**Complejidad Algorítmica**

**Curso : Complejidad Algorítmica**

**Trabajo Final**

**Profesor : Luis Martin Canaval Sanchez**

**Autores :**

* **Raúl Bigoria**
* **Gonzalo Nuñez**
* **Luis Rivera**

**Noviembre del 2016**

**TRABAJO FINAL**

**1. Introducción**

Los algoritmos son un conjunto ordenado de operaciones sistemáticas que permite hacer un cálculo y hallar la solución de un tipo de problemas. En el curso se aprende el fundamento matemático para analizar algoritmos de un nivel alto de complejidad. Como parte del proyecto final del curso se solicita elaborar un juego en el cual se pueda utilizar algoritmos de búsqueda y algoritmos codiciosos o uso de programación dinámica para solucionar un problema.

De esta manera, el trabajo final estará basado en el juego de Slither.io. El cual se trata de una nueva versión del famoso Agar.io. En este juego el personaje principal es una lombriz. El sistema de juego es sencillo, en el cual se mueve con el cursor por entornos 3D repletos de esferas de colores y lombrices enemigas donde puedes comer todo lo que encuentres y seguir creciendoEl juego se encuentra en la plataforma web y móvil.

**a. Problema**

Se necesita elaborar y analizar algoritmos de búsqueda y greedy o dinámico que permitan determinar qué camino tomar para que la lombriz coma la mayor cantidad de esferas de luz en la menor cantidad de movimientos.

**b. Objetivos**

El objetivo del trabajo es desarrollar algoritmos de búsqueda y algoritmos codiciosos o programación dinámica para resolver nuestro problema y poder analizarlos. Como resultado se tendrá algoritmos desarrollados con la teoría aprendida durante el curso, los cuales permitirán a la lombriz encontrar de manera eficaz un camino para llegar a la comida. Además, analizar el algoritmo de una manera cuantitativa para estimar el tiempo que demora en ejecutarse y poder elegir de esta manera un algoritmo eficiente y adecuado para el caso propuesto.

**2. Marco conceptual**

**Corte Mínimo en un Grafo**

En [la teoría](https://en.wikipedia.org/wiki/Graph_theory) de [grafos](https://en.wikipedia.org/wiki/Graph_theory) , un corte mínimo de ungrafo es uncorte (una [p](https://en.wikipedia.org/wiki/Partition_of_a_set)ar[tición](https://en.wikipedia.org/wiki/Partition_of_a_set) de los vértices de un gráfico en dos subconjuntos disjuntos que están unidas por al menos un borde) que es mínimo en algún sentido. Por ejemplo, el número de aristas.

Para hallar un corte mínimo de un grafo no dirigido según el número de aristas se utiliza el algoritmo de Karger, el cual proporciona un método aleatorio para hallarlo.

**Algoritmo de Karger**

El algoritmo de Karger es un algoritmo aleatorio, inventado por David Karger en 1993, que es utilizado para calcular el corte mínimo de conexión en un grafo conectado. El algoritmo utiliza un modelo en el cual se va reduciendo el grafo mediante la contracción de aristas uno por uno hasta que sólo queden dos nodos entre los cuales se realiza el corte. El algoritmo al ser aleatorio, genera distintos resultados.

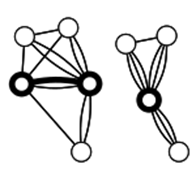
**Pseudocódigo**

1) Crear una copia CG(Grafo contractado) del Grafo

2) Mientras que el grafo tenga más de dos vértices

a) Escoger aleatoriamente una arista (u, v) del grafo contractado

b) Unir o contractar ‘u’ y ‘v’ en un solo vértice (actualizar el gráfico).



c) Remover los bucles

3)Retornar el corte representado por 2 vértices.

Este algoritmo no será utilizado para el desarrollo del trabajo final porque no queremos hacer un corte mínimo para ir de un camino a otro, sino que buscamos ir por varios puntos recorriendo lo menos posible.

**Algoritmos de búsqueda**

Un [algoritmo](https://es.wikipedia.org/wiki/Algoritmo) de búsqueda es aquel que está diseñado para localizar un elemento con ciertas propiedades dentro de una [estructura de datos](https://es.wikipedia.org/wiki/Estructura_de_datos).

