Linkipedia

Design Specification

Version 2.0
April 12th, 2020

Group 06
Jay Mody, Jessica Lim, Maanav Dalal, Pranay Kotian

SFWRENG 2XB3
Department of Computing and Software McMaster

Table of Contents

Table of Contents 2

Revision Page 4

Contributions Page 5

Executive Summary 6

Internal Design Review 7

1 Node Module 8
- 1.1 Abstract Data Type 8
- 1.2 Uses 8
- 1.3 Syntax 8
- 1.4 Semantics 8

2 Graph Module 10
- 2.1 Abstract Data Type 10
- 2.2 Uses 10
- 2.3 Syntax 10
- 2.4 Semantics 11

3 DataParser Module 13
- 3.1 Module 13
- 3.2 Uses 13
- 3.3 Syntax 13
- 3.4 Semantics 13

4 Pair Module 15
- 4.1 Module 15
- 4.2 Uses 15
- 4.3 Syntax 15
- 4.4 Semantics 15

5 Search Module 17
- 5.1 Generic Module 17
- 5.2 Uses 17
- 5.3 Syntax 17
- 5.4 Semantics 17

6 Sort Module 19
- 6.1 Generic Module 19
- 6.2 Uses 19
- 6.3 Syntax 19
- 6.4 Semantics 19

7 API Module **21**
7.1 Generic Module 21
7.2 Uses 21
7.3 Syntax 21
7.4 Semantics 21

8 Serving Web Content Application Module **23**
8.1 Module 23
8.2 Uses 23
8.3 Syntax 23
8.4 Semantics 23

9 Search Controller Module **24**
9.1 Module 24
9.2 Uses 24
9.3 Syntax 24
9.4 Semantics 24

10 Results Controller Module **25**
10.1 Module 25
10.2 Uses 25
10.3 Syntax 25
10.4 Semantics 25

Revision Page

Implementation revision history: Link to github repo commit history
https://github.com/JessicaLim8/Linkipedia/commits/master

Summary of design decisions through our meeting minutes:

Date	**Summary**
Jan 22nd	Brainstormed potential ideas for our final project.
Feb 26th	Project idea finalized (Linkipedia). Overall outline and scope of project being fleshed out. Possible graphing and sorting algorithms being explored.
Feb 28th	Outline of our module specification has been completed with details for the sorting and searching modules finalized. The sorting module included mergesort and quicksort functionality (later changed to just quicksort). The searching module included a binary search and lexicographical search. The other modules included: Data parser, API, Graph, Node, and Front End. (The front end module is later changed to Results Controller, Search Controller, and Serving Web Content Application modules. Additionally, a Pair module is created which serves as a tuple ADT.)
Mar 19th	Module interface specification has been formally written with the uses, syntax and semantics for each module clearly outlined. Having finalized how the modules will be interacting with each other, the modules are split between the group members to work on individually.
Mar 24th	WIth the majority of the backend completed (in Java) the last major design decision was our front end implementation. We considered creating a simple implementation using Java Swing or a command line interface since the focus of the project is on the backend implementation of our sorting, graphing, and searching functions. However, we ultimately decided on creating a website using Java Spring.
Apr 7th	FrontEnd module changed to Results Controller, Search Controller, and Serving Web Content Application modules for the Java Swing front end. No other major design decisions were made at this time. We simply worked on finalizing the front end implementation, creating test cases, and general debugging.

By virtue of submitting this document we electronically sign and date that the work being submitted by all the individuals in the group is their exclusive work as a group and we consent to make available the application developed through [CS] or [SE]-2XB3 project, the reports, presentations, and assignments (not including my name and student number) for future teaching purposes.

