|  |  |  |
| --- | --- | --- |
| **ipn** | **INSTITUTO POLITÉCNICO NACIONAL**  **ESCUELA SUPERIOR DE CÓMPUTO** |  |

**Análisis de Algoritmos**

**“Proyecto – Algoritmos Greedy vs Fuerza Bruta”**

Resumen

El resumen debe ser preciso de todo lo que trata el reporte completo. Su función es indicar los contenidos del reporte para que el lector pueda ver si vale la pena leerlo completo o no.

**Por:**

**González Barrios Alan Ernesto**

**Romero Gamarra Joel Mauricio**

**Zavala Pérez René**

Profesor:

FRANCO MARTÍNEZ EDGARDO ADRIAN

Noviembre 2017

**Índice**

Contenido

[Introducción: 1](#_Toc500343855)

[Análisis Teórico: 6](#_Toc500343856)

[Software (librerías, paquetes, herramientas): 6](#_Toc500343857)

[Procedimiento: 6](#_Toc500343858)

[Resultados 6](#_Toc500343859)

[Discusión: 6](#_Toc500343860)

[Conclusiones: 7](#_Toc500343861)

[Referencias: 7](#_Toc500343862)

[Código 7](#_Toc500343863)

# Introducción:

Los algoritmos ávidos o voraces (Algoritmos Greedy) son algoritmos que toman decisiones de corto alcance, basadas en información inmediatamente disponible. Este tipo de algoritmos suele ser bastante rápidos para encontrar una solución, que, por cierto, la mayoría de las veces son problemas de optimización (ya sea minimizar un costo o maximizar un beneficio). [1]

Estos algoritmos tienen una serie de características o elementos que lo conforman:

1. **Conjunto de candidatos (C):** Son las entradas del problema.
2. **Función solución:** Comprueba si el subconjunto actual de candidatos elegidos forma o no una solución (sin importar si es óptima o no).
3. **Función de selección:** Indica cual es el elemento más prometedor para completar la solución. Cada elemento puede solamente ser aceptado o rechazado.
4. **Función de factibilidad:** Indica si a partir de un conjunto se puede llegar a una solución.
5. **Función objetivo:** Es la que se quiere maximizar o minimizar.

Lo que hace a los algoritmos Greedy bastante rápidos para hallar la solución es que, en cada paso, van escogiendo la solución óptima, para al final combinar las soluciones, y como todas esas pequeñas soluciones fueron óptimas, la respuesta final al problema será óptima, sin embargo, no sabemos si fue la mejor solución. Esto tiene algunas ventajas y desventajas mostradas a continuación:

**Ventajas:**

* Son mucho más rápidos que los algoritmos que encuentran la mejor respuesta ya que resuelven en problema en 1 solo ciclo.
* La programación de los mismos es bastante simple, sin embargo, encontrar la lógica no lo es tanto.
* Al resolver problemas de optimización, tiene aplicaciones como encontrar el camino más corto en un grafo de 1 punto a otro (Telecomunicaciones).
* Si toman una decisión, no se regresan a ver si la decisión tomada fue la mejor, pero sabemos que es óptima (evitan redundancia).
* La solución entregada, es óptima.

**Desventajas:**

* No garantizan que la solución entregada del problema sea la mejor solución posible.
* No todos los problemas pueden ser resueltos por un algoritmo Greedy (solo aquellos que deriven de optimización).
* Al no obtener la mejor solución, se debe comprobar al final que la respuesta entregada fue óptima.

Como podemos ver, existen más ventajas para los algoritmos voraces que desventajas, esto indica que siempre que se presente algún problema de maximizar un beneficio o minimizar un costo, un algoritmo voraz es la respuesta debido a su gran velocidad de resolver el problema en un solo ciclo sin regresar a ver si la decisión tomada (que se agrega al conjunto solución), fue la óptima.

