Análisis de Algoritmos | IC4301

Proyecto #2

Profesor. Joss Pecou Johnson

Pertenece a:

• Juan Mora Acuña | 2024126028

• Angelo Piedra Castro | 2024101262

• Poll Garro Vargas | 2024129001

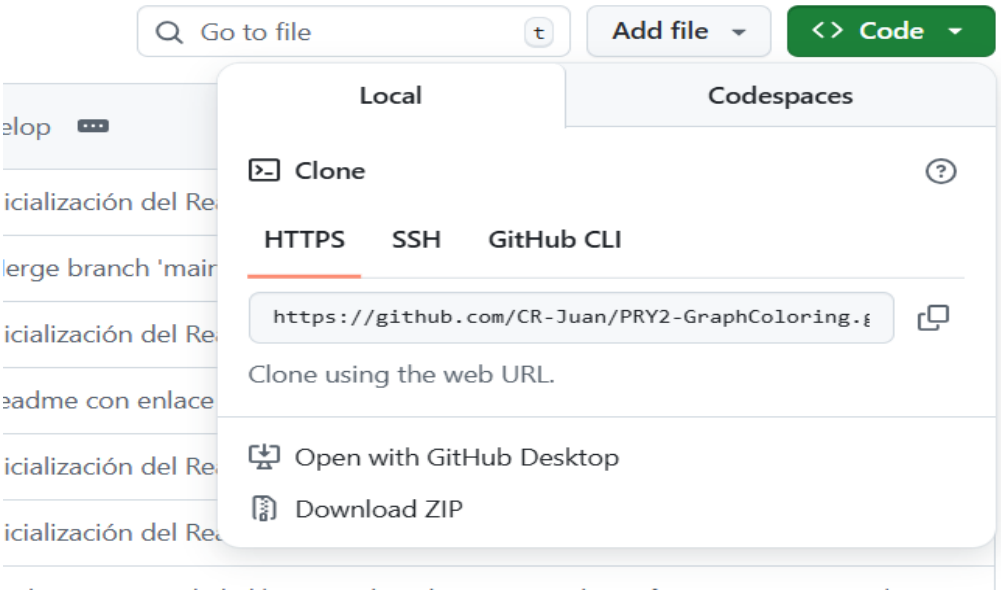
Fecha de entrega: 03/12/2025

Grupo | 60

II Semestre | 2025

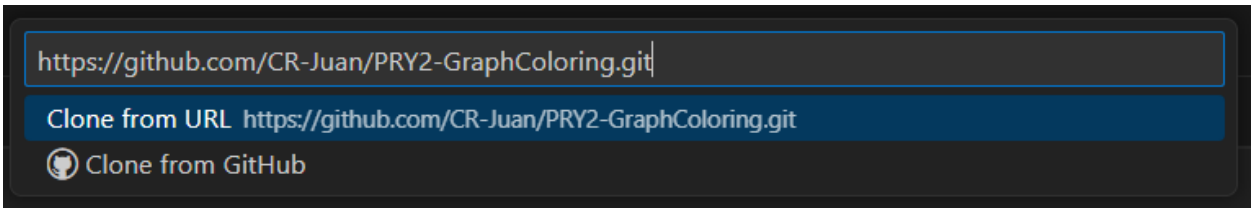
Instrucciones de instalación, ejecución y despliegue

Primero debe ingresar al repositorio en **GitHub**, acceder al proyecto y copiar el siguiente enlace del repositorio:



Luego ingrese a **Visual Studio Code** (de preferencia) y clone el repositorio utilizando el siguiente comando:

Haga clic en la opción **Clone Git Repository** y, en la barra de búsqueda, pegue el enlace del repositorio.



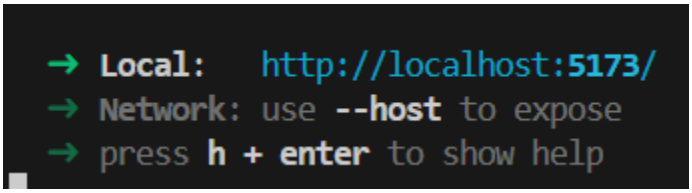
De esta manera podrá tener acceso completo al proyecto. Una vez dentro del repositorio, cree una nueva terminal y ejecute el siguiente comando:

npm install

Después de ejecutar el comando anterior, ejecute el siguiente comando:

npm run dev

Esto generará un enlace directo al proyecto. Basta con hacer clic sobre él para acceder a la aplicación, o bien copiarlo y pegarlo en su navegador de preferencia. El enlace tendrá un formato similar al siguiente:



Y listo, esto lo llevará directamente al programa, el cual debería verse de la siguiente manera:

Descripción detallada de los algoritmos utilizados

Algoritmo Monte Carlo

El algoritmo Monte Carlo es un algoritmo aleatorio en el que se permite la posibilidad de obtener una solución incorrecta o incompleta, pero a cambio se obtiene un tiempo de ejecución acotado y predecible. Es decir, el algoritmo siempre termina en un número de pasos definido (por ejemplo, un número máximo de intentos), aunque eso implique que, en algunos casos, no encuentre una solución válida aun cuando exista.

En el contexto general, un algoritmo Monte Carlo funciona así:

1. **Inicialización de parámetros**
    a. Se define un número máximo de intentos o iteraciones.
    b. Se fija la estructura del problema (por ejemplo, el grafo a colorear y el valor de k colores).
2. **Generación aleatoria de soluciones**
    a. En cada intento, el algoritmo genera una solución candidata de forma aleatoria (por ejemplo, asignando un color aleatorio a cada nodo del grafo).
    b. Esta generación puede ser completamente aleatoria o estar guiada por ciertas heurísticas (por ejemplo, evitar reusar colores en nodos vecinos cuando sea posible).
3. **Evaluación de la solución**
    a. La solución generada se evalúa según los criterios del problema.
    b. En el caso de coloración de grafos, se verifica cuántos conflictos hay (es decir, cuántas aristas conectan nodos con el mismo color).
    c. Se puede registrar información como: número de conflictos, tiempo de ejecución, cantidad de intentos usados, etc.
4. **Actualización del mejor resultado**
    a. Se conserva la mejor solución encontrada hasta el momento (por ejemplo, la que tenga menos conflictos).
    b. Aunque no se logre una solución perfecta, el algoritmo devuelve esta mejor solución como resultado aproximado.
5. **Criterio de parada**
    a. El algoritmo se detiene al alcanzar el número máximo de intentos o cuando se encuentra una solución sin conflictos, lo que ocurra primero.
    b. Incluso si no se encuentra una solución perfecta, el algoritmo siempre termina.

