DOI: 10.5281/zenodo.5535815

METHODS FOR STORING AND FINDING DATA IN THE BUSINESS LOGIC FOR ECONOMIC APPLICATIONS

Emilia VASILE, PhD Professor
Athenaeum University, Bucharest, Romania
rector@univath.ro

Dănuţ-Octavian SIMION, PhD Associate Professor
Athenaeum University, Bucharest, Romania
danut_so@yahoo.com

Abstract: *The paper presents the methods for storing and finding data in the business logic for economic applications. Along with the definition of the business strategy, it is necessary to define the strategy of the information system and this because the IT system supports the managers, through the information provided, in the management and control activity in order to achieve the strategic objectives of the organization, IT systems are open and flexible, constantly adapting to the imposed requirements the dynamic environment in which the company operates, promoting IT solutions supports the organization in consolidating and developing the business (eg: electronic commerce, e-banking, etc.), the computer system provides the necessary information to control the fulfillment and adaptation of the plans operational and strategic aspects of the organization, the organization must know and control the risks related to the implementation of the new ones technologies and adaptation of the computer system to the new requirements, establishing standards at the level of the information system that are meant to specify the hardware and software features and performance of the components to be purchased and what methodologies are to be used in the development of the system. Storing and finding data is an important action for the IT system in the role to provide fast and secure informations that are used in business logic and flow available to managers for the decision-making process. The methods include complex algorithms and the reason of these is to get fast responses at different types of queries embedded in specific modules of an economic application.*

Keywords: *management and control activity, solutions supports, data structures, data graphs, data trees, economical data, information processes, business flows, application modules*

JEL Classification: *C23, C26, C38, C55, C81, C87*

1. Introduction

The management domains correspond to each of the homogeneous activities carried out within the company - research-development, commercial, production, personnel, financial-accounting - taking into account the interactions between them. Moreover, the approach of these fields is made in a hierarchical vision leading to the identification of the following levels:

– Transactional in which elementary operations are performed;
– Operational where current operations are carried out, the decisions taken at this level are current, by routine;
– Tactical corresponding to control activities and short-term decisions;
– Strategic characteristic of long-term decisions and / or that globally engage the company.

Management information systems are defined in the business world, following two approaches:

a) starting from the information and its support;
b) starting from the function that the management information system must perform.

• in the first case, the management information systems represent the set of information used within the company, of the means and procedures of identification, collection, storage and processing of information (Crysup et al., 2019; Jansson, 2019).

• in the second approach of defining the management information systems, it starts from its purpose, namely to provide the information requested by the user in the desired form and at the appropriate time in order to substantiate decisions.

Management information systems (GIS) involve the definition of: management domains, data, models, management rules.

Data is the „raw material" of any management system. All data transmitted and processed are taken into account regardless of their nature, their formal or informal nature or the media on which they are located (Erciyes, 2021; Harish, 2020).

2. Binary search trees for data in applications

Finding certain information or pieces of information from a large volume of previously stored / stored data is a fundamental operation, called searching, of most computer applications. The data is organized as items or items each with a key that is used in the search. The purpose of the search is to find items that have keys that match the search key.

The purpose of the search is to access the information in the article for processing.

Search - Definition: A symbol table is a data structure that supports two basic operations: inserting a new item and returning an item with a given key. Symbol tables are also called dictionaries by analogy with the secular system of giving definitions of words by listing them alphabetically in a reference book:

- the keys are the words
- articles are records associated with words and contain definition, pronunciation and etymology.

The advantages of symbol tables on the computer:
– have efficient search algorithms,
– efficient insertion operations,
– efficient deletion or modification operations,
– operations of combining 2 tables into one. Indispensable in organizing computer software: keys are symbolic names, and articles contain information that describes the named object (Khadda, 2020; Edappanavar, 2019).