Contributions Page

Name	Role(s)	Contributions
Jay Mody 400195508	Developer (Backend) Q&A (Testing)	- Implemented Graph related algorithms and code - Assisted with the design of the sort and search modules - Created Unit tests for Node, Graph, Search, and Sort modules - Created UML diagrams
Maanav Dalal 400178117	Developer (Frontend) UI Design	- Integrated Spring Boot backend into application - Designed mockups for front-end - Implemented front-end and integrated using MVC design pattern - Assisted with bug fixes and performed rigorous user testing
Jessica Lim 400173669	Developer (Backend) Backend-Frontend integration	- Created the pair modules to use as a Tuple data type - Created DataParser Module to allow for datasets to be parsed - Wrote API module to allow for backend code and frontend to interact with one another - Wrote Design Specification for some of modules
Pranay Kotian 400198425	Developer (Backend) Documentation	- Implemented Sort and Search modules - Documented group meetings through meeting minutes - Summarized meeting minutes in revision history - Wrote design specification for some of the modules

Executive Summary

Inspired by the wiki game, our project, Linkipedia, helps users find connections between two Wikipedia pages. We completed this project over the span of about 4 months. We began with a general outline of our module interface specification in order to more easily separate the workload and enable parallelization of our efforts. From here, we completed multiple scrum-like sprints with a focus on creating a minimum viable product with fully operational features. These sprints were documented using meeting minutes and our github repo commit history. Some changes were made to our MIS over time, with our final working product being deliverable for April 12th 2020.

The following is an outline of how our product functions and how the various modules interact with one another. The data parser module converts the input from the source files into usable data. Next, the graph and node modules use the pair, sort, and search modules to build the graph and find the connections between the two topics that have been queried by the user. Finally, the results are displayed to the user through the API module as well as the search controller, results controller and serving web content application modules which handle the front end Java Spring implementation.

Internal Design Review

Our design includes a backend that processes the data and performs the algorithms required to find nodes and determine paths as quickly as possible. The frontend is a web application which improves the usability of our project, and allows individuals to explore the connections between two seemingly unrelated wikipedia pages. The frontend itself does no computing; it simply includes controllers that call the backend program via the API module. This allows for optimal user-friendliness, which was a requirement outlined in the requirements document.

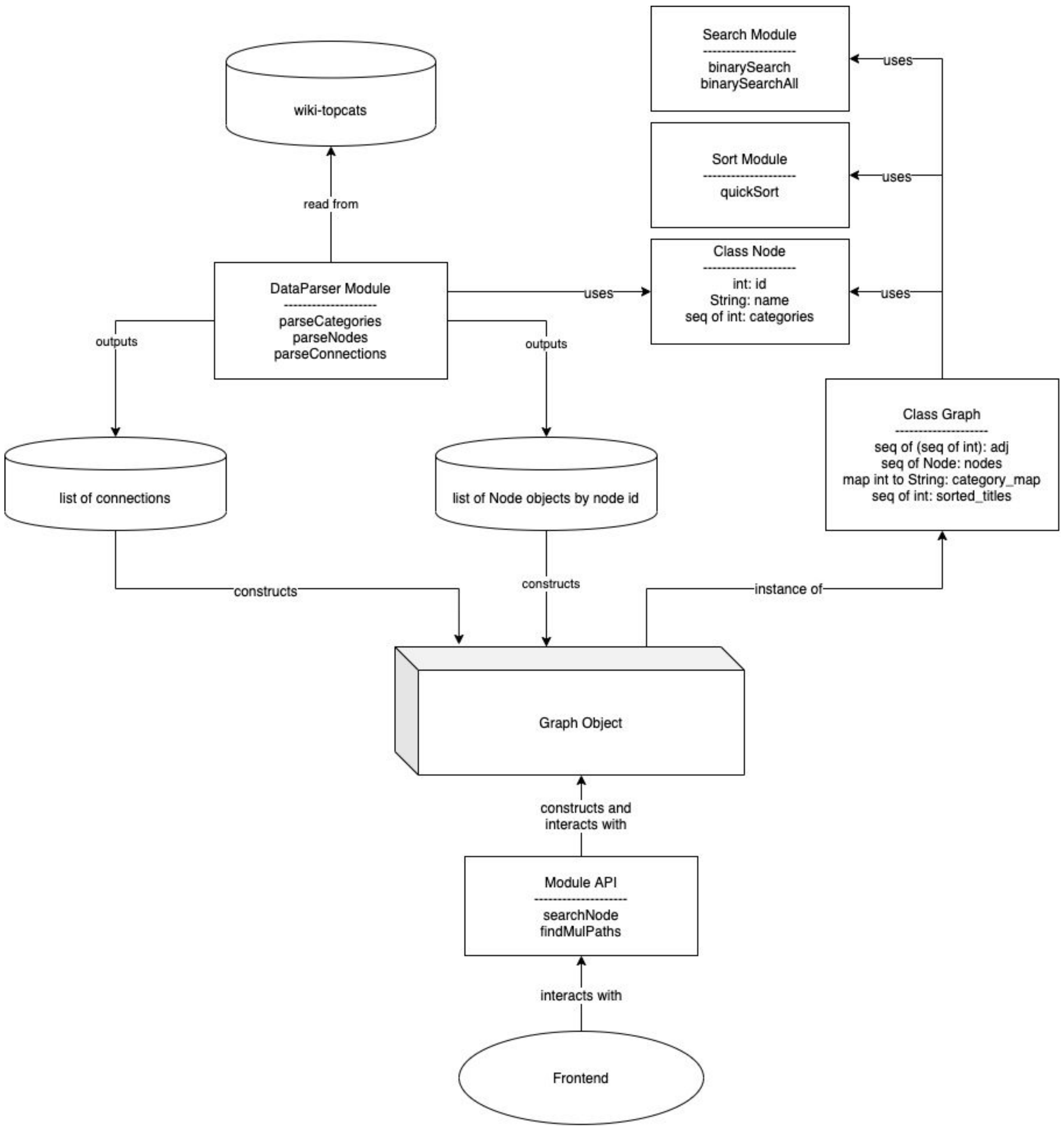
Our program includes an exhaustive data source with millions of nodes and connections. However, this causes some design issues as the data takes a long time to process. To parse the data and create the graph, it takes approximately 50 seconds to run, and 5 minutes to run on a web server. This is definitely a critique upon the user friendliness of our design, as we were not able to find a way to cache data. However, to ensure that this data processing occurs infrequently, the graph is only processed once, when the application is first loaded, and then any queries (e.g. searches, path finding, etc.) can be performed without reloading the graph. Because the queries are quite fast once data is loaded, it follows our requirements.

