Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 — Estructuras de Datos y Algoritmos
2020 - 1

Tarea 2

Fecha de entrega código: Domingo 14 de junio
Fecha de entrega informe: Domingo 14 de junio

Objetivos

- Experimentar con distintas variaciones en la implementación de una tabla de hash
- Experimentar con parámetros de la tabla de hash
- Implementar una función de hash *ad-hoc* al problema
- Analizar eficiencia de manera empírica y sacar conclusiones a partir de esto

Introducción

Abrumados por el deficiente catálogo de Spoogle Spacetadia y enfurecidos por el intergaláctico lag de su conexión, los pasajeros del Pod P137 se encuentran decepcionados por la tecnología. En un afán ludista que haría a RoboGhandi llorar lágrimas robóticas e inspirados por cierta banda de SpaceCalifornia, decidieron descargar su *ira contra la máquina*, en este caso, el pobre RoboKasparov.

El maltratado androide terminó con sus circuitos fritos, entre ellos su módulo de libre albedrío pre-programado y su tabla de hash, no pudiendo soñar con ovejas eléctricas, recordar estados de SpaceChess, ni resolver el más simple *halting problem*. Ya que un robot sin tabla de hash apropiada en su memoria principal viola la Declaración Espacial de Derechos Robóticos del año 3945, los pasajeros intentaron programar una nueva, sin embargo sus esfuerzos fueron en vano. Ni hablar del módulo de libre albedrío pre-programado.

Debido a que el equipo docente se encuentra en la prisión espacial de SpaceColinA-12 tras cierto percance con el Fisco Galáctico, es tu labor mandar una versión mejorada de la tabla de hash y reparar a RoboKasparov.

Afortunadamente, los pasajeros encontraron entre lágrimas en la lluvia un antiguo Flip Puzzle con el que podrás probar la efectividad de tu tabla antes de inyectarla en el PCB del lastimado robot.

Flip puzzle

El flip puzzle consiste en una cuadricula de $h \times w$ con números. El objetivo del puzzle es lograr hacer que cada celda de la cuadrıcula vuelva a tener el valor que tenia originalmente en el puzzle ordenado. Estos valores van de 0 a $hw - 1$, como se muestra en la siguiente figura:

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

El tablero resuelto, S_f

Los movimientos sobre el tablero corresponden a inversiones de una fila o una columna especifica, los que llamaremos flips. Al aplicar flip en una columna o fila, se intercambia el orden de los elementos dejándolos en el orden inverso. Los abreviaremos como sigue:

- Flip de fila $y \implies R_y$
- Flip de columna $x \implies C_x$

Estos se pueden ver como funciones que se aplican sobre el tablero: al aplicar una de ellas, se obtiene un nuevo tablero con una distinta configuración.

0	1	14	3
4	5	10	7
8	9	6	11
12	13	2	15

$C_2(S_f)$

0	1	14	3
7	10	5	4
8	9	6	11
12	13	2	15

$R_1(C_2(S_f))$

Espacio de Estados

En problemas como este, se le llama **estado** a una configuración en particular del tablero. Se puede definir un grafo $G(V, E)$ como el espacio de estados del problema, donde:

- V son todos los estados del problema
- $(u, v) \in E$ si y solo si existe una operación P tal que $P(u) = v$

A partir de un estado inicial, se realizara una búsqueda a través del espacio de estados hasta encontrar el estado donde el Flip Puzzle este resuelto.

Código Base

Este problema ya viene completamente resuelto en el código que te ha sido provisto. En particular, podrás encontrar 3 módulos:

- `src/state/`: Contiene la modelación del problema y sus operaciones.
- `src/graph/`: Maneja el grafo de estados. Utiliza una tabla de hash para almacenar los estados distintos.
- `src/solver/`: Utiliza BFS para recorrer el grafo de estados en busca de la ruta más corta a la solución.

El módulo `src/graph/` ha sido provisto por los sobrevivientes del Pod, sin embargo esta gente claramente no recuerda lo visto en clases de SpaceEDD, por lo que su implementación, si bien es correcta, es muy ineficiente.

