UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

ESTRUCTURAS COMPACTAS DINÁMICAS MÁS EFICIENTES PARA BASES DE DATOS DE GRAFOS CON ATRIBUTOS

RESULTADOS DE LOS EXPERIMENTOS SOBRE TRES $K^2 - TREES$

DAVID DE LA PUENTE VÉLIZ

PROFESOR GUÍA:
GONZALO NAVARRO
PROFESOR GUÍA 2:
DIEGO ARROYUELO

SANTIAGO DE CHILE
2021

Preliminar

En este documento se encuentran los resultados de varios experimentos hechos en tres distintas estructuras que representan $k^2 - trees$, las cuales son: 1) basado en un arreglo de bits dinámicos [2], al que llamaremos $dyn - arr$, 2) basado en bloques de arreglos de bits dinámicos [1], al que llamaremos "blocks", 3) basado en la unión de k^2-trees estáticos [3], al que llamaremos "Ustatic". Para estas estructura medimos su rendimiento para las operaciones de inserción y borrado de aristas. En lo que sigue del informe se presentan seis experimentos básicos de los cuales se desprenden algunas conclusiones y otros experimentos.

datasets

Inicialmente se empieza con un archivo que contiene 194 millones de aristas para 750000 de vértices, descritas en código de Morton, es decir cada arista se representa con 23 caracteres del 0 al 3. Este archivo es bien recibido por la estructura blocks, sin embargo las otras dos estructuras reciben las aristas como coordenadas (x,y). Por lo que para testear estas estructuras, tuvimos que transformar el archivo en Morton, a uno en coordenadas

Para lo anterior lo que hacemos es: por cada string en Morton m, buscar el par ("x","y") que lo representa. Para encontrar "x" hacemos una búsqueda binaria sobre I =[0,7500000] tal que los caracteres 0 y 1 en Morton dividen I dejándonos con su mitad izquierda, y los caracteres 2 y 3 dividen I dejándonos con su mitad derecha. Para encontrar "y" hacemos una búsqueda binaria sobre I =[0,7500000] tal que los caracteres pares dividen I a su mitad izquierda y los impares dividen I a su mitad derecha.

exp1: Insertar 194M

Este experimento consiste en insertar 194 millones de aristas en las estructuras.

tiempo por arista

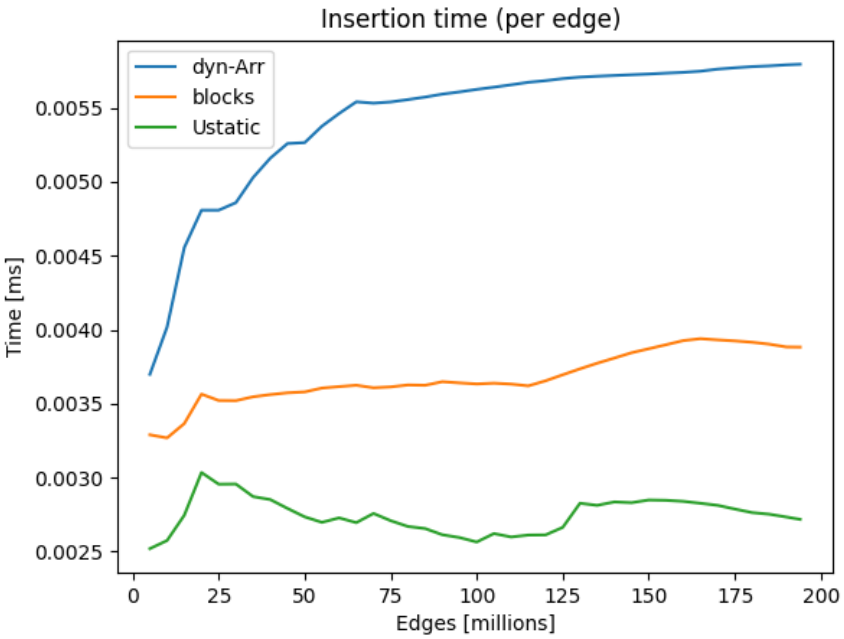

Figura 1: Tiempo de inserción por arista

Podemos ver en la figura 1 que todas las estructuras tienen comportamientos logarítmicos. Ustatic tiene un comportamiento mas irregular. Para esta ultima estructura podemos ver unos picos en las 25M, 75M, 100M y 125M aristas, esto puede deberse a que la estructura tiene que reconstruir sus k^2-trees estáticos en ese momento. Notamos que Ustatic es el mejor para este experimento.

