C·O·L

CEMCA

Design, Development and Implementation of Data Structure Experiments: Graph Traversal, Minimum Spanning Tree and Shortest Path Algorithms

(Final Report submitted to CEMCA)

Venkatesh Choppella
IIIT Hyderabad

[2022-03-14 Mon]

Contents

List of Tables

List of Figures

1 Executive Summary

1.1 Experiment Completion Summary

Table 1 is a summary of the expected and completed work measured in terms of number of experiments built and hosted.

Table 1: Experiment Completion Summary

No. of Experiments contracted to be built	10-12
No. of Experiments built	14
No. of Experiments hosted	14

1.2 Hosting Summary

Table 2 is a summary of the URL's of the hosted labs and also their source code.

Table 2: URLs of Hosted Labs

Lab/Source	URL
Graph Traversal	`http://algodynamics.io/graphTraversal/index.html`
Minimum Spanning Tree	`http://algodynamics.io/mst/index.html`
Shortest Path	`http://algodynamics.io/shortestpath/index.html`
Source code	`https://gitlab.com/vxcg/pub/cemca-ph2-exps`

2 Introduction and Background

This document is the final report of the work supported by CEMCA to build and deploy Virtual Lab Experiments on Data Structures and Algorithms based on the Algodynamics approach. The experiments are grouped into three labs, corresponding to three families of problems: Graph Traversal, Minimum Spanning Tree and Shortest Path. These are quite well-known algorithmic problems, widely covered in a variety of standard text books [2, 3].

2.1 Background

2.1.1 The relevance of Data Structures and Algorithms to CS Education.

Data Structures and Algorithms (DSA) are fundamental to all of computer science. The course on DSA is usually the first course taught in computer science curricula after basic programming and discrete mathematics.

In many DSA courses today, algorithms are taught in a *fait-accompli* manner; algorithms are written out on the blackboard and presented as it is. The laboratory for data structures usually involve programming. This approach has two main drawbacks: first, algorithms are expressed either in pseudo-code or programming notation, often obscuring a more abstract (mathematical) representation which facilitates reasoning and is also more portable. The 'programming first' approach leads to a vocabulary for reasoning about algorithms in terms of the syntax of the programs implementing them (e.g., while loops, function calls, assignment statements, etc.) rather than a more abstract vocabulary consisting of states, observations, actions and dynamics. Second, since algorithms are closed systems, i.e., non-interactive, they are inherently unsuitable for exploration. Students are therefore limited to trace that algorithm's execution, not control it. As a result, the laboratory for Data Structures and Algorithms is usually restricted to coding algorithms rather than interacting with them.

2.1.2 Algodynamics Approach

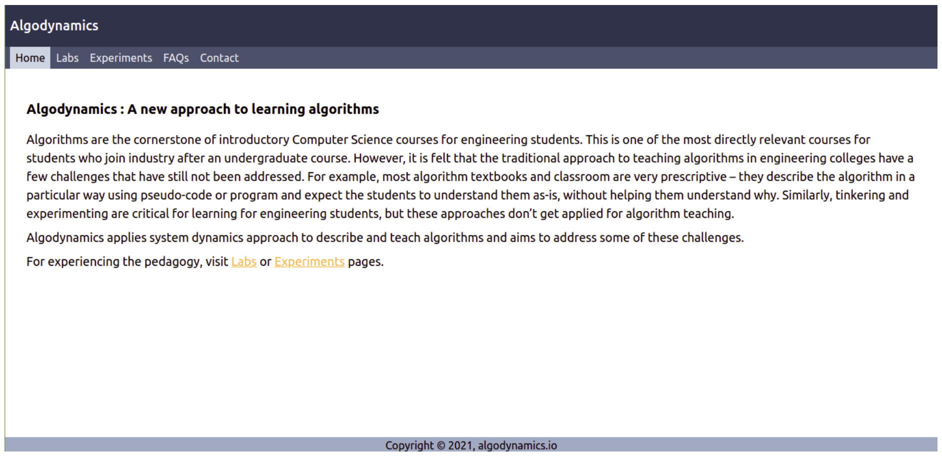
The algodynamics approach is based on three insights that drive the learning of algorithms. First, well-established theories from the learning sciences attest that learning is facilitated by interaction, exploration and strategy building. As a result, the approach of learning algorithms through interactive simulations holds great promise. Second, algorithms are closed systems, and therefore need to be 'opened up' to allow for exploration and tinkering. Third, it should be possible to start from an completely interactive simulation of an algorithm and arrive at a completely automated simulation of the algorithm's execution, exploring several intermediate interactive alternatives in the process. This idea is inspired from the idea of successive refinement, a well-known technique in computer science[4].

Internally, the simulations and the algorithm are uniformly represented as interactive transition systems, which facilitate reasoning using the vocabulary of dynamical systems: state spaces, actions, maps and fixed points. The theoretical basis of the soundness and practicality of this approach are described elsewhere[1]. However, these formal transition systems are not a prerequisite to using the laboratory of interactive experiments.

3 User interface requirements for Labs and Experiments

The user interface of the experiments specify the visual and control elements that a student uses for interacting with the experiment.

A fundamental requirement of the laboratory experiments is that they need to be available online and be accessible to anyone with a laptop or desktop computer and an internet connection. The experiments are hosted on the web at `https://algodynamics.io` webpage. Figure 1 is a screenshot of the homepage of the algodynamics site.



Algodynamics : A new approach to learning algorithms

Algorithms are the cornerstone of introductory Computer Science courses for engineering students. This is one of the most directly relevant courses for students who join industry after an undergraduate course. However, it is felt that the traditional approach to teaching algorithms in engineering colleges have a few challenges that have still not been addressed. For example, most algorithm textbooks and classroom are very prescriptive – they describe the algorithm in a particular way using pseudo-code or program and expect the students to understand them as-is, without helping them understand why. Similarly, tinkering and experimenting are critical for learning for engineering students, but these approaches don't get applied for algorithm teaching.

Algodynamics applies system dynamics approach to describe and teach algorithms and aims to address some of these challenges.

For experiencing the pedagogy, visit Labs or Experiments pages.

Copyright © 2021, algodynamics.io

Figure 1: Landing page of `https://algodynamics.io`

The current set of labs are among a large collection of virtual labs in computer science. Clicking on the labs link on the main webpage shows a gallery of labs (Figure 2).

