Proyecto final de programación: Primer Avance

Donald Vargas (C18231)

Escuela de Ingeniería Eléctrica, Universidad de Costa Rica, Estructuras de Datos Abstractas y Algoritmos para Ingeniería.

3 de noviembre de 2025

1. Introducción

El objetivo de este primer avance es definir y validar la arquitectura de software y la solución técnica propuesta para el proyecto final del curso "Buscador de Rutas con Grafos y Cola de Prioridad". Se espera un documento que sirva como guía maestra para el desarrollo. El objetivo de este proyecto es implementar el algoritmo de Dijkstra para encontrar la ruta más corta en un grafo. Requiere construir una estructura no lineal (grafo) y una estructura de datos especializada (cola de prioridad) completamente desde cero. Para este primer avance se desea validar la arquitectura de la solución, identificar los componentes clave y planificar el desarrollo de las próximas semanas, con lo estipulado en cada punto de este documento.

2. Definición Formal de la Solución

El objetivo de este proyecto se basa en implementar un buscador de rutas basado en el algoritmo Dijkstra, que permitirá encontrar el camino más corto entre dos vértices de un grafo.
El grafo se verá representado por medio de una lista de adyacencia, la cual constará de un vector dinámico y una lista enlazada.
Con el fin de una gestión correcta de los vértices con menor distancia provisional, se implementará una cola de prioridad, la cual estará implementada con un binary heap construido sobre el vector dinámico. El sistema permitirá ingresar los vértices y aristas, definir un nodo de inicio y fin, con el fin de mostrar la ruta más corta y su costo total.

3. Diseño de Estructuras (Diagrama de Clases UML)

Para la elaboración del Diagrama de clases, se utilizó la guía presente en la página "Geek for Geeks", y para la realización gráfica del diagrama se utilizó la página web PlantUML.

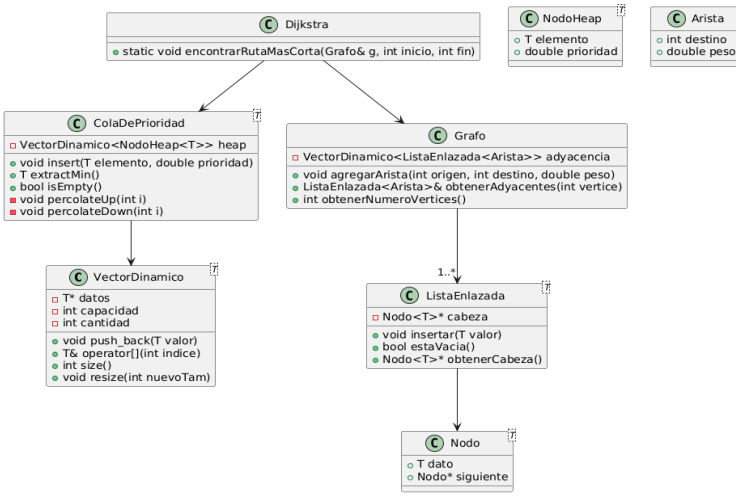

Figura 1. Diagrama de clases y jerarquías para la elaboración del código.

4. Métodos Principales

En esta sección se planea dejar previsto y definidas las interfaces de los componentes más críticos del sistema, en caso de requerirlo, estos códigos pueden estar sujetos a cambios futuros.

```cpp
template <typename T>
class VectorDinamico {
public:
    VectorDinamico();// Constructor (crea vector con capacidad inicial)
    ~VectorDinamico();// Destructor (libera la memoria dinámica)
    void push_back(const T& valor);//Inserta un nuevo elemento al final del vector creado
    T& operator[](int indice);//Permite acceder a un elemento por el índice otorgado
    int size() const;//Devuelve el número de elementos que fueron almacenados almacenados
    void resize(int nuevoTam);//Cambia la capacidad del vector

private:
    T* datos;//Puntero a los datos en memoria dinámica
    int capacidad;//Capacidad de almacenamiento del vector
    int cantidad;//Cantidad actual de elementos en el vector
};
```

Figura 2. Código para la función del vector dinámico.

```cpp
template <typename T>
struct Nodo {
    T dato;//Valor que fue almacenado en el nodo
    Nodo* siguiente;//Puntero al nodo siguiente
};

template <typename T>
class ListaEnlazada {
public:
    ListaEnlazada();//Constructor(lista vacia)
    ~ListaEnlazada();//Destructor (libera todos los nodos)

    void insertar(const T& valor);//Inserta un nuevo elemento al final de la lista
    bool estaVacia() const;//Indica si la lista está vacía
    Nodo<T>* obtenerCabeza() const;//retorna el puntero al primer nodo de la lista
private:
    Nodo<T>* cabeza;//Puntero que apunta al inicio de la lista
};
```