Memoria total

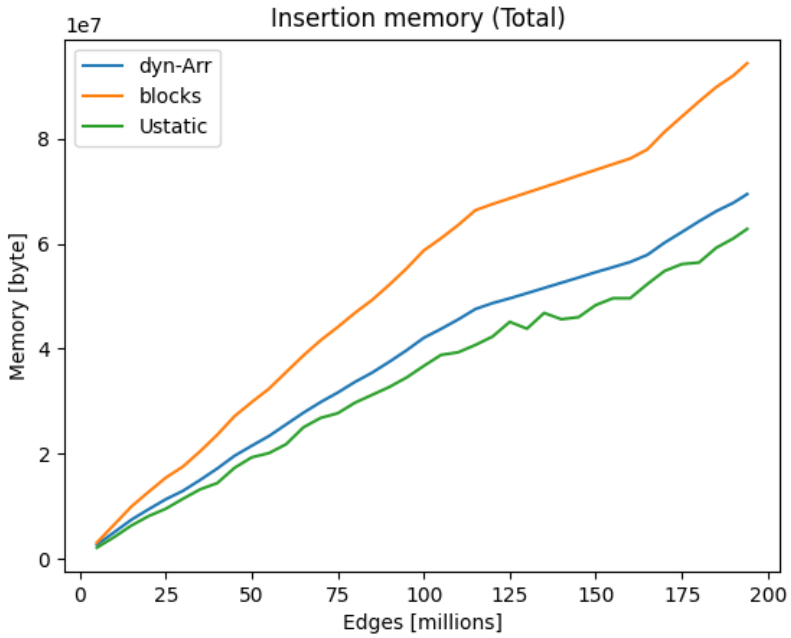

Figura 2: Memoria total ocupada por la estructura

En la figura 2 podemos ver como crece la memoria que ocupa la estructura a medida que se le van insertando aristas. dyn-arr y Ustatic tienen un comportamiento muy parecido, mientras que blocks se separa un poco mas de ellos. Notamos que en este experimento gana Ustatic.

Memoria por arista

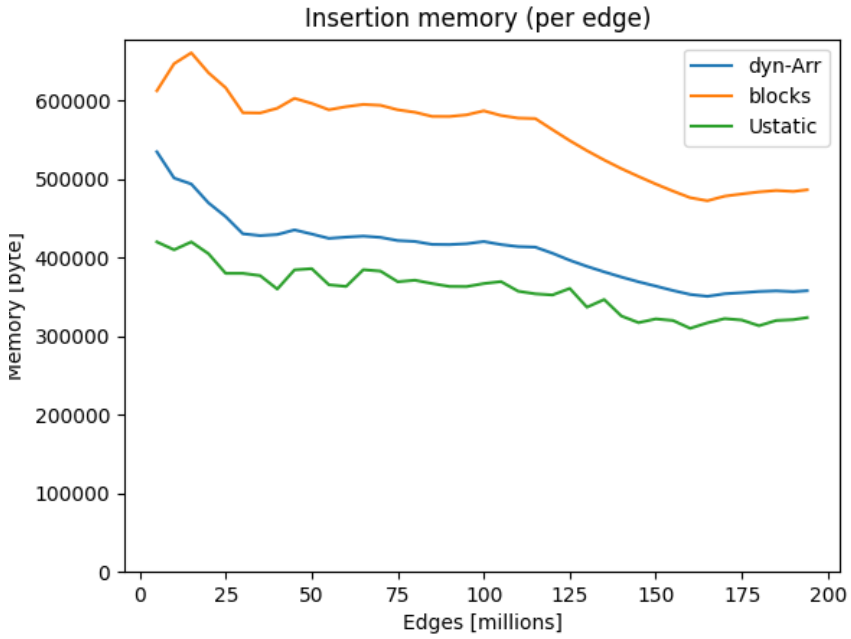

Figura 3: Memoria por arista ocupada en la estructura

El gráfico de la figura 3 muestra los mismos datos del experimento anterior, pero esta vez dividimos la memoria ocupada, por el numero de aristas que posee cada estructura (de esta forma, para el punto (x, y) en el gráfico x corresponde a un k^2-tree de tamaño x, e y corresponde a cuanta memoria ocupa cada una de sus aristas). Podemos observar que el gráfico tiene un comportamiento constante, que se corresponde con el comportamiento lineal del gráfico de la figura 2.

exp2: insertar y buscar 194M

Este experimento consiste en insertar 194 millones de aristas en las estructuras, pero apenas insertamos una arista, la buscamos inmediatamente. De esta forma podemos saber como crece el tiempo de búsqueda.