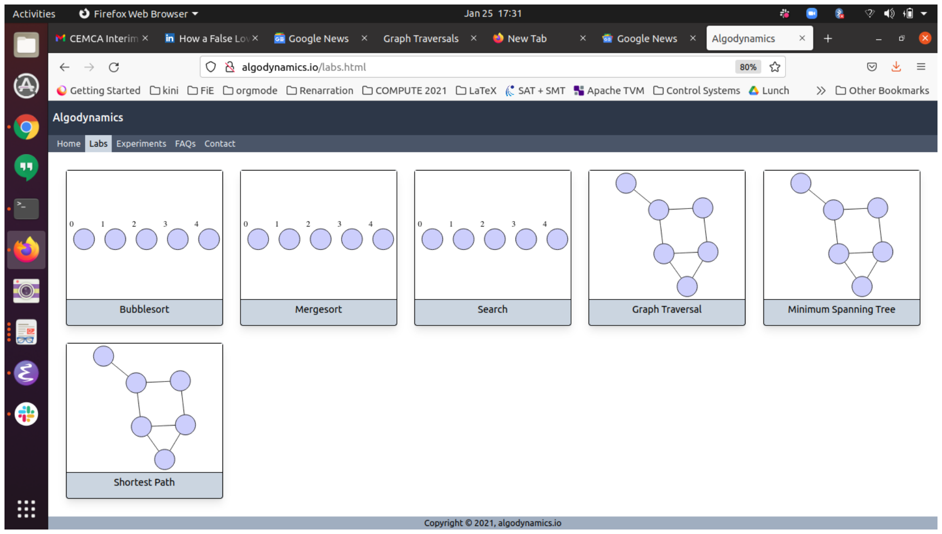
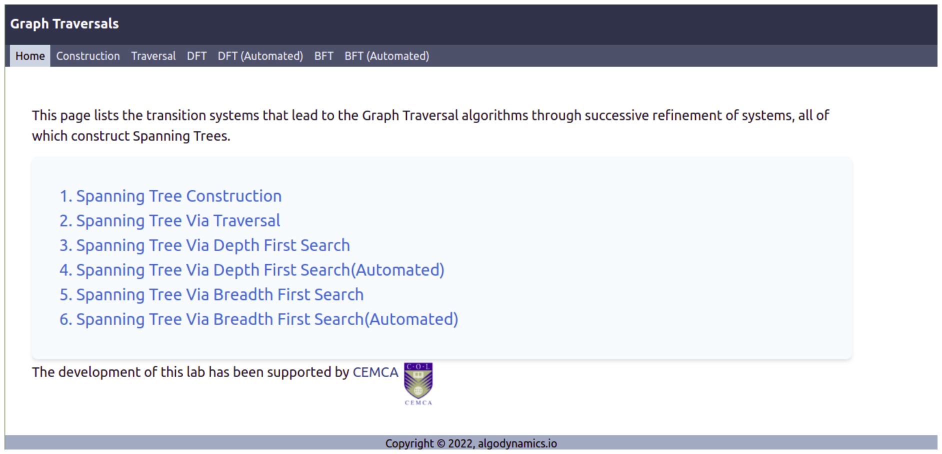


Figure 2: Gallery of data structures labs at `https://algodynamics.io`

Each lab page lists the experiments under that lab (Figure 3). The experimentse are available as tabs.

Each experiment has two panes: an instruction pane and an experiment pane. These are described below.



This page lists the transition systems that lead to the Graph Traversal algorithms through successive refinement of systems, all of which construct Spanning Trees.

1. Spanning Tree Construction
2. Spanning Tree Via Traversal
3. Spanning Tree Via Depth First Search
4. Spanning Tree Via Depth First Search(Automated)
5. Spanning Tree Via Breadth First Search
6. Spanning Tree Via Breadth First Search(Automated)

The development of this lab has been supported by CEMCA

Copyright © 2022, algodynamics.io

Figure 3: List of experiments under a lab (Graph Traversal)

3.1 Instructions pane

The instructions pane has the following sections:

Objectives section This section contains the learning objectives of the experiment.

Experimental Setup section This section describes the experiment interface and how to interact with the various elements of the experiment.

Procedure section This section lists the steps to perform the experiment.

3.2 Experiment Pane

The experiment interface is divided into the following elements:

Problem Display This sub-pane contains the visual representation of the problem. Typically, it is the a diagram of a data structure, which is randomly generated. The graph is chosen small enough so that it easily fits on a screen.

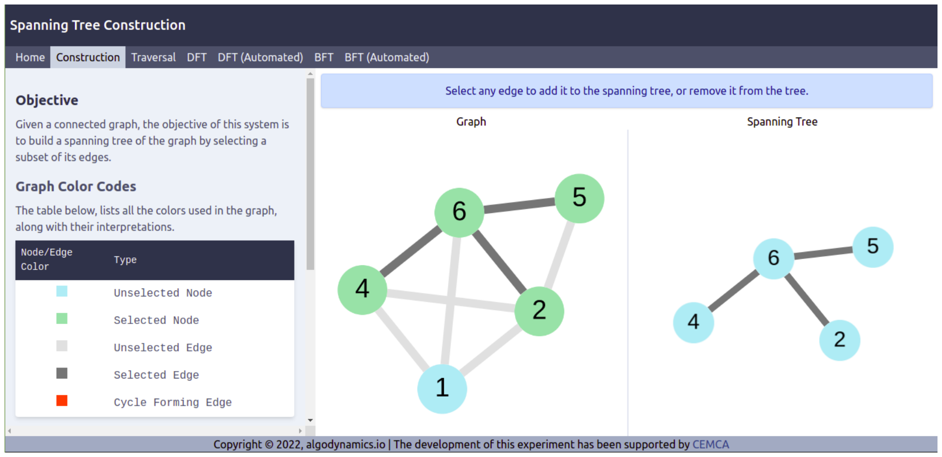
Solution Display This sub-pane contains the partial solution being constructed at any given time.

Control buttons This sub-pane consists of buttons and controls that allow the student to perform operations required by the experiment.

Auxiliary Data Structures This sub-pane shows the auxiliary data structures like stacks or queues used as part of the underlying machinery of the experiment.

Prompt This sub-pane contains the feedback message to the student after each interaction. The feedback indicates the result of the latest action and hints towards subsequent possible actions.

Figure 4 shows a screenshot depicting a typical user interface of an experiment (in this case Spanning Tree construction in the Graph Traversal experiment). The instruction pane is on the left and the experiment pane is on the right. The experiment pane contains the problem and the solution parts.



Objective

Given a connected graph, the objective of this system is to build a spanning tree of the graph by selecting a subset of its edges.

Graph Color Codes

The table below, lists all the colors used in the graph, along with their interpretations.

Node/Edge Color	Type
🟦	Unselected Node
🟩	Selected Node
⬜	Unselected Edge
⬛	Selected Edge
🟥	Cycle Forming Edge

Select any edge to add it to the spanning tree, or remove it from the tree.

Copyright © 2022, algodynamics.io | The development of this experiment has been supported by CEMCA

Figure 4: Screenshot depicting User Interface of a typical experiment (Graph Traversal)

4 Learning objectives and pedagogical design

The requirements of the experiments pertain to the learning objectives, outcomes and the pedagogical approach employed in the experiments. The learning objectives focus on skills that complement the coding or theoretical exercises like the correctness and complexity analysis of the algorithm.

As briefly discussed earlier, algorithms, by their very nature are closed systems and impervious to interaction. The lab experiments are designed to 'open' up algorithms to yield interactive transition systems. Interaction brings with the opportunity to explore, make mistakes, learn from them, develop intuitions and insights into the algorithm's mechanisms, and invent strategies for problem solving.

The approach of Virtual Lab experiments that we discuss here is expected to complement the following important elements that are part of a holistic understanding of an algorithm:

Coding this is already done via programming assignments or laboratories. The interactive experiments are expected to *precede* the coding exercise. Interaction helps in gaining intuition, understanding and insights that help code the algorithm as a program.

Correctness The interactive labs rely on the student's intuition to appreciate the correctness of the algorithm. The goal is not to have the student prove the algorithm's correctness, although this is part of the larger Algodynamics approach, which emphasizes the use of transition systems and reasoning about them. Typically, the correctness proofs are to be done as separate paper and pencil exercises and are not part of the effort of implementing the virtual labs.

