Table of Contents

[**1.** **Introduction 4**](#_Toc72728908)

[1.1 Purpose 4](#_Toc72728909)

[1.2 Scope 4](#_Toc72728910)

[1.3 Intended Audience 4](#_Toc72728911)

[1.4 References 4](#_Toc72728912)

[1.5 Definitions, Acronyms, and Abbreviations 4](#_Toc72728913)

[**2.** **System Overview 5**](#_Toc72728914)

[2.1 System Context Diagram 6](#_Toc72728915)

[**3.** **Design Considerations 6**](#_Toc72728916)

[3.1 Assumptions and Dependencies 6](#_Toc72728917)

[3.2 General Constraints 6](#_Toc72728918)

[3.3 Goals and Guidelines 6](#_Toc72728919)

[3.4 Development Methods 7](#_Toc72728920)

[**4.** **Pseudo Code 7**](#_Toc72728921)

**1. Introduction**

**1.1 Purpose of this document**

  The Software Design Specification (SDS) sections provides the guidelines related to the structure and the contents of SDS document. This software design is made with the purpose of outlining the design of graphical simulation of BFS and DFS in detail. The document will provide developers an insight in meeting users needs efficiently and effectively. Moreover the document facilitates communication and understanding of the system by providing several views of the system design.

**1.2 Scope of the development project**

The software design document would demonstrate how the design will accomplish the functional and non-functional requirements captured in the Software Requirement Specification (SRS). The document will provide a framework to the programmers through describing the high level components and architecture, subsystems, interfaces, database design and algorithm design. This is achieved through the use of architectural patterns, sequence diagrams, class diagrams, relational models and user interfaces.

**1.3 Definitions, acronyms, and abbreviations**

Graph - A graph data structure consists of a finite set of vertices or nodes or points, together with a set of unordered pairs of these vertices for an undirected graph or a set of ordered pairs for a directed graph. These pairs are known as edges or arcs or lines for an undirected graph and as arrows or directed edges or directed arcs or directed lines for a directed graph.

Breadth First Search (BFS) - It is an algorithm for traversing or searching tree or graph data structures. It starts at the tree root and explores the neighbour nodes first, before moving to the next level neighbours.

Depth First Search(DFS) - It is an algorithm for traversing or searching tree or graph data structures. One starts at the root and explores as far as possible along each branch before backtracking.

**1.4 References**

  1) Python.org.

### 2) Python for dummies by Eric Tyson.

**1.5 Overview of document**

Our SDS involves various components like Architectural System Description, Structure and relationships.

**2. System architecture description**

**Overview of modules / components**

This project consists of one program with 4 modules

* Main module to continuously interact with user in drawing graph.
* Node module consisting of two classes describing the characteristics of nodes and edges of a graph.
* Menu module which creates buttons in order to perform an action triggered by button click.
* BFS and DFS module which has methods to traverse graph using BFS and DFS algorithms.

**2.2 Structure and relationships**

The relationship between application and the user is done by handling mouse events. When a mouse is clicked on the screen, a vertex is created and added to the list of nodes. When a source and destination vertex is selected, an edge is created between them and it is added to the list of edges.

**2.3 User interface issues**

In this project we have created an application in which Graphical user interface (GUI) is provided to **:**

* To draw graph through mouse clicks.
* To choose between BFS and DFS traversal simulation.
* To view modifications made in stack and queue contains in DFS and BFS respectively.

**3. Detailed description of components**

**3.1 Component template description**

The components used in this project and their description is given below.

**3.2 Vertex creation:**

* On a mouse click , a vertex is created.
* Its coordinates(x,y) are stored .
* Each vertex is labelled with a numeric value.
* Then the vertex is added to the list of vertices.

**3.3 Edge creation:**

* A source node is selected, if the coordinates of the mouse click is within the vertex area.
* Similarly destination vertex has to be selected.
* Now an edge is created between source and destination vertices.
* Then the edge is added to the list of edges.

**3.4 BFS Simulation:**

* It starts at the start vertex.
* Then it explores the neighbour vertex first, before moving to the next level neighbours.
* Shows the queue contents at each iteration.
* Prints the result.

**3.5 DFS Simulation:**

* It starts at the start vertex.
* Then it explores as far as possible along each branch before backtracking.
* Shows the stack contents at each iteration.
* Prints the result.

**4.0 Reuse and relationships to other products**

For teams doing enhancement work, reuse is an important issue. Most enhancement work should focus on extending, rather than replacing, the design and product development from earlier designs. For teams doing new development, reuse can also be an important strategy. In some cases, there is freeware that could be incorporated. In other cases, there are existing modules or classes that could be adapted. Another possibility is the use of special tools that produce open source results and thus permissible under the terms of this course.

* Reusable code —GUI to create graph.
* This code can be reused in many other simulations in which we need to create a graph, trees.
* This graph created can be used for various simulations like simulation of Floyd’s algorithm.

**5.0 Design decisions and trade-offs**

We have completely designed this project using Python language, coming to the design decisions

1. User interface through GUI.
2. A vertex is created on a mouse click.
3. When a source and destination vertex is selected, an edge is created between them and it is added to the list of edges.
4. Four buttons are provided for various operations.
5. Exit when clicked on close button.

**6.0 Pseudo code for components**

Pseudo codes for the Server and client program is given below

**6.1 Program Pseudo code:**

Step 1: Create a vertex on a mouse click.

Step 2: Select a source node and destination node to create an edge between them.

Step 3: Repeat step1 and step2 until the graph is created.

Step 4: If new button is clicked move to step1.

Step 5: If BFS button is clicked, move to step8.

Step 6: If DFS button is clicked, move to step9.

Step 7: If UNDO button is clicked delete the most recent operation.

Step 8: Visit start vertex, insert neighbouring vertices into queue. Visit the vertex at the front of the queue and perform dequeue operation until the queue is empty.

Step 9: Visit the start vertex.

Step 10: Visit the next adjacent vertex and push on the top of the stack until end of the branch. Pop the vertex at the top of the stack and go to step10.

Step 11: Close the application when mouse is clicked on close button.

**SDS component template**

## Program

|  |  |
| --- | --- |
| Identification | main.py |
| Type | A program that runs on the machine, but considered one of the modules/components of the system |
| Purpose | To constantly receive input through user interaction.  Provide a GUI to the user.  Close the application when the mouse click is made on close button. |
| Function | In brief, the component provides a graphical user interface to create graph. Graph is created by handling mouse events. Four buttons are provided to perform various operations. New button takes a fresh graph input . BFS performs BFS simulation and prints the result. DFS performs DFS simulation and prints the result. UNDO deletes the most recent operation. |
| Subordinates | There are no subordinate components. |
| Dependencies | The component is made up of several modules and does not have dependencies on any other component. |
| Interfaces | There is only one user interface – through the GUI  The application communicates to the user by handling mouse events and then by passing messages through different classes.  The possible errors are :   * ERROR too many vertices * ERROR use of UNDO before adding any vertex. |
| Resources | CPU execution time  Memory – primary for execution. |
| Data | No external data in files is accessed.  Internally, global and local variables are used to store vertices or edges etc. |