When searching, it was important to find any results that began with the searched substring. We did this using a modified binary search, where a match is found if the substring being searched is a prefix to the string at the current index. To modify it in a way that a perfect match would be presented first, and then the rest of the possibilities would be listed in lexicographical order, after the first instance, the program will simply iterate to the left until the string at the current index no longer contains the inputted string. This works well for the most part, although if a very short string is provided as input (for example a single letter or an empty string) the searching goes from log(n) complexity to linear time, which is much slower.

To allow for multiple paths to be found, we modified BFS so that the adjacency list would be shuffled prior to being added to the queue. In doing so, this allowed us to find different results on our data every time BFS was run. Each path found using BFS would be compared to previously found values, until sufficient shortest paths are found. This implementation made for a quick fix to finding multiple paths, as other algorithms proved to be quite slow. However, it is difficult to verify if we are finding all the possible paths, as our program will deem that no other new paths exist if three consecutive existing paths are found. While this is sound in terms of mathematical probability, if our program returns two paths on a request of five paths, we cannot verify that there were not more paths that exist, although in most cases that would be the case.

To ensure the code works as desired, and as per the requirements, test were run on all algorithmic and data structure modules. While some design decisions reduce the optimality of the program, for the most part, our program is quite efficient and extremely friendly.

UML Class Diagrams



Static representation of our application classes and relationship between classes

1 Node Module

1.1 Abstract Data Type

Node() uses Comparable<Node>

1.2 Uses

1.3 Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine Name	**In**	**Out**	**Exceptions**
new Node	int, String	self	
id		int	
title		String	
categories		ArrayList<String>	
addCategory	String		
toString			
compareTo	Node	int	

1.4 Semantics

State Variables

id: int
title: String
categories: ArrayList<String>

State Invariant

Assumptions

id, title, and categories are all valid information taken from the input dataset.