Tiempo de búsqueda por arista

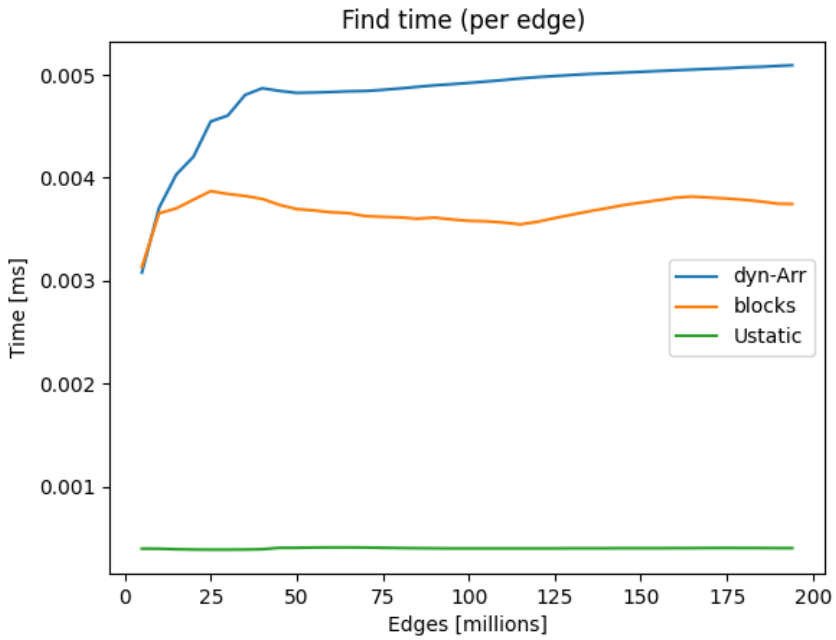

Figura 4: Tiempo de búsqueda por arista

Podemos ver en la figura 4, que en todas las estructuras, el tiempo de búsqueda crece de manera logarítmica a medida que crece la estructura. Notamos que Ustatic gana en este experimento.

exp3: insertar 194M y borrar 194M

El tercer experimento consiste en insertar 194 millones aristas y luego borrar los 194 millones de aristas. Anotamos el tiempo de borrado y memoria ocupada cada 5 millones de aristas borradas.

Tiempo de borrado por arista

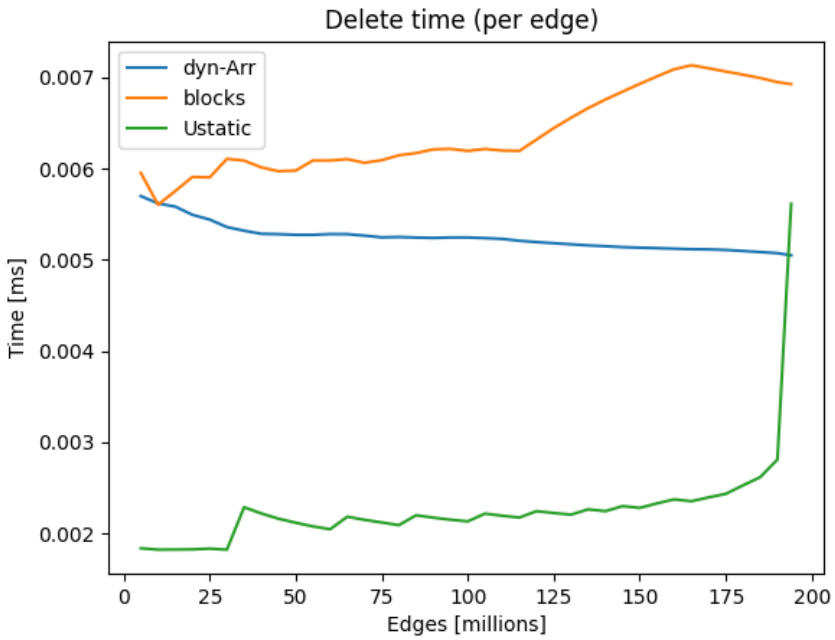

Figura 5: Tiempo de borrado por arista

En la figura 4 podemos ver los tiempos de borrado por arista. Se puede ver que blocks tiene un comportamiento logarítmico (recordemos que se demora el doble de la inserción, que tambien es logarítmica), por otro lado podemos ver que dyn-array tiene un comportamiento constante al borrar aristas, esto significa que no importa cuantas aristas borremos de la estructura, siempre nos vamos a demorar lo mismo por cada una de ellas (veremos en la siguiente figura que esto trae repercusiones en la memoria de la estructura).

Nuevamente Ustatic vuelve a ganar en este experimento, aunque podemos notar un comportamiento extraño al borrar toda la estructura. Podemos ver que cuando se borran las ultimas aristas, el tiempo de borrado explota. Al principio pensé que podía ser un error del código o una coincidencia. Repetí el experimento varias veces y siempre obtuve el mismo resultado. Releí el documento que habla de la implementación de la estructura [3]. Los autores mencionan que para el borrar una arista, solo apagan el bit correspondiente en el k^2-tree estático. Esto suele ser mucho mas rápido, pero trae consecuencias sobre la memoria ocupada por la estructura y consecuencias sobre el tiempo de borrado de las últimas aristas.