Características principales del algoritmo Monte Carlo

- **Tiempo de ejecución controlado**:
  El límite de intentos o iteraciones garantiza que el algoritmo siempre termine en un tiempo predecible.
- **Soluciones aproximadas**:
  Puede devolver soluciones con conflictos. Sin embargo, sirve para obtener resultados promedio, estudiar el comportamiento del algoritmo y comparar diferentes configuraciones de parámetros.
- **Dependencia de la aleatoriedad**:
  Resultados diferentes pueden obtenerse al ejecutar el algoritmo varias veces con la misma configuración, debido al uso de números aleatorios.

Aplicación en la coloración probabilística de grafos

En el problema de colorar un grafo con k colores:

- Monte Carlo intenta varias coloraciones aleatorias del grafo.
- Para cada intento, se calcula el número de conflictos (aristas cuyos extremos comparten color).
- Se registra cuál fue la mejor coloración encontrada (por ejemplo, la que presenta menos conflictos).
- Al finalizar el número de intentos establecido, el algoritmo devuelve:
  - El mejor resultado encontrado.
  - Métricas como: número total de intentos, tiempo de ejecución, probabilidad de encontrar solución válida (según las pruebas realizadas), etc.

Esto permite comparar el rendimiento del algoritmo para distintos valores de k y diferentes grafos, analizando tanto la calidad de las soluciones como el tiempo requerido.

Algoritmo Las Vegas

El algoritmo Las Vegas es también un algoritmo aleatorio, pero con una diferencia fundamental respecto a Monte Carlo. Cuando el algoritmo termina, la solución que devuelve es siempre correcta. La aleatoriedad afecta el tiempo de ejecución (cuántos intentos necesita), pero no la validez del resultado final.

En otras palabras, el algoritmo Las Vegas nunca devuelve una solución incorrecta, pero su tiempo de ejecución es aleatorio y, en teoría, puede ser muy grande (o incluso no terminar, si no se impone algún límite práctico).

En el contexto general, un algoritmo Las Vegas funciona así:

1. **Inicialización del problema**
    a. Se define el problema (por ejemplo, el grafo y el valor de k colores).
    b. Se inicializan estructuras para medir el número de intentos y el tiempo de ejecución.
2. **Bucle principal aleatorio**
    a. El algoritmo entra en un ciclo donde, en cada iteración, genera una solución candidata usando aleatoriedad (por ejemplo, coloraciones aleatorias del grafo).
    b. Cada solución generada se verifica completamente.
3. **Verificación estricta de la solución**
    a. Si la solución candidata cumple todas las restricciones del problema (por ejemplo, no hay ninguna arista que conecte nodos con el mismo color), se considera que se ha encontrado una solución válida.
    b. En ese momento, el algoritmo se detiene y devuelve la solución correcta.
4. **Repetición en caso de fallo**
    a. Si la solución candidata no cumple las condiciones (por ejemplo, hay conflictos en la coloración), se descarta por completo.
    b. El algoritmo realiza otro intento, generando una nueva solución aleatoria y repitiendo el proceso.
5. **Criterio de parada**
    a. De forma teórica, un algoritmo Las Vegas podría continuar hasta encontrar una solución válida, sin límite.
    b. En la práctica, para evitar tiempos infinitos o excesivamente largos, se suele imponer un límite máximo de intentos o tiempo. Sin embargo, si se añade este límite, existe la posibilidad de que el algoritmo termine sin solución: en ese caso, se considera que no pudo encontrar una solución en los recursos permitidos, pero nunca entrega una solución incorrecta como válida.

Características principales del método Las Vegas

- **Solución siempre correcta (cuando la hay)**:
  Solo termina cuando encuentra una solución válida que cumple todas las restricciones del problema.
- **Tiempo de ejecución aleatorio**:
  El número de intentos necesarios puede variar mucho entre una ejecución y otra. En algunos casos encuentra solución muy rápido; en otros, puede tardar más.
- **Uso intensivo en problemas donde se requiere exactitud**:
  Es útil cuando no es aceptable devolver soluciones aproximadas o con errores.

Aplicación en la coloración probabilística de grafos

En el problema de colorar un grafo con k-colores:

- El algoritmo genera coloraciones aleatorias del grafo.
- Para cada coloración:
  - Verifica si existe algún conflicto (aristas cuyos extremos tienen el mismo color).
  - Si hay conflictos, descarta esa coloración y vuelve a intentarlo.
  - Si no hay conflictos, devuelve la coloración como una solución válida de k-coloración.
- En la teoría pura de Las Vegas, el algoritmo seguirá intentando hasta encontrar una coloración válida (si existe).
- En la práctica, muchas implementaciones añaden:
  - Un contador de intentos.
  - Un límite máximo para evitar que el algoritmo quede ejecutándose indefinidamente en grafos o configuraciones muy difíciles.