**Algoritmos de optimización(algoritmos codiciosos o dinámicos)**

Los algoritmos codiciosos o greedy se utilizan generalmente para resolver problemas de optimización (obtener el máximo o el mínimo). Estos algoritmos toman decisiones en función de la información que está disponible en cada momento y una vez tomada la decisión, ésta no vuelve a replantearse en el futuro. Estos suelen ser rápidos y fáciles de implementar, pero no siempre garantizan alcanzar la solución óptima.

La programación dinámica es un método para reducir el tiempo de ejecución de un [algoritmo](https://es.wikipedia.org/wiki/Algoritmo) mediante la utilización de [subproblemas superpuestos](https://es.wikipedia.org/w/index.php?title=Subproblema_superpuesto&action=edit&redlink=1) y [subestructuras óptimas](https://es.wikipedia.org/w/index.php?title=Estructura_%C3%B3ptima&action=edit&redlink=1).

**3. Metodología**

Se seleccionará un juego con un problema que pueda ser solucionado con algoritmos de búsqueda y codiciosos. Por esta razón, elegimos Slither.io.

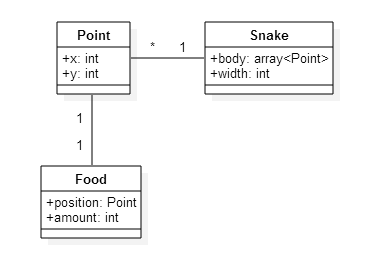
El proyecto posee tres partes importantes, las cuales sucederá lo siguiente: en la primera parte se analizará el juego que se ha elegido, determinando los posibles algoritmos a utilizar y los tipos de datos abstractos que posee. En la segunda parte se implementará un algoritmo de búsqueda. Por último, se optimizará la solución realizando algoritmos codiciosos o programación dinámica.

Los algoritmos se aplicarán en un bot, el cual seleccionará qué movimientos realizar según el algoritmo de búsqueda decida. El bot deberá trasladar al gusano por el escenario para recoger las esferas de luz tomando decisiones sobre las rutas a seguir.El bot será implementado bajo el idioma javascript y se implementará en tiempo real de manera que se jugará contra otras personas en línea.

El análisis del juego se realizará utilizando el lenguaje de modelado de sistemas de software UML para la elaboración del diagrama de clases para representar los tipos de datos abstractos.

**4. Ejecución**

**a. Tipos de datos abstractos**



**b. Algoritmos**

**Algoritmo de búsqueda**

El primer algoritmo que se utiliza busca entre el arreglo de comidas y encuentra el que es más cercano, iterando y comparando el valor más bajo encontrado hasta el momento con los demás y que sea mayor a 100. De esta manera, se encuentra la mejor opción para direccionar al bot.

**Problema del viajante**

En el Problema del Agente Viajero - TSP (Travelling Salesman Problem), el objetivo es encontrar un recorrido completo que conecte todos los nodos de una red, visitandolos tan solo una vez y volviendo al punto de partida, y que además minimice la distancia total de la ruta.

**c. Espacio o espacios de búsqueda**

El estado inicial de nuestro problema es el tamaño inicial de la lombriz. El estado final es el mayor tamaño posible que puede tener esta lombriz. Las transiciones son números altos siendo que el personaje se mueve en direcciones de coordenadas y el algoritmo es el encargado de seleccionar qué coordenada tomar para que la lombriz coma y se vuelva más grande lo más rápido posible. Así como descartar los estados donde la lombriz choca con un enemigo (tamaño sería 0).

