**Integrative Task #2 Computation and Discrete Structures 1**

**Engineering Method:**

1. **Identification of the problem:**

**Context:**

It is necessary to design a system that allows the implementation of a game that alludes to a Maze. However, with some modifications. This game will allow the user to play with another player, creating a 1 vs 1 match. They will compete in a type of box game, where each participant can only move toward his adjacent vertexes. Based on the fact that this game needs to be done through the implementation of graphs, each square of the board where players can fall will be modeled as if it were a vertex. To avoid the fact that the implementation of the graph structure is wasted, in other words, where the weight of each edge of the vertexes is always one, the game will be modeled with different weights between vertexes. In this way, it will provide a better playability and allow the algorithms that we will implement to be efficient.

**Needs and Symptoms:**

* Board Representation with Graphs:

Need for an efficient data structure to represent the game board as a graph.

Implementation of nodes to represent squares and edges to model the connections between them. That is, it is necessary to be able to add vertices and to add edges in order to be able to carry out a graphical representation of the game board.

* Graph Search Algorithms:

Need for graph search algorithms to determine player paths and movements along the board.

* Event and mini-game management:

Development of a system to manage in-game events, such as activation of mini-games or special events upon reaching certain squares.

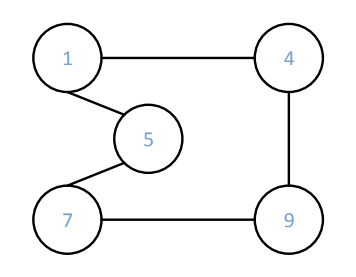
**Problem definition:**

A 1 vs 1 game needs to be created. It must allow players to move to the different vertexes but only to the adjacent ones, until the first one reach the exit of the labyrinth. However, it will also be evaluated the smallest weighting as the winner one, since the movement between vertexes (edge) does not always represent 1 weight.

1. **Compilation of information**

**Graph:** Graphs find applications across diverse fields, including electrical engineering, chemistry, industrial engineering, compilers, operating systems, organization, and information retrieval. Serving as data structures, graphs function as abstract data types frequently employed for problem modeling. A graph is a set of objects or physical entities with inherent relationships. Comprising vertices, sometimes referred to as nodes (representing the objects), and a collection of edges that depict connections between vertices, graphs are versatile structures known as arcs or edges in English.

In essence, graphs are visually represented by vertices linked by lines or edges, which can be either directed or undirected. Describing a graph G involves specifying the pair of vertices and edges, denoted as G = (V, E)



In the image above, the previously described relationship can be observed, where the vertices are denoted as V = {1, 4, 5, 7, 9}, and the set of edges is given by A = {(1, 4), (4, 1), (5, 1), (1, 5), (7, 9), (9, 7), (7, 5), (5, 7), (4, 9), (9, 4)}. It's important to note that this graph is undirected.

On the other hand, there are also directed graphs, which means they have arrows indicating the direction of the relationships between vertices or nodes. For instance, Figure 2 depicts the directed graph G = (V, A) as follows: its vertices are V = {A, E, I, O, U}, and its edges are A = {(A, O), (O, U), (E, A), (E, I), (I, E)}.

Another interesting property of graphs is that they can have priorities or weights assigned to their edges. In other words, the edges are associated with a magnitude known as weight.

Source:<https://posgrados.inaoep.mx/archivos/PosCsComputacionales/Curso_Propedeutico/Programacion_Estructuras_Datos/Capitulo_10_Grafos.pdf>

**Directed Graph:** A directed graph, also known as a digraph, is a type of graph where edges have a specific direction. This directional aspect is typically represented by arrows on the edges. More formally, if v and w are vertices, an edge is an unordered pair {v, w}, whereas a directed edge, referred to as an arc, is an ordered pair (v, w) or (w, v). The arc (v, w) is visually depicted as an arrow pointing from vertex v to vertex w. In cases where a graph includes both arcs (v, w) and (w, v), they are not considered as a "multiple edge" since each arc is distinct. It is possible to have multiple arcs between the same vertices; for example, an arc (v, w) may be included multiple times in the multiset of arcs. Similar to undirected graphs, a digraph is termed simple if it lacks loops or multiple arcs between the same pair of vertices.

A graph serves as both a mathematical and visual representation of a set comprising vertices and edges. This set is non-empty, and the edges establish connections between the nodes or vertices. Nodes can be understood as representations of objects, while edges denote the connections between these objects. The use of an arrow (→) signifies the direction in which the graph is traversed. Additionally, edges in a graph may sometimes carry weights, indicating the strength or magnitude of each connection between vertices.