Al analizar los resultados, se mide:

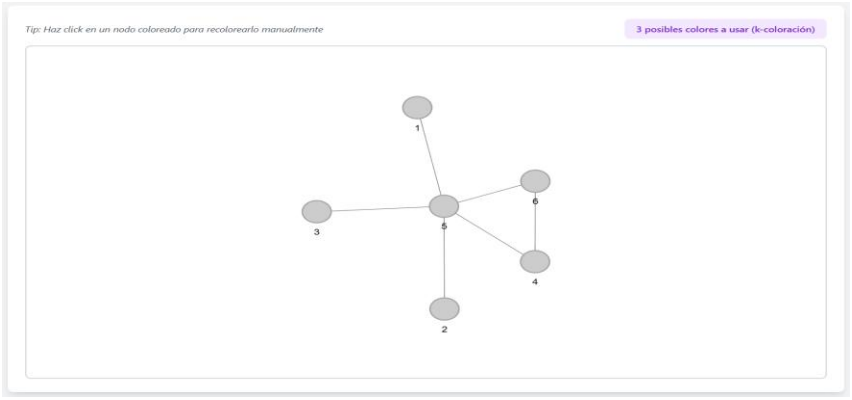
- Cuántos intentos en promedio necesita para encontrar una solución válida.
- Cuánto tiempo de ejecución requiere en comparación con el algoritmo Monte Carlo.
- Para qué valores de k y para qué tamaños de grafos se vuelve más eficiente o costoso.

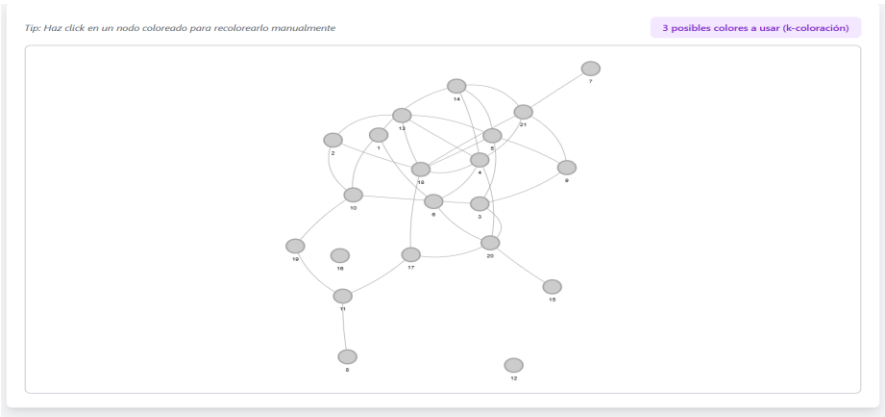
Capturas de pantalla del funcionamiento del sistema.

Tenemos tres partes importantes en el sistema de nodos que son:

En estos es que se van a manejar los nodos donde se puede agregar un nodo extra a él grafo y también se pueden conectar dos nodos o varios nodos desde dos índices con esto empezaremos por el grafo aleatorio.

Generar aleatoriamente: En este se genera grafos grises aleatoriamente desde 5 hasta 120 aquí algunos ejemplos:

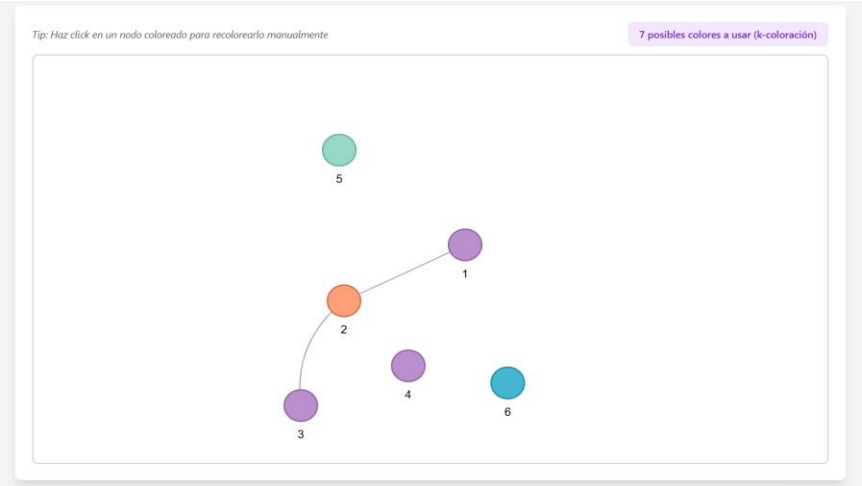




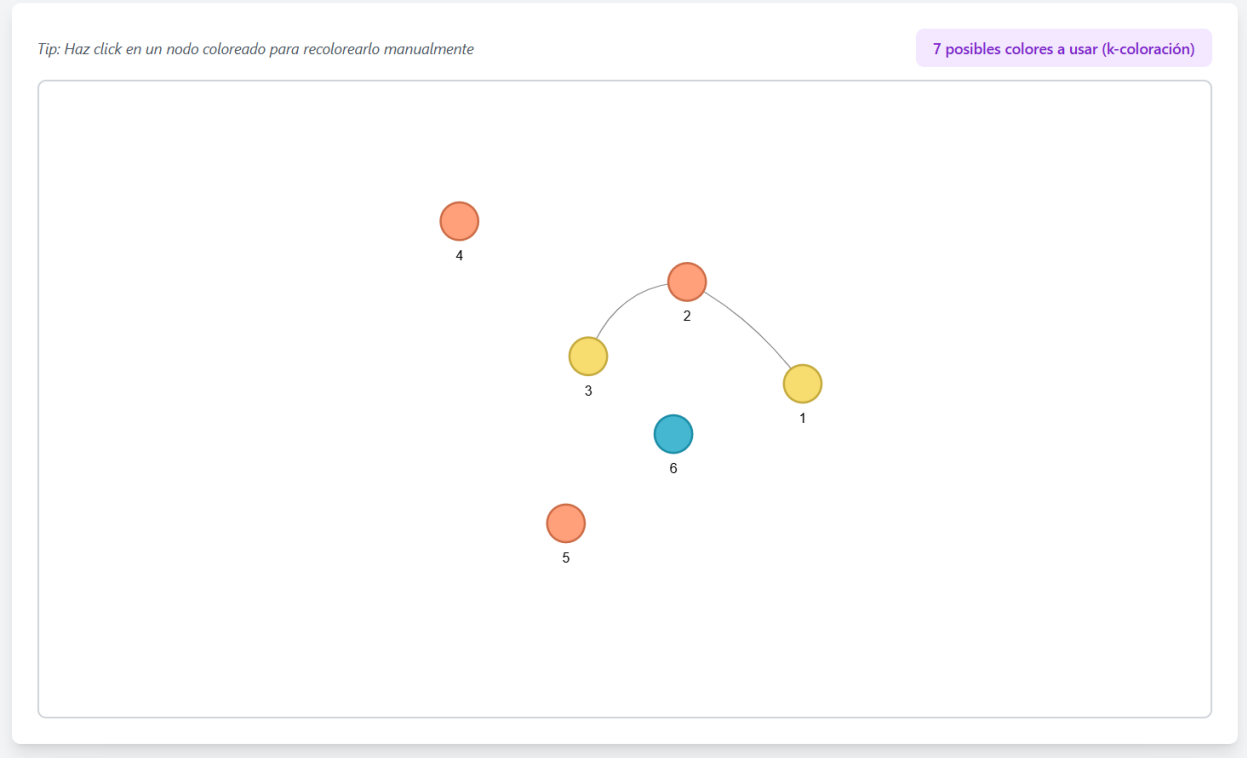
Ahora en esta sección se describe el proceso de creación del grafo y la aplicación de los algoritmos de coloración probabilística Monte Carlo y Las Vegas, los cuales permiten analizar el comportamiento del sistema frente a soluciones aproximadas y soluciones estrictamente válidas, respectivamente aquí algunos ejemplos:

Las Vegas:

Esta con auto incremento en K:



Ahora **Las Vegas** sin auto incremento en K:



Estadísticas Detalladas

Última Ejecución

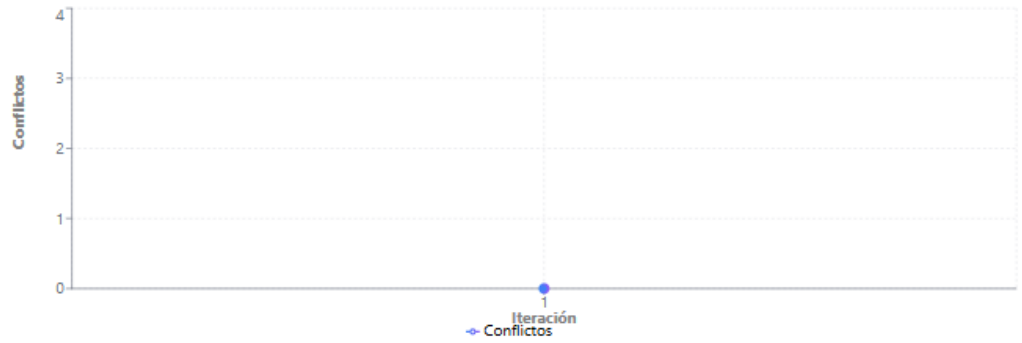
Algoritmo	Iteraciones	Tiempo	Conflictos	Estado
Las Vegas	1	0.10ms	0	✓ Válida