Operations for binary search trees:
– Insert a new item.
– Search for an item / items with a given key.
– Delete a specified item.
– Select the k-th item in a symbol table.
– Sort symbol table (visit all items in order of keys)
– Join two symbol tables

Other operations:
– Initialize
– Test ifempty
– Destroy
– Copy

TAD Symbol table

– Void STinit (int); int STcount (); void STinsert (Item);
– Item STsearch (Key); void STdelete (Item);
– Item STselect (int); void STsort (void (* visit) (Item));
– TAD interface Symbol table

A Binary Search Tree (BST) is a binary tree that has a key associated with each of its internal nodes, with the added property that the key of each node is greater than or equal to the keys in all nodes of its left subtree and more small or equal to the keys in all nodes of its right subtree.

```
#include <stdlib.h>
#include "Item.h"
typedef struct STnode* link;
struct STnode { Item item; link l, r; int N; };
static link head, z;
link NEW(Item item, link l, link r, int N)
{ link x = malloc(sizeof *x);
 x->item = item; x->l = l; x->r = r; x->N = N;
 return x;
}
void STinit()
{ head = (z = NEW(NULLitem, 0, 0, 0)); }
int STcount() { return head->N; }
Item searchR(link h, Key v) //search in BST
{ Key t =  key(h->item);
  if (h == z) return NULLitem;
  if eq(v, t) return h->item;
  if less(v, t) return searchR(h->l, v);
  else return searchR(h->r, v);
}
Item STsearch(Key v)
 { return searchR(head, v); }
link insertR(link h, Item item) //insert in BST
 { Key v = key(item), t = key(h->item);
   if (h == z) return NEW(item, z, z, 1);
   if less(v, t)
         h->l = insertR(h->l, item);
    else h->r = insertR(h->r, item);
    (h->N)++; return h;
  }
 void STinsert(Item item)
 { head = insertR(head, item); }
```

Symbol table implemented with binary search tree

We define the successor of a node x, the node y with the lowest value of the key but with key [y] ≥ key [x] We define the predecessor of a node x, the node y with the highest value of the key but with key [y] - key [x]

```
link rotR(link h)
{
      link x =  h->l; h->l = x->r; x->r = h;    return x;
}
      link rotL(link h)
{
      link x = h->r; h->r x->l; x->l = h;  return x;
}
```

These two routines perform the rotation operation in a BST tree. A right rotation makes the old root the right tree of the new root (the old left subtree of the root); the rotation to the left makes the old root the left subtree of the new root.

– The search requires on average about 2lnN ~ 1.39lnN comparisons in a binary search tree formed with N random keys.
– Unsuccessful insertion and search requires on average about 2lnN ~ 1.39lnN comparisons in a binary search tree formed with N random keys.
– In the worst case, a search in a binary tree of search with N keys requires N comparisons.

By traversing BST

```
void sortR (link h, void (* visit) (Item))

{
if (h == z) return;
 sortR (h-> l, visit);
 visit (h-> item); sortR (h-> r, visit);
 } void STsort (void (* visit) (Item)) {
   sortR (head, visit);
}
```

Sort with binary search tree

3. Methods for traversing Graphs that contain data

Graphs are useful for modeling various problems and are implemented in multiple practical applications:

- Computer networks
- Web pages
- Social networks
- Road maps
- Graphic modeling

The graph can be modeled as a pair of sets G = (V, E). The set V contains the vertices, and the set E contains the edges, each edge establishing a neighborhood relationship between two nodes. A wide variety of problems are modeled using graphs, and solving them involves exploring space. A traversal aims to discuss each node of the graph, exactly once, starting from a chosen node, hereinafter called the source node.

The memory representation of graphs is usually done with adjacent lists or adjacent matrices. However, other data structures can be used, for example a pair map << source, destination>, cost>.