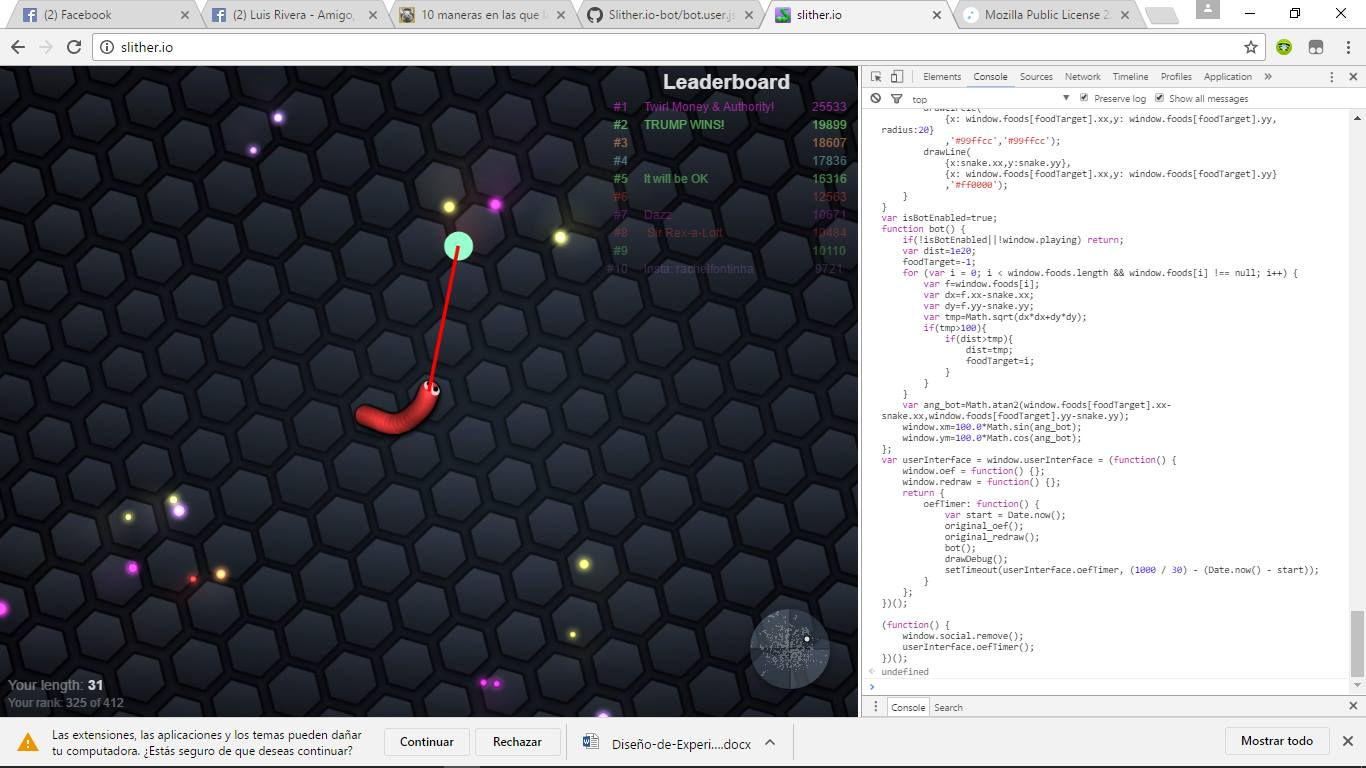
**d. Aplicación de algoritmos codiciosos**

Para la segunda parte hemos aplicado un algoritmo greedy el cual va hacia las comidas más cercanas utilizando el concepto de clusterización y también detecta los cuerpos enemigos e intenta ir a la normal de los mismos, o sea alejarse lo más posible de estos. Para poder permanecer en el juego.

**5. Análisis de complejidad de secciones críticas de la aplicación**

* La solución de búsqueda simple tiene una complejidad lineal pues hace una iteración por cada elemento, manteniendo la comida más cercana en el frame actual.
* La solución greedy por clustering tiene una complejidad n^2, pues para cada elemento se modifican los elementos cercanos añadiendoles un peso según el inverso de su distancia, emulando así pesos por clustering en el mapa.

**6. Imágenes**



**7. Conclusiones**

* Se implementó un algoritmo de búsqueda de tiempo O(n) para encontrar el alimento más cercano.
* Se implementó un algoritmo Greedy de tiempo O(n^2) para poder detectar los cuerpos enemigos y poder alejarse de ellos.
* Se logró una correcta y eficiente implementación de algoritmos.
* Se realizó el análisis cuantitativo de los algoritmos dando como resultado resultados óptimos que permiten un uso fluido de la función.

**8. Bibliografía**

* <http://www.minijuegos.com/juego/slither-io>
* <https://es.wikipedia.org/wiki/Algoritmo_de_b%C3%BAsqueda>
* <https://en.wikipedia.org/wiki/Karger%27s_algorithm>
* <http://www.geeksforgeeks.org/kargers-algorithm-for-minimum-cut-set-1-introduction-and-implementation/>
* <https://es.wikipedia.org/wiki/Programaci%C3%B3n_din%C3%A1mica>
* http://www.ingenieriaindustrialonline.com/herramientas-para-el-ingeniero-industrial/investigaci%C3%B3n-de-operaciones/problema-del-agente-viajero-tsp/