Evolución de Conflictos

Expandir

Mín. Conflictos	Promedio	Máx. Conflictos
0	0.00	0

Zoom gráfica: 100% Restablecer

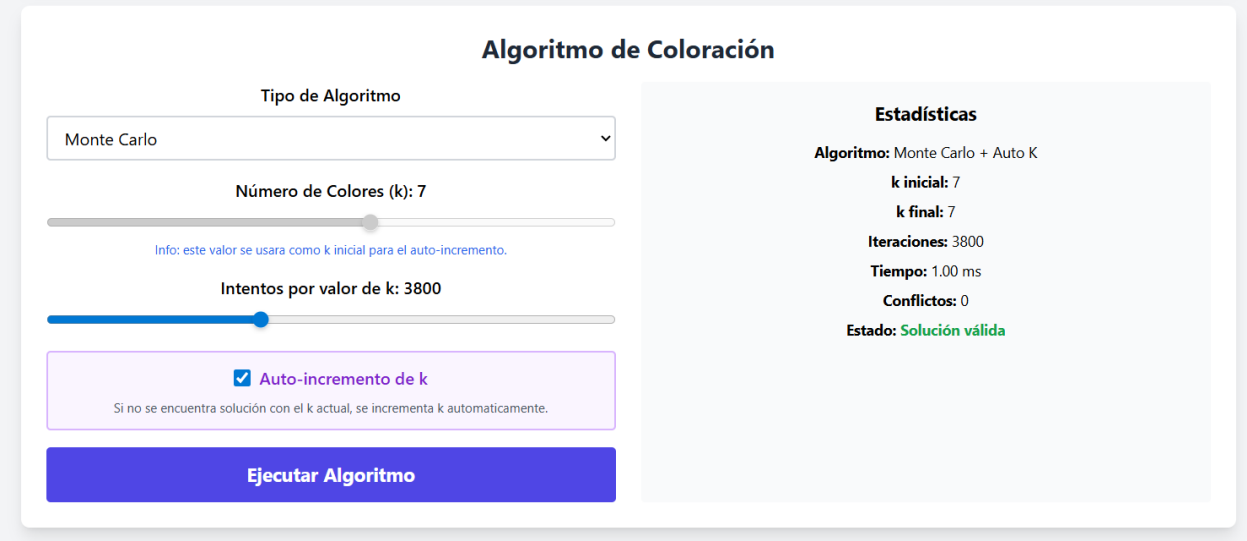


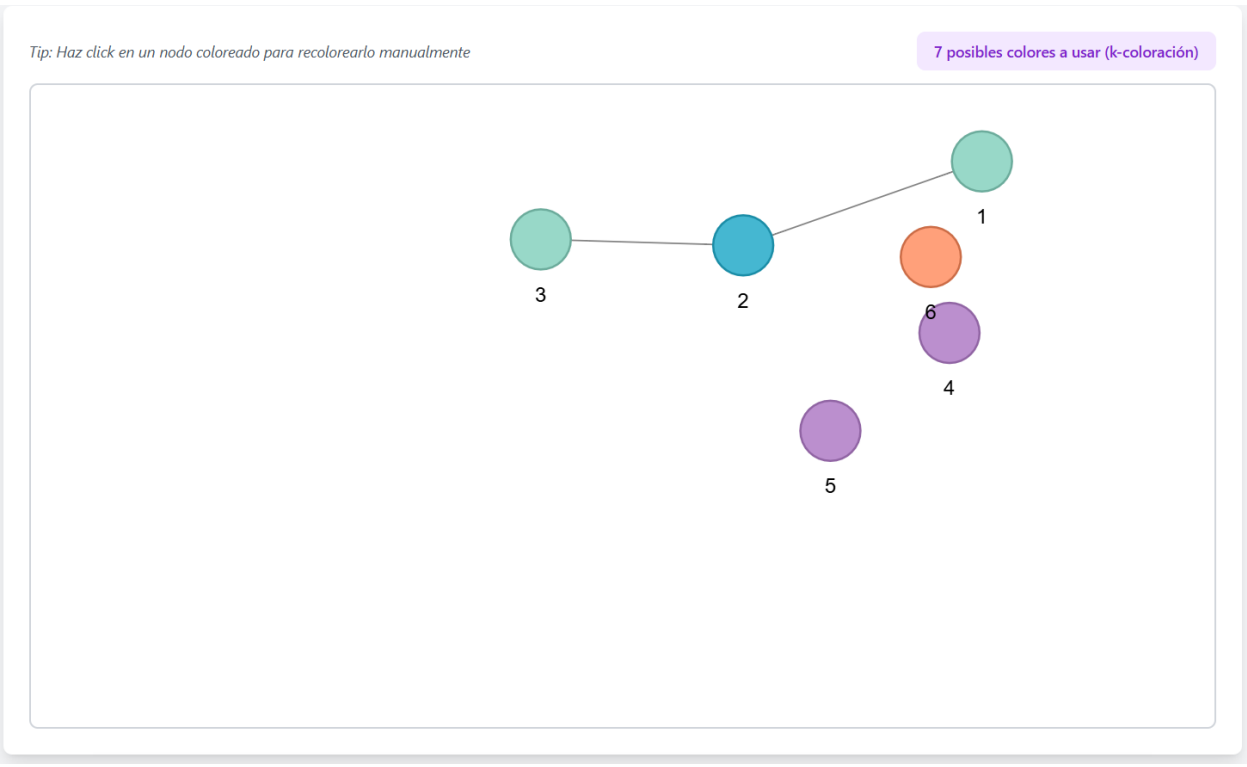
Historial de Ejecuciones (2 total)

#	ALGORITMO	ITERACIONES	TIEMPO (MS)	CONFLICTOS	ESTADO
2	Las Vegas	1	0.10	0	✓ Válida
1	Las Vegas + Auto K	1	1.20	0	✓ Válida

Monte Carlo:

Ahora seguimos con el de **Monte Carlo** y primero la probaremos con el auto incremento en K:





Estadísticas Detalladas

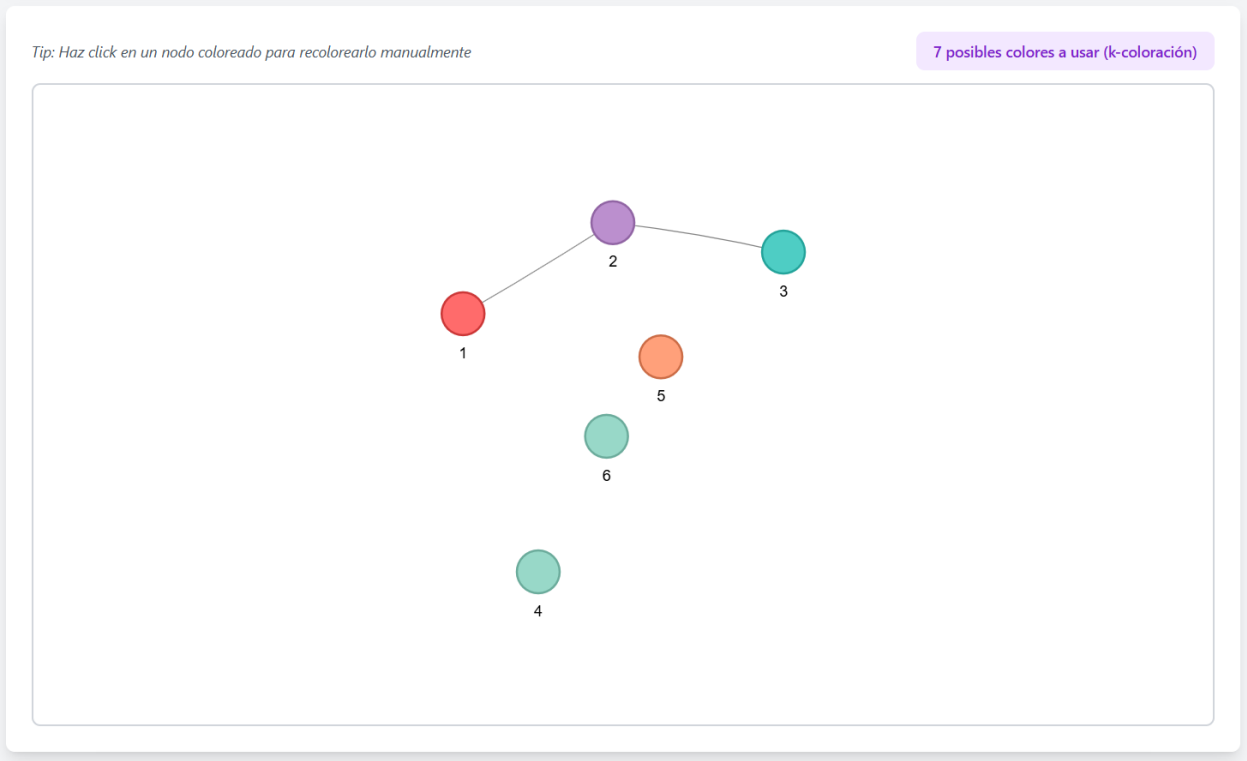
Última Ejecución

Algoritmo	Iteraciones	Tiempo	Conflictos	Estado
Monte Carlo + Auto K	3 800	0.60ms	0	✓ Válida