Complexity The interactive experiments illustrate the mechanisms behind the execution of an algorithm but not their cost. It is assumed that asymptotic complexity of the algorithms is part of the lecture material. Therefore it is outside the scope of the interactive experiments.

4.1 Strategies and Versions

The Algodynamics approach emphasizes the presentation of experiments as a sequences of refinements of a base, interactive system. We identify two axes along which the experiments progress.

Strategy An algorithmic problem typically admits a multitude of solutions. A solution is distinguished by the strategy it employs. A strategy, simply put, is a rule that determines which action (out of a choice of many) is taken next. Examples of well-known strategies are 'recursive', 'depth-first', 'breadth-first', etc.

Version An experiment may be done completely interactively, or in a completely automated fashion or with some combination of interaction and automation. The former relies on interaction by the student. The latter allows the student to observe the execution of the algorithm by simply pressing a 'next' button. Each class address a different concern: while the interactive versions allow the student to explore, the non-interactive ones allow the student to understand the steps undertaken by the algorithm.

Currently, three types of versions are implemented. Each experiment strategy has been mapped to a subset of the possible versions. The versions are listed below:

Tutorial This is a highly interactive version where the student has the opportunity to explore. Immediate feedback informs the user of any mistakes and offers a chance to undo an action.

Semi-Automated The experiment allows for some interaction, while some aspects of the experiment are automated.

Automated In this version, the student follows the implementation of the algorithm by clicking 'next' and observing the various data structures manipulated by the algorithm.

In the next few sections, we list the experiments designed for the three labs: Graph Traversal, Minimum Spanning Tree and Shortest Path.

5 Detailed List of Experiments

5.1 Graph Traversal: List of Experiments and Requirements

The overall objective of this lab is to introduce the student to an important class of graph traversal algorithms. Graph traversal is employed when systematically searching for a given vertex or trying to construct a spanning tree, which is the problem we solve in this set of experiments. A spanning tree of a graph G is a connected subgraph of G that includes all vertices of the G and has no cycles.

Table 3 is a summary of the Graph Traversal experiments that have been implemented.

Table 3: Graph Traversal Experiments

No.	Problem	Strategy	Version	Status
1	Build Spanning Tree	Arbitrary Choice of edges	Tutorial	Hosted
2	Build Spanning Tree	Traversal	Tutorial	Hosted
3	Build Spanning Tree	Depth First Traversal	Tutorial	Hosted
4	Build Spanning Tree	Depth First Traversal	Automated	Hosted
5	Build Spanning Tree	Breadth First Traversal	Tutorial	Hosted
6	Build Spanning Tree	Breadth First Traversal	Automated	Hosted

5.1.1 Spanning Tree (Arbitrary construction strategy)

The objective of this experiment is to demonstrate the process of building any spanning tree of a connected graph by selecting a subset of its edges. The edges may be selected in any arbitrary order; hence the intermediate structure could be a forest.

Learning outcomes The student should be able to observe the following after performing this experiment:

1. **A spanning tree is a special subgraph of a given graph** The student should be able to observe that the spanning tree is a connected sub-

graph of the original graph, which has all the vertices of the original graph.

2. **Spanning Trees are acyclic** A student must observe that all the spanning trees are acyclic.

3. **Many spanning trees exist for a graph** There is no unique spanning tree, different choices of edges result in different spanning trees.

A screenshot of this experiment is given in Figure 4.

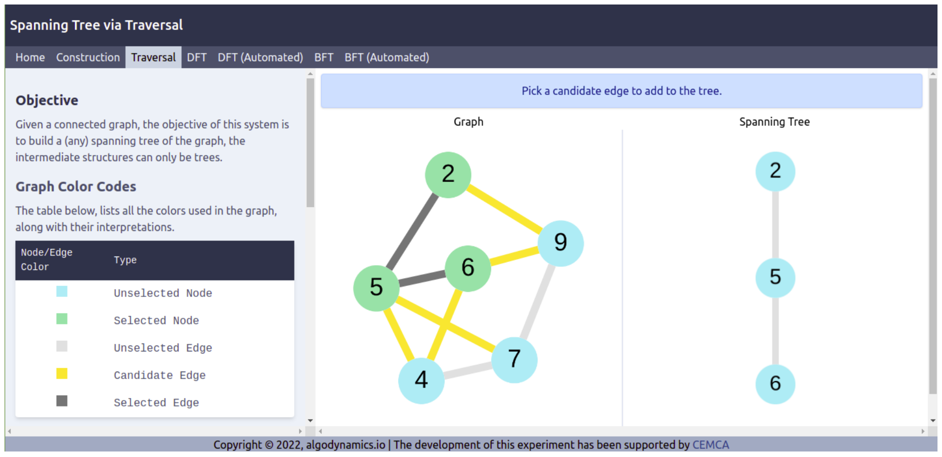
5.1.2 Spanning Tree Traversal (Subgraph is a tree)

The objective of this experiment is to demonstrate the process of building any spanning tree of a connected graph by adding an arbitrarily selected edge from a node already present in the substructure. This ensures that the intermediate structure is always a tree. This strategy may be seen as a refinement of the arbitrary selection of edges employed in the previous experiment.

Learning Outcomes The student should be able to learn the following after performing this experiment:

1. **Graph Traversal** The student should understand the notion of a *traversal*, a sequence of a selection of edges (and vertices) such that intermediate structure is always connected.

2. **The intermediate structures in this process are trees** The student should observe that in a traversal, we add a node that we have not visited yet to the tree along with the edge. This process ensures that all the intermediate graphs before the final spanning tree are also trees.


Spanning Tree via Traversal

Home Construction Traversal DFT DFT (Automated) BFT BFT (Automated)

Objective

Given a connected graph, the objective of this system is to build a (any) spanning tree of the graph, the intermediate structures can only be trees.

Graph Color Codes

The table below, lists all the colors used in the graph, along with their interpretations.

Node/Edge Color	Type
🟦	Unselected Node
🟩	Selected Node
⬜	Unselected Edge
🟨	Candidate Edge
⬛	Selected Edge

Pick a candidate edge to add to the tree.

Copyright © 2022, algodynamics.io | The development of this experiment has been supported by CEMCA

Figure 5: Screenshot of the Spanning Tree (Traversal) experiment in the Graph Traversal Lab

A screenshot of this experiment is given in Figure 5.

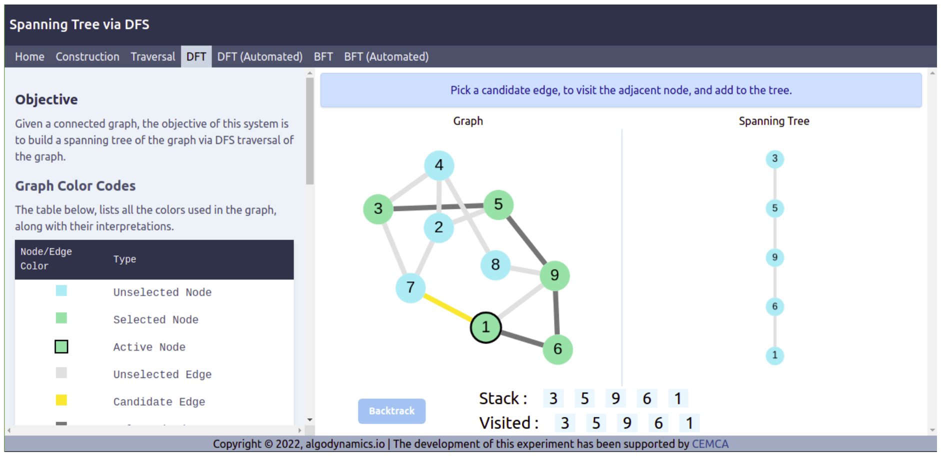
5.1.3 Spanning Tree: Depth First Traversal (Tutorial and Automated Versions)

The objective of this experiment is to demonstrate the process of building any spanning tree of a graph by performing a specific type of traversal called Depth First Traversal (DFT). In strategy in depth first traversal, is to choose a neighbouring vertex, then its neighbour, then the neighbour's and so on, until one reaches a vertex with no neighbours. At that point, the strategy involves backtracking and continuing the depth first search from their. The algorithm terminates when one comes back to the starting node with all vertices of the connected graph visited.