Un algoritmo voraz trabaja por etapas, ya que, en cada una de éstas, toma la decisión que le parece más conveniente sin pensar en las repercusiones que esto pudiera tener un futuro, ya que como en ese momento, ese candidato produce un óptimo local y el algoritmo hace la suposición que provocará un óptimo global para dar una buena solución al problema. [1]

En este proyecto, nos enfocamos en 2 problemas que pueden ser resueltos con un algoritmo voraz, en uno de ellos se nos pide maximizar y en el otro, minimizar. Estos 2 problemas son descritos a continuación:

**Scarecrow [2]**

Taso adquiere un terreno muy grande en el que planea sembrar diferentes cosechas en la próxima temporada de crecimiento. Sin embargo, el terreno está lleno de cuervos y Taso tiene miedo de que los cuervos se vayan a comer sus cosechas. Para evitar esto, en la temporada de crecimiento, está planeando poner espantapájaros en diferentes puntos del terreno (que está modelado como de 1 x N m2), algunas partes del terreno son infértiles, por lo tanto, es inútil poner un espantapájaros ahí ya que no se podrá sembrar nada.

Lo que Taso sabe, es que cuando coloca un espantapájaros, cubre el espacio en el que se puso, y el espacio una posición a la derecha y una posición a la izquierda, como se muestra en la Figura 1.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Resultado de imagen para cosecha | Resultado de imagen para cosecha | Resultado de imagen para cosecha | Resultado de imagen para cosecha | Resultado de imagen para cosecha |

Figura . Ejemplo del área cubierta por un espantapájaros

Sin embargo, Taso desea minimizar el número de espantapájaros que debe poner en su terreno para proteger su cosecha de los cuervos, recordando que, si una parte del terreno es infértil, no tiene caso proteger esa área.

La entrada para este problema (en el caso particular de este proyecto) es:

* La forma del terreno (el usuario arrastra las imágenes de la cosecha (maíz) y de la tierra infértil a su gusto).

Restricciones:

* El terreno tiene una longitud de 1 x N, con N menor o igual a 10 (para apreciar bien la animación).

En las Figura 2, se muestra la pantalla de inicio para el usuario.

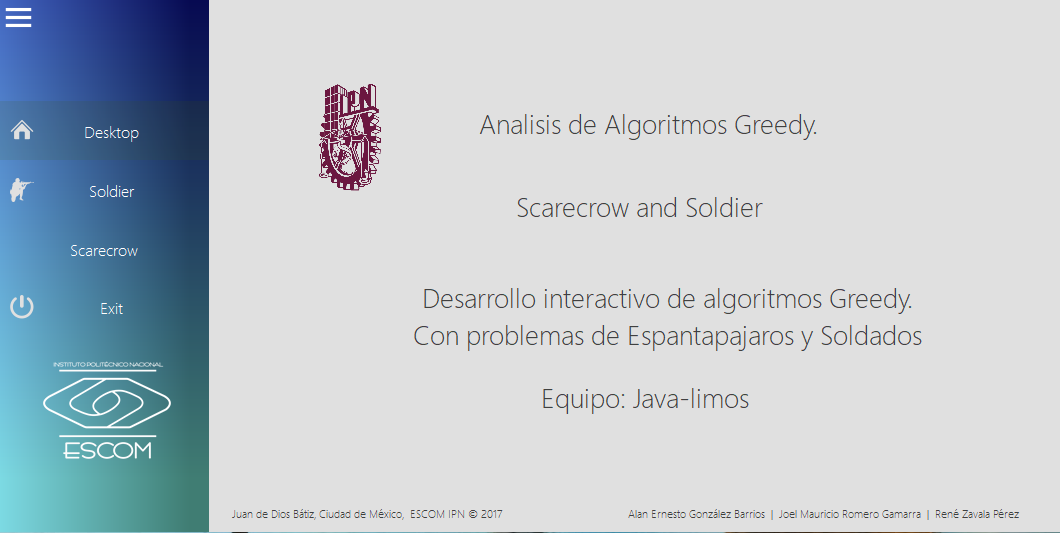


Figura . Pantalla de inicio

A continuación, en las Figuras 3 y 4 se muestra la interfaz en la opción para Scarecrow.