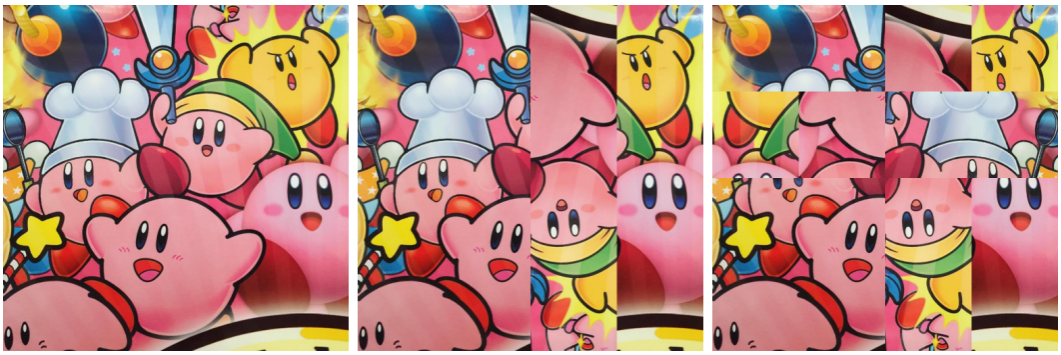
Tu tarea consiste en arreglar este código para hacerlo eficiente, pero debes tener en mente que deberás justificar tus decisiones con un análisis, ya que si no, no aceptarán tus *pull requests galácticos*.

La tabla utiliza direccionamiento abierto con *Linear Probing*. Todo el código se encuentra exhaustivamente documentado mediante comentarios.

Eres libre de editar el contenido de los archivos dentro de la carpeta `/src/graph/`, pero **no podrás** modificar nada que esté fuera de esta carpeta.

Interfaz gráfica

Se ha preparado una interfaz gráfica que permite la visualización de tanto los estados como las operaciones:



(a) S_f (b) $C_2(S_f)$ (c) $R_1(C_2(S_f))$

Ejemplos de flip en la interfaz

Esta interfaz utiliza la librería GTK, que se puede instalar en Linux, MacOS y Windows Linux Subsystem. Para esto, sigue las instrucciones disponibles en la wiki del curso en el siguiente link.

Esta definida en `/src/watcher`. Al ejecutar su tarea, aparecerá una ventana con el estado inicial, la cual permanecerá congelada hasta que se llegue a un estado solución para luego reproducir los pasos necesarios para resolverla.

Experimentos

La idea de esta tarea es que prueben libremente con distintas mejoras al funcionamiento de su tabla de hash, realizando experimentos pertinentes y documentando sus resultados. Se detallan a continuación las **categorías** sobre las cuales pueden experimentar:

A. **Distribución del Hash:** Cómo distribuye la función de hash.

B. **Complejidad del Hash:** El costo en tiempo de calcular la función de hash.[1]

C. **Método de Ajuste:** Regla según la que los valores de hash se ajustan al tamaño de la tabla.

D. **Función de *Probing*:** Regla según la cual se busca la siguiente celda dentro de la tabla.

E. **Condición de Rehashing:** Regla que indica en que momento la tabla debe aumentar su tamaño.

De esta forma, deberán realizar:

1. Tres experimentos individuales sobre categorías distintas. Por ejemplo: A, D y E.
2. Dos experimentos combinando dos experimentos realizados en 1. Por ejemplo: A+D, D+E
3. Un experimento usando las tres variables de las elegidas en 2. Por ejemplo: A+D+E

Se espera que por cada categoría pruebes distintas configuraciones y que uses diversas métricas relevantes, tales como número de colisiones, tiempo de ejecución, entre otras. De esta manera, podrás constatar y contrastar las mejoras con el rendimiento del código base y tus propias propuestas.

Puedes hacer más experimentos si quieres, pero a la hora de entregar tu tarea deberás elegir cuales dejar para corrección de manera coincidan con los experimentos solicitados.