Learning Outcomes The experiment is designed such that the student should be able to learn the following after performing this experiment:

1. **DFT is a particular strategy of traversal** The student should observe that we explore the graph aggressively i.e., we always select such an edge that connects the last visited node and an unvisited node. This strategy signifies that we are moving along the depth of the graph.
2. **The Backtrack operation** The student should understand the use of the *backtrack* operation. This operation is used when we can not go any deeper in the graph, but some unexplored nodes still exist. In such a case, we move back to the last explored node and try to explore the remaining unexplored nodes.
3. **Use of auxiliary data structures** The student should observe the role of auxiliary variables to keep track of the last visited nodes and their parents.

A screenshot of this experiment (tutorial version) is given in Figure 6.



Objective

Given a connected graph, the objective of this system is to build a spanning tree of the graph via DFS traversal of the graph.

Graph Color Codes

The table below, lists all the colors used in the graph, along with their interpretations.

Node/Edge Color	Type
🟦	Unselected Node
🟩	Selected Node
🟩	Active Node
⬜	Unselected Edge
🟨	Candidate Edge

Pick a candidate edge, to visit the adjacent node, and add to the tree.

```
Stack :   3  5  9  6  1
Visited : 3  5  9  6  1
```

Copyright © 2022, algodynamics.io | The development of this experiment has been supported by CEMCA

Figure 6: Screenshot of the Spanning Tree (Depth First Traversal, tutorial version) experiment

5.1.4 Spanning Tree: Breadth First Traversal (Tutorial and Automated Versions)

The objective of this experiment is to demonstrate the process of building any spanning tree of a graph by performing a specific type of traversal called Breadth First Traversal (BFT). All adjacent nodes are visited before continuing the traversal from any of them.

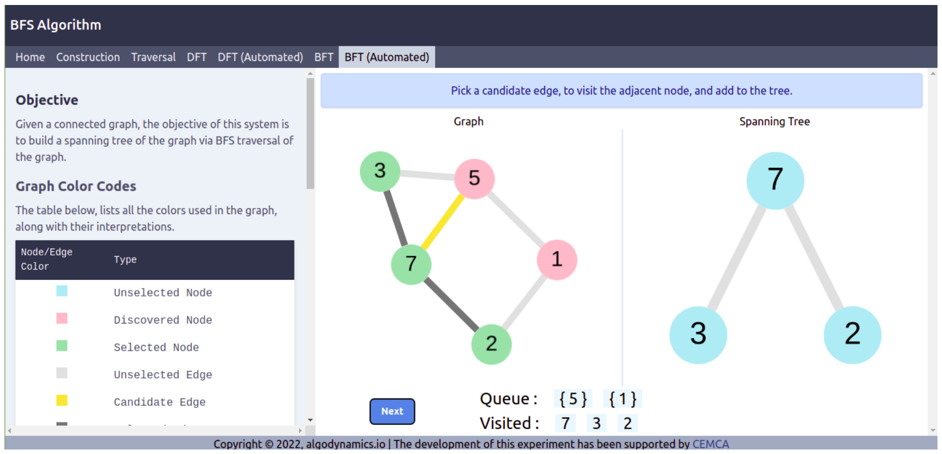
Learning Outcomes The student should be able to learn the following after performing this experiment:

1. **BFT is a particular strategy of traversal** The student should observe that the exploration of the graph is done 'level' wise: a vertices, neighbours are all visited, then the neighbour's neighbours in turn, etc. The student should observe the difference in the sequence obtained via depth-first and breadth-first strategies.

3. **Use of auxiliary data structures** The student should observe the use

of auxiliary queue data structure that helps choose the next vertex to be considered for visiting.

A screenshot of this experiment (automated version) is given in Figure 7.



Pick a candidate edge, to visit the adjacent node, and add to the tree.

Objective

Given a connected graph, the objective of this system is to build a spanning tree of the graph via BFS traversal of the graph.

Graph Color Codes

The table below, lists all the colors used in the graph, along with their interpretations.

Node/Edge Color	Type
🟦	Unselected Node
🟥	Discovered Node
🟩	Selected Node
⬜	Unselected Edge
🟨	Candidate Edge

Queue : {5} {1}
Visited : 7 3 2

Copyright © 2022, algodynamics.io | The development of this experiment has been supported by CEMCA

Figure 7: Screenshot of the Spanning Tree (Breadth First Traversal, automated version) experiment

5.2 Minimum Spanning Tree: List of Experiments and Requirements

The objective of the Minimum Spanning Tree (MST) lab is to demonstrate the process of building a minimum spanning tree for a given Graph. Another goal is to demonstrate different 'greedy' strategies of Prim's and Kruskal's to solve the MST problem.

Table 4 is a summary of the Minimum Spanning Tree experiments designed.

Table 4: Minimum Spanning Tree Experiments

No.	Problem	Strategy	Version	Status
1	Build MST	Arbitrary	Tutorial	Hosted
2	Build MST	Prim's Algorithm	Tutorial	Hosted
3	Build MST	Kruskal Algorithm	Tutorial	Hosted

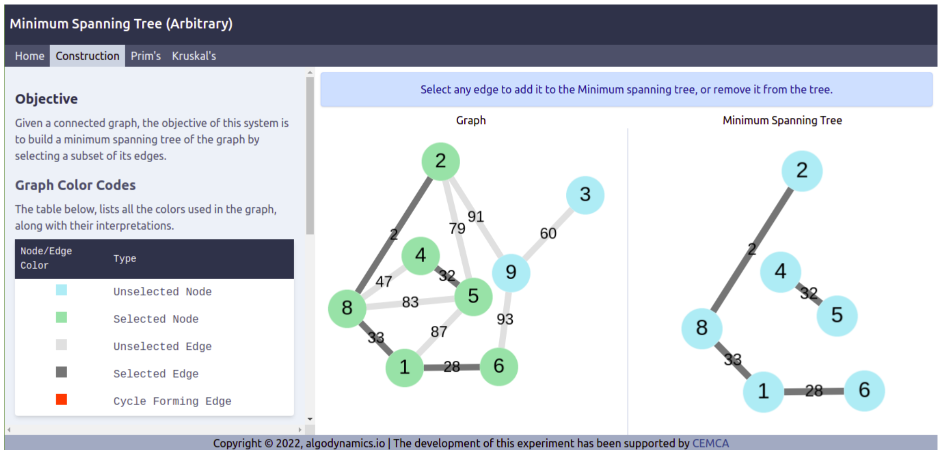
5.2.1 Minimum Spanning Tree Construction using Arbitrary strategy

Objective The objective of this experiment is to demonstrate the process of building any minimum spanning tree of a connected graph by selecting a subset of its edges. The edges may be selected in any arbitrary order; hence the intermediate structure could be a forest.

