**Contexto Problemático**

En colombia anteriormente existían trenes intermunicipales que conectaban las principales ciudades colombianas, un medio de transporte muy utilizado por los colombianos. Hoy en día se quiere simular estaciones de trenes en diferentes ciudades de colombia para evaluar un nuevo proyecto en el que viene trabajando el gobierno colombiano, dependiendo del buen uso de esta interfaz y que las simulaciones logren salir de la mejor manera, el proyecto será evaluado y se instalarán metros en las principales ciudades colombianas.

**Paso 1: Identificación del problema**

**Definición del programa:**

Este proyecto consiste en un sistema digital de tren subterráneo, la idea es simular dicho sistema mediante el uso de grafos. El programa estará en la capacidad de darle al usuario el camino más corto entre dos estaciones (vértices), el tiempo de menor tráfico entre las estaciones y entre otras funcionalidades que aún están en desarrollo. Creemos que este programa sería muy útil para las personas que viven en ciudades que cuentan con este servicio rápido y efectivo; además pensamos que el uso de un programa como este hará mejor el servicio y ayudará mucho más a cada uno de los usuarios.

**Identificación de necesidades y síntomas:**

* El programa debe estar en la capacidad de entregar el camino más corto, dadas dos estaciones de tren.
* El programa debe estar en la capacidad de agregar estaciones cuando lo desee el usuario.
* El programa debe estar en la capacidad de dar el camino más corto para recorrer las estaciones anteriormente agregadas por el usuario.

**Paso 2: Recopilación de información**

Con el objetivo de modelar de la mejor forma el problema, se ha hecho una profunda búsqueda de información sobre las estructuras de datos y los métodos que apliquen a este problema.

**Definiciones**

**Estructura de Datos**

En ciencias de la computación, una estructura de datos es una forma particular de organizar datos en una computadora para que puedan ser utilizados de manera eficiente. Diferentes tipos de estructuras de datos son adecuados para diferentes tipos de aplicaciones, y algunos son altamente especializados para tareas específicas.

**Grafos**

Los grafos son un conjunto de puntos, de los cuales algún par de ellos está conectado por unas líneas. Si estas líneas son flechas, hablaremos de grafo dirigido (digrafo), mientras que si son simples líneas estamos ante un grafo no dirigido. Más formalmente se pueden definir como un conjunto de vértices y un conjunto de aristas. Cada arista es un par (u,v), donde u y v pertenecen al conjunto de vértices. Si este par es ordenado el grafo es dirigido. Un grafo G se dice conexo si, para cualquier par de vértices u y v en G, existe al menos una trayectoria (una sucesión de vértices adyacentes que no repite vértices) de u a v.

**Algoritmo Djisktra**

Es un algoritmo para la determinación del camino más corto, dado un vértice origen, hacia el resto de los vértices en un grafo que tiene pesos en cada arista. Este algoritmo consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen hasta el resto de los vértices que componen el grafo, el algoritmo se detiene.

**Algoritmo de Floyd-Warshall**

Es un algoritmo de análisis sobre grafos para encontrar el camino mínimo en grafos dirigidos ponderados. El algoritmo encuentra el camino entre todos los pares de vértices en una única ejecución.

**Algoritmo de Prim**

Es un algoritmo perteneciente a la teoría de los grafos para encontrar un árbol recubridor mínimo en un grafo conexo, no dirigido y cuyas aristas están etiquetados. El algoritmo encuentra un subconjunto de aristas que forman un árbol con todos los vértices, donde el peso total de todas las aristas en el árbol es el mínimo posible. Si el grafo no es conexo, entonces el algoritmo encontrará el árbol recubridor mínimo para uno de los componentes conexos que forman dicho grafo no conexo.

**Algoritmo de Kruskal**

Es un algoritmo de la teoría de grafos para encontrar un árbol recubridor mínimo en un grafo conexo y ponderado. Es decir, busca un subconjunto de aristas que, formando un árbol, incluyen todos los vértices y donde el valor de la suma de todas las aristas del árbol es el mínimo.