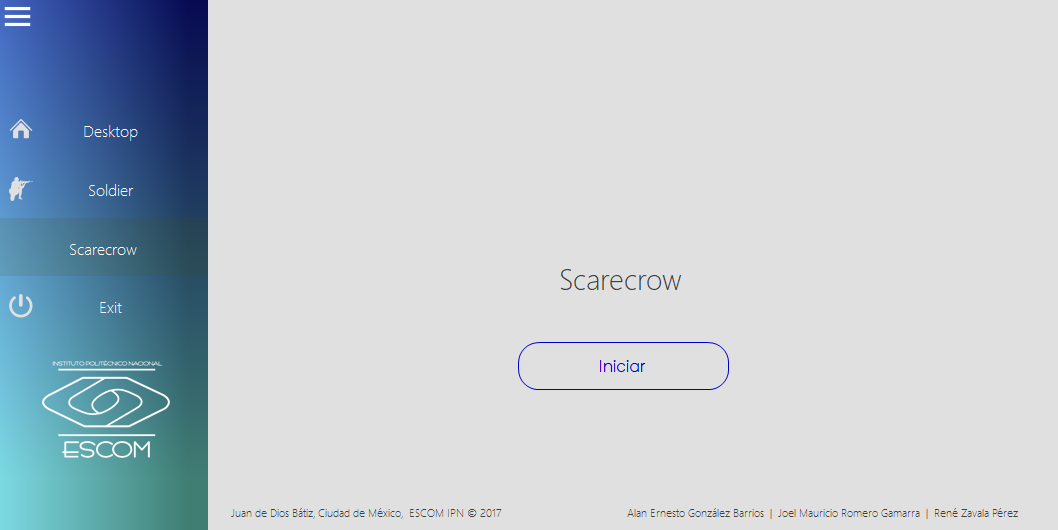


Figura . Interfaz en la opción de Scarecrow

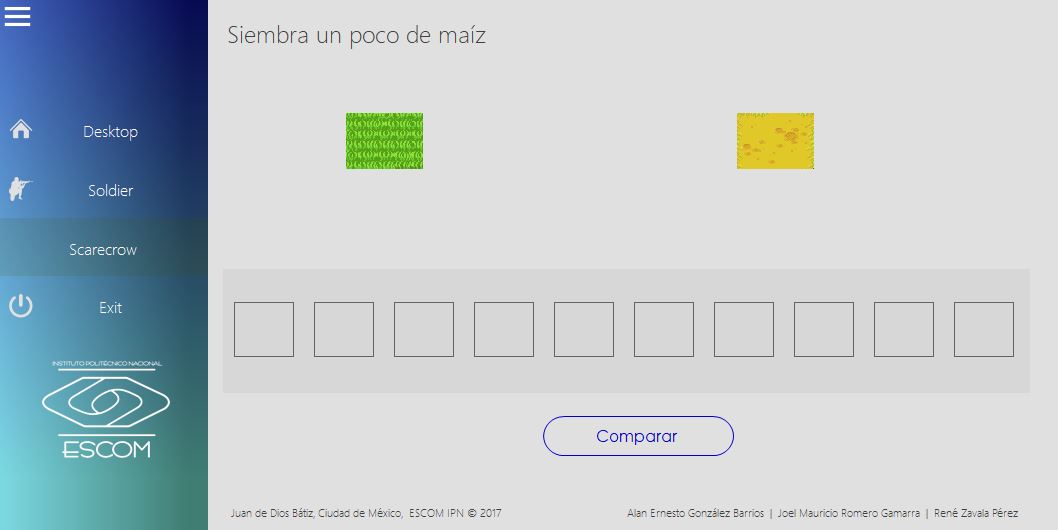


Figura . Interfaz para colocar las cosechas y tierra infértil

Como se puede ver en la parte superior de la Figura 4, las imágenes están puestas para poderlas arrastrar y formar la cadena en los espacios de la parte inferior.

