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

IIC2133 — Estructuras de Datos y Algoritmos
2021 - 1

Tarea 2

Fecha de entrega código e informe: viernes 11 de junio

Objetivos

- Modelar un problema y encontrar una solución eficiente usando funciones de hash.
- Analizar distintas técnicas de hash en la eficiencia de una solución.
- Documentar el diseño de la solución de un problema algorítmico.

Introducción

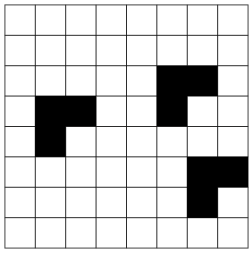
¡Pobre Kevin! ¡Ha sido funado! A pesar del intento de Kevin de expresar su arte, a muchos no les gustó por su complejidad. Los amigos de Kevin ingresaron a varios canales anónimos de confesiones, en donde se percataron de que sus obras eran objeto de discusiones polémicas. Sumergido en la melancolía, le pidió a su cliente, `Gepe`, más plazo para la entrega de su trabajo y renunció a la idea de volver a hacer arte.

A la izquierda, Kevin llorando a color. A la derecha, Kevin llorando con el filtro de la T1.

Tras una semana de receso, Kevin conversa con su amigo (y proveedor) `Walter White`, quien le comenta que **(no spoiler)** tiene una plaga de moscas y de otros insectos que le imposibilitan trabajar. Sumido en la desesperación, Walter le pide a Kevin que le ayude con un programa para identificar rápidamente distintos tipos de insectos en fotos de paredes.

Problema

Walter White tiene imágenes cuadradas de $n \times n$ píxeles de paredes con plagas, en donde cada píxel es blanco si forma parte de la pared o negro si forma parte de un posible insecto.

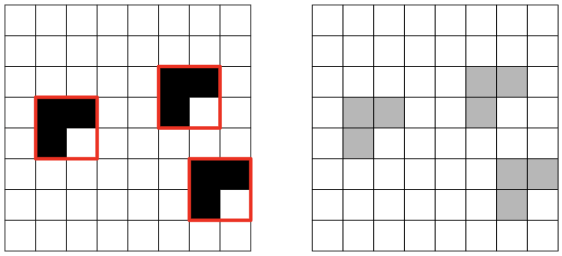


Ejemplo: Pared de 8×8 píxeles.

Para cada imagen de una pared, Walter le entrega a Kevin otras imágenes cuadradas más pequeñas de $m \times m$ píxeles ($m < n$). La sub-imagen representa al tipo de insecto que Walter quiere que tu programa identifique.

Ejemplo: Insecto de 2×2 píxeles.

Tu programa deberá identificar todas las ocurrencias de la sub-imagen en la imagen principal y después de recorrer la imagen, deberá pintar de gris los píxeles negros de las sub-imágenes seleccionadas. A continuación, un ejemplo utilizando las dos imágenes anteriores:

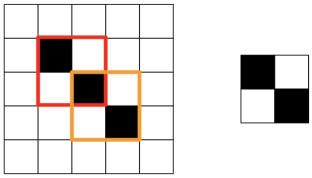


A la izquierda en rojo, las sub-imágenes seleccionadas. A la derecha, las sub-imágenes coloreadas con gris.

Deberás escribir un programa en **C** que, dada una imagen de la pared y un conjunto de `Q` sub-imágenes de insectos, retorne `Q` imágenes de la pared, cada una identificando con gris las ocurrencias de una sola sub-imagen de un insecto. Para esta labor, se recomienda **fuertemente** la utilización de hash. Es tu deber identificar una función adecuada al problema, pudiendo codificar eficientemente los datos entregados. También deberás implementar una *Hash Table* apropiada, ajustando correctamente los parámetros asociados a esta, tales como factor de carga, direccionamiento, función de *resize*, función de *probing*, entre otros.

Aclaraciones

- Toda sub-imagen que representa a un insecto tiene al menos un píxel negro.
- Si no hay ninguna ocurrencia de una sub-imagen dentro de una imagen, el programa retorna la imagen sin alteraciones.
- Dos o más ocurrencias de una misma subimagen pueden traslaparse.

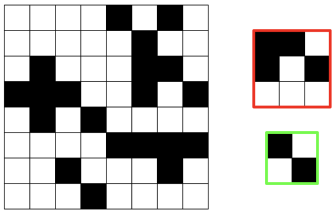


La pared (izq) de tiene dos insectos (der) que se traspalan.

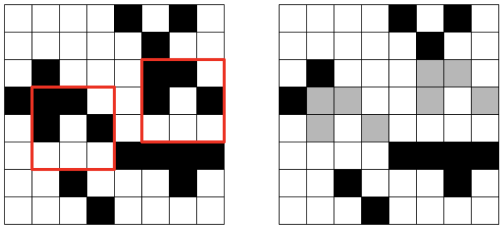
- Colorear con gris los píxeles negros de las ocurrencias se hace una vez que se hayan revisado todos los casos posibles. No se realiza inmediatamente después de haber identificado un caso.