Formal definitions for different types of graphs exist based on edge characteristics, leading to the creation of various terminologies. The definitions may vary in different applications, providing flexibility in how nodes and edges are defined. For example, in modeling a social network's friendships, individuals are represented as nodes in the graph, while friendships are depicted by edges, showcasing the versatility of graphs in various applications.

Source:<https://www.javatpoint.com/directed-and-undirected-graph-in-discrete-mathematics>

<https://www.whitman.edu/mathematics/cgt_online/book/section05.11.html#:~:text=A%20directed%20graph%2C%20also%20called,or%20(w%2Cv)>.

**Dijkstra:**

Dijkstra's algorithm was developed by computer scientist Edsger W. Dijkstra in 1956 and has widespread applications in various fields, including network routing, transportation planning, and robotics. The algorithm's efficiency lies in its ability to consistently choose the locally optimal path at each step, ensuring that the selected paths are indeed the shortest. This approach is particularly useful in scenarios where finding the most efficient route between points is critical.

Dijkstra's algorithm stands as a greedy approach designed to discover the shortest path from a specified source vertex to all other vertices within a weighted graph. The core of the algorithm revolves around maintaining two sets of vertices: one set includes vertices whose shortest distance from the source is known, and the other set involves vertices with unknown shortest distances. In each iteration, the algorithm selects the vertex with the smallest known distance, incorporates it into the known set, and proceeds to update the distances of its neighboring vertices.

This algorithm serves as a robust solution for tackling shortest path problems in graphs with weighted edges. The provided Java implementation serves as an illuminating resource, offering a lucid and succinct insight into the algorithm's inner workings. Whether you're a student delving into algorithmic principles, a seasoned professional seeking a reliable solution, or simply someone with a curiosity about algorithms, this article aims to provide clarity on the mechanics and effectiveness of Dijkstra's algorithm.

Source:<https://medium.com/@haochenglin/understanding-dijkstras-algorithm-with-java-dcc568b4ae3a>

1. **Search for solutions**

**Search Vertex:**

1. Dijkstra
2. Floyd
3. Prim
4. BFS
5. DFS
6. Kruskal

**Complement:**

1. Graph
2. Hash maps
3. LinkedList
4. Array
5. ArrayList
6. **Preliminary Designs:**

**Search Vertex:**

1. Floyd: The Floyd-Warshall algorithm is used to find the shortest paths between all pairs of vertices in a weighted graph. However, in the game, the edge weights are different and represent the movement between maze vertices.
2. BFS: While BFS is an efficient algorithm for finding the shortest path in unweighted graphs, in the game, edge weights are essential for gameplay. Since you've chosen to model different weights between vertices, BFS might not be the best choice as it does not consider edge weights and simply focuses on breadth-first exploration.
3. DFS: Like BFS, DFS also does not consider edge weights. Although DFS is useful for exploring graphs and finding solutions, it does not provide an efficient solution for finding the shortest path in weighted graphs.
4. Kruskal: Kruskal is used to find the minimum spanning tree in an undirected weighted graph. Kruskal is not applicable to the context. Additionally, in your case, implementing a 1 vs. 1 game in a maze, finding a minimum spanning tree may not be relevant to the game objective.

**Complement:**

1. Array: Arrays in a system do not have flexible and dynamic range, frequent insertions or deletions, can be more efficient with more suitable data structures than arrays.
2. **Testing and Solution Selection:**

Criteria must be defined to evaluate the solution alternatives and, based on this result, choose the solution that best meets the needs of the problem. The solution that best meets the needs of the problem posed. The objective is to establish a weight that indicates which of the possible values of each criterion have more weight.

**Parameter A.** Accuracy of the solution.

[2] Exact (an exact solution is preferred).

[1] Approximate

**Parameter B.** Efficiency. A solution with better efficiency than the others considered is preferred.

[4] Constant

[3] Greater than constant

[2] Logarithmic

[1] Linear

**Parameter C**. Completeness. A solution that finds all solutions is preferred.

[3] All

[2] More than one if any, but not all.

[1] Only one or none

**Parameter D**. Ease of algorithmic implementation:

[2] Compatible with the basic arithmetic operations of modern computer hardware.

[1] Not fully compatible with the basic arithmetic operations of a modern computer.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Search Vertex | **Parameter A** | **Parameter B** | **Parameter C** | **Parameter D** | **Total** |
| Dijkstra | 2 | 2 | 3 | 2 | 9 |
| Prim |  |  |  |  |  |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Complements | **Parameter A** | **Parameter B** | **Parameter C** | **Parameter D** | **Total** |
| Graph | 2 | 2 | 3 | 2 | 9 |
| Hash maps | 2 | 1 | 3 | 2 | 8 |
| LinkedList | 2 | 1 | 3 | 2 | 9 |
| ArrayList | 1 | 1 | 2 | 2 | 6 |