Historial de Ejecuciones (3 total)

#	ALGORITMO	ITERACIONES	TIEMPO (MS)	CONFLICTOS	ESTADO
3	Monte Carlo + Auto K	3 800	0.60	0	✓ Válida
2	Las Vegas	1	0.10	0	✓ Válida
1	Las Vegas + Auto K	1	1.20	0	✓ Válida

Ahora lo veremos en el algoritmo **Montecarlo** sin el auto incremento en K:



Estadísticas Detalladas

Última Ejecución

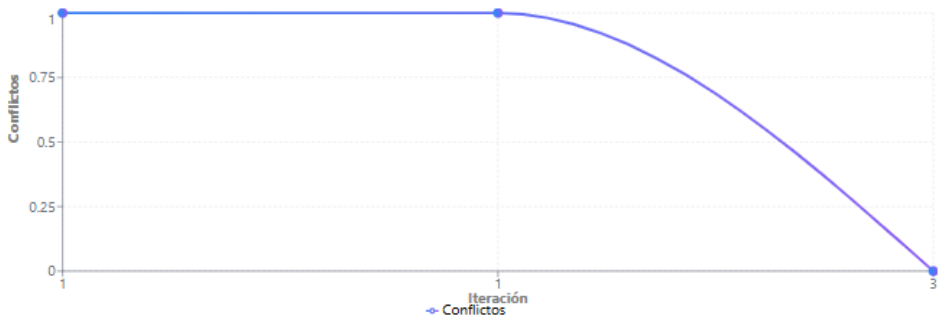
Algoritmo	Iteraciones	Tiempo	Conflictos	Estado
Monte Carlo	3 800	0.00ms	0	✓ Válida

Evolución de Conflictos

Expandir

Mín. Conflictos	Promedio	Máx. Conflictos
0	0.67	1

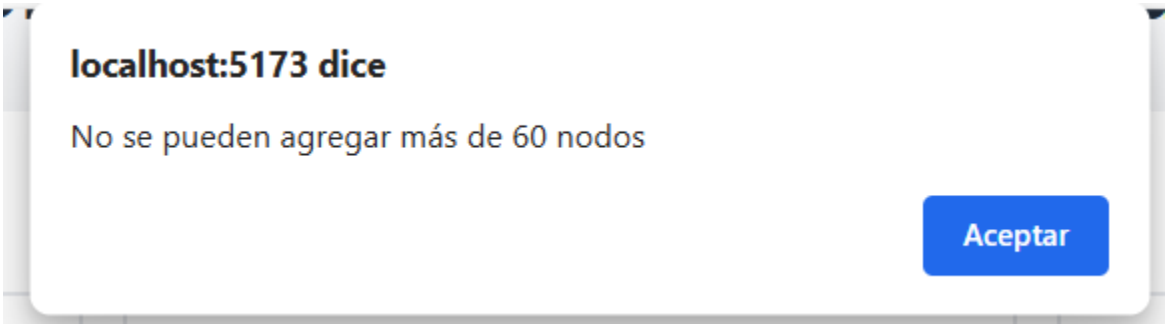
Zoom gráfica: 100% Restablecer



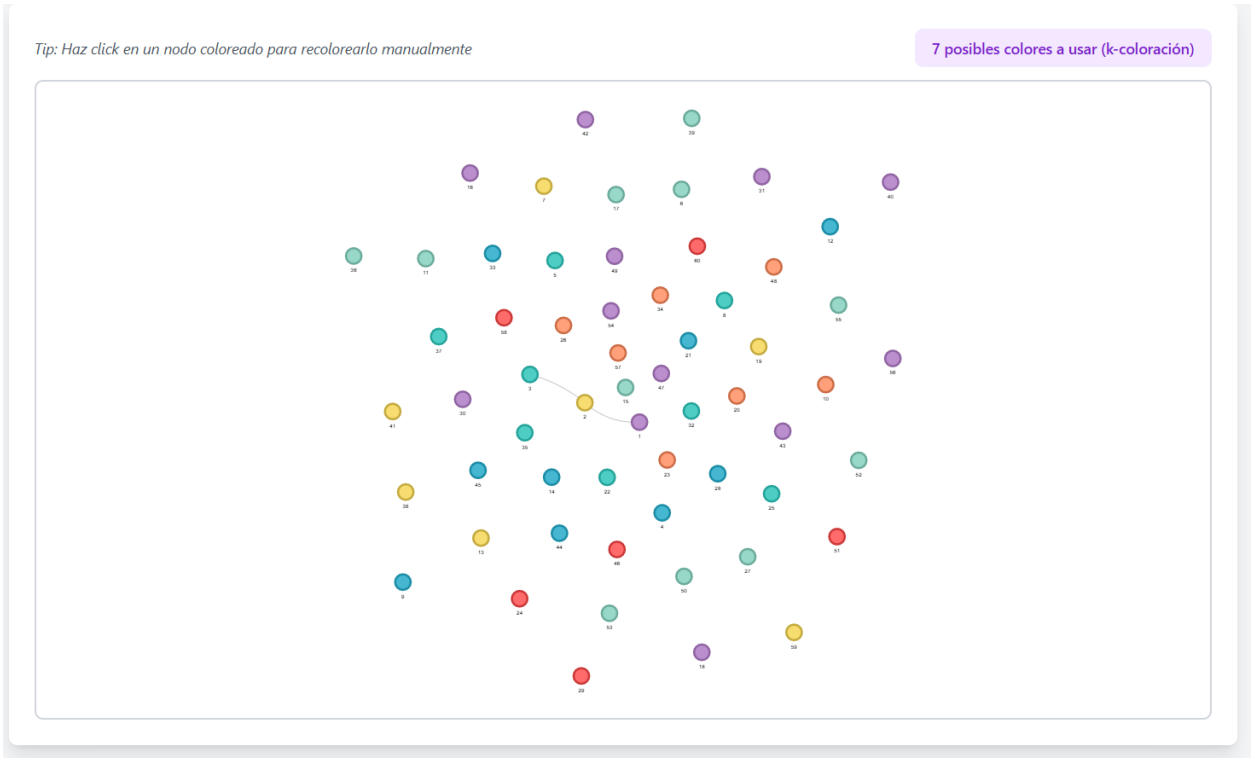
Historial de Ejecuciones (4 total)