Learning Outcomes The student should be able to learn the following after performing this experiment:

1. **A minimum spanning tree is a special spanning tree of a given graph** The student should be able to observe that the minimum spanning tree is a spanning tree of the original graph, for which the sum of all the edge weights is minimum.
2. **The intermediate structures could be a forest** The student should observe that the intermediate structures could be disconnected i.e., a forest.
3. **Act as an intuition for the greedy algorithms** The student may play with this experiment for a while and develop an intuition that he needs to select those edges which minimize the sum, and this might give rise to the strategy of selecting edges in increasing order of weights.

A screenshot of this experiment is given in Figure 8.



Objective

Given a connected graph, the objective of this system is to build a minimum spanning tree of the graph by selecting a subset of its edges.

Graph Color Codes

The table below, lists all the colors used in the graph, along with their interpretations.

Node/Edge Color	Type
🟦	Unselected Node
🟩	Selected Node
⬜	Unselected Edge
⬛	Selected Edge
🟥	Cycle Forming Edge

Select any edge to add it to the Minimum spanning tree, or remove it from the tree.

Copyright © 2022, algodynamics.io | The development of this experiment has been supported by CEMCA

Figure 8: Screenshot of Minimum Spanning Tree (Arbitrary Strategy)

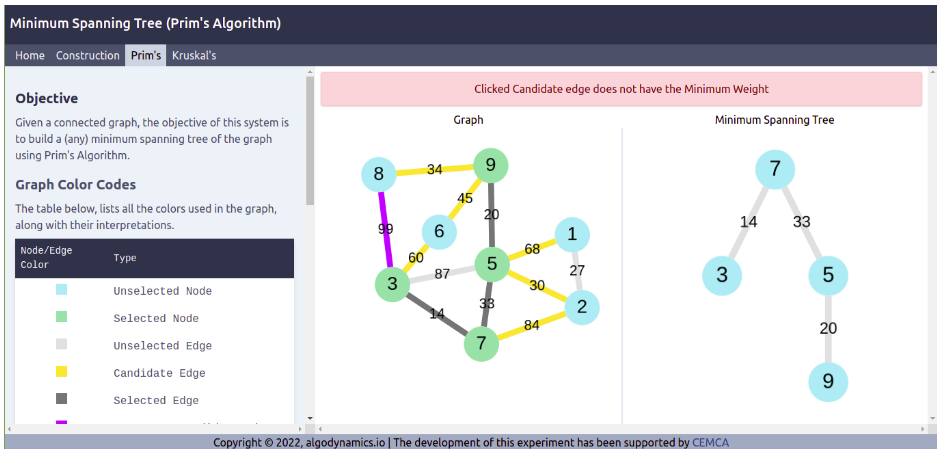
5.2.2 Minimum Spanning Tree Using Prim's Algorithm

Objective The objective of this experiment is to demonstrate the process of building any minimum spanning tree of a connected graph by using Prim's algorithm.

Learning Outcomes The student should be able to learn the following after performing this experiment:

1. **Prim's algorithm** The student should understand the Prim's algorithm.
2. **The intermediate structures are trees** The student should observe that the intermediate graphs before the completion of the algorithm have one connected component i.e. a Tree.
3. **Learn about greedy algorithm** The student should notice that in this strategy, at each step, from the chosen vertex, the edge from that vertex with the smallest weight is picked. This makes the algorithm greedy.

A screenshot of this experiment is given in Figure 9.


Minimum Spanning Tree (Prim's Algorithm)

Home Construction Prim's Kruskal's

Objective

Given a connected graph, the objective of this system is to build a (any) minimum spanning tree of the graph using Prim's Algorithm.

Graph Color Codes

The table below, lists all the colors used in the graph, along with their interpretations.

Node/Edge Color	Type
🟦	Unselected Node
🟩	Selected Node
⬜	Unselected Edge
🟨	Candidate Edge
⬛	Selected Edge

Clicked Candidate edge does not have the Minimum Weight

Copyright © 2022, algodynamics.io | The development of this experiment has been supported by CEMCA

Figure 9: Screenshot of Minimum Spanning Tree (Prim's tutorial version)

5.2.3 Minimum Spanning Tree Using Kruskal's Algorithm

Objective The objective of this experiment is to demonstrate the process of building any minimum spanning tree of a connected graph by using Kruskal's algorithm.

Learning Outcomes The student should be able to learn the following after performing this experiment:

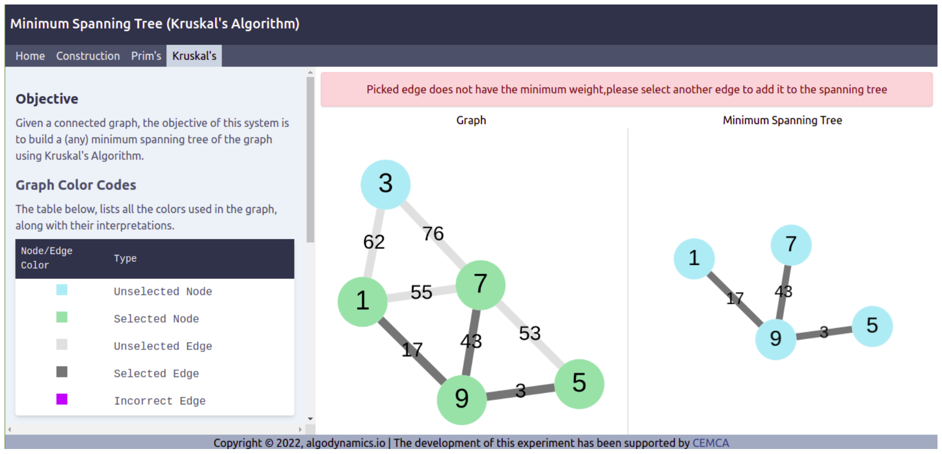
1. **Basic strategy of the Kruskal's algorithm** The student should understand the basic strategy of Kruskal's algorithm: edges are first ordered and at any time the smallest weighted edge is picked provided it does not cause a cycle.

2. **The intermediate structures could be a forest** The student should observe that the intermediate graphs before the completion of the

algorithm can have more than one connected component i.e., a Forest.

3. **Compare the Kruskal's algorithm with Prim's algorithm** The student should appreciate the difference in the strategy between the two algorithms.

A screenshot of this experiment is given in Figure 10. Note the feedback being given to the user student upon choosing an incorrect edge.


Minimum Spanning Tree (Kruskal's Algorithm)

Objective

Given a connected graph, the objective of this system is to build a (any) minimum spanning tree of the graph using Kruskal's Algorithm.

Graph Color Codes

The table below, lists all the colors used in the graph, along with their interpretations.

Node/Edge Color	Type
🟦	Unselected Node
🟩	Selected Node
⬜	Unselected Edge
⬛	Selected Edge
🟪	Incorrect Edge

Picked edge does not have the minimum weight,please select another edge to add it to the spanning tree

Copyright © 2022, algodynamics.io | The development of this experiment has been supported by CEMCA

Figure 10: Screenshot of Minimum Spanning Tree (Kruskal's tutorial version)

5.3 Shortest Path: List of Experiments and Requirements

The objective of this lab is to demonstrate the process of finding the shortest path from a single source to every node in a connected directed acyclic graph. After performing the experiments in this lab, the student is expected to achieve the following:

- Have a clear understanding of the formulation of the shortest path problem.