Access Routine Semantics

new Node(i, name):

- Transition: id, title := i, name

- Output: out := self
- Exception: None

id():
- Transition: None
- Output: out := id
- Exception: None

title():
- Transition: None
- Output: out := title
- Exception: None

categories():
- Transition: None
- Output: out := categories
- Exception: None

addCategories(cat):
- Transition: categories := categories.add(cat)
- Output: None
- Exception: None

toString():
- Transition: None
- Output: out := id + “, “ + title
- Exception: None

compareTo(that):
- Transition: None
- Output: out := title.compareTo(that.title)
- Exception: None

2 Graph Module

2.1 Abstract Data Type

Graph()

2.2 Uses

Node Module
Search Module
Sort Module

2.3 Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine Name	In	Out	Exceptions
new Graph	Node[] || ArrayList<Node>	self	
N		int	
E		int	
node	int	Node	
adj	int	Iterable<int>	
addEdge	int, int		
numNodes		int	
search	String	ArrayList<Node>	
shortestPath	Node, Node	ArrayList<Node>	
nShortestPaths	Node, Node	ArrayList<Node>	
private pathExists	ArrayList<ArrayList<Node>>, ArrayList<Node>	boolean	
private validNode	Node		IllegalArgumentException

2.4 Semantics

State Variables
N: int
E: int
nodes: ArrayList<Node>
adj: List<ArrayList<int>>
sorted_nodes: ArrayList<int>

State Invariant

Assumptions

- Graph is Immutable
- The index of each Node in the nodes state variable corresponds to the Node's Id at that index
- sorted_nodes must contain the node ids sorted lexicographically by title

Access Routine Semantics
new Graph(nodes):

- Transition: nodes, N, E, shortedNodes, adj := nodes, nodes.length, 0, nodes.quickSort(), new List[n]
- Output: out := self
- Exception: None

N():

- Transition: None
- Output: out := N
- Exception: None

E():

- Transition: None
- Output: out := N
- Exception: None

node(n):

- Transition: None
- Output: out := nodes[n]
- Exception: None

adj(n):

- Transition: None
- Output: out := adj[n].shuffle()
- Exception: None

addEdge(int src, int dst):

- Transition: E, adj[src] := E++, adj[src].add(dst)

    - Add every edge to graph
- Output: None
- Exception: None

search(title):
- Transition: None
- Output: out := results = for item in sortedNodes | title.subString(item) ? results.add(item)
    - Find all instances of the title that are present in the list of nodes
- Exception: None

shortestPath(src, dst):
- Transition: None
- Output: out := path = shortestPath(src, dst)
    - Find shortest path from source to destination node
- Exception: None

nShortestPaths(src, dst, n):
- Transition: None
- Output: out := paths = nShortestPath(src, dst, n)
    - Find n shortest paths from source to destination node
- Exception: None

pathExists(paths, path):
- Transition: None
- Output: out := for p in paths; path == p -> True | False
    - If path matches one of the paths, return true
- Exception: None

validNode(n):
- Transition: None
- Output: None
- Exception: n.id < 0 | n.id > N -> IllegalArgumentException

3 DataParser Module

3.1 Module
DataParser()

3.2 Uses
Node Module
Pair Module

3.3 Syntax
Exported Constants
None

Exported Types
None

Exported Access Programs

Routine Name	In	Out	Exceptions
parseNodes	String, String	Arraylist<Node>	FileNotFound
parseCategories	String	ArrayList<Pair(String , ArrayList<Int>)>	FileNotFound
parseConnections	String	ArrayList<Integer[]>	FileNotFound
private setCategories	Arraylist<Node>, ArrayList<Pair(String, ArrayList<Int>)>	None	

3.4 Semantics
State Variables

Assumptions
Inputted files will be valid

Access Routine Semantics
parseNodes(nodes, categories):

- Transition: nList := For x connections.newLine(); nlist.append(Node(id = x[0] name = x[1]))
  nList := setCategories(nList)
    - For every line, create a node with the given id and name, and add note to list, set categories to list of nodes
- Output: out := nList
- Exception: File nodes || File categories not found -> FileNotFound

parseCategories(categories):

- Transition: catList := For x in categories.newLine(); catList.append(Pair(x[0], x[1]))
    - i.e. append every category and the list of nodes in that category
- Output: out := catList
- Exception: File nodes || File categories not found -> FileNotFound

parseConnections(connections):

- Transition: cList := For x connections.newLine(); clist.append([x[0], x[1]])
    - For every line, add a 2 element list with the source and destination path
- Output: out := cList
- Exception: File nodes || File categories not found -> FileNotFound

setCategories(nodeList, categories):