Tiempo de borrado en Ustatic

Para demostrar que Ustatic libera la memoria cuando se borran las últimas aristas, replicamos el mismo experimento anterior, pero para estructuras con menos aristas. Específicamente replicamos el experimento en un k^2-tree con 10 millones de aristas, en otro con 100 millones y en otro con 190 millones.

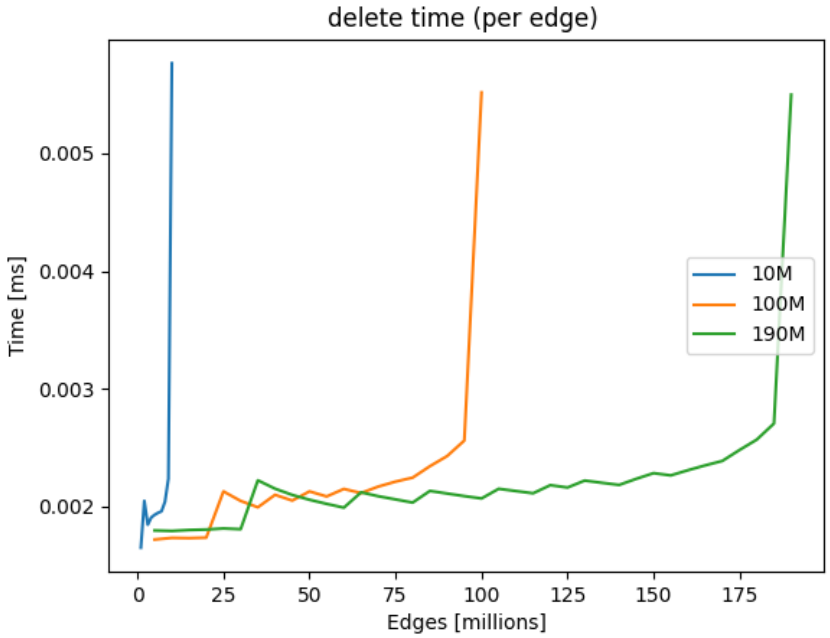

Figura 6: Tiempo de borrado por arista para Ustatics de distintos tamaños

En la figura 6 notamos que efectivamente el tiempo de borrado en Ustatic explota cuando la estructura borra sus últimas aristas.

Memoria total

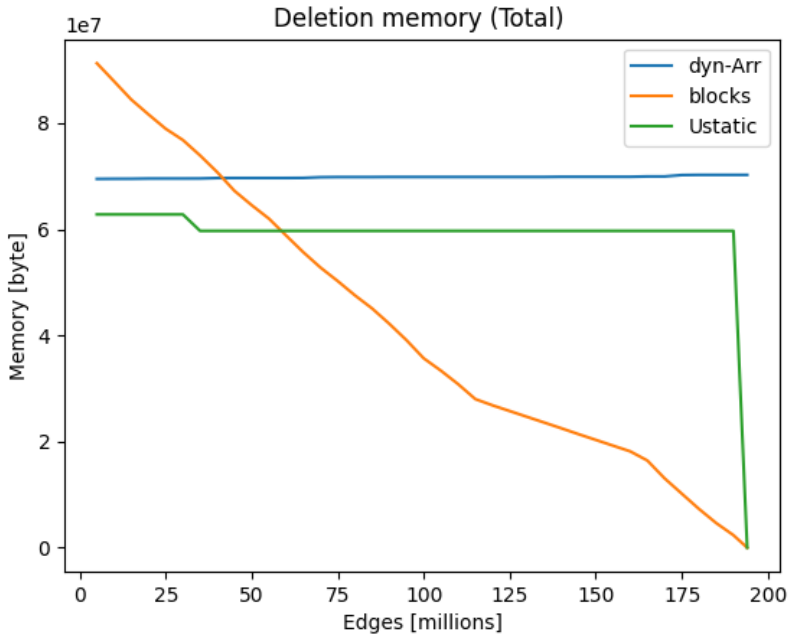

Figura 7: Memoria total al borrar

En la segura 7, podemos observar como decrece la memoria de la estructura a medida que se le van borrando aristas. dyn-array mantiene el tamaño de la estructura a pesar de que se borren todas las aristas (esto se debe a que la estructura no fue pensada para borrarle aristas). Ustatic tambien mantiene la memoria constante al borrar ya que solo va apagando bits y no borra las aristas hasta el ultimo momento.

En este experimento, el ganador es blocks ya que mantiene una buena gestión de la memoria que ocupa la estructura.