- Be able to identify the situations where a problem can be modelled and solved as the shortest path problem, for example, finding the shortest route between two cities on a map.
- Understand the basic operations involved in the shortest path algorithm.
- Learn about the dynamic programming paradigm in algorithms.

Table 5 is a summary of the experiments designed and implemented for the Shortest Path Algorithms lab.

Table 5: Experiments for Shortest Path

Problem	Problem	Strategy	Version
1.	Shortest Path	Arbitrary	Tutorial
2.	Shortest Path	Topological Sort	Tutorial
3.	Shortest Path	Topological Sort (V2)	Semi-automated
4.	Shortest Path	Topological Sort	Automated
5.	Shortest Path	Using Dijkstra's Algorithm	Tutorial
6.	Shortest Path	Using Dijkstra's Algorithm	Automated

5.3.1 Shortest Path using the 'Arbitrary' Strategy

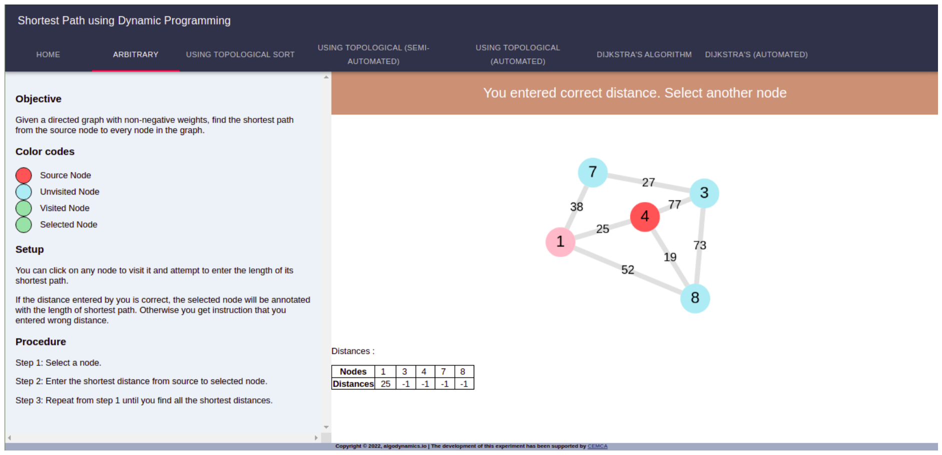
Objective The objective of this experiment is to construct a function (table) that maps each vertex v to the length of the shortest path from a given source vertex s to v. The strategy employed is 'arbitrary', i.e., the student could choose vertices in any order, but has to correctly guess the minimum distances from the source vertex.

Learning Outcomes The student should be able to learn the following after performing this experiment:

1. **Understand the shortest path problem** The student should be able to formulate and understand the shortest path problem.

2. **Develop an intuition for finding a strategy to compute shortest path** The student should conjecture strategies for patterns while finding the shortest path to each vertex from the source vertex in a systematic manner.

A screenshot of this experiment is given in Figure 11.


Shortest Path using Dynamic Programming
HOME
ARBITRARY
USING TOPOLOGICAL SORT
USING TOPOLOGICAL (SEMI-AUTOMATED)
USING TOPOLOGICAL (AUTOMATED)
DIJKSTRA'S ALGORITHM
DIJKSTRA'S (AUTOMATED)
7
27
3
38
77
4
25
1
73
19
52
8

You entered correct distance. Select another node

Objective

Given a directed graph with non-negative weights, find the shortest path from the source node to every node in the graph.

Color codes

- 🔴 Source Node
- ● Unvisited Node
- 🟢 Visited Node
- 🟢 Selected Node

Setup

You can click on any node to visit it and attempt to enter the length of its shortest path.

If the distance entered by you is correct, the selected node will be annotated with the length of shortest path. Otherwise you get instruction that you entered wrong distance.

Procedure

Step 1: Select a node.

Step 2: Enter the shortest distance from source to selected node.

Step 3: Repeat from step 1 until you find all the shortest distances.

Distances :

Nodes	1	3	4	7	8
Distances	25	-1	-1	-1	-1

Copyright © 2022, algodynamics.io | The development of this experiment has been supported by CEMCA

Figure 11: Screenshot of Shortest Path Experiment (Arbitrary Strategy)

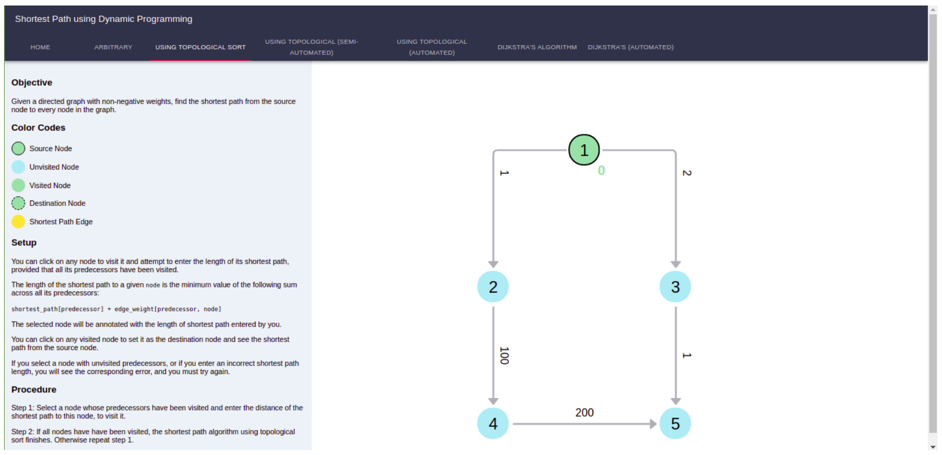
5.3.2 Shortest Path using Topological Sort And Dynamic Programming

Objective The objective of this experiment is to construct a function (table) that maps each vertex v to a number which is the length of the shortest path from a given source vertex s to v. The strategy employed is to construct these tables using Topological Sorting and Dynamic Programming.

Learning Outcomes The student should be able to have the following concepts reinforced after performing this experiment:

1. **Topological sorting** The student should observe the role of topological sorting in the experiment (sorting of weighted edges).
2. **Dynamic programming** The student should be able to observe the strategy of dynamic programming being employed in the experiment.
3. **Optimal substructure property** The student should observe how the dynamic programming strategy helps maintain the optimal substructure property of the problem: the paths computed are shortest for the subgraph considered so far.

A screenshot of this experiment is given in Figure 12.


Shortest Path using Dynamic Programming
HOME
ARBITRARY
USING TOPOLOGICAL SORT
USING TOPOLOGICAL (SEMI-AUTOMATED)
USING TOPOLOGICAL (AUTOMATED)
DIJKSTRA'S ALGORITHM
DIJKSTRA'S (AUTOMATED)
1
0
2
3
4
5
1
2
100
1
200

Objective

Given a directed graph with non-negative weights, find the shortest path from the source node to every node in the graph.

Color Codes

●	Source Node
●	Unvisited Node
●	Visited Node
◌	Destination Node
●	Shortest Path Edge

Setup

You can click on any node to visit it and attempt to enter the length of its shortest path, provided that all its predecessors have been visited.

