Problema 1.21: suma dels 2^i-èssims

A.22 Grup 2

Enunciat

Tenim un vector $A[1, \ldots, n]$ no ordenat amb claus no necessàriament numèriques, però que pertanyen a un conjunt totalment ordenat de claus. Sigui x_i el 2^i-èssim element més petit en A. Doneu un algorisme per calcular la suma dels valors x_i, per $1 \leq 2^i \leq n$, en $\Theta(n)$ passos.

Solució

Codi en C++

Per a simplificar, el codi treballarà amb un vector (que implícitament usa claus numèriques), però la mateixa idea serviria si les claus fossin d'un altre tipus.

```
#include<vector>
#include<cmath>
using namespace std;

int partition(vector<double>& v, int l, int r){
    int p, g;
    p = g = l + 1;

    // invariant: v[l] pivot, v[l + 1...p - 1] elements < pivot, v[p...g - 1] elements >= pivot
    while (g <= r) {
        if (v[g] < v[l]) swap(v[p++], v[g]);
        ++g;
    }

    swap(v[l], v[p - 1]);
    return p - 1;
}

// pre: 1 <= quantil <= r - l + 1
// post: retorna el valor del quantil-essim valor mes petit del subvector v[l...r] i
// r passa a apuntar a l'element anterior (r') en el vector desordenat
// de manera que v[l...r'] son tots elements de valor mes petit que el valor retornat
double select_n_divide(vector<double>& v, int l, int& r, int quantil){
    int index_pivot = partition(v, l, r);

    if (index_pivot - l < quantil - 1) {
        return select_n_divide(v, index_pivot + 1, r, quantil - index_pivot + l - 1);
    }
    else {
        r = index_pivot - 1;
        if (index_pivot - l > quantil - 1) return select_n_divide(v, l, r, quantil);
        else return v[index_pivot];
    }
}

double suma2iessims(vector<double>& v) {
    double res = 0;

    int l = 0, r = v.size() - 1;
    int quantil = pow(2,int(log2(v.size())));

    while (quantil >= 1) {
        res += select_n_divide(v, l, r, quantil);
        quantil /= 2;
    }

    return res;
}
```

Idea general

Per poder sumar els elements x_i, caldrà trobar-los i visitar-los a tots. Per fer-ho, farem servir l'algoritme de selecció basat en el Quicksort, el Quickselect.

El Quickselect té cost $\Theta(n)$ en el cas mig. Sigui X el conjunt dels elements que hem de sumar, és a dir, $X = \{x_0, x_1, \ldots, x_k\}$ on $k = \lfloor \log_2 n \rfloor$. Com que $|X| = \lfloor \log_2 n \rfloor + 1$, si apliquéssim el Quickselect sobre el vector original per trobar cada un dels elements, el cost seria $\Theta(n \log n)$.

Ara bé, com que busquem els elements 2^i-èssims, serà possible començar buscant el 2^k-èssim amb un Quickselect, i quedar-nos amb una meitat del vector que encara conté la resta dels x_i. A continuació, de la mateixa manera, podrem cercar el 2^{k-1}-èssim, i tornar a quedar-nos amb la meitat del vector un altre cop. Així, recursivament fins a trobar x_0, i dividint sempre per 2 la mida del vector útil, aconseguirem el cost lineal buscat.

Correcció i terminació

No demostraré la correcció de la funció *partition*(), perquè s'ha vist a classe de teoria, i és molt fàcil de veure amb l'invariant que hi ha comentat.

La funció *select_n_divide*() acaba. Sigui $F(v, l, r, quantil) = r - l$ la funció de cota de la operació, hem de veure que $r - l$ decrementa com a mínim en 1 a cada crida recursiva, i que quan $F(v, l, r, quantil) = 0$, la funció retorna.

1. L'element escollit com a pivot en una determinada crida mai hi serà al subvector que passem a la crida recursiva, i per tant és evident que $r - l$ com a mínim decreix en 1 en cada crida
2. Si $r - l = 0$ aleshores treballem amb un subvector d'un sol element, i per precondició, tenim que $quantil = 1$. L'únic element serà seleccionat com a pivot