#	ALGORITMO	ITERACIONES	TIEMPO (MS)	CONFLICTOS	ESTADO
4	Monte Carlo	3 800	0.00	0	✓ Válida
3	Monte Carlo + Auto K	3 800	0.60	0	✓ Válida
2	Las Vegas	1	0.10	0	✓ Válida
1	Las Vegas + Auto K	1	1.20	0	✓ Válida

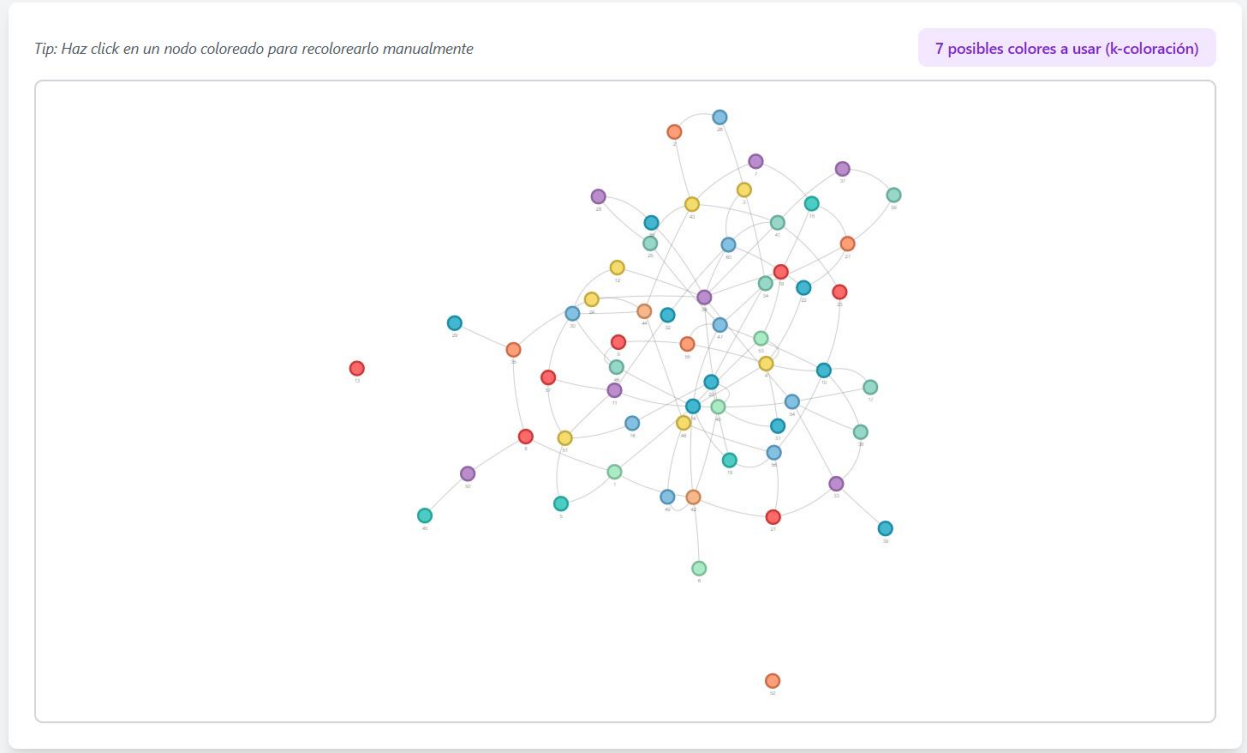
Vemos una validación de que no se pueden agregar más de 60 nodos:



Y vemos como se vería:



Y de esta manera se vería varios nodos conectados:



Análisis de resultados experimentales (tiempo, éxito, conflictos).

En esta sección se analizan los resultados obtenidos al ejecutar los algoritmos **Monte Carlo** y **Las Vegas** sobre grafos generados aleatoriamente, utilizando diferentes configuraciones de número de nodos, probabilidad de conexión y número de colores. El análisis se centra en tres métricas principales: **tiempo de ejecución**, **tasa de éxito** y **cantidad de conflictos**.

Tiempo de ejecución

Algoritmo Monte Carlo

El algoritmo Monte Carlo presenta un tiempo de ejecución **estable y predecible**, ya que este está directamente controlado por el número máximo de intentos configurado.

- El tiempo de ejecución crece de forma directamente proporcional al número de intentos.
- A mayor cantidad de nodos:
    - Mayor tiempo de verificación.
    - Mayor consumo computacional por iteración.
- El aumento de la probabilidad de conexión incrementa el número de conflictos posibles, haciendo que la validación por iteración sea ligeramente más costosa.

Observación clave:
Monte Carlo garantiza tiempo limitado, aunque no garantiza una solución válida.

Algoritmo Las Vegas

El tiempo de ejecución del algoritmo Las Vegas es **no determinista**.

- Algunas ejecuciones encuentran solución rápidamente.
- Otras ejecuciones pueden tardar significativamente más.
- En grafos densos y con bajo número de colores:
    - El tiempo puede crecer de forma exponencial.
- En grafos dispersos o con mayor número de colores:
    - El tiempo disminuye considerablemente.

Observación clave:
Las Vegas no garantiza tiempo de finalización, pero garantiza validez.

Tasa de éxito

Monte Carlo

- No garantiza éxito.
- La tasa de éxito depende directamente de:
    - Número de iteraciones.
    - Número de colores k.
    - Densidad del grafo.
- En casos difíciles:
    - Puede terminar sin encontrar una solución válida.
- Aumentar el número de intentos incrementa la probabilidad de éxito.

