1. **IDENTIFICATION OF THE PROBLEM:**

* Create a free choice theme program which uses at least two different types of graph techniques, such as BFS, DFS, PRIM, etc.…
* The system must use a tree graph with at least 50 vertexes.
* The system must permit the user to enter a username.
* The system must load information of a flat text file into the database for later use in the program.
* The system must have a full 50 room connected paths to get to the goal.
* The system must randomly show the player a mathematical equation every time he enters a new room (Vertex).
* The system must send the player to the fastest path to the goal initially. Every time the player messes up the answer of an equation, the system must send them to a slower path.
* The system must have a timer with the amount of time the player has left to win the game. If the player does not reach the goal in the established time, he will face a game over.

1. **RECOPILATION OF THE NECESSARY INFORMATION:**

It is now necessary for us to search for information regarding all the concepts related to the problem. In this way we would be able to gather ideas of how we could later on create the solution for the problem.

**DEFINITIONS:**

* **Generics in Java:**

Generic programming in Java allows us to create methods, classes and interfaces with parameterized types, which makes it possible for these entities to work with different data types.

With this type of programming we can use the dynamic classes on Java such as ArrayList, HashMap, Queue, etc. and create them in a generic way.

* **Generic method:**

Generic Java method takes a parameter and returns some value after performing a task. It is exactly like a normal function; however, a generic method has type parameters that are cited by actual type. This allows the generic method to be used in a more general way. The compiler takes care of the type of safety which enables programmers to code easily since they do not have to perform long, individual type castings

* **BFS:**

BFS or Breadth-First search is a graph traversal technique where a node and its neighbors are visited first and then the neighbors of neighbors. In simple terms, it tries to explore all the neighbors it can reach from the current node. First, it traverses level 1 nodes (direct neighbors of source nodes) and then level 2 nodes (neighbors of source node) and so on. The BFS can be used to determine the level of each node from a given source node. It will use a queue data structure.

* **DFS:**

DFS or Depth-first search is graph traversal technique where a root node (it is selected an arbitrary node as the root node) explores as far as possible along each branch before backtracking. First, it starts from the root or any arbitrary node and marks the node and moves to the adjacent unmarked node and continues this loop until there is no unmarked adjacent node. Then backtrack and check for other unmarked nodes and traverse them. Finally, print the nodes in the path. This will use a stack data structure.

* **PRIM:**

Prim is a graph technique for finding the MST or Minimum Spanning tree. First, it starts with an empty spanning tree. The idea is to maintain two sets of vertices. The first set contains the vertices already included in the MST; the other set contains the vertices not yet included. At every step, it considers all the edges that connect the two sets and picks the minimum weight edge from these edges. After picking the edge, it moves the other endpoint of the edge to the set containing MST.

* **KRUSKAL:**

Kruskal is a graph technique for finding the MST or Minimum Spanning tree. First, it sorts all the edges in non-decreasing order of their weight. Pick the smallest edge. Then, it checks if it forms a cycle with the spanning-tree formed so far. If the cycle is not formed, include this edge. Else, discard it. Finally, it repeats the previous step until there are (V-1) edges in the spanning tree. Kruskal’s algorithm to find the minimum cost spanning tree uses the greedy approach. The Greedy Choice is to pick the smallest weight edge that does not cause a cycle in the MST constructed so far

* **FLOYD-WARSHALL:**

Floyd-Warshall is a graph technique for solving all pairs shortest path problems. First, initialize the solution matrix same as the input graph matrix as a first step. Then update the solution matrix by considering all vertices as an intermediate vertex. The idea is to one by one pick all vertices and update all shortest paths which include the picked vertex as an intermediate vertex in the shortest path. When we pick vertex number k as an intermediate vertex, we already have considered vertices {0, 1, 2, ... k-1} as intermediate vertices. For every pair (i, j) of the source and destination vertices respectively, there are two possible cases:

k is not an intermediate vertex in the shortest path from i to j. We keep the value of dist[i][j] as it is.

k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j] if dist[i][j] > dist[i][k] + dist[k][j]

* **DIJKSTRA:**

Dijkstra is a graph technique for solving all pairs shortest path problems. First, it generates a SPT (shortest path tree) with a given source as a root. Maintain two sets, one set contains vertices included in the shortest-path tree, other set includes vertices not yet included in the shortest-path tree. At every step of the algorithm, find a vertex that is in the other set (set not yet included) and has a minimum distance from the source.

* **Java FX:**

JavaFX is a Java library and a GUI toolkit designed to develop and facilitate Rich Internet applications, web applications, and desktop applications. The most significant perk of using JavaFX is that the applications written using this library can run on multiple operating systems and several. This characteristic makes it very versatile across operating systems and different platforms.

1. **Search for creative solutions:**

In this case, we are using **“brainstorm”** in order to look up for creative solutions to the problem that we have. In this case, we are looking for a solution that helps us find the shorter path from the beginning to the end of the graph, so, using **“brainstorm”**, we end up with the following solutions:

**DFS:** The use of the DFS structure to explore the nodes of the program.

**BFS:** The use of the BFS structure to explore the nodes of the program.

Is important to take into account that, according to the problem that we want to solve, DFS is more efficient than BFS.

**PRIM:** this algorithm searches for the connection of all the nodes with a minimum weight. This algorithm passes through each node multiple times.