Figura 3. Código para la función de la lista enlazada.

```cpp
template <typename T>
struct NodoHeap {
    T elemento;//Elemento almacenado en la cola
    double prioridad;//Prioridad respectiva para cada elemento de la cola
};
template <typename T>
class ColaDePrioridad {
public:
    ColaDePrioridad();//Constructor (heap vacío)
    void insert(const T& elemento, double prioridad);//Inserta un elemento con su respectiva prioridad
    T extractMin();//Extrae el elemento con menor prioridad
    bool isEmpty() const;//Verifica si la cola está vacía

private:
    VectorDinamico<NodoHeap<T>> heap;//Vector que almacena el heap
    void percolateUp(int indice);//Restaura la propiedad del heap hacia arriba
    void percolateDown(int indice);//Restaura la propiedad del heap hacia abajo
};
```

Figura 4. Código para la función de la cola de prioridad.

```
struct Arista {
    int destino;//Vértice destino
    double peso;//Costo
};
class Grafo {
public:
    Grafo(int numVertices);//Constructor (inicializa la estructura base)
    void agregarArista(int origen, int destino, double peso);//Agrega una arista al grafo
    ListaEnlazada<Arista>& obtenerAdyacentes(int vertice);//Retorna la lista de adyacentes
    int obtenerNumeroVertices() const;//Retorna el número de vértices
private:
    VectorDinamico<ListaEnlazada<Arista>> adyacencia;//Lista de adyacencia del grafo
};
```

Figura 5. Código para la función del grafo.

```
class Dijkstra {
public:
    static void encontrar_Ruta_Mas_Corta(Grafo& g, int inicio, int fin);
    //Encuentra la ruta más corta entre dos nodos usando Dijkstra
private:
    static void imprimir_Ruta(const VectorDinamico<int>& predecesores, int inicio, int fin);
    //Imprime la ruta más corta encontrada
};
```

Figura 6. Código para la implementación del algoritmo Dijkstra.

5. Explicación de la Lógica

Lineamiento que describe el comportamiento y orden de ejecución del código:

1. El usuario define el grafo a utilizar
2. Se llama a Dijkstra con la función de encontrar la ruta más corta.
3. Dijkstra inicializa las distancias y usa la cola de prioridad para obtener el vértice a menor distancia.
4. Con la lista de adyacencia, actualiza las distancias en el Grafo.
5. Finalmente, calcula el costo total y lo imprime junto a la ruta.

6. Cronograma

Cronograma de trabajo correspondiente a las próximas 4 semanas, sujeto a cambios dado las instrucciones del profesor:

Tabla 1. Cronograma a efectuar el último mes para la realización del proyecto.

Semana	Fecha	Trabajo por efectuar
1	Nov 1- Nov 6	Implementar vector dinámico y lista enlazada.
2	Nov 7 - Nov 13	Implementar cola de prioridad con montículo binario. Entrega Avance 2.
3	Nov 14 - Nov 20	Implementar grafo y pruebas de inserción y recorrido.
4	Nov 21 - Nov 27	Implementar algoritmo de Dijkstra y pruebas integradas, documentar y limpiar código.

7. Prototipo de Estructura de Archivos

Para la elaboración de la estructura se basó en el diagrama UML del punto 3, así como también del código propuesto en el punto 4, manteniendo jerarquías y orden.

```
/Proyecto_Dijkstra
|
|-- /include
|   |-- VectorDinamico.hpp
|   |-- ListaEnlazada.hpp
|   |--- ColaDePrioridad.hpp
|   |-- Grafo.hpp
|   |-- Dijkstra.hpp
|
|-- /src
|   |-- VectorDinamico.cpp
|   |-- ListaEnlazada.cpp
|   |-- ColaDePrioridad.cpp
|   |-- Grafo.cpp
|   |-- Dijkstra.cpp
|   |-- main.cpp
|
|-- /docs
|   |-- avance_1.pdf
|   |-- diagrama_uml.png
|
|-- Makefile
|-- README.md
```

Figura 7. Prototipo de estructura de archivos para la realización del proyecto.