The length of the shortest path to a given `node` is the minimum value of the following sum across all its predecessors:

```
shortest_path[predecessor] + edge_weight[predecessor, node]
```

The selected node will be annotated with the length of shortest path entered by you.

You can click on any visited node to set it as the destination node and see the shortest path from the source node.

If you select a node with unvisited predecessors, or if you enter an incorrect shortest path length, you will see the corresponding error, and you must try again.

Procedure

Step 1: Select a node whose predecessors have been visited and enter the distance of the shortest path to this node, to visit it.

Step 2: If all nodes have have been visited, the shortest path algorithm using topological sort finishes. Otherwise repeat step 1.

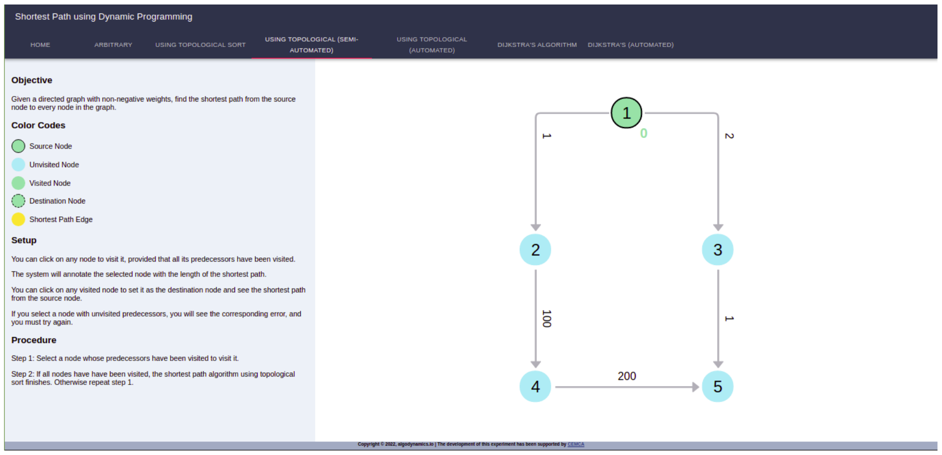
Figure 12: Screenshot of Shortest Path Experiment (topological sort strategy, tutorial version)

5.3.3 Shortest Path using Topological Sort And Dynamic Programming (Semi-Automated)

Objective The objective of this experiment is to construct a function (table) that maps each vertex v to a number which is the length of the shortest path from a given source vertex s to v. The strategy employed is to construct is Topological Sorting and Dynamic Programming. This version is semi-automated: the student chooses the next vertex and the system automatically computes the shortest path from the source to the vertex.

Learning Outcomes The learning outcomes are the same as the previous version, except that the student is now expected to focus on the choosing of the next vertex. The calculations of the shortest path are automated.

A screenshot of this experiment is given in Figure 13.


Shortest Path using Dynamic Programming
HOME
ARBITRARY
USING TOPOLOGICAL SORT
USING TOPOLOGICAL (SEMI-AUTOMATED)
USING TOPOLOGICAL (AUTOMATED)
DIJKSTRA'S ALGORITHM
DIJKSTRA'S (AUTOMATED)
1
0
1
2
2
3
100
1
4
200
5

Objective

Given a directed graph with non-negative weights, find the shortest path from the source node to every node in the graph.

Color Codes

- Source Node
- Unvisited Node
- Visited Node
- Destination Node
- Shortest Path Edge

Setup

You can click on any node to visit it, provided that all its predecessors have been visited.

The system will annotate the selected node with the length of the shortest path.

You can click on any visited node to set it as the destination node and see the shortest path from the source node.

If you select a node with unvisited predecessors, you will see the corresponding error, and you must try again.

Procedure

Step 1: Select a node whose predecessors have been visited to visit it.

Step 2: If all nodes have have been visited, the shortest path algorithm using topological sort finishes. Otherwise repeat step 1.

Copyright © 2022, algodynamics.io | The development of this experiment has been supported by CEMCA

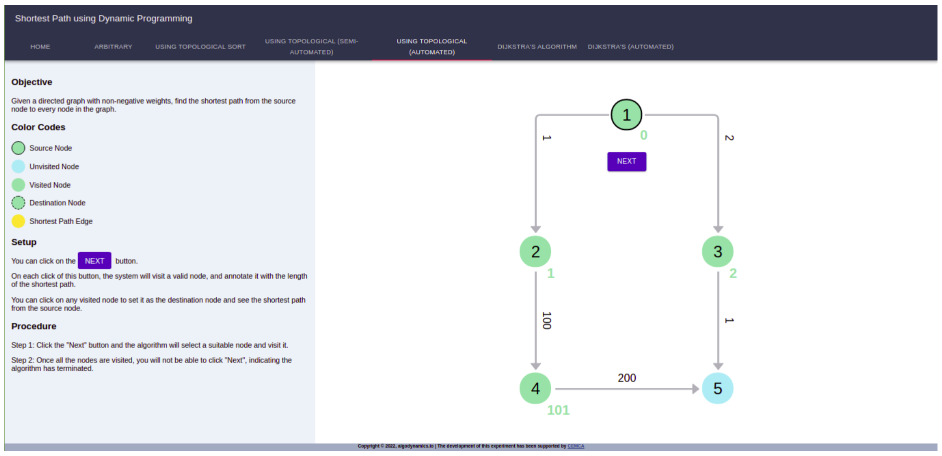
Figure 13: Screenshot of Shortest Path Experiment (topological sort strategy, semi-automated version)

5.3.4 Shortest Path Using Topological Sort And Dynamic Programming (Automated)

Objective The objective of this experiment is to automatically construct a function (tables) that maps each vertex v to a number which is the length of the shortest path from s to v. The strategy to construct these tables uses Topological Sorting and Dynamic Programming approach. The experiment is automated and demonstrates the steps involved in the process.

Learning Outcomes The student should be able to recall the stragegy of using topological sorting, specially the steps involved in the algorithm and the order in which they need to be executed.

A screenshot of this experiment is given in Figure 14.


Shortest Path using Dynamic Programming
HOME
ARBITRARY
USING TOPOLOGICAL SORT
USING TOPOLOGICAL (SEMI-AUTOMATED)
USING TOPOLOGICAL (AUTOMATED)
DIJKSTRA'S ALGORITHM
DIJKSTRA'S (AUTOMATED)
NEXT
1
0
2
1
3
2
4
101
5
100
200

Objective

Given a directed graph with non-negative weights, find the shortest path from the source node to every node in the graph.

Color Codes

- Source Node
- Unvisited Node
- Visited Node
- Destination Node
- Shortest Path Edge

Setup

You can click on the NEXT button.

On each click of this button, the system will visit a valid node, and annotate it with the length of the shortest path.

You can click on any visited node to set it as the destination node and see the shortest path from the source node.

Procedure

Step 1: Click the "Next" button and the algorithm will select a suitable node and visit it.

Step 2: Once all the nodes are visited, you will not be able to click "Next", indicating the algorithm has terminated.

