bibliographyAdaptive

Jeremy Barbay

October 25, 2012

1 Concept Questions

1.1 Cual es el interes del algoritmo `Greedysort`, en particular en comparacion con `Splitsort`? :EduardoVillouta:

En la clase de Splitsort vimos que había que cuando se recorría un arreglo y en una posición i se encontraba un valor menor que en i-1 había que sacar ambos elementos del arreglo y ponerlos en 2 nuevos arreglos Xg y Gs, para así disminuir el número de comparaciones promedio. Sin embargo, Elmasry y Hammad presentan Greedysort, que solo va quitando un elemento si es que es menor que el anterior. Por qué en este caso es relevante mencionar este algoritmo?

1. El rendimiento teorico de `Greedysort` es mejor que `Splitsort`.
2. `Greedysort` teoricamente genera menos cache misses que `Splisort`.
3. `Greedysort` es más rapido que `Splitsort` en practica (cache miss o no)
4. `Greedysort` es más rapido que `Splitsort` en practica (cuando hay cache miss)
5. Solo se usa para mostrar un nuevo algoritmo. Siempre es mejor usar `Splitsort`

1.2 es posible hacer Bucket Sort adaptativo? :Francisco-Havon:

1. no, pues hay que saber previamente todos los inputs para cada bucket
2. ya es adaptativo
3. no, pues no es un algoritmo que use comparaciones binarias
4. no tiene sentido la pregunta
5. si es posible

6. no, aunque se puede combinar con uno que si lo sea
7. otra cosa

1.3 Qué seria una variante adaptativa de `Quicksort`? :CarlosDettoni:

Hint: puede usarse un algoritmo de ordenamiento adaptativo auxiliar.

1. Podría comenzar usando un algoritmo de ordenamiento adaptativo como `Insertion-Sort` para que cuando el arreglo esté casi ordenado, aplico `Quicksort`.
2. Podría comenzar usando `Quicksort` hasta que se haya hecho un número arbitrario de particiones, logrando un arreglo semi-ordenado y finalmente al arreglo semi-ordenado resultante se le aplica un algoritmo adaptativo de ordenamiento como `Insertion-Sort`.
3. Se puede encontrar un pivote recomendable que permite romper la barrera del $O(n \log n)$ cuando nos acercamos al mejor caso.
4. No es posible hacerlo adaptativo, pues su naturaleza es ser $O(n \log n)$ en promedio y en el mejor caso, y eso absorberá a cualquier sub-rutina auxiliar.

1.4 Which adaptive algorithms could be made adaptive in external memory and how?

1. Adaptive MergeSort, with a merging degree of size $m = M/B$
2. Local Insertion Sort with a B-Tree
3. HeapSort, with a B-Heap
4. (add your own here)

1.5 What would be required to make `Adaptive Merge sort` adaptive in External Memory?

1. A merging degree of size $m = M/B$, where M is the amount of internal memory
2. A merging degree of size B
3. a difficulty measure adequate for external memory: runs are not necessarily meaning full if shorter than the size of a page
4. Nothing, it is already natually adaptive in external memory

1.6 What would be required to make `Adaptive Heap sort` adaptive in External Memory?

1. An Adaptive B-Heap
2. Nothing, it is already natually adaptive in external memory
3. It cannot be made adaptive in external memory: the heap has no spacial locality

1.7 What would be required to make `Local Insertion sort` adaptive in External Memory?

1. An B-tree with the finger search property
2. It's impossible, the Local Insertion sort algorithm has no spacial locality
3. Nothing, it is already natually adaptive in external memory

1.8 Los tiempos de ejecución obtenidos en las pruebas con los algoritmos de ordenamiento adaptativos analizados eran cercanos a los de quicksort para permutaciones casi ordenadas y para aquellas con un alto grado de desorden su rendimiento era bastante bajo en comparación, entonces, tienen algún sentido estos algoritmos adaptativos? :EduardoEscobar:

1. No. Las mejoras en los tiempos para permutaciones casi ordenadas son insignificantes.
2. No. Porque para que las mejoras en los tiempos tengan algún sentido, es necesario ordenar permutaciones muy grandes (además de estar casi ordenadas), pero todos los algoritmos estudiados son in-place, por lo tanto, no es una alternativa viable.
3. Si. Para un arreglo pequeño con algún nivel de orden su rendimiento es mejor que el de quicksort y es posible permitirse los costos extras de memoria.
4. Si. Si bien las mejoras para un arreglo pequeño las mejoras no son muy significativas, si se aplica a varios arreglos pequeños se pueden obtener mejoras considerables. Se puede construir un algoritmo híbrido entre quicksort y TD Splaysort que tome lo mejor de cada uno. Si para sub arreglos de un tamaño menor o igual que un m dado, quicksort en lugar de aplicarse recursivamente podría utilizar TD Splaysort para ordenarlos.

1.9 Como se puede hacer `splaysort` adaptativo en memoria externa? :BenjaminSoto:

1. Usando B tree de manera simple.
2. Usando un B tree, pero adaptando de tal forma que dejando sierto espacio por hoja en memoria principal, se puedan fusionar las hojas que quedan cercanas a la nueva raiz al momento de hacer inserción.
3. de una forma distinta.
4. no es posible adaptar splaysort a memoria externa.