Memoria por arista

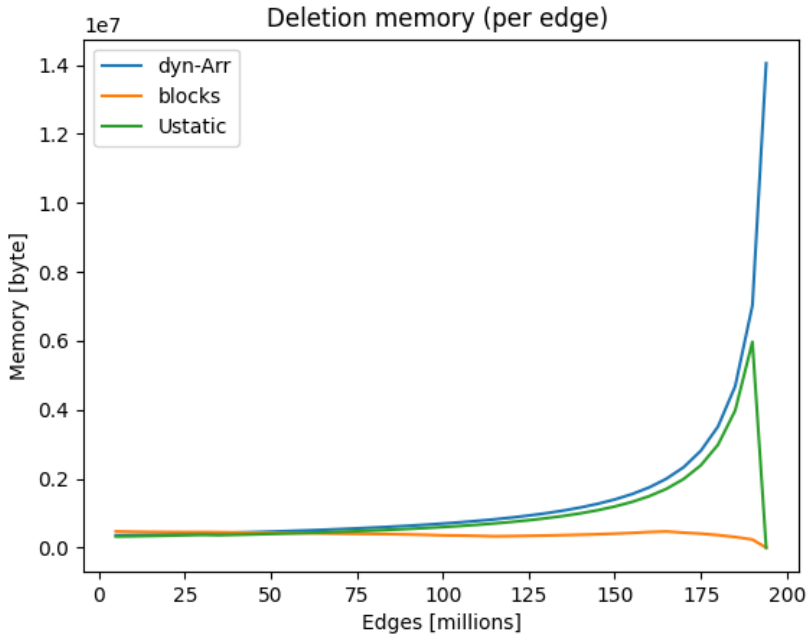

Figura 8: Memoria por arista al borrar

en la figura 8, vemos que en dyn-arr y Ustatic, la memoria por arista crece de manera exponencial, esto se debe a que el numero de aristas disminuye pero la memoria se mantiene constante. Por otro lado la memoria por arista en blocks disminuye recorriendo un logaritmo al revés.

exp4: insertar 194M borrar los últimos X

El siguiente experimento consiste en insertar 194 millones de aristas, borrar la estructura completa pero esta vez tomar el tiempo cuando se borran las últimas x aristas. De esta forma sabemos cuanto se demora en borrar las últimas 5M aristas, las últimas 10M, las últimas 15M,... las últimas 190M y las últimas 194M (todas). En este experimento no mostramos la memoria ya que no varía con respecto al experimento anterior.

Tiempo total

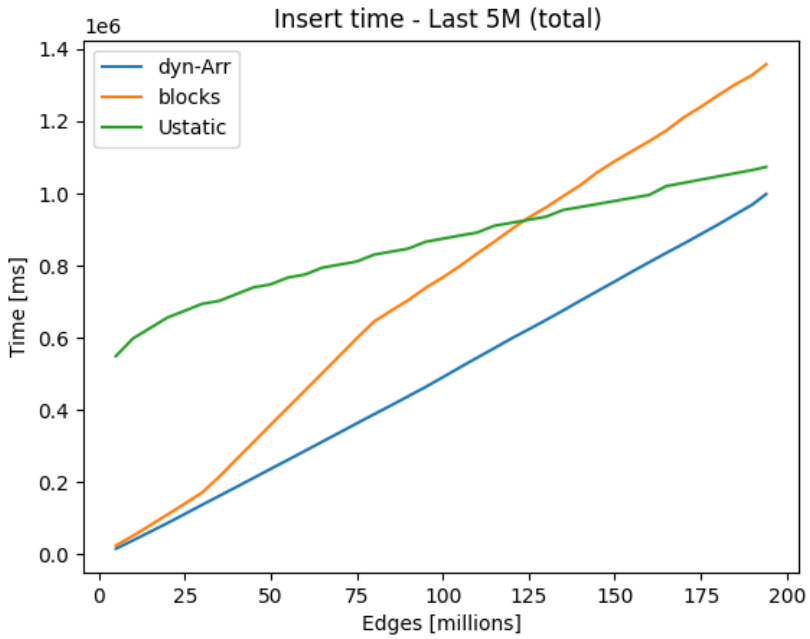

Figura 9: Tiempo total al borrar las últimas x aristas

en la figura 9, vemos un gráfico donde el eje "x" representa las ultimas "x" aristas borradas de un k^2-tree que inicialmente tenia 194 millones de aristas. E "y" representa cuanto se demora en borrar esas aristas. Por ejemplo si "x" = 5 millones, entonces "y" = a cuanto se demoró en borrar las ultimas 5 millones de aristas.

Notamos que el tiempo de borrado es creciente por que a medida que nos movemos en "x", tenemos que borrar mas aristas. Otra cosa interesante que notamos es que ahora Ustatic no tiene tan buen desempeño, ya que al borrar las ultimas x aristas, la estructura se vacía y como vimos en el experimento anterior, cuando la estructura se vacía el tiempo de borrado explota (es como si explotara en cada punto).