**KRUSKAL:** This algorithm searches for the connection of all the nodes, using the edges that have less weight and, with this, we have the connection path with the minimum weight possible. This algorithm passes through each node only once.

**DIJKSTRA**: this algorithm looks up for the shorter path from an initial node to a specific node.

**FLOYD-WARSHALL:** this algorithm helps finding the minimum path in ponderate graphs. Meaning that this method uses the weight of the edges.

Now that we have all the algorithms that we can use in our program, let’s do some combinations or alternatives in order to decide which one is the best.

**ALTERNATIVE 1 BFS & FLOYD-WARSHALL:**

We could use both a BFS search structure and a Floyd-Warshall graph technique in order to find the shorter path in all the node pairs.

**ALTERNATIVE 2 DFS & KRUSKAL:**

We could use DFS in order to check that all the nodes are connected and kruskal in order to get the graph with the shorter path from one point to another.

**ALTERNATIVE 3 DFS & PRIM:**

In this case, we could use DFS to check if all the graphs are connected and, with prim, we can get a graph that contains the shorter path from one node to another.

**ALTERNATIVE 4 BFS & DIJKSTRA:**

Last but not least, we think we can use BFS to check that all the nodes are connected and Dijkstra to find the shorter path from a node to another one.

**ALTERNATIVE 5 DFS & DIJKSTRA:**

With this alternative, we can check, with DFS, if all the nodes are strongly or not strongly connected. And, with dijkstra, we can find the shortest path to the goal.

**ALTERNATIVE 6 BFS & PRIM:**

It’s worth to say that all the combinations and algorithms previously mentioned have very similar objectives, so the final decision about what pair of algorithms we should use, will be defined by the little differences that each algorithm has.

1. **Transition from ideas to preliminary designs:**

In this step it's important to discard some ideas previously mentioned that are not workable to develop a solution to the problem, therefore, we need to set a correct structure of the design problem.

The alternatives we are going to initially discard would be the follow ones:

**ALTERNATIVE 2 DFS & KRUSKAL:** with the DFS we can verify that all the nodes are connected, but we can still discard this option because the kruskal algorithm works in a way that we don't need since the objective of the kruskal algorithm is to search the shorter path in which every node is connected, that’s something that is unnecessary to our program. What we really need is to look up for the shorter path between 2 nodes and the kruskal algorithm doesn’t work like that.

**ALTERNATIVE 3 DFS & PRIM:** we can discard this option because the prim algorithm have a similar functionality to kruskal’s algorithm, both have the objective of finding the shorter path to connect all the nodes and, as we mentioned before, that is something that we don’t really need in our program.

**ALTERNATIVE 6 BFS & PRIM:** with this option we can check if the graph is strongly connected or not, but we can still discard it because of prim, since prim is an algorithm that look up for the connection of all the vertex, we can’t use it because, in the program, that’s something we don’t really eats’’’

Regarding all the requirements and the general objective of the program, the alternatives that could accomplish them and would later on be examined to determined the best solution would be the follow ones:

**ALTERNATIVE 1 BFS & FLOYD-WARSHALL:** we can take into account this option because floyd-warshall’s algorithm has an interesting functionality that is finding the shorter path between every path of nodes. That is something that could help us a lot in our problem solving because we need to know the shortest path between the initial node in which the games starts and the “final node” in which the games end. Also, the BFS algorithm can help us in our need of knowing if every node is connected, something that is necessary in our problem solving.

**ALTERNATIVE 4 BFS & DIJKSTRA:** we can take this solution into account because dijkstra is an algorithm that have the objective of finding the shorter path between 2 nodes, this helps us in our problem solving because we need to know the shortest distance between the initial node (the start of the game) and the final node (the end of the game), and dijkstra make it easy to know. Also, the BFS is a good algorithm that, as we said before, can help us by finding out if all the nodes are connected or not.

**ALTERNATIVE 5 DFS & Dijkstra:** we can take this pair into account because of similar reasons as the BFS & Dijkstra, since BFS and DFS have very similar functionality, so with DFS I can still check if the graph is strongly connected or not and, with dijkstra, we fastest path to our goal.

1. **Examination and selection of the best solution:**

We establish the following set of criteria to select the final alternative which is going to be use on the program:

Criteria A. Precision of the solution

[2] Exact

[1] Approximate

Criteria B. Efficiency

[4] Constant

[3] Greater than constant

[2] Logarithmic

[1] Linear

Criteria C. Entirety

[3] Complete

[2] Almost Complete

[1] None

Criteria D. Easiness in implementation

[2] Compatible with the basic arithmetic operations of a modern computer equipment

[1] Non-compatible with the basic arithmetic operations of a modern computer equipment

|  | Criteria A | Criteria B | Criteria C | Criteria D | Total |
| --- | --- | --- | --- | --- | --- |
| Alternative 1 | [2] | [3] | [3] | [1] | 9 |
| Alternative 4 | [2] | [4] | [3] | [2] | 11 |
| Alternative 5 | [2] | [3] | [2] | [1] | 8 |

Based on the previous results, we have established that the ideal solution for the problem we consider is the Alternative 4 which uses the BFS searching structure and the Dijkstra graph technique.

**Group members:**

**A00381777 | Belalcazar Bonelo, Santiago Jose**

**A00381157 | Escobar Vacaflor, Daniel**

**A00381323 | Pinillos Sanchez, Luis Fernando**