During the running of the traversal algorithms, a node can have 3 colors:

- White = undiscovered
- Gray = has been discovered and is being processed
- Black = was processed

An analogy can be made with a black spot that extends over a white space. The gray knots are on the border of the black spot. Scrolling algorithms can be characterized by completeness and optimality. A complete exploration algorithm will always find a solution, if the problem accepts the solution. An optimal exploration algorithm will discover the optimal solution to the problem from the perspective of the number of steps to be performed (Khuller, 2021; Watanobe et al., 2020).

Breadth-first Search (BFS) is a graphical search algorithm in which, when reaching an unvisited node v, all unvisited nodes adjacent to v are visited, then all unvisited peaks adjacent to the peaks. adjacent to v, etc.

BFS depends on the start node. Starting from a node, only the connected component of which it is part will be traversed. For graphs with several connected components, several covering trees will be obtained.

Following the application of the BFS algorithm on each connected component of the graph, a coverage tree is obtained (by eliminating the edges that we do not use when traversing). In order to be able to reconstruct this tree, the identity of its parent is kept for each given node. If there is no cost function associated with the edges, BFS will also determine the minimum paths from the root to any node (Crysup et al., 2019; Harish, 2020).

A queue is used to implement BFS. When added to the queue, a knot should be colored gray (it has been discovered and is to be processed).

The BFS exploration algorithm is complete and optimal.

Algorithm:

```
BFS (s, G) {
foreach (u in V) {
p (u) = null; // initialization
dist (s, u) = inf;
c (u) = white;
}
dist (s) = 0; // the distance to the source is 0
c (s) = gray; // we start processing the node, so the color
turns gray
Q = (); // use a queue with the nodes to be processed
Q = Q + s; // add the source to the queue
while (! empty (Q)) {// how long do I have nodes to process
u = top (Q); // determine the node at the top of the queue
foreach v in succs (u) {// for all neighbors
if (c (v) = white) {// node not found, not in queue
```

```
// update the data structure
dist (v) = dist (u) + 1;
p (v) = u;
c (v) = gray;
Q = Q + v;
} // close if
      } // close foreach
      c (u) = black; // I finished processing the current node
      Q = Q - u; // the node is removed from the queue
      } // close while
      }
```

Complexity:

- with list: O (| E | + | V |)
- with matrix: O (| V | 2)

Depth-First Search (DFS) starts from a given node (start node), which is marked as being processed. The first unvisited neighbor of this node is chosen, it is also marked as being processed, then the first unvisited neighbor is also searched for this neighbor, and so on. When the current node no longer has unvisited neighbors, it is marked as already processed and returns to the previous node. The first unvisited neighbor is searched for this node. The algorithm repeats until all nodes of the graph have been processed (Erciyes, 2021; Edappanavar, 2019).

Following the application of the DFS algorithm on each connected component of the graph, a covering shaft is obtained for each of them (by eliminating the edges that we do not use when traversing). In order to be able to reconstruct this tree, we keep the identity of its parent for each given node. For each node we will remember:

- time of discovery
- completion time
- the parent
- color

The DFS exploration algorithm is neither complete (in case of a search on an infinite subtree), nor optimal (it does not find the node with the minimum depth) (Khadda, 2020; Watanobe et al., 2020). Unlike BFS, a stack (LIFO approach instead of FIFO) is used to implement DFS. Although this replacement can be made in the above algorithm, it is often more intuitive to use recusivity.

Algorithm:

```
DFS (G) {
V = nodes (G)
```

```
foreach (u in V) {
// initialize the data structure
c (u) = white;
p (u) = null;
}
time = 0; // keep the distance from the root to the current node
foreach (u in V)
if (c (u) = white) explore (u); // explore the node
}
explore {u
d (u) = time ++; // node discovery time u
c (u) = gray; // node being explored
foreach (v in success (u)) // I try to process the neighbors
if (c (v) = white) {// if they have not already been processed
p (v) = u;
exploration (v);
}
c (u) = black; // I finished processing the current node
f (u) = time ++; // node completion time u
}
```

Complexity:

- with list: O (| E | + | V |)
- with matrix: O (| V | 2)

Giving an acyclic oriented graph, the topological sorting achieves a linear arrangement of the nodes according to the edges between them. The orientation of the edges corresponds to an order relation from the source node to the destination node. Thus, if (u, v) is one of the edges of the graph, u must appear before v in a row. If the graph were cyclic, there could be no such sequence (no order can be established between the nodes that make up a cycle).

