DOMAIN SPECIFIC LANGUAGE - Data Structure Visualization (DSV)

Eugeniu CHETRAR[1*], Ana DVORAC[1], Artiom GHERMAN[1]

[1]*Technical University of Moldova, Faculty of Computers, Informatics and Microelectronics, Department of Software Engineering and Automatics, group FAF-193, Chisinau, Moldova*

*Corresponding author: Chetrar Eugeniu, chetrar.eugeniu@isa.utm.md

Abstract: *This language was created in order to help people with visualization of the data structure. For example, a teacher wants to draw a binary tree to show it for students. Mostly, will spend a lot of time in photo editors, in this case, teachers, using Data Structure Visualization (DSV) will write some lines of code and get the result and will take less than one minute.*

Keywords*: data structure, graphs, binary tree, stack, queue.*

Introduction

DSV is resolving specific problems. It resolves the problem with representation of the data structures: binary tree, graph, queue, stack. In some cases, if user wants to draw a graph, in most cases, user should open Photoshop and is wasting time in order to represent it. For instance, it was created a test, where was opened Photoshop and tried to visualize binary tree and it took more than 7 minutes. This language will help user to waste less than 2 minutes.

As a base language DSV is using C++, there have been some searching for similar realization and have been found d3.js library for JavaScript. Mostly it is used for diagrams, but it can be used to draw graph and tree, but another data structures it doesn't support. As well, there is an app Algorithm Visualizer, with some of data structures, but it has only description and a code example, without visualization. This language is user-friendly, it means that it won't be a big deal to represent data structure or to create it [1].

Data Structure

Binary search tree, also called an ordered or sorted binary tree, is a rooted binary tree whose internal nodes each store a key greater than all the keys in the node's left subtree and less than those in its right subtree [2]. A binary tree is a type of data structure for storing data such as numbers in an organized way. Binary search trees allow binary search for fast lookup, addition and removal of data items, and can be used to implement dynamic sets and lookup tables. The order of nodes in a BST means that each comparison skips about half of the remaining tree, so the whole lookup takes time proportional to the binary logarithm of the number of items stored in the tree.

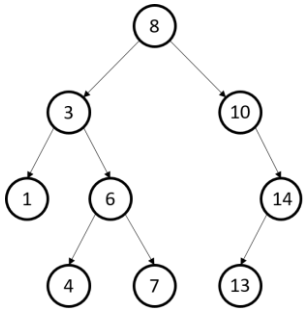

Figure 1. Binary Tree

Graph is a structure amounting to a set of objects in which some pairs of the objects are in some sense "related" [3]. The objects correspond to mathematical abstractions called vertices (also called nodes or points) and each of the related pairs of vertices is called an edge (also called link or line).

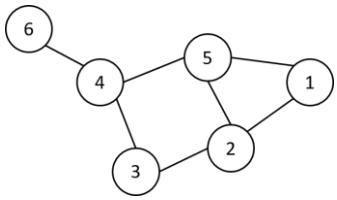

Figure 2. Graph

Stack is an abstract data type that serves as a collection of elements, with two main principal operations: push, which adds an element to the collection, and Pop, which removes the most recently added element that was not yet removed [4]. The order in which elements come off a stack gives rise to its alternative name, LIFO (last in, first out).

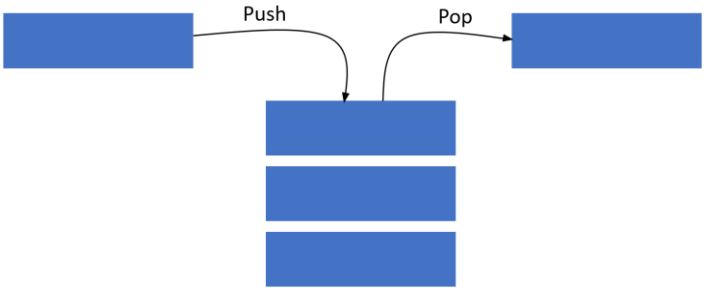

Figure 3. Stack

Queue is a collection of entities that are maintained in a sequence and can be modified by the addition of entities at one end of the sequence and the removal of entities from the other end of the sequence [5]. By convention, the end of the sequence at which elements are added is called the back, tail, or rear of the queue, and the end at which elements are removed is called the head or front of the queue, analogously to the words used when people line up to wait for goods or services.

