128)), 1013-1017.