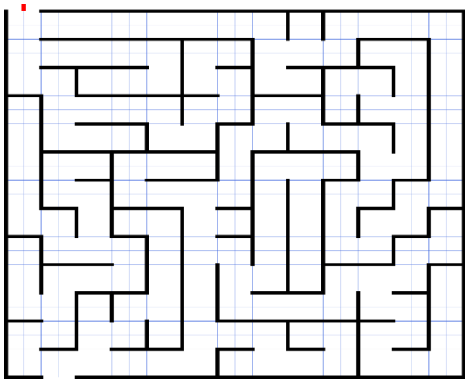
FACULTY OF ENGINEERING AND APPLIED SCIENCE

Department of Electrical, Computer and Software Engineering

SOFE 2715 Data Structure / Group Project (20%)

Project 1

One way to construct a ***maze*** starts with an $n \times n$ grid such that each grid cell is bounded by four unit-length walls. We then remove two boundary unit-length walls, to represent the start and finish. For each remaining unit-length wall not on the boundary, we assign a random value and create a graph G, called the ***dual***, such that each grid cell is a vertex in G and there is an edge joining the vertices for two cells if and only if the cells share a common wall. The weight of each edge is the weight of the corresponding wall. We construct the maze by finding a minimum spanning tree T for G and removing all the walls corresponding to edges in T. Write a program that uses this algorithm to generate mazes and then solves them. Minimally, your program should draw the maze and, ideally, it should visualize the solution as well.



Project 2

Decremental Tree Connectivity: Consider the following problem. You're given an initial tree T. We will begin deleting edges from T and, as we do, we'd like to be able to efficiently determine whether arbitrary pairs of nodes are still connected in the graph. This is an example of a dynamic graph algorithm: in the case where we knew the final tree, it would be easy to solve this problem with some strategic breadth-first searches, but it turns out to be a lot more complex when the deletes and queries are intermixed. By using a number of techniques similar to the Four Russians speedup in Fischer-Heun, it's possible to solve this problem with linear preprocessing time and constant deletion and query times.

Project 3

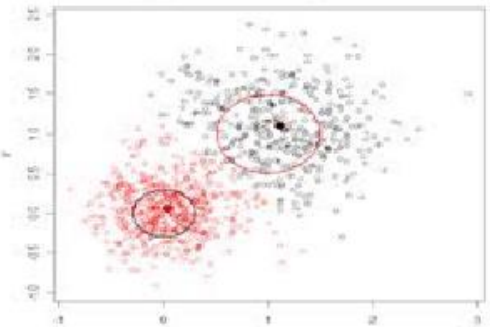
A finger tree is a B-tree augmented with a "finger" that points to some element. The tree is then reshaped by pulling the finger up to the root and letting the rest of the tree hang down from the finger. These trees have some remarkable properties. For example, when used in a purely functional setting, they give an excellent implementation of a double-ended queue with amortized efficient insertion and deletion.

Project 4

Write a program that can play Tic-Tac-Toe effectively. To do this, you will need to create a ***game tree*** *T*, which is a tree where each position corresponds to a ***game configuration***, which, in this case, is a representation of the Tic-Tac-Toe board. The root corresponds to the initial configuration. For each internal position *p* in *T*, the children of *p* correspond to the game states we can reach from *p*'s game state in a single legal move for the appropriate player, *A* (the first player) or *B* (the second player). Positions at even depths correspond to moves for *A* and positions at odd depths correspond to moves for *B*. Leaves are either final game states or are at a depth beyond which we do not want to explore. We score each leaf with a value that indicates how good this state is for player *A*. We construct the entire game tree and score leaves as +1, 0, −1, indicating whether player *A* has a win, draw, or lose in that configuration. A good algorithm for choosing moves is ***minimax***. In this algorithm, we assign a score to each internal position *p* in *T*, such that if *p* represents *A*'s turn, we compute *p*'s score as the maximum of the scores of *p*'s children (which corresponds to *A*'s optimal play from *p*). If an internal node *p* represents *B*'s turn, then we compute *p*'s score as the minimum of the scores of *p*'s children (which corresponds to *B*'s optimal play from *p*).

Project 5

K-Means Clustering: The K-means clustering algorithm is one of the most widely used methods for finding clusters (patterns) within data and is widely used in the field of Machine Learning. Your task is to implement the K-means clustering method algorithm as well as to use data structures to ensure that the algorithm performs well even with large amounts of data.



Project 6

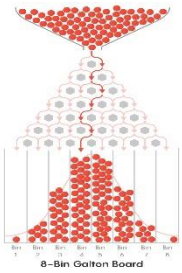
Dictionaries are widely used and we often take spellchecking for granted. You will be required to create a dictionary that not only provides the ability to search a word, but also to add new words, delete existing words, and edit the definition of a word in the dictionary. For this project you must ensure that you meet all of the following criteria.

1. You must implement a dictionary that contains not only words but also their definition.
2. You will be provided with a list of six different Comma-Separated Values (CSV) files containing words and their definitions, as well as for each a list of operations to perform.
3. Your solution must be able to read in the data provided and create the dictionary as well as process the accompanying file with the list of operations to perform.
4. Your program should also have a CLI so that a user can perform each of the operations from the terminal such as searching for a word to retrieve the definition.

Project 7

Implement a Galton Board Simulation: The Galton board simulates Central Limit Theorem (CLT), which roughly states that given enough independent random variables the total aggregate outcome of their results converges to an approximately normal distribution. For this project you must make sure to meet the following criteria.