**Bear and Row 01 [3]**

Limak es un oso polar que está jugando un videojuego, hay una fila con N espacios, cada una está vacía u ocupada por un soldado. El objetivo del juego es mover a todos los soldados de la fila hacia la derecha, como se muestra en las Figuras 5 y 6.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

Figura . Fila de soldados al iniciar el juego

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |

Figura . Fila de soldados al terminar el juego

La única manera de lograrlo es escoger un soldado y darle la orden de que se mueva lo más a la derecha posible, sin embargo, escoger un soldado y darle la orden tarda 1 segundo, además, un soldado tarda 1 segundo en moverse 1 posición, el soldado al que se le dio la orden se detiene inmediatamente si el espacio que sigue ya está ocupado por un soldado.

Como a Limak le gusta mucho este juego, desea jugar el mayor tiempo posible, sin embargo, el no puede dar una orden a otro soldado hasta que el soldado que se está moviendo en ese momento termine de moverse. El objetivo, es decirle a Limak ¿cuántos segundos puede jugar máximo?

La entrada y restricciones para este problema son exactamente las mismas que para el problema de los espantapájaros, con una longitud máxima de la fila de 10 elementos.

En las Figuras 7 y 8, se muestra la interfaz en la opción para Bear and Row 01.

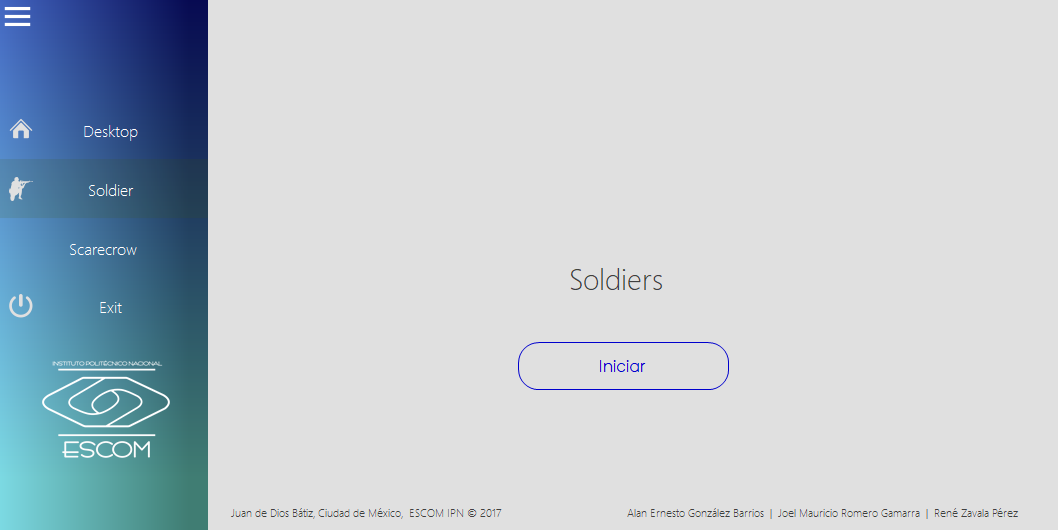


Figura . Interfaz en la opción de Bear and Row 01

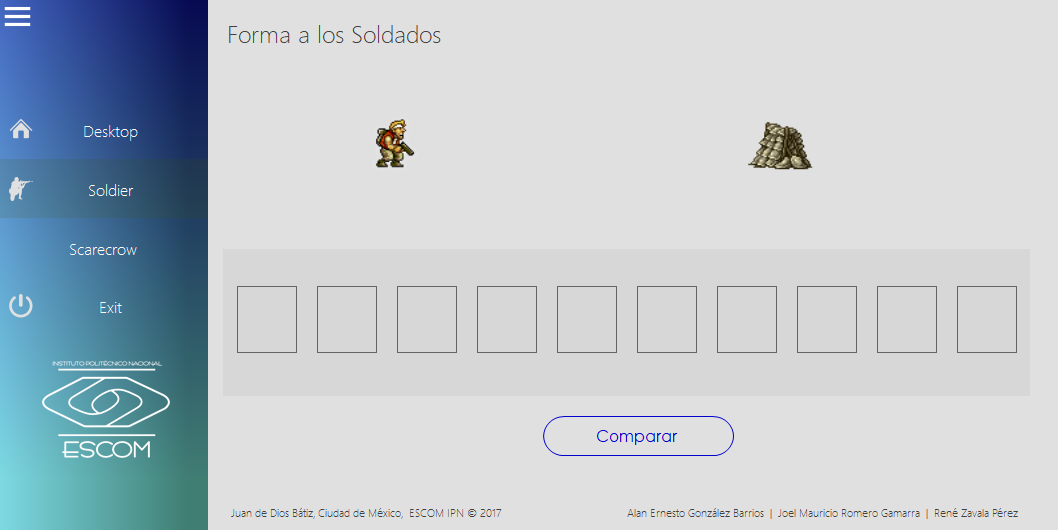


Figura . Interfaz para formar a los soldados y obstáculos

a

# Análisis Teórico:

Esta sección necesita proveer un entendimiento del teórico, matemático y conceptual del contexto, antecedentes y justificación del trabajo.

Se pueden incluir diagramas, fórmulas, algoritmos, etc.

# Software (librerías, paquetes, herramientas):

* Visual Studio Enterprise 2017: IDE [4]
* Bunifu: .NET UI Framework [5]

# Procedimiento:

\* Diagramas de Flujo / Diagrama a Bloques  
\* Agregar detalles (paso a paso) del procedimiento de manera que cualquier persona que lea pueda repetir el experimento.

# Resultados

\* Esta sección debe incluir cualquier tabla de datos, observaciones, imágenes, etc.  
\* Todas las tablas y gráficas deben estar debidamente etiquetadas.

Esta sección describe, pero no explica los resultados obtenidos.

