Proyecto de Curso

IE0217 – Estructuras Abstractas de Datos y Algoritmos para Ingeniería
Prof: Ing. Karen Tovar Parra

Escuela de Ingeniería Eléctrica
Universidad de Costa Rica

1 Objetivos

- Aplicar estructuras de datos avanzadas en la solución de problemas computacionales, mediante la implementación de algoritmos eficientes en C++.
- Desarrollar habilidades de programación estructurada y orientada a objetos, implementando soluciones modulares, reutilizables y escalables.
- Diseñar e implementar interfaces gráficas interactivas que permitan visualizar y manipular los datos de manera intuitiva para mejorar la experiencia del usuario.
- Fomentar el trabajo en equipo y la gestión de proyectos, distribuyendo tareas de manera eficiente y colaborando en la resolución de problemas.
- Comprender e implementar mecanismos de persistencia de datos, permitiendo guardar y recuperar el estado de la aplicación en archivos para garantizar la continuidad de la información.
- Utilizar estructuras de datos específicas (pilas, colas, árboles, listas enlazadas, grafos, tablas hash) según la naturaleza del problema, justificando su selección en cada caso.
- Optimizar el rendimiento de las soluciones implementadas, evaluando la complejidad de los algoritmos utilizados y buscando mejoras en la eficiencia computacional.

2 Evaluación

Para la evaluación de este proyecto, se considerarán tanto los criterios establecidos en la rúbrica específica de cada opción de proyecto como en la rúbrica general.

Toda documentación o evidencia adicional relevante al proyecto deberá incluirse en un archivo README o en documentos referenciados dentro del mismo. Como mínimo, el README debe contener los nombres completos y números de carné de los integrantes del equipo.

- La fecha de entrega del proyecto es el 27 de febrero de 2025 a las 23:59. No se aceptarán entregas fuera de este plazo.
- El proyecto debe ser desarrollado en GitHub, utilizando un repositorio personal de uno de los integrantes, el cual deberá ser compartido con el docente.

- Se requiere un mínimo de 20 commits distribuidos a lo largo del desarrollo del proyecto para evidenciar progreso.
- Cada integrante del equipo deberá realizar al menos 3 commits.
- El incumplimiento de esta norma resultará en una penalización de 15 puntos en la calificación final del proyecto.

3 Modalidad

Cada estudiante deberá realizar el laboratorio de forma grupal de 4 integrantes.

Introducción

Este documento contiene los enunciados detallados para proyectos de curso en la materia de Estructuras de Datos. Los estudiantes trabajarán en grupos de 4 personas . Se requiere el uso del lenguaje de programación C++ y elementos de interfaz gráfica.

Proyectos Propuestos

3.1 Juego de 4 en Línea

Requerimientos específicos:

1. **Lógica del juego:**
   - Tablero de 7x6 representado con una matriz dinámica o lista de listas.
   - Validar movimientos del jugador (columna llena, posición válida).
   - Detectar victoria horizontal, vertical o diagonal.
   - Implementar la IA utilizando un **árbol de decisión**, donde cada nodo represente un estado del tablero y las ramas sean las posibles jugadas. La IA seleccionará la jugada óptima explorando una cantidad limitada de niveles en el árbol.
2. **Modo IA:**
   - Implementar un nivel básico (jugadas aleatorias).
   - Implementar un nivel intermedio, donde la IA evalúe el árbol de decisión para bloquear al oponente o maximizar sus propias posibilidades.
3. **Interfaz gráfica:**
   - Utilizar una librería como **SFML** o **Qt** para representar gráficamente el tablero.
   - Permitir a los jugadores seleccionar columnas haciendo clic en ellas.
   - Mostrar los turnos y resaltar al ganador o declarar empate al final del juego.
4. **Persistencia:**

- Implementar un sistema para guardar el estado del tablero en un archivo de texto o binario, de manera que se pueda reanudar la partida en cualquier momento.
- La información almacenada debe incluir el tablero actual, el turno en curso y el modo de juego (Jugador vs Jugador o Jugador vs IA).

Estructuras de datos obligatorias:

- **Matrices dinámicas** o listas enlazadas para el tablero.
- **Árbol de decisión** para la IA.

Sugerencia para la interfaz gráfica: Utilizar **SFML** por su simplicidad para manejar gráficos 2D. Cada celda del tablero puede representarse como un rectángulo con colores que cambien según el jugador. Los eventos de clic se pueden utilizar para determinar la columna seleccionada.

Implementación de persistencia: Guardar el tablero como una matriz en un archivo de texto o binario, junto con el turno y el modo de juego. Para cargar, leer los datos y reconstruir el estado del juego.

Simulador de Red Social

Requerimientos específicos:

1. **Gestión de usuarios y conexiones:**
   - Añadir, eliminar y modificar usuarios (nodos en el grafo).
   - Conectar y desconectar usuarios (aristas en el grafo).
2. **Sugerencias de amigos:**
   - Proporcionar sugerencias basadas en amigos en común.
   - Utilizar **listas enlazadas** para representar las conexiones de cada usuario.
3. **Interfaz gráfica:**
   - Mostrar la red social gráficamente como un grafo, donde los nodos representen usuarios y las aristas, las amistades.
   - Permitir la interacción con el grafo para agregar o eliminar usuarios y conexiones.
4. **Persistencia:**
   - Guardar el grafo en un archivo, utilizando una representación basada en listas enlazadas.
   - Al cargar, reconstruir la red social desde el archivo.

Estructuras de datos obligatorias:

- **Grafo no dirigido** para modelar la red social.
- **Listas enlazadas** para representar conexiones entre usuarios.
- Algoritmos como **DFS o BFS** para analizar relaciones y sugerir amigos.