Tiempo por arista

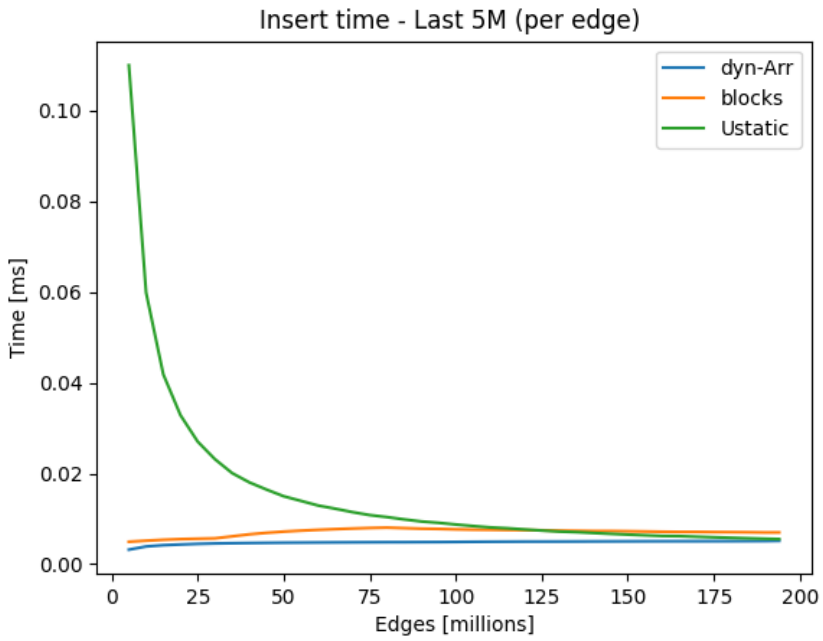

Figura 10: Tiempo total al borrar las últimas x aristas

En la figura 10, se muestran los mismos resultados que en el gráfico anterior, pero ahora el tiempo es por arista.

exp5: Insertar 194M, borrar 75 %, re insertar 75 %

En este experimento insertamos 194 millones de aristas, luego borramos el 75 % de cada estructura y volvemos a insertar el 75 % borrado. Cuando re insertamos vamos anotando el tiempo y memoria ocupada cada 5 millones de aristas.

Tiempo por arista (re inserción)

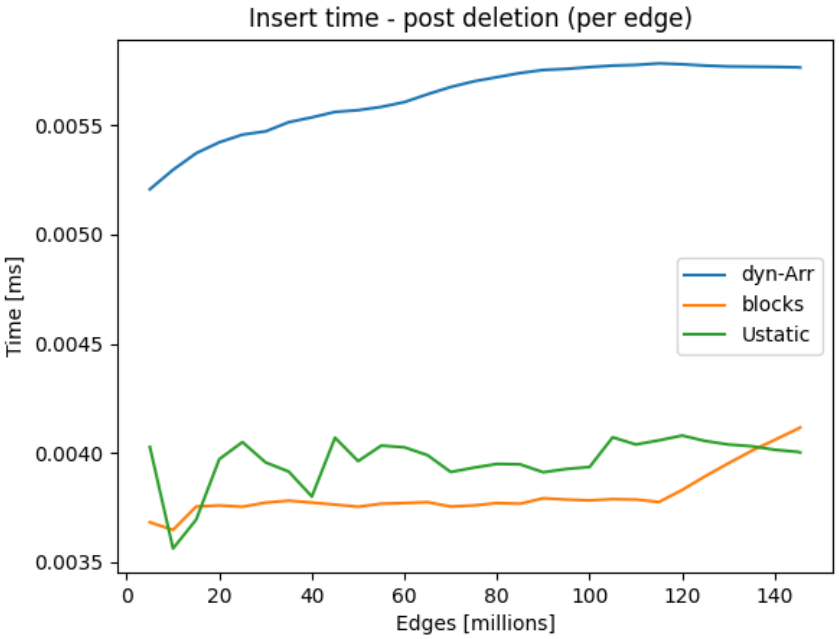

Figura 11: Tiempo de re inserción por arista

En la figura 11 podemos ver que al re insertar luego de borrar, las estructuras se ven afectadas en su desempeño. En particular la mas afectada es Ustatic, que duplica su tiempo de inserción por arista. Notamos que blocks gana este experimento.