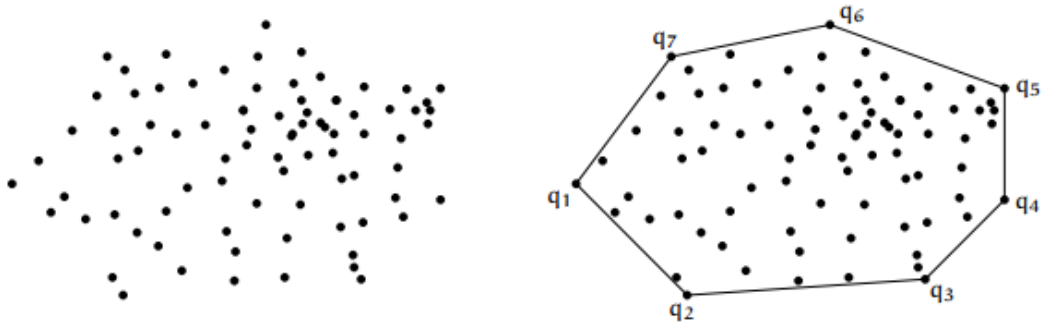
1. You will be provided with six different numbers for the amount of random balls to drop (e.g. 1000, 10000, etc.).
2. For each random ball dropped (random value generated) you must show an animation of the Galton Board being created with each additional random point being generated. **You do not need to animate balls dropping,** but you must animate showing how the Galton Board statistical distribution is being created with each additional random data generated.



Project 8

Finding the Convex Hull: The convex hull consists of finding the set of outermost points that encompass within them all of the other points. The Convex Hull problem can be thought of visually as a set of nails partially hammered into a board, where a rubber band has been wrapped around the outside edge of nails. For this project you must implement a solution to the convex hull problem that meets all of the following criteria

1. You will be given six different sets of points (x and y coordinates) in Comma-Separated Values (CSV) files.
2. While you may base your solution on any of the well-known algorithms for solving the Convex Hull problem, you must design and **implement the algorithms in your own code** and cannot use existing libraries or tools to solve the problem.
3. You must plot an image showing the collection of points provided as well as the Convex Hull found that encloses all of the points within it. Your results should look similar to the following image.



(a) Input. (b) Output.

Project 9

Sorting: There are many different ways to sort data. In this project you will compare several sorting algorithms. You will need to:

- Implement a hash (bucket) sort.
- Implement shell sort.
- Implement sorting using an ordered tree
- Write a test program that will compare the efficiencies of selection sort, insertion sort, quick sort, merge sort, heap sort, and the new sorts you have implemented.

Project 10

Graphs: there are many possible projects involving graphs, each beginning by implementing a graph class and operations (via external or internal iterators) allowing for depth-first and breadth-first traversals.

Varying implementations: Implement several versions of a graph class using adjacency-matrix, edge-list, and edge-set representations. Write a test program to compare the relative efficiencies for these implementations.

- Write a reachability operation that given two nodes, *i* and *j*, determines whether there exists a path in the graph from *i* to *j*. Write an operation that given node *i*, return a list or

set of all the nodes reachable from *i*. Write an operation that computes a reachability matrix so future reachability queries can be answered in constant time. Include a test program.

- Implement a version of the graph class (perhaps by using inheritance) that allows for information to be stored at the edges as well as the nodes. A *weighted graph* is a graph with a number associated with each edge. Recall, a *spanning tree* is a tree consisting of all the nodes in a graph and a collection of edges from the graph that connect the nodes to make a tree (no loops). A *minimum spanning tree* of a weighted graph is a spanning tree of a graph such that the sum of all the weights along the edges of the tree is less than or equal to the sum of the weights along the edges of any other spanning tree for that graph. Write an operation to compute the minimum spanning tree of a weighted graph.
- A *shortest path* between nodes *i* and *j* in a graph is a path from *i* to *j* such that the sum of the weights of the edges along the path is less than or equal to the sum of the weights along the edges of any other path from *i* to *j*. (In a graph without edge weights, we can pretend each edge has a weight of 1 and then a shortest path from *i* to *j* refers to a path from *i* to *j* with the fewest edges.) Write an operation to compute the shortest path between any two nodes in a graph.

Deliverables

For your project, you must submit a zip file containing your report in addition to all of the source code and related data for your project. You **must also ensure that you provide a README file** that briefly explains how to run the project so that the instructor is able to easily evaluate your results. (The code should be written in Java)

1. You must provide your solution (all source codes) for your project problems, your solution must meet all of the criteria specified for that project. Ensure that you also include all of the project implementation (solutions, images, results, etc.) in addition to your source code.

2. During the project presentation you will demonstrate your solution and shows the performance.
   - Demonstration of a working solution of the project problems **[5 marks]**
   - Demonstration of the performance (time complexity, plots) **[5 marks]**

3. You must also provide a detailed report that has an introduction and explains each of the problems, shows the time complexity analysis of your algorithms **[5 marks].**
   The report should include:
   - Introduction and explanation of the project's problem as well as the challenges and your solution.
   - Algorithm/s you used (pseudocode) and time complexity analysis for your algorithm/s.
   - Performance plots for each project solution comparing the execution time for each of the six datasets.
   - Conclusion and summary of your solution and your findings.

4. Self and peer evaluation: 3 marks