- Transition: for newLine in categories, for every id in line, add title to node(id).categories
    - For every line, add every category to the appropriate nodes
- Output: None
- Exception: None

4 Pair Module

4.1 Module

Pair()

4.2 Uses

4.3 Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine Name	In	Out	Exceptions
Pair	A, B		None
getA		A	None
setA	A		None
getB		B	None
setB	B		None
equals	Pair<A, B>	boolean	None

4.4 Semantics

State Variables

private A a;
private B b;

State Invariant

Assumptions

Access Routine Semantics

Pair(A a, B b):

- Transition: a, b := a, b
- Output: None
- Exception: None

getA():

- Transition: None
- Output: out := a
- Exception: None

setA(A a):
- Transition: this.a := a
- Output: None

getB():
- Transition: None
- Output: out := b
- Exception: None

setB(B b):
- Transition: this.b := b
- Output: None
- Exception: None

equals(Pair<A, B> that):
- Transition: None
- Output: out := (this.a == that.getA() && this.b == that.getB())
- Exception: None

5 Search Module

5.1 Generic Module

Search()

5.2 Uses

5.3 Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine Name	In	Out	Exceptions
binarySearch	Comparable[], Comparable	int	None
binarySearch	T[], T, Comparator<T>	int	None
binarySearchAll	Comparable[], Comparable, int	ArrayList<Integer>	None
binarySearchAll	T[], T, int, Comparator<T>	ArrayList<Integer>	None

5.4 Semantics

State Variables

None

State Invariant

None

Assumptions

The list passed into the binary searching algorithm is sorted with respect to the type of the identifier (ID, Title).

Access Routine Semantics

binarySearch(arr, target):

- Transition: None
- Output: out := if value found: return mid, else return -1
    - Index position of value in array (-1 if not found)

binarySearch(arr, target, c):

- Transition: None
- Output: out := if value found: return mid, else return -1
    - Index position of value in array (-1 if not found)

binarySearchAll(arr, target, N):

- Transition: None
- Output: out := ArrayList<Integer> of (max N) index positions of matches

binarySearchAll(arr, target, N, c):

- Transition: None
- Output: out := ArrayList<Integer> of (max N) index positions of matches

6 Sort Module

6.1 Generic Module

Sort()

6.2 Uses

6.3 Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine Name	In	Out	Exceptions
quickSort	Comparable[]		
private quickSort	Comparable[], int, int		
private partition	Comparable[], int, int	int	
quickSort	T[], Comparator<T>		
private quickSort	T[], int, int, Comparator<T>		
private partition	T[], int, int, Comparator<T>	int	
private exch	T[], int, int		

6.4 Semantics

State Variables

None

State Invariant

None

Assumptions

None

Access Routine Semantics

quickSort(arr):

- Transition: None
  - Calls quickSort with lo and hi values
- Output: None

private quickSort(arr, lo, hi):

- Transition: j := partition(arr, lo, hi)
    - j is set to the partitioning element and the two subarrays are recursively sorted in-place
- Output: None

private partition(a, lo, hi):

- Transition: exch(a, i, j);
    - partitioning element is chosen such that there is no smaller entry to the left and there is no larger entry to the right
    - partitioning element is swapped to its required position within the array
- Output: out := j
    - index of partitioning element

quickSort(arr, c):

- Transition: None
    - Calls quickSort with lo and hi values, and the comparator
- Output: None

private quickSort(arr, lo, hi, c):

- Transition: j := partition(arr, lo, hi, c)
    - j is set to the partitioning element and the two subarrays are recursively sorted in-place
- Output: None

private partition(a, lo, hi, c):

- Transition: exch(a, i, j);
    - partitioning element is chosen such that there is no smaller entry to the left and there is no larger entry to the right
    - partitioning element is swapped to its required position within the array
- Output: out := j
    - index of partitioning element

private exch(a, p1, p2):

- Transition: t = a[p1], a[p1] = a[p2], a[p2] = t
    - Simple swap function with temp variable t
- Output: None

