Product Requirements Description

By: Anahit Doshoyan & Elizabeth Vickery

Version 1.02

1. Introduction

1.1 Document Identifier

This is a product requirement document. It summarizes and includes the product requirements description (PRD) in regards to our Data Structures and Algorithms project. This document also encompasses the scope of our product, a glossary including terms and acronyms included in the PRD, functional, platform, implementation, performance, verification, and documentation requirements. It also introduces the means of product and project management, the dependencies, assumptions, and risks associated with the implementation, the effort estimations and schedule of deliverables, and acceptance criteria.

1.2 Scope

Introduction section provides overview of the document content as well as the brief description of the project requirements. Section Definitions of terms and acronyms includes description of all the terms and acronyms used in the document. Section References has links to the documentation and sources used to build this document. Overview Section briefly describes the requirements and purpose of the project. All the requirements are described in subsections of Requirements section. The features that the project should support are described in section Functional Requirements. Platform Requirements specifies all the hardware and software requirements of the project, including hardware models and operating system versions. Implementation Requirements section includes requirements on the tools, libraries, applications, programming languages. Performance Requirements section contains information of the performance of the developed product, in particular run- time of the applications developed for the project. The methods used to verify the project are specified in Verification Requirements section. Documentation Requirements project lists all the documents that should be created for the project. Project Management section includes Error: Reference source not found, Dependencies, Assumptions, Risks and Schedule and

Effort Estimations sections which describe the project state and estimations with consideration of the dependencies and risks.

- Project goal:
  - Using and getting acquainted with data structures within the boundaries of our Data Structures and Algorithms course
  - Using a self-balancing binary search tree to encode and decode Morse Code

- Deliverables:
  - A working C code

- Tasks/roles:
  - Conducting research
  - Writing code
  - Using binary tree data structure
  - Troubleshooting
  - Debugging

1.3 Definitions of Terms and Acronyms

Glossary

Binary Search Tree – A binary search tree is a rooted binary tree data structure with internal nodes, of which each stores a key or element greater than all of the keys in the node's left subtree and less than those in the node's right subtree. The left and right subtrees of a binary search tree must also be binary search trees.

Child – a node protruding from a parent node (or the node above it)

Parent – a node from which other nodes emerge to create a new subtree

Brother – nodes on the same level emerging from the same parent node are brother nodes

Key/Element – the data stored or kept in any of the nodes of the binary tree

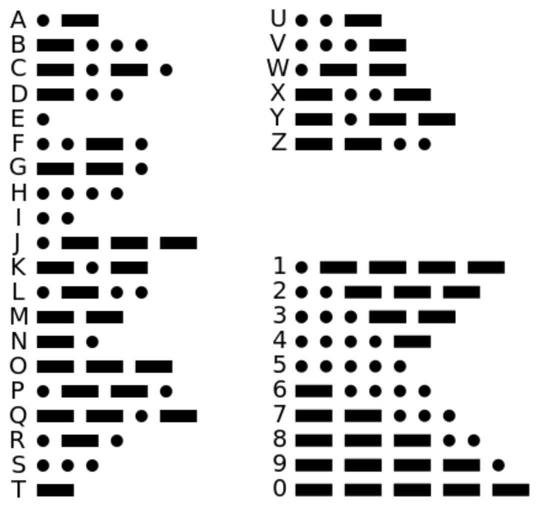
Self-balancing – A self-balancing binary search tree means that when we add new data or values to our initial set the binary tree will sort and balance itself out. We will not have to manually carry out new iterations or implementations in order for the tree to be balanced. It will be automated.

Node – A node is a basic unit of a data structure, such as a linked list or tree data structure. Nodes contain data and also may link to other nodes. A node represents the information contained in a single data structure.

Root node – the topmost node in a binary search tree

Height – the longest path from the root node to a leaf node

Leaf – a node with two empty subtrees or child nodes
Comparable module –
Hash table – a data structure that can map a key to a value
Treap – a hybrid between trees and heaps
Heap – a tree-based data structure in which the tree is a complete binary tree **Stack –** a linear data structure that follows a particular order in which the operations are performed
Dynamic balancing –
Morse code – Morse code is a method used in telecommunication to encode text characters as standardized sequences of two different signal durations, called dots and dashes, or dits and dahs.
Degree – the number of children a node has
Morse Code



A	• —	U	• • —
B	— • • •	V	• • • —
C	— • — •	W	• — —
D	— • •	X	— • • —
E	•	Y	— • — —
F	• • — •	Z	— — • •
G	— — •		
H	• • • •		
I	• •		
J	• — — —		
K	— • —	1	• — — — —
L	• — • •	2	• • — — —
M	— —	3	• • • — —
N	— •	4	• • • • —
O	— — —	5	• • • • •
P	• — — •	6	— • • • •
Q	— — • —	7	— — • • •
R	• — •	8	— — — • •
S	• • •	9	— — — — •
T	—	0	— — — — —

Operating System

The project is being implemented on macOS using Terminal and GitHub.

1.4 References

1. https://en.wikipedia.org/wiki/Node_(computer_science)
2. https://en.wikipedia.org/wiki/Morse_code
3. https://www.mathworks.com/content/dam/mathworks/mathworks-dot-com/moler/exm/chapters/morse.pdf

1.5 Overview

The project being implemented is a balanced binary search tree. A binary tree is a tree data structure. Each of its nodes has a maximum of two “children”, also referred to as the left and right children. Each “child” is another node. A self-balancing search tree automatically tries to keep its height, the number of levels of nodes underneath the root, minimal by performing transformations at relevant times.

These binary trees can be used for constructing and maintaining ordered lists, associative arrays, and key-value pairs. They can be altered to record additional data and perform other operations, which can help optimize database queries and different algorithms.

Using a self-balancing binary search tree, we are going to sort Morse code.

2. Requirements

2.1 Functional Requirements

- It should be able to be supported on all platforms that support C or C++.

2.2 Platform Requirements

- Any platform or medium that supports C or C++ should be able to support our project/product.

- 4 GB RAM
- Pentium 4 2GHz CPU clock or equivalent
- 5 GB free HDD space
- Supported OS:
  - CentOS 6 (64-bit)
  - Ubuntu 14.04 (64 bit)

2.3 Implementation Requirements

- The code is written in C programming language.

2.4 Performance Requirements

- In general the time complexity for the binary tree is O(logn), since we are using the tree to store the English alphabet, the number of levels is 4 so it becomes $\log_2 4$. Depending on the text size the taken time to decode it will differ.

2.5 Verification Requirements

N/A

2.6 Documentation Requirements

N/A

3. <u>Project Management</u>

3.1 Dependencies, Assumptions, Risks

Dependencies

- Time
- Sufficient knowledge of data structures, especially trees

Assumptions

- The user has an internet connection.
- The user has access to Github.
- The user has a device to be able to connect to the internet and interact with our project.

Risks

- Inability to manage time properly in order to allocate time towards the project implementation.
- Failure to make code work properly or at all.

3.2 Schedule and Effort Estimations

Schedule:

Date	Deliverable
29/11/21	Completion of PRD Version 1.0
01/12/21	Implementation initiation, start writing code
03/12/21	Continue writing code
06/12/21	Troubleshooting, debugging
08/12/21	Make presentation to present on 10/12/21
10/12/21	Completion of entire project, presentation

Effort Estimations:

We estimated that each of us will have to spend at least 10 hours for the project to be fully developed and implemented before the due date.

3.3 Acceptance Criteria

N/A