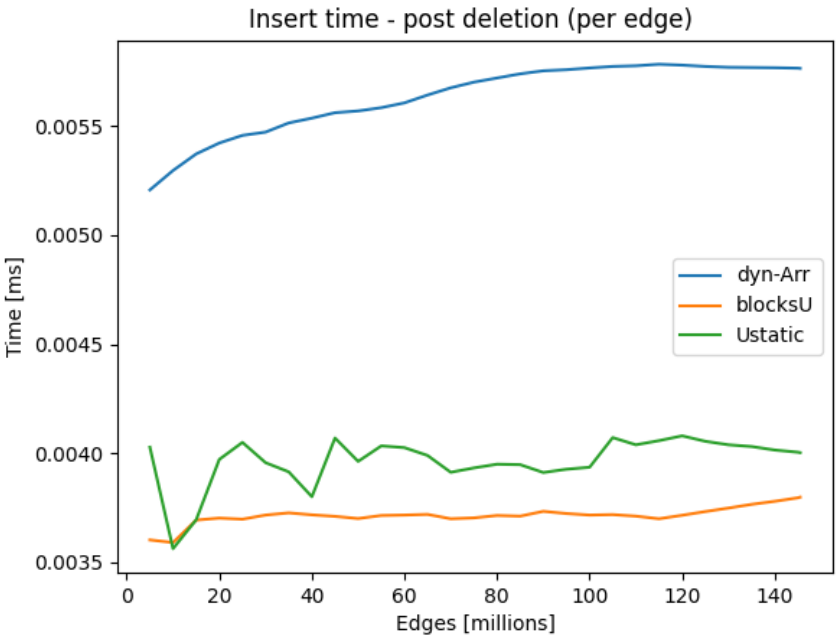

Figura 12: Tiempo de re inserción por arista (con blocksU)

En la figura 12 mostramos los resultados del mismo experimento anterior, pero esta vez usamos la variante de blocks que se encarga de ir uniendo los bloques que están muy vacíos, a esta variante la llamamos blocksU.

Memoria Total

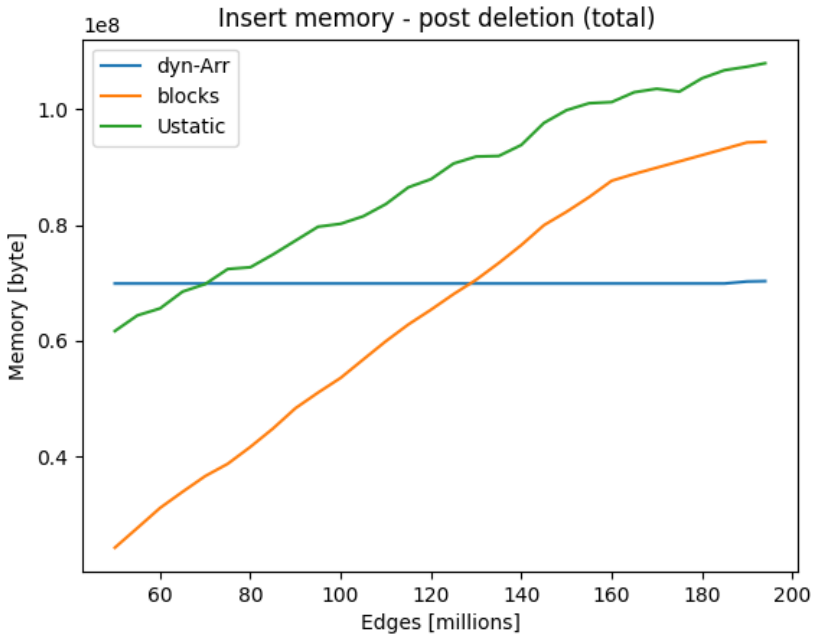

Figura 13: Memoria total en la re inserción

en la figura 13 podemos ver como aumenta la memoria total en cada estructura, a medida que se le re insertan aristas. dyn-array mantiene una memoria constante ya que no liberó espacio cuando se le borraron aristas. Ustatic tampoco libera su memoria cuando se borra el 75 % de sus aristas, es por esto que al re insertar la memoria ocupada empieza a crecer desde un punto mucho mayor que blocks, empeorando su desempeño. Blocks gana en este experimento al mantener una mejor gestión de su memoria.