Topological sorting can also be seen as the placement of nodes along a horizontal line so that all edges are directed from left to right (Harish, 2020; Khadda, 2020).

(a) Each muche (u, v) means that the garment u must be dressed before the garment v. The discovery times d (u) and completion times f (u) obtained after traversing the DFS are noted next to the nodes.

(b) The same graph, sorted topologically. Its nodes are arranged from left to right in descending order of f (u). Notice that all edges are oriented from left to right. Now Trudy can dress quietly.

Algorithm:

There are two known algorithms for topological sorting.

DFS-based algorithm:

- DFS traversal to determine times
- descending sorting depending on the completion time

Another algorithm is the one described by the following example:

```
SortM1 (G) {
V = nodes (G)
L = empty; // the list that will contain the sorted items
// initialize S with nodes that have no edges
foreach (u -> V)
if (u has no edges)
   S = S + u;
}
while (! empty (S)) {// how long do I have nodes to process
u = random (S); // remove a node from the set S
L = L + u; // add U to the final list
foreach v -> succs (u) {// for all neighbors
  delete u-v; // delete muchia u-v
  if (v has no edges)
   S = S + v; // add v to the set S
} // close foreach
} // close while
if (G has edges)
  print (error); // cyclic graph
else
  print (L); // topological order
}
```

Optimal complexity: O (| E | + | V |)

Graphs are very important for representing and solving a multitude of problems. The most common ways to represent a graph are:

- adjacency lists
- adjacency matrix

The two usual ways to go through an uninformed graph are:

- BFS - width traversal
- DFS - deep traversal

Topological sorting is a way of arranging nodes according to the edges between them. Depending on the starting node of DFS, different sorts can be obtained, but keeping the general properties of the topological sort.

4. Conclusions

Operational IT systems processes data generated and used in business operations. Depending on the role they have, there are several categories: transaction processing systems - record and process data resulting from transactions, update databases and produce a variety of documents and reports; process control systems - provide operational decisions that control physical

processes; automated service systems - those that support communications (Khuller, 2021; Harish, 2020). Economic automated systems have always been necessary for the processing of data generated and used in business operations. Operational systems produce a variety of information, but they (information) do not highlight which information products are best for managers. For this reason, further processing by computer systems is required (Khadda, 2020; Jansson, 2019). The complexity of data needs advanced algorithms to get economic indicators that managers can use in the process of making decisions. Graphs offer an optimal way to get the stored data using methods that implement the business logic that is used in economical processes that are vital for companies, such as supplies, orders, sales, processing, customer relations and other actions that are specific.

References

Crysup, B., Woerner, A. E., King, J. L., Budowle, B. (2019). Graph Algorithms for Mixture Interpretation. *MDPI Journal, Genes.* 12, 185.

Erciyes, K. (2021). *Distributed Graph Algorithms for Computer Networks.* Springer Science Publishing.

Harish, P., Narayanan, P.J. (2020). Accelerating Large Graph Algorithms on the GPU Using CUDA. Springer, *International Conference on High-Performance Computing.*

Jansson, J. (2019). Special Issue on Graph Algorithms. *MDPI Journal, Algorithms.*

Khuller, S., Raghavachari, B. (2021). Graph and network algorithms, *ACM Computing Surveys.*

Watanobe, Y., Mirenkov, N.N., Yoshioka, R., Monakhov, O. (2020). Filmification of methods: A visual language for graph algorithms. *Journal of Visual Languages & Computing,* Elsevier.