1.10 Como generar datos aleatorizados por `Osc` y n fijados? :DiegoGajardo:

En el paper se ha visto un método de generación de inputs experimentales con el cual se controla el número esperado de inversiones en la permutación resultante; el objetivo de generar inputs de esta forma es poder probar bajo condiciones "reales" el rendimiento de los distintos algoritmos Inv-óptimos estudiados en el experimento. Si se quisiera realizar un estudio similar al de este paper, pero en función de Osc, sobre varios algoritmos Osc-óptimos, cuál(es) de las siguientes ideas podrían servir en la generación de inputs experimentales?

1. Realizar la misma operación presentada en el paper, la cual también serviría para controlar el número esperado de Osc(X)
2. Subdividir el arreglo en m bloques de igual largo (mayor o igual a 2, excepto posiblemente 1 bloque) e intercambiar el primer y último índice de cada bloque entre sí
3. Subdividir el arreglo en m bloques de igual largo (mayor o igual a 2, excepto posiblemente 1 bloque), y para cada bloque escoger 2*k (para un entero positivo k fijo, k menor o igual que |Bloque|/2) elementos al azar e intercambiar sus posiciones entre sí
4. Subdividir el arreglo en m bloques de igual largo (mayor o igual a 2, excepto posiblemente 1 bloque), escoger dos elementos del bloque B (para todo B, excepto quizás 1) al azar e intercambiarlos entre sí, y luego, para cada uno de los demás elementos E del bloque B (para todo B), escoger otro bloque B' al azar e intercambiar B[i] con B'[i] (donde i es la posición de E dentro del bloque B)
5. Ninguna de las anteriores serviría
6. Otra idea, pero parecida a las anteriores
7. Otra idea, pero muy distinta y más difícil pues, dada la definición de Osc(X), es mucho más complejo generar inputs experimentales razonables para Osc que para Inv

1.11 Supongamos queremos adaptar el algoritmo del paper que usaba un bosque de árboles AVL para que tenga menos cache misses. Cómo lo podemos hacer? (puede elegir más de una alternativa): :Samir:

1. Reemplazar los árboles AVL por BTREES
2. Usar un BTREE para almacenar los rangos
3. Cuando se necesite escribir datos en cache, reservar un poco de espacio y escribir parte del contenido de los árboles que estan inmediatamente atras y adelante, que posiblemente se encuentren en el rango anterior o posterior respectivamente
4. Algo completamente distinto

1.12 CANC Is Windows XP commonly used for simulations? (Find five other experimental articles, for instance from JEA, and do statistics on which operating system they used) :CANC:

1.12.1 Dixit Francisco Claude: OK

It should be fine, I've seen it in other papers. A function is useful as long as it has enough precision to measure whatever you are trying to measure. The functions provided by Windows seem to be okay for most cases (would have to take a closer look to see a particular scenario). Here is a link that explains how to use them if you want to go down that path ;-). http://www.drdobbs.com/windows/win32-performance-measurement-options/184416651

1.13 CANC Is it common to "omit" data in the results? :CANC:

("each point is the average of 100 sample runs after omitting the most biased ten values out of 110 values."

1.13.1 Dixit Francisco Claude: OK

It's a common practice. It would be interesting to know where those points landed though ;-).

1.13.2 Dixit Gonzalo Navarro: This is not strange

it is common in statistics to get rid of outliers. It is usually done when something can, in rare cases, take very extreme values that could ruin a mean. Or one uses the median in that case. Using it for quicksort, for example, may be justified.

1.14 Al considerar el caso de los distintos algoritmos vistos y sus variantes y recordar otras medidas vistas a lo largo del curso(no sólo Inv) qué se puede mencionar respecto al concepto de "adaptive optimality" de éste respecto a dichas métricas? :FreddyCea:

1. Existen implementaciones que han conseguido algoritmos in-place, pero respecto a una métrica a la vez como `Rem`.

2. Existen implementaciones que han conseguido algoritmos in-place, pero respecto a varias métricas a la vez, como por ejemplo `Rem` e `Inv`.

3. Se ha podido solucionar el problema para algunos métodos de ordenamiento, por ende no existen variantes para todos los algoritmos en las cuales se respete la idea de "adaptive" junto con el in-place. Un ejemplo sería `Heapsort` y sus variantes.

4. No se ha podido solucionar el problema para ningún método de ordenamiento, por lo que no existen variantes en las cuales se respete la idea de "adaptive" junto con el in-place

5. La pregunta no tiene sentido.

1.15 Qué estructuras de datos podrían ser útiles introduciendo algunos de los contenidos revisados? :EduardoVillouta:

Vimos que se podían agrupar el nodo padre con sus 2 hijos, para poder acceder rapidamente entre ellos, dado que desde el padre se va necesariamente a uno de los 2. Siguiendo esta lógica, se podría realizar lo mismo para los Tries, aprovechando de que almacenan más información, agrupando el nodo padre, con todos sus nodos hijos.

1.16 Supongamos que quisieramos modificar quicksort aleatorizado para que las comparaciones se puedan hacer mas rapido, en este caso imaginen que se puede implementar los `indices comprimidos`. En que momento el uso de estos indices comprimidos no resultaria util para el algoritmo? :HectorMoraga:

1. Siempre resultarian utiles, ya que podriamos dividir todos los numeros en bloques (de acuerdo a estos indices) y asi ordenar cada bloque y luego unirlos, dejando ordenado el arreglo.

2. Son utiles hasta que el numero de bits de los indices comprimidos no sea mayor que $\lceil \frac{n}{2} \rceil$.

3. Solo resultan utiles en la medida que mejoren el tiempo $O(n \log n)$ de quicksort, y eso se da con pocos indices.

4. No resultarian utiles, ya que es complicado de calcular estos indices en primer lugar, y segundo, no es posible asociar los numeros a ordenar a estos indices.