Memoria por arista

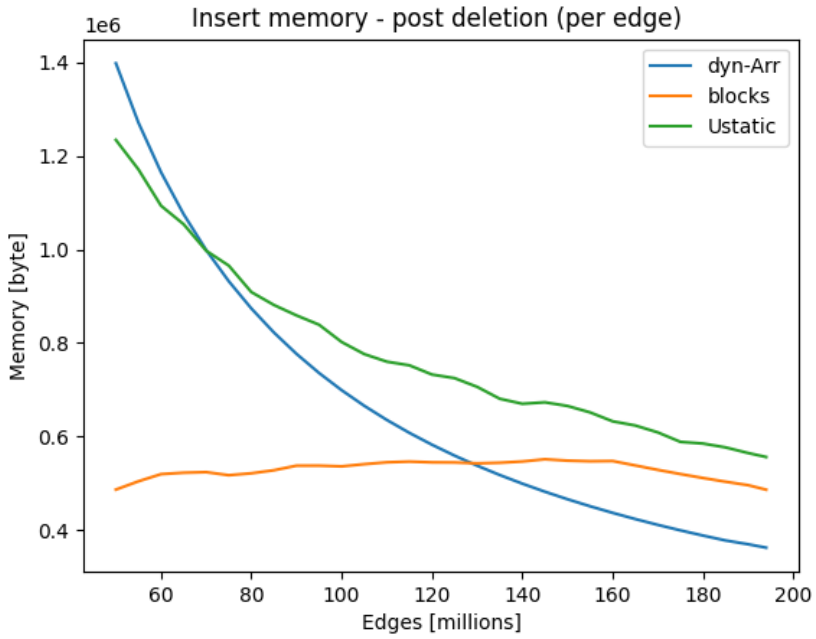

Figura 14: Memoria por arista en la re inserción

El grafico de la figura 14 muestra la memoria ocupada por arista al re insertar aristas (Pareciera que Ustatic y dyn-arr decrecieran, pero en verdad solo parten de mas arriba, luego crecen logarítmicamente).

exp6: insertar 194M, borrar 75 %, re insertar y buscar 75 %

El sexto experimento consiste en hacer lo mismo que en el experimento anterior, pero cada vez que re insertamos, buscamos la misma aristas. De esta forma podemos ir midiendo el tiempo de buscar al momento de re insertar.

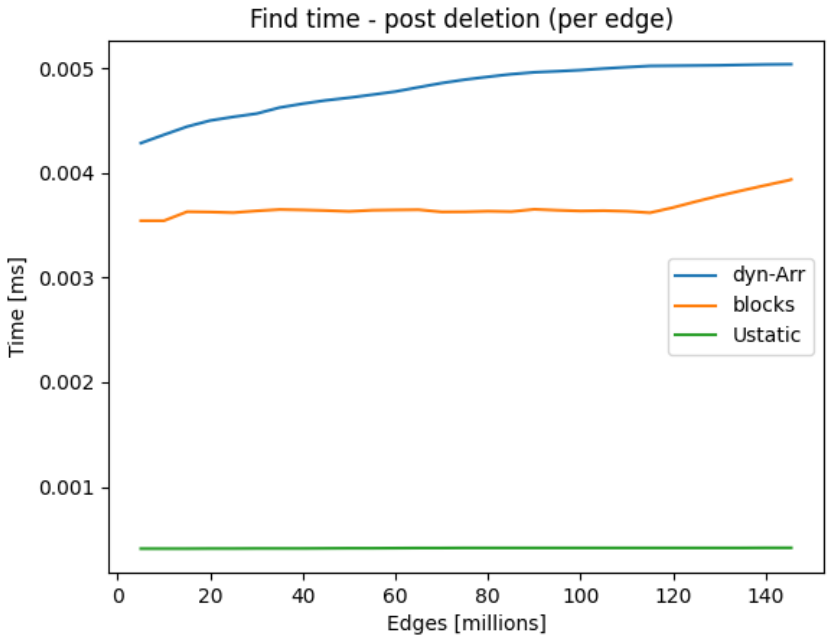

Figura 15: Tiempo de búsqueda en la re inserción

Aunque el tiempo de Ustatic creció un poco, sigue siendo el mejor para buscar.

Conclusiones

1) Para decidir cual de los k^2-trees es el mejor, hay que elegir entre blocks y Ustatic, Pues dyn-arr se queda atrás en varios experimentos.

2) Ustatic tiene muy buen desempeño para buscar aristas, y es muy rápido cuando solo hay inserciones.

3) Ustatic tiene mala gestión de memoria al borrar y por lo mismo al re insertar obtiene peores tiempos que blocks.

4) Blocks es el mas equilibrado de los tres k^2-trees, quizás no es el mejor en todos los experimentos, pero administra bien su memoria.

5) Falta un experimento que ponga a prueba las estructuras con varias inserciones y borrados de aristas mezclados.

Referencias

[1] Arroyuelo, Diego, Guillermo de Bernardo, Travis Gagie y Gonzalo Navarro: *Faster Dynamic Compressed d-ary Relations*. Lecture Notes in Computer Science, página 419–433, 2019, ISSN 1611-3349. http://dx.doi.org/10.1007/978-3-030-32686-9_30.

[2] Brisaboa, Nieves R., Ana Cerdeira-Pena, Guillermo de Bernardo y Gonzalo Navarro: *Compressed Representation of Dynamic Binary Relations with Applications*, 2017.

[3] Coimbra, Miguel E., Alexandre P. Francisco, Luís M. S. Russo, Guillermo de Bernardo, Susana Ladra y Gonzalo Navarro: *On dynamic succinct graph representations*, 2019.