Copyright © 2022, algodynamics.io | The development of this experiment has been supported by CEMCA

Figure 14: Screenshot of Shortest Path Experiment (topological sort strategy, automated version)

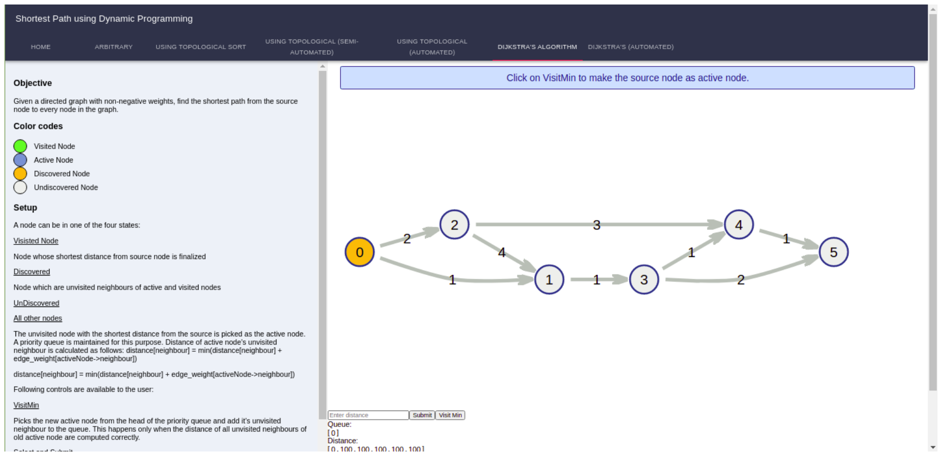
5.3.5 Shortest Path Using Dijkstra's Algorithm

Objective The objective of this experiment is to construct a function (tables) that maps each vertex v to a number which is the length of the shortest path from s to v. The strategy to construct these tables uses topological sorting and dynamic programming approach using Dijkstra's algorithm. The graph here can be any connected directed graph.

Learning Outcomes The student should be able to learn the following after performing this experiment:

1. **Learn about Dijkstra's algorithm** The student should be able to learn about the Dijkstra's algorithm and the operations involved in it.
2. **Compare the Dijkstra's algorithm with previous algorithms** The student should be able to compare the optimal substructure of both algorithms, and should also be able to compare their running time complexities.

A screenshot of this experiment is given in Figure 15.



Objective

Given a directed graph with non-negative weights, find the shortest path from the source node to every node in the graph.

Color codes

🟢 Visited Node
🔵 Active Node
🟠 Discovered Node
⚪ Undiscovered Node

Setup

A node can be in one of the four states:

Visisted Node

Node whose shortest distance from source node is finalized

Discovered

Node which are unvisited neighbours of active and visited nodes

UnDiscovered

All other nodes

The unvisited node with the shortest distance from the source is picked as the active node. A priority queue is maintained for this purpose. Distance of active node's unvisited neighbour is calculated as follows: distance[neighbour] = min(distance[neighbour] + edge_weight[activeNode->neighbour])

distance[neighbour] = min(distance[neighbour] + edge_weight[activeNode->neighbour])

Following controls are available to the user:

VisitMin

Picks the new active node from the head of the priority queue and add it's unvisited neighbour to the queue. This happens only when the distance of all unvisited neighbours of old active node are computed correctly.

Click on VisitMin to make the source node as active node.

Queue:
[0]
Distance:
[0 , 100 , 100 , 100 , 100 , 100]

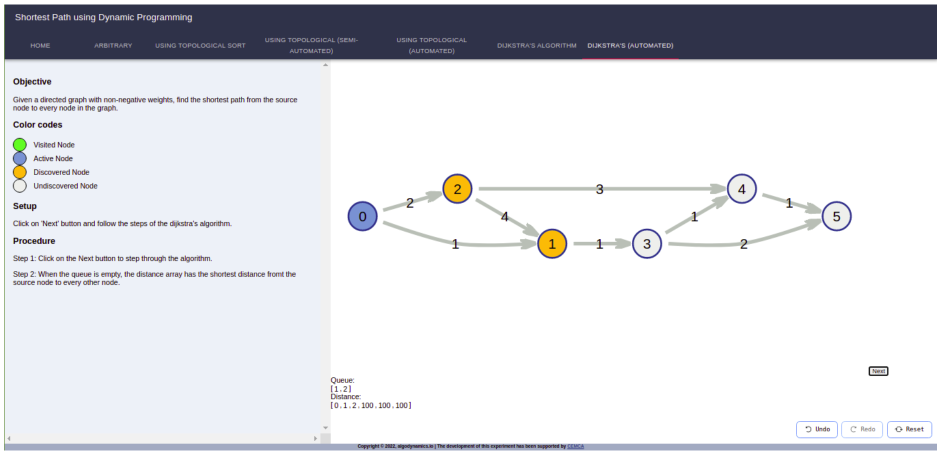
Figure 15: Screenshot of Shortest Path Experiment (Dijskstra's algorithm, tutorial version)

5.3.6 Shortest Path Using Dijkstra's algorithm(Automated)

Objective The objective of this experiment is to demonstrate the steps involved in the Dijkstra's algorithm. The graph in this experiment could be Connected Directed Graph.

Learning Outcomes The student should be able to review the understanding of Dijkstra's algorithm by viewing each step of the algorithm after pressing the 'next' button.

A screenshot of this experiment is given in Figure 16.



Objective

Given a directed graph with non-negative weights, find the shortest path from the source node to every node in the graph.

Color codes

🟢 Visited Node
🔵 Active Node
🟠 Discovered Node
⚪ Undiscovered Node

Setup

Click on 'Next' button and follow the steps of the dijkstra's algorithm.

Procedure

Step 1: Click on the Next button to step through the algorithm.

Step 2: When the queue is empty, the distance array has the shortest distance fromt the source node to every other node.

```
Queue:
[1,2]
Distance:
[0,1,2,100,100,100]
```

Copyright © 2022, algodynamics.io | The development of this experiment has been supported by CEMCA

Figure 16: Screenshot of Shortest Path Experiment (Dijskstra's algorithm, automated version)

6 Implementation

All the experiments have been implemented using HTML, CSS, Javascript and Elm. The code for the experiments is available on gitlab.

7 Conclusions

The project required the implementation of 12 experiments spanning three different labs in Data Structures and Algorithms: Graph Traversal, Minimum Spanning Tree and (Single Source) Shortest Path. A total of 14 experiments were delivered.

Going forward, two intertwined efforts should to be undertaken to establish impact of this work. First, a larger repertoire of experiments should be built, both in data structures and algorithms, and other domains of computing like programming language execution models, concurrent and distributed algorithms. Second, faculty development workshops for capacity building and awareness should be conducted on a large scale. These workshop will expose faculty to algodynamics, the theoretical foundation of the interactive approach illustrated via the experiments reported here.

7.1 Acknowledgements

The implementation of the experiments were carried out by the following students of the Algodynamics research team at IIIT Hyderabad: Archit Goyal, Prince Varshney, Ganne Jyotheeswar, Anukul Sangwan and Mrityunjay Kumar.

8 Bibliography

References

[1] Venkatesh Choppella, Kasturi Viswanath, and Mrityunjay Kumar. "Algodynamics: Algorithms as systems". In: *2021 IEEE Frontiers in Education Conference (FIE)*. 2021, pp. 1–9. DOI: 10.1109/FIE49875.2021.9637441.

[2] Thomas H Cormen et al. *Introduction to algorithms*. MIT press, 2009.

[3] Robert Sedgewick. *Algorithms*. Pearson Education, 1988.

[4] Niklaus Wirth. "Program Development by Stepwise Refinement". In: *Communications of the ACM* 14.4 (1971), pp. 221–227.