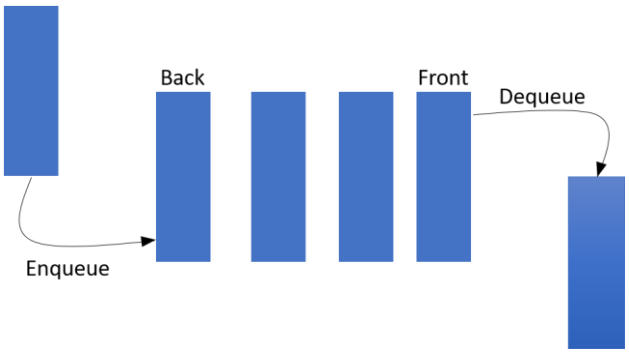

Figure 4. Queue

Grammar representation

All keywords are snakecase (create_graph), each command should include opened and closed bracket and semicolon. For example, draw(); The reserved words are: create, draw, pop, push, front, add, node, delete, tree, inorder, preorder, postorder, find, back, empty, top. This commands are connected with other methods from different libraries, such as SFML, STL.

Reference Grammar

Vn = {add_node, delete_node, delete_tree, inorder, preorder, postorder, find, push, pop, back, front, empty}

Vt = {start, commands, end, _, graph, binaryTree, stack, queue, (, 0,1,2.,9,a,b…z,A,B…,Z,), ;}

Table 1

Meta-notation

x*	means 0 or more occurrences of x
<foo>	means foo is a nonterminal
foo	means is a terminal
,	separates alternativities

P = {

<program> → <start><commands>*<end>

<start>→<start_command><separate><data_structures><brackets><semicolon>

<start_command> → **create**

<data_structures> → **graph|binaryTree|stack|queue**

<commands>→**add_node**<brackets>|**push**<brackets>|**pop**<brackets>|**front**<brackets>|**empty**<brackets>|**top**<brackets>|**back**<brackets>|**inorder**<brackets>|**postorder**<brackets>|**preorder**<brackets>

<end> → <end_command><brackets><semicolon>

<end_command> → **draw**

<brackets> → **(**<variable*>,<variable>*,<variable>***)**

<variable> → **0,1,2.,9,a,b…z,A,B…,Z**

<semicolon> → '**;**'

<separate> → '**_**'

}

Semantic Rules

- Everything under method draw() won't be included.
- Method draw() must be under method create_<data_structure>.
- Method without semicolon, brackets won't be considered as a correct input.
- Method create_<data_structure> must be one and in the beginning.
- For each data structure exist their own methods.

A derivation

<program> → <start><commands><end>
→ < start_command><separate><data_structures><brackets><semicolon><commands><end>
→ **create**<separate><data_ structures><brackets><semicolon><commands><end>
→ **create_**<data_ structures><brackets><semicolon><commands><end>
→ **create_graph**<brackets><semicolon><commands><end>
→ **create_graph()**<semicolon><commands><end>
→ **create_graph();**<commands><end>
→ **create_graph();add_node**<brackets><end>
→ **create_graph();add_node**(<variables>*)<end>
→ **create_graph();add_node(2)<**end>
→ **create_graph();add_node(2)**<end_command><brackets><semicolon>
→ **create_graph();add_node(2)draw**<brackets><semicolon>
→ **create_graph();add_node(2)draw**(<variables>*)<semicolon>
→ **create_graph();add_node(2)draw()<**semicolon>
→ **create_graph();add_node(2)draw();**

Conclusion

In this scientist work was shown Data Structure Visualization language, which was invented in order to help people with faster and easier data structure representing. C++ was chosen as a based language and SFML as a graphic library. In the future it is planning to add algorithms, right now it is planning to realize: Ford-Fulkerson algorithm for oriented and weight graph, binary tree traversals.

References

1. AHO A.V., LAM M.S., SETHI R and ULLMAN J. *Compilers: Principles, Techniques, and Tools (2nd edition)*. Publisher: Addison Wesley, 2007
2. Geeks for geeks, Binary Tree [online]. [27.02.2021] https://www.geeksforgeeks.org/binary-tree-data-structure/
3. Geeks for geeks, Graph [online]. [27.02.2021] https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/
4. Geek for geeks, Stack [online]. [27.02.2021] https://www.geeksforgeeks.org/stack-data-structure/
5. Geek for geeks, Queue [online]. [27.02.2021] https://www.geeksforgeeks.org/queue-data-structure/