Conclusión:
Monte Carlo es útil para exploración y análisis probabilístico, pero no para sistemas que requieren soluciones exactas.

Las Vegas

- La tasa de éxito es teóricamente del **100% cuando existe solución**.
- En implementaciones prácticas:
    - Si existe un límite de intentos, el éxito puede bajar.
    - Sin límite, eventualmente siempre encuentra solución (si es posible).
- En grafos imposibles de colorear con k colores:
    - El algoritmo nunca termina (sin límite).

Conclusión:
Las Vegas es confiable cuando se requiere certeza absoluta.

Conflictos

Monte Carlo

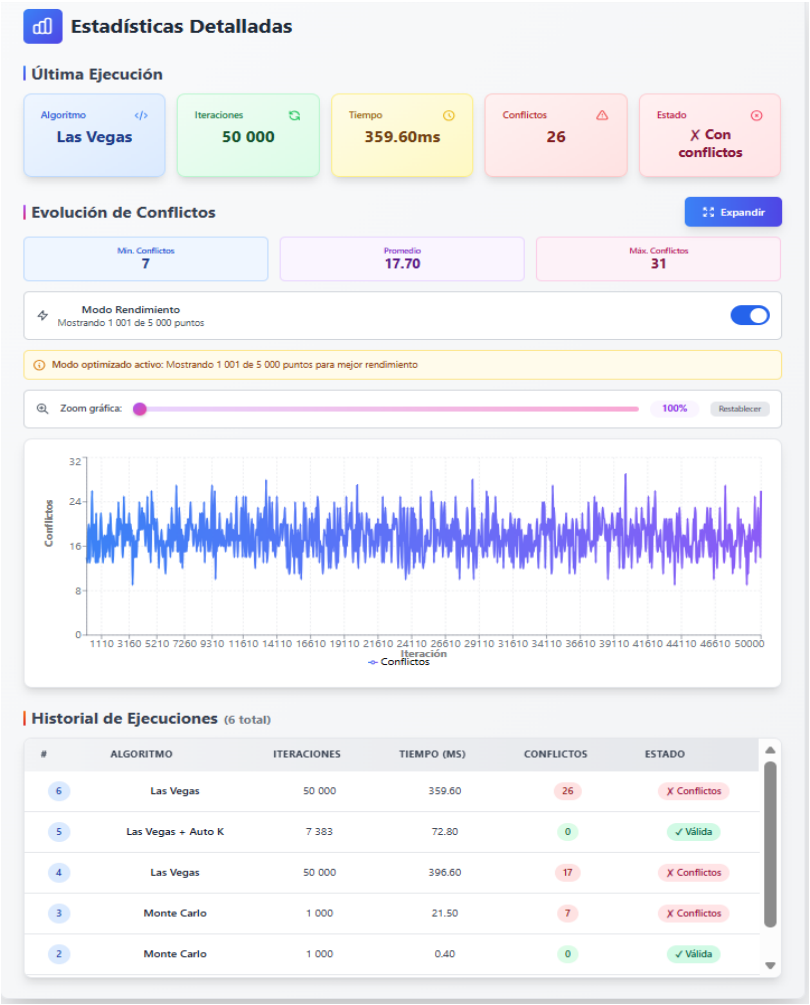
- Puede devolver soluciones con conflictos.
- Mejores soluciones aparecen tras más iteraciones.
- En muchos casos:
    - El número de conflictos disminuye gradualmente.
- Sin embargo:
    - No se puede asegurar que llegue a cero.

Las Vegas

- No existen conflictos al finalizar.
- Todas las soluciones devueltas son válidas.
- Los conflictos solo ocurren durante intentos descartados.

Panel de estadísticas

Se muestra un panel de estadísticas del algoritmo **Las Vegas**:



Se muestra el panel de estadísticas del algoritmo **Monte Carlo**:

Estadísticas

Algoritmo: Monte Carlo
Iteraciones: 1000
Tiempo: 21.50 ms
Conflictos: 7
Estado: Coloración con conflictos

Estadísticas Detalladas

Última Ejecución

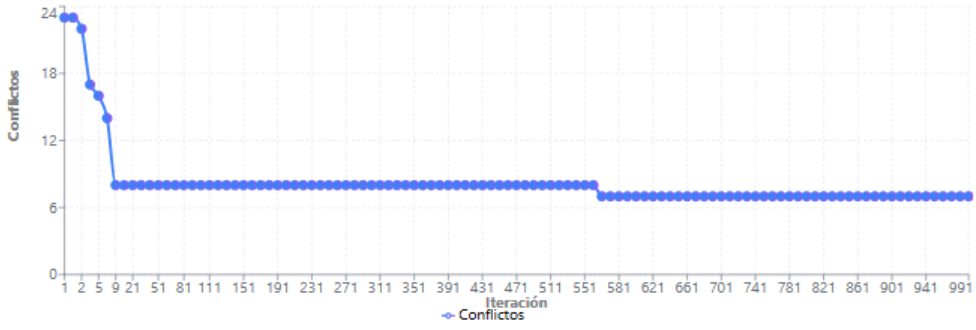
Algoritmo	Iteraciones	Tiempo	Conflictos	Estado
Monte Carlo	1 000	21.50ms	7	✗ Con conflictos

Evolución de Conflictos

Expandir

Mín. Conflictos	Promedio	Máx. Conflictos
7	8.21	23

Zoom gráfica: 100% Restablecer



Historial de Ejecuciones (3 total)

#	ALGORITMO	ITERACIONES	TIEMPO (MS)	CONFLICTOS	ESTADO
3	Monte Carlo	1 000	21.50	7	✗ Conflictos
2	Monte Carlo	1 000	0.40	0	✓ Válida
1	Las Vegas	1	0.20	0	✓ Válida

Acceso al proyecto

Enlace del deploy: **pr2-graph-coloring.netlify.app**

Enlace del Github: **https://github.com/CR-Juan/PRY2-GraphColoring.git**