7 API Module

7.1 Generic Module
API()

7.2 Uses
Graph Module
Node Module
Pair Module

7.3 Syntax
Exported Constants
None

Exported Types
None

Exported Access Programs

Routine Name	In	Out	Exceptions
main	String, String, int, int || None		
createGraph		Graph	
searchNode	Graph, String, int	ArrayList<Node>	
searchSingleNode	Graph, String	Node	
findPath	Graph, Node, Node	ArrayList<Node>	
findMulPath	Graph, Node, Node, int	ArrayList<ArrayList<Node>>	
pathToString	Graph, Node, Node, int	Arraylist<String>	

7.4 Semantics
Assumptions
- Command line arguments for Main will match given format OR they will not be passed in at all
- Program is being run on the web topcats files

Access Routine Semantics
main(src, dst, searchNum, pathNum):

- Transition: graph := createGraph(), nodeA := searchSingleNode(graph, src, searchNum), nodeB := searchSingleNode(graph, dst, searchNum), pathToString(graph, nodeA, nodeB, pathNum)
    - Calls functions in order to find pathNum number of paths from the source to destination string, using the node with the first result
- Output: None
    - All results are printed to commandline
- Exception: None

createGraph():
- Transition: graph := parseNodes() && parseConnections() && for pair in connections | graph.addEdge(pair)
    - Parse the nodes and connections, and add all pairs to graph
- Output: out := graph
- Exception: None

searchNode(graph, title, max):
- Transition: list := graph.search(title).substring(0, max);
    - Search graph for given title with maxlength max
- Output: out := list
- Exception: None

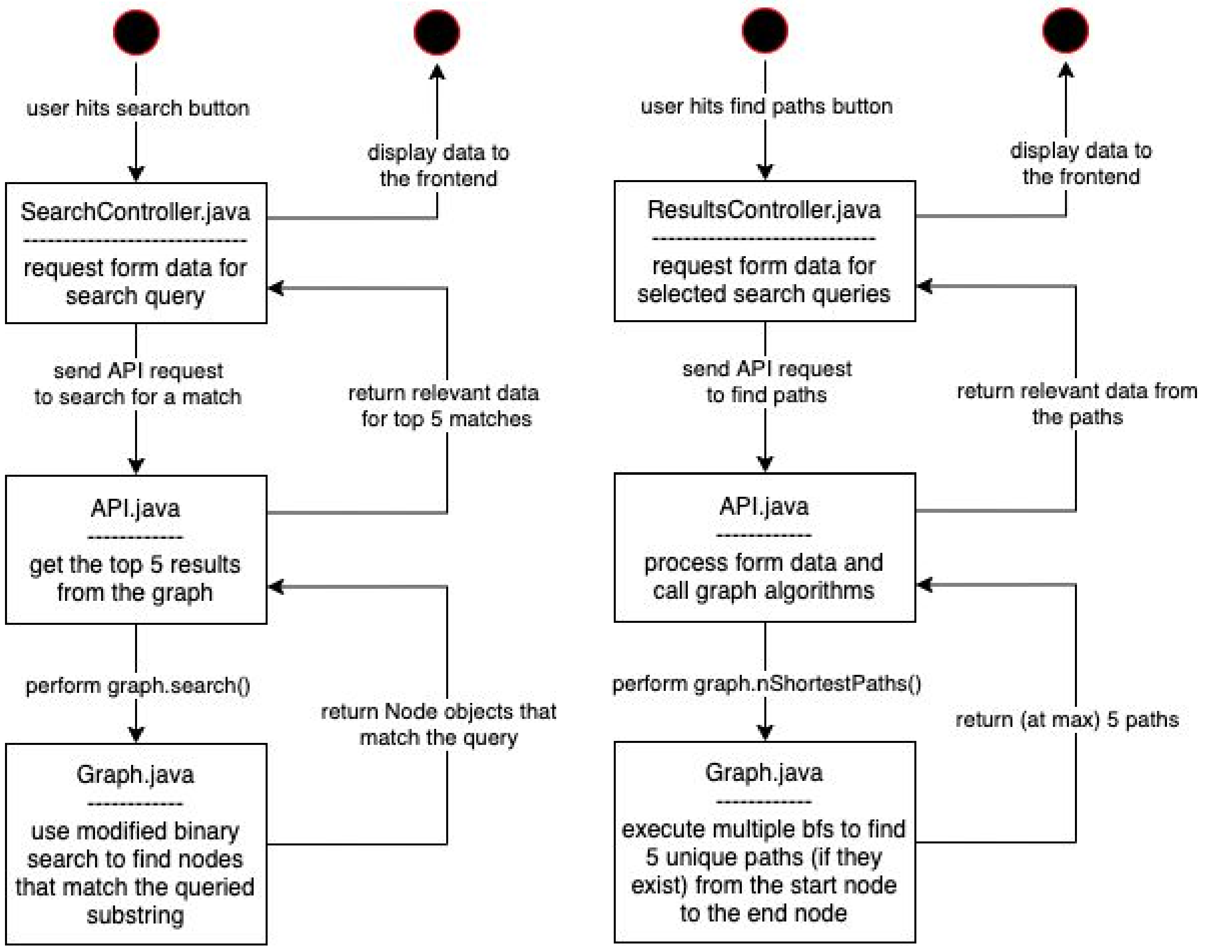
searchSingleNode(graph, title):
- Transition: result := graph.search(title)[0] || null;
    - Find first instance of the result in graph, or null if no result is found
- Output: out := result
- Exception: None