Librería de Análisis Estadístico

Con el fin de hacer tu análisis más ameno, se ha preparado una *suite* en Python que generará gráficos y un pequeño análisis estadístico de los resultados recolectados de tu solución. Se recomienda fuertemente familiarizarte con las distintas clases y métodos, así no implementarás funcionalidades que se han entregado listas. Estás en tu libertad de añadir más *features* y tests, así como de no usar los gráficos entregados. Se proveen las siguientes clases:

1. `RegressionGenerator`: Útil para comparar el efecto de una variable sobre otra y producir un *scatterplot* de los puntos. Opcionalmente genera una regresión con los datos y entrega los resultados de un test de correlación de Pearson.
2. `HistogramGenerator`: Útil para generar los histogramas que denoten colisiones del hash o del probing en la tabla de hash.
3. `HashTester`: Útil para analizar la distribución de la función de hash y generar un histograma de sus resultados. Opcionalmente genera un test KS de fit de distribución con la hipótesis H_1 : Normalized Hash $\sim Uniform(0, 1)$.

Añadido a esto, se incluye el script `run_homework.py`, que automatiza el proceso de recolección de datos así como los distintos tests y generación de gráficos.

[1] Considera usar métodos incrementales para calcular tu función de hash a partir del hash del estado anterior y la operación que produjo el nuevo estado.

Análisis

Cada experimento deberá estar en tu repositorio como un commit independiente y documentado en formato **Markdown** dentro de una *issue* en el repositorio de tu tarea. GitHub permite la inclusión de referencias de código en las issues, por lo que **deberás** hacer las referencias pertinentes usando esta funcionalidad. **No se revisarán issues que no hagan referencia explícita al código**, lo que implicaría 0 puntos en ese experimento. En este link puedes encontrar una guía para referenciar código efectivamente.

Puedes hacer el análisis tan simple o complejo como consideres necesario. Lo importante es que argumentes con bases empíricas y datos concretos los resultados que obtengas. Es importante recalcar que en cada experimento puedes modificar **la categoría a estudiar**, pero el resto **deben** quedar constantes. En caso contrario el experimento se considerará inválido y tendrás 0 puntos en esa sección.

Finalmente, deberás incluir una *issue* a modo de conclusión. Esta debe mostrar la versión final de tu implementación utilizando las mejores configuraciones previamente encontradas, así como evidencia que demuestre su mejora significativa con respecto al código base. Recuerda que **debes** incluir referencias al código final.

Si decides dejar un experimento fuera de la corrección, no lo borres, simplemente cierra la issue ya que así queda el registro de tus experimentos anteriores. Esto puede serte útil si necesitas recorregir.

Input

El programa recibe el siguiente input `./solver puzzle` donde `puzzle` es la ruta al archivo con el puzzle. Esta además puede recibir 3 parámetros adicionales para la generación de estadísticas, pero se recomienda usar el script `generate_report.py` provisto para este propósito.

Output

El output del programa es la visualización de la solución en la interfaz. Si ejecutas el programa desde los scripts de Python no aparecerá la ventana para agilizar el proceso, pero la solución seguirá siendo registrada.

Evaluación

- **Código: 50%** Se realizarán diversos tests de performance con la mejor versión de tu tabla de hash. Será recolectado desde su rama `master`, y se utilizará el codigo base para todo menos los contenidos de `/src/graph/`. Si el output de tu programa no es exactamente el mismo que el del código base, se considerará que rompiste la funcionalidad de BFS, por lo que tendrás 0 puntos en ese test.[2].

  Si tu algoritmo demora más de 10 segundos en un test, será cortado y tendrás 0 puntos en ese test. Tu programa se probará con diversas configuraciones de complejidad creciente.

- **Informe: 50%** Por cada issue se evaluará que los experimentos sean pertinentes, estén bien desarrollados, tengan referencias directas a su código y justificación de la elección final.

[2]El código base por defecto entrega resultados correctos, esto tiene como fin que no modifiquen la modelación del problema

Entrega

- **Código:** GIT - Repositorio asignado. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.
- **Informe:** GIT - Issues en repositorio asignado. Markdown. Sigue las instrucciones de la sección Informe. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.

Bonus

- **Gotta go fast:** Se otorgará un bonus de hasta 10 décimas (a la nota final de la tarea) a los 20 estudiantes cuya solución corra más rápido. 10 décimas a los dos primeros lugares, 9 al tercer y cuarto puesto, etc.

Política de recorrección

- **Código**: Se mantiene la política de recorrección de tareas pasada, excepto que en este caso, será el ayudante quien decida si el cambio es válido o no.
- **Informe**: En caso de que uno de tus experimentos sea rachazado, podrás cambiarlo por alguno que se encuentre en una *issue* cerrada.