Sugerencia para la interfaz gráfica: Usar **Qt** o **SFML**. Representar usuarios como nodos circulares y conexiones como líneas. Permitir clics para agregar nodos y conexiones dinámicamente.

Explicación de la lista enlazada: Cada usuario tendrá una lista enlazada que contenga punteros a sus amigos, representando las conexiones. Esto facilita la implementación del BFS/DFS para las sugerencias de amigos.

Simulador de Cola para Parque de Atracciones

Requerimientos específicos:

1. **Simulación de colas:**
   - Implementar colas de prioridad para manejar visitantes VIP.
   - Implementar una cola normal para visitantes regulares.
   - Manejar grupos de visitantes (todos los integrantes del grupo deben ser atendidos juntos).
2. **Estimación de tiempos:**
   - Calcular tiempos de espera basados en una tasa de atención configurable.
3. **Interfaz gráfica:**
   - Mostrar visualmente las colas en tiempo real.
   - Permitir agregar, mover o eliminar visitantes mediante la interfaz.

Estructuras de datos obligatorias:

- **Cola de prioridad** para visitantes VIP.
- **Cola regular** para visitantes regulares.
- **Listas enlazadas** para manejar grupos.

Sugerencia para la interfaz gráfica: Usar **SFML** para mostrar colas como bloques rectangulares con colores diferentes según el tipo de visitante. Actualizar la interfaz en tiempo real para reflejar cambios en las colas.

Generador de Laberintos y Solucionador

Requerimientos específicos:

1. **Generador de laberintos:**
   - Implementar un algoritmo como *DFS*, *Prim* o *Kruskal* para generar laberintos aleatorios.
2. **Resolución del laberinto:**
   - Implementar *BFS* o *A** para encontrar la ruta más corta entre dos puntos.
   - Mostrar el proceso de búsqueda paso a paso.
3. **Interfaz gráfica:**
   - Dibujar el laberinto generado.
   - Permitir al usuario seleccionar puntos de inicio y fin con el mouse.
   - Animación que muestre el proceso de resolución.

Estructuras de datos obligatorias:

- **Grafos representados con listas de adyacencia o matrices.**
- **Colas (para BFS)** o estructuras de prioridad (para A*).

Sugerencia para la interfaz gráfica: Usar **SFML** para dibujar el laberinto como una cuadrícula. Los caminos abiertos se representan con celdas blancas, y las paredes, con celdas negras. Permitir que el usuario seleccione puntos interactivos para el inicio y fin.

Sistema de Reserva de Vuelos

Requerimientos específicos:

1. **Gestión de vuelos:**
   - Crear y eliminar ciudades (nodos del grafo).
   - Añadir, modificar y eliminar conexiones (aristas con peso que representan distancia o costo).
2. **Rutas óptimas:**
   - Implementar *Dijkstra* o *A** para calcular rutas óptimas entre ciudades.
3. **Interfaz gráfica:**
   - Mostrar un mapa interactivo con nodos (ciudades) y aristas (vuelos).
   - Permitir seleccionar ciudades para calcular rutas.
4. **Persistencia:**
   - Guardar el grafo y las conexiones en un archivo para su recuperación.

Estructuras de datos obligatorias:

- **Grafo ponderado dirigido** para representar vuelos.
- **Cola de prioridad** para implementar *Dijkstra* o *A**.
- **Mapas o vectores** para manejar pesos y conexiones.

Sugerencia para la interfaz gráfica: Usar **Qt** para mostrar un mapa interactivo donde cada ciudad sea un nodo y los vuelos, líneas entre ellos. Permitir al usuario seleccionar ciudades haciendo clic en los nodos.

4 Rúbrica

Rúbrica General (10% del total)

Criterio	Puntos %
Documentación técnica	5%
Manejo de GitHub (20 commits, 3 por integrante)	5%
Presentación de la funcionalidad del proyecto	10%

Tabla 1: Rúbrica General

Rúbricas por Proyecto (20% del total)

Juego de 4 en Línea

Criterio	Puntos %
Implementación de la lógica básica del juego	5%
IA funcional (movimientos válidos y bloqueos básicos)	5%
Interfaz gráfica intuitiva y funcional	5%
Persistencia (guardar/cargar partida)	5%

Tabla 2: Rúbrica para Juego de 4 en Línea

Simulador de Red Social

Criterio	Puntos %
Implementación del grafo y funcionalidades básicas	5%
Sugerencias de amigos funcionales	5%
Interfaz gráfica para visualizar el grafo	5%
Persistencia de la red social	5%

Tabla 3: Rúbrica para Simulador de Red Social

Criterio	Puntos %
Implementación de las colas (regular y prioridad)	5%
Gestión eficiente de grupos	5%
Interfaz gráfica funcional	5%
Estimación de tiempos de espera	5%

Tabla 4: Rúbrica para Simulador de Cola para Parque de Atracciones

Simulador de Cola para Parque de Atracciones

Generador de Laberintos y Solucionador

Criterio	Puntos %
Generador de laberintos funcional	5%
Algoritmo de resolución correcto	5%
Interfaz gráfica intuitiva y animada	5%
Visualización del proceso de resolución	5%

Tabla 5: Rúbrica para Generador de Laberintos y Solucionador

Sistema de Reserva de Vuelos

Criterio	Puntos %
Implementación del grafo con rutas funcionales	5%
Cálculo correcto de rutas óptimas	5%
Interfaz gráfica clara e interactiva	5%
Persistencia del sistema	5%

Tabla 6: Rúbrica para Sistema de Reserva de Vuelos