**Algoritmo DFS**

Es un algoritmo de búsqueda no informada utilizado para recorrer todos los nodos de un grafo o árbol de manera ordenada, pero no uniforme. Su funcionamiento consiste en ir expandiendo todos y cada uno de los nodos que va localizando, de forma recurrente, en un camino concreto.

**Algoritmo BFS**

Es un algoritmo de búsqueda no informada utilizado para recorrer o buscar elementos en un grafo. Intuitivamente, se comienza en la raíz (eligiendo algún nodo como elemento raíz en el caso de un grafo) y se exploran todos los vecinos de este nodo. A continuación para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el árbol.

**Paso 3: Búsqueda de soluciones creativas**

Las siguientes son las alternativas que se tienen para solucionar el problema

*Alternativa 1: Usar pilas, colas y hashtables propias*

* El implementar nuestras propias colas, pilas y hashtables nos facilita de gran manera nuestra tarea ya que podemos acondicionar estas estructuras de acuerdo a las necesidades que tenemos.

*Alternativa 2: Usar arreglos y estructuras no dinámicas*

* El uso de arreglos y estructuras no conocidas, tienen un grado de implementación y entendimiento muy menor, ya que se tiene un tiempo respetable trabajando con estas estructuras.

*Alternativa 3: Usar arboles(ABB,Roji-Negro,AVL)*

* El uso de estas estructuras de datos nos son de gran ayuda, ya que con estas estructuras podemos organizar los datos de una manera muy organizada y clara. Además al momento de requerir datos fácilmente los podemos encontrar si está organizados en una de estas estructuras.

*Alternativa 4: Usar Grafos*

* El uso de estas estructuras de datos nos son de gran ayuda, ya que con estas estructuras se logra organizar y plantear una solución, a parte de los diferentes algoritmos que esta estructura nos permite realizar, podemos implementar fácilmente algoritmos que nos ayuden con la solución del problema presentado.

**Paso 4: Transición de las Ideas a los Diseños Preliminares**

Se descarta la alternativa número 2, ya que el trabajar solamente con arreglos y estructuras no dinámicas nos dificulta la realización del trabajo y no se podría llevar de la mejor manera las necesidades que pide el programa.

La revisión cuidadosa de las otras alternativas nos conduce a lo siguiente:

*Alternativa 1: Usar pilas, colas y hashtables propias*

* Estas estructuras de datos pueden servir para algunos de los requerimientos pero con otros nos pueden causar dificultad por la manera en que son almacenados los datos en estas estructuras.

*Alternativa 3: Usar arboles(ABB,Roji-Negro,AVL)*

* Estas estructuras permiten una gran manera de organizar la información, pero no cuenta con algoritmos útiles para dar una completa solución al problema que se presenta.

*Alternativa 4: Usar Grafos*

* Esta estructura es ideal para dar solución al problema presentado, a parte de la fácil organización que podemos hacer en ella, los algoritmos que se pueden implementar satisfacen complemente lo requerido por el problema.

**Paso 5: Evaluación y Selección de la Mejor Solución**

* Criterio A. Precisión de la solución. La alternativa entrega una solución:

- [2] Exacta (se prefiere una solución exacta)

­ - [1] Aproximada

* ­ Criterio B. Eficiencia. Se prefiere una solución con mejor eficiencia que las otras consideradas. La eficiencia puede ser:

­ - [4] Constante

­ - [3] Mayor a constante

­ - [2] Logarítmica

­ - [1] Lineal

* ­ Criterio C. Completitud. Se prefiere una solución que encuentre todas las soluciones. Cuántas soluciones entrega:

­ - [3] Todas

­ - [2] Más de una si las hay, aunque no todas

­ - [1] Sólo una o ninguna

* ­ Criterio D. Facilidad en implementación algorítmica:

­ - [2] Compatible con las operaciones aritméticas básicas de un equipo de cómputo moderno

­ - [1] No compatible completamente con las operaciones aritméticas básicas de un equipo de cómputo moderno

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Criterio A** | **Criterio B** | **Criterio C** | **Criterio D** | **Total** |
| **Alternative(1)** | **1** | **2** | **1** | **2** | **6** |
| **Alternative(3)** | **1** | **2** | **1** | **2** | **6** |
| **Alternative(4)** | **2** | **2** | **3** | **2** | **9** |