Puesto que se están presentando los resultados y no las cifras/figuras que representan los resultados, debemos asegurarnos de que nos referimos explícitamente a nuestros resultados y nada más, no sólo a las cifras/figuras (gráficas o tablas). Al describir resultados particulares en el texto de esta sección, **debemos asegurarnos de consultar la figura correspondiente entre paréntesis después de mencionarla en los resultados**. Las figuras deben ser insertadas en el texto lo más pronto posible después de haberlas mencionado.

# Discusión:

La sección de discusión tiene 2 objetivos principales:

* Interpretar y explicar los resultados del estudio.
* Explorar la importancia del estudio, encontrando, calificando y explorando la importancia teórica de los resultados.

La discusión es también un espacio en el reporte donde cualquier calificación o reservación que se tiene sobre la investigación debe ser mencionada.

# Conclusiones:

* Lista una cosa que hayas aprendido y describe como lo aplicarías a una situación de la vida real.
* Discute los posibles errores que podrían haber ocurrido en la colección de los datos (errores experimentales).
* ¿Cómo se aplicarían los resultados obtenidos generalmente?
* ¿Hubo algún defecto en el diseño experimental o en el procedimiento?

# Referencias:

**[1]** Edgardo Adrián Franco Martínez, “Análisis de Algoritmos: Algoritmos Ávidos”, Noviembre 2017. [Online]. Disponible en: [http://www.eafranco.com/docencia/analisisdealgoritmos/files/09 /Tema09.pdf](http://www.eafranco.com/docencia/analisisdealgoritmos/files/09%20/Tema09.pdf)

**[2]** UVa Online Judge, “12405 - Scarecrow”. [Online]. Disponible en: <https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=657&page=show_problem&problem=3836>

**[3]** Kamil Debowski, “Bear and Row 01”, April 2017. [Online]. Disponible en: <https://www.codechef.com/problems/ROWSOLD>

**[4]** Microsoft, “Visual Studio Enterprise”. Disponible en: <https://www.visualstudio.com/es/vs/enterprise/>

**[5]** Bunifu, “.NET UI Framework – Empowering Developers to build faster, beautiful with less”. Disponible en: <https://devtools.bunifu.co.ke/>

# Código

Incluir todo el código fuente, comentar todo el código reutilizado y mostrar referencias.

Se debe usar el siguiente link para darle formato al código.

<https://tohtml.com/c/>