Ejemplo más complejo

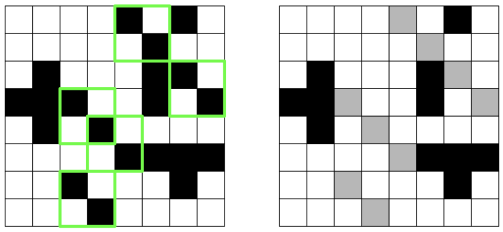
Supongamos una pared de 8×8 y con dos insectos: uno 3×3 (rojo) y otro 2×2 (verde).



Para el insecto identificado con el color rojo, el programa debería identificar las siguientes sub-imágenes (imagen izquierda) y retornar la imagen de la derecha:



El mismo razonamiento se aplica para el insecto asociado con el verde:



Ejecución

Tu programa se debe compilar con el comando `make` y debe generar un binario de nombre `fumigate` que se ejecuta con el siguiente comando:

```
./fumigate queries
```

Donde `queries` es la ruta a un archivo de texto que incluye las rutas de la imagen inicial y una serie de consultas.

Input

El input está estructurado como sigue:

- Una línea con la ruta de la imagen inicial. La estructura `imagen` contiene sus dimensiones.
- Una línea conteniendo el número `Q` de consultas.
- `Q` líneas, cada una con la ruta de una imagen con $2 \leq n < N$ y la ruta de escritura del output.

Por ejemplo, el input `test_0.txt` para el ejemplo y las 2 consultas que hicimos:

```
test_0.png
2
test_0_1.png out_0_1.png
test_0_2.png out_0_2.png
```

La primera línea indica que se analiza la imagen `test_0.png`. Luego se indica que se realizarán 2 consultas. La primera con el *insecto* `test_0_1.png`, cuyo resultado se guardará en `out_0_1.png`

Cada imagen se lee como un array de enteros que puede ser blanco o negro. Los valores ya están definidos en el código base como `BLACK` y `WHITE`.

Output

Tu output deberá consistir de `Q` imágenes llamadas, cada una con el resultado descrito anteriormente. Para la imagen de output también está definido el color gris como `GRAY`.

Evaluación

La nota de tu tarea se separa en 2 partes: nota de código y una nota especial asociada a un documento de diseño. A continuación se detalla cada uno de estas partes.

Documento de diseño: 50 % de la nota

Deberás escribir un documento de diseño **simple**[1] donde expliques cómo resolver el problema usando *hashing*. Se recomienda **fuertemente** escribir este informe antes de escribir el código en **C**. En particular, se espera que:

- Expliques cómo usar un diccionario para resolver este problema de manera eficiente.
- Diseñes una función de hash uniforme para una imagen de dimensiones $n \times n$, que pueda calcularse de manera incremental[2].
- Demuestres que tu función de hash es uniforme.
- Justifiques la elección de los parámetros de la tabla, tales como factor de carga, direccionamiento, función de *resize*, función de *probing*, entre otros.

Código: 50 % de la nota

Tu código será evaluado con tests de dificultad creciente. Estos tests están separados en 3 categorías:

- Easy: Para debugear manualmente. No serán evaluados.
- Normal: Usados para testear la correctitud de tu implementación. Serán evaluados.
- Hard: Usados para testear la eficiencia de tu implementación. Serán evaluados.

Para cada test de evaluación, tu programa deberá entregar el output correcto en menos de 10 segundos. De lo contrario, recibirás 0 puntos en ese test.

Restricción: Para esta tarea también se evaluará el uso no excesivo de memoria. Por esto, tu tarea no debe usar más de 1 GB de RAM, de lo contrario no se asignará puntaje.

Entrega

Código: GIT - Repositorio asignado. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.

Informe: SIDING - En el cuestionario correspondiente, en formato PDF. Sigue las instrucciones del cuestionario. Se entrega a más tardar el día de entrega a las 23:59 hora de Chile continental.

[1] Se mantiene el largo máximo de 5 páginas.
[2] Puedes usar el hash obtenido para una sub-imagen de $n \times n$ para calcular el hash de la sub-imagen de $(n+1) \times (n+1)$

Bonus

Los siguientes bonus solo aplican si tu nota correspondiente es mayor o igual a 4.

Manejo de memoria perfecto: +5 % a la nota de código

Recibirás este bonus si `valgrind` reporta que tu programa no tiene ni leaks ni errores de memoria en los tests que logres resolver.

Diseño impecable: +5 % a la nota de diseño

Recibirás este bonus, a criterio del corrector, si tu documento de diseño está especialmente bien estructurado y contiene un análisis sólido. Para este bonus el diseño y análisis de tu función de hash es de especial importancia.

Hash goes as fast as VTR (+X décimas a la nota final de la tarea)

Se otorgará un bonus de hasta 10 décimas (a la nota final de la tarea) a las 20 soluciones que corran más rápido. 10 décimas a los dos primeros lugares, 9 al tercer y cuarto puesto, etc.

Para participar en esta carrera, tu programa deberá resolver el test `Lunatic`, de elevada dificultad, correctamente y en menos de 10 segundos. Este test no será considerado dentro de la evaluación, solo permite la entrada a la gran carrera.