findPath(graph, src, ds):
- Transition: result := src & dst != null -> graph.shortestPath(src, dst) |null;
    - Find path from src to dst, return null ot
- Output: out := result
- Exception: None

findMulPath(graph, src, dst, n):
- Transition: result := src & dst != null -> graph.nshortestPaths(src, dst) |null;
    - Find path from src to dst, return null ot
- Output: out := result
- Exception: None

pathToString(graph, src, dst, n):
- Transition: printedPaths := for result in findMulPath(graph, src, dst, n), for node in result | printedPaths.appened(node.title + "->")
    - Provide a toString version of the path
- Output: out := printedPaths
- Exception: None

7.5 UML State Diagrams (API)


user hits search button
SearchController.java
request form data for search query
display data to the frontend
send API request to search for a match
return relevant data for top 5 matches
API.java
get the top 5 results from the graph
perform graph.search()
return Node objects that match the query
Graph.java
use modified binary search to find nodes that match the queried substring
user hits find paths button
ResultsController.java
request form data for selected search queries
display data to the frontend
send API request to find paths
return relevant data from the paths
API.java
process form data and call graph algorithms
perform graph.nShortestPaths()
return (at max) 5 paths
Graph.java
execute multiple bfs to find 5 unique paths (if they exist) from the start node to the end node

State diagrams for searching for nodes matching inputted strings (searchNode()) and finding appropriate paths given specific nodes (findMulPaths())

8 Serving Web Content Application Module

8.1 Module

ServingWebContentApplication()

8.2 Uses

API Module

8.3 Syntax

Exported Constants

None

Exported Types

None

Exported Access Programs

Routine Name	In	Out	Exceptions
main	String[] args		
buildGraph			
getGraph		Graph	

8.4 Semantics

State Variables

g: Graph

State Invariant

Assumptions

Access Routine Semantics

main():

- Transition: None
- Output: None

buildGraph():

- Transition: g := API.createGraph()
- Output: None

getGraph():

- Output: out := g

9 Search Controller Module

9.1 Module

SearchController()

9.2 Uses

API Module
ServingWebContentApplication Module

9.3 Syntax

Exported Constants
None

Exported Types
None

Exported Access Programs

Routine Name	In	Out	Exceptions
search	String, String, Model	String	

9.4 Semantics

Access Routine Semantics
search(firstInput, secondInput, model):

- Transition:

result1-5 := API.searchNode(ServingWebContentApplication.getGraph(), firstInput, 5)
end1-5 := API.searchNode(ServingWebContentApplication.getGraph(), secondInput, 5)
model := result1-5 + end1-5

- Output: out := search view (With updated model)

10 Results Controller Module

10.1 Module

ResultsController()

10.2 Uses

API Module
ServingWebContentApplication Module

10.3 Syntax

Exported Constants
None

Exported Types
None

Exported Access Programs

Routine Name	In	Out	Exceptions
results	String, String, Model	String	

10.4 Semantics

State Variables

State Invariant

Assumptions

Access Routine Semantics

results(from, to, model):

- Transition:
    - path1-5 := shortest 5 between from and to
    - Model := path1-5
- Output: out := result view (With updated model)