Engineering Design Method

Discrete Structures and Computing I

Members:

Ommhes Samuel Leon Diaz- A00406645

Luis Fernando Soto Bedoya- A00406591

Miguel Pérez Ojeda-A00407054

Story

In a highly surveilled digital world, a solitary hacker must infiltrate a network of servers distributed across different virtual cities to recover secret keys without being detected. This network is composed of nodes (servers) and transmission channels (links) with various latencies.

The player, controlling the hacker, will have to navigate this network intelligently by applying search algorithms, shortest paths, and minimum spanning trees, all through an interactive graphical interface that simulates this hostile environment.

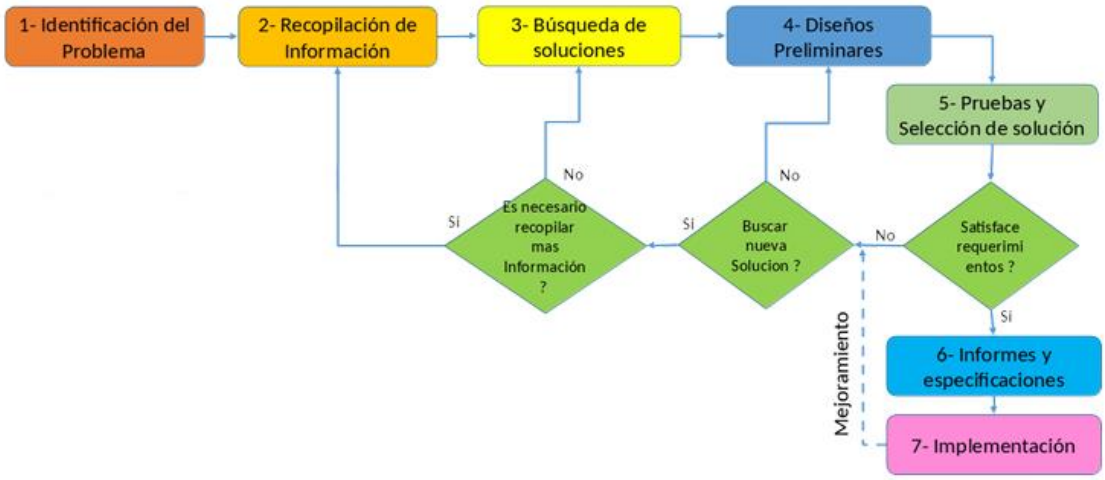
Problem Context

As part of the course "Discrete Structures and Computing I," the challenge is to develop an interactive game whose logic and structure are based on graphs. The objective of the project is to practically apply concepts learned in graph theory, including traversals, optimal path searches, and minimum spanning trees.

The system must implement two functional versions of the Graph ADT, each with its own unit tests, and include a graphical interface that allows the user to visualize and interact with the graph. The solution must include at least 50 vertices and 50 edges and apply at least two course algorithms, such as BFS, DFS, Dijkstra, Floyd-Warshall, Prim, or Kruskal.

Solution Development

To solve the design and development of the interactive game *The Traveling Hacker*, we will use the Engineering Method, applying a systematic and organized approach that allows modeling, building, and validating a digital network represented through graphs, and navigating it optimally using specialized algorithms.



Example A of the Engineering Method. Based on *Introduction to Engineering* by Paul H. Wright.

PHASE 1: PROBLEM IDENTIFICATION

The system seeks to address the need to create an interactive game that simulates navigating a digital network using graphs. In this context, the player must act as a hacker who infiltrates a network of virtual servers to recover secret keys while optimizing their movement.

The challenge lies in integrating the theoretical concepts of graph structures into a practical and visual environment, allowing a deeper understanding of how algorithms like BFS, DFS, Dijkstra, Floyd-Warshall, Prim, and Kruskal work.

System requirements include:

- Visualizing the graph of the digital network, showing nodes and connections with weights.
- Selecting start and end nodes to perform traversals or calculate routes.
- Exploring the network using BFS and DFS.
- Calculating optimal paths using Dijkstra or Floyd-Warshall..
- Switching between adjacency list and adjacency matrix implementations.
- Displaying routes, keys found, and movement costs.
- Ensuring the graph has at least 50 vertices and 50 edges.
- Running automatic unit tests.

PHASE 2: INFORMATION GATHERING

To develop the interactive system for the game *The Traveling Hacker*, information was gathered in the following key areas:

- **Graph Theory**: Concepts such as vertices, edges, weights, directed/undirected graphs, and representations (lists and matrices) were studied as a foundation.
- **Graph Algorithms**: BFS and DFS (for exploration and key searches), Dijkstra and Floyd-Warshall (for shortest paths), and Prim and Kruskal (for minimum spanning trees) were examined.
- **Graph Representation**: Two versions of the Graph ADT were defined:
    - Adjacency list (efficient in sparse graphs)
    - Adjacency matrix (useful in dense graphs and for quick lookups)
- **Interactive Graphical Interfaces**: JavaFX principles were reviewed to:
    - Visualize graphs and paths
    - Select nodes and apply algorithms
    - Switch graph implementations
- **Unit Testing**: Strategies were explored to test both structural operations of the graph and the correct behavior of the algorithms.

Definiciones:

Fuente

https://es.wikipedia.org/wiki/Grafo

https://es.wikipedia.org/wiki/B%C3%BAsqueda_en_anchura

https://es.wikipedia.org/wiki/B%C3%BAsqueda_en_profundidad

https://es.wikipedia.org/wiki/Algoritmo_de_Dijkstra

https://es.wikipedia.org/wiki/Algoritmo_de_Floyd-Warshall

https://es.wikipedia.org/wiki/Algoritmo_de_Prim

https://es.wikipedia.org/wiki/Algoritmo_de_Kruskal

https://openjfx.io/

Graph: A data structure that represents a set of objects (vertices or nodes) connected by links (edges). It is used to model relationships between entities, such as server networks.

Adjacency List: A graph representation in which each vertex has a list of vertices to which it is connected. It is memory efficient for sparse graphs.

Adjacency Matrix: A matrix-based representation of a graph, where rows and columns indicate whether two vertices are connected. Useful for dense graphs.

BFS (Breadth-First Search): An algorithm that explores all the neighbors of a node before moving to the next level. Useful for finding shortest paths in unweighted graphs.

DFS (Depth-First Search): An algorithm that explores as far down a path as possible before backtracking. It is used for full network traversal or detecting cycles.

Dijkstra: An algorithm that calculates the shortest path from a source node to all others in graphs with non-negative weights.

Floyd-Warshall: An algorithm that finds the shortest paths between all pairs of nodes. Ideal when multiple path queries are needed.

Prim: An algorithm for building a minimum spanning tree (MST) from a starting node by adding the lowest-weight edge that connects to the tree.

Kruskal: An algorithm for building an MST by adding the lowest-weight edges without forming cycles, especially when edges are sorted by weight.

Minimum Spanning Tree (MST): A subgraph that connects all vertices in a graph with the lowest total cost possible, without cycles.

JavaFX: A Java library for creating modern and interactive graphical interfaces. Used to visually represent the graph and allow user interaction.

PHASE 3: CREATIVE SOLUTION GENERATION

For the development of the game *The Traveling Hacker*, several solution alternatives were considered. Each represents a different approach to modeling and resolving navigation through the network of virtual servers:

- **Alternative 1: Use of static traditional structures**

  Arrays and simple lists are used to represent node connections. This approach simplifies implementation but severely limits scalability and flexibility.

- **Alternative 2: Use of customized dynamic structures**

  The system is designed using data structures such as stacks, queues, and hash tables to store and manage node connections and data.

- **Alternative 3: Graph modeling with specialized algorithms**

  The system is built using a weighted directed graph structure, represented by both an adjacency list and matrix.

PHASE 4: FROM IDEA FORMULATION TO PRELIMINARY DESIGN

In this phase, the alternatives generated were analyzed, and those not viable for the system's needs were discarded.

Discarded Alternative:

- **Alternative 1: Use of static traditional structures** Although easy to implement, this option lacks the scalability and flexibility required for representing a complex and evolving network, making it inadequate for the game's design.

Selected Alternatives:

- **Alternative 2: Use of customized dynamic structures** This alternative allows greater flexibility and efficiency in data management. Structures like stacks, queues, and hash tables support user interactions, temporary key storage, and control of information flow dynamically, adapting to player behavior and system changes.
- **Alternative 3: Graph modeling with specialized algorithms** This is the most complete and robust option, aligning well with the core of the game. Weighted directed graphs accurately model the server network, and algorithms like Dijkstra and BFS are essential for optimal pathfinding and exploration.

Preliminary Designs:

- ***Alternative 2: Use of customized dynamic structures***
    - **Stacks:** for allowing undoing recent moves.
    - **FIFO Queues:** for managing event or node visit order.
    - **Priority Queues:** to prioritize more efficient or lower-latency paths.
    - **Hash Tables:** to store nodes, retrieved keys, and visited paths efficiently.
- ***Alternative 3: Graph modeling with specialized algorithms***
    - **Weighted Graph ADT using:**
        - Adjacency list
        - Adjacency matrix
    - **Algorithms:**
        - bfsPath() for exploration and key search
        - dijkstraPath() for optimal pathfinding
        - reconstructPath() for visual path reconstruction
    - **JavaFX Interface for:**
        - Selecting origin/destination nodes
        - Displaying paths, costs, and retrieved keys
        - Modifying the graph in real time
    - **Scalable:** design supporting at least 50 nodes and 50 edges

FASE 5: EVALUACIÓN Y SELECCIÓN DE LA MEJOR SOLUCION

Based on the previously selected alternatives, a comparative evaluation was conducted using the following criteria: data processing efficiency, implementation complexity, alignment with academic objectives, and estimated development time.

Alternatives	Efficiency	Complexity	Alignment	Development Time	Total
Use of customized dynamic structures	4	3	3	3	13
Graph modeling with specialized algorithms	5	4	5	4	18

Selection:

After analyzing the scores obtained by each alternative, the graph modeling with specialized algorithms was chosen as the most appropriate solution for the system. This alternative received the highest total score in terms of efficiency, academic alignment, and development feasibility. It also allows a faithful and flexible representation of the digital network required by the game. Although dynamic structures provide a functional solution, their limited alignment with pathfinding logic restricts their potential to meet the project's objectives. For these reasons, the graph-based alternative using algorithms like Dijkstra is selected as the final solution for implementing the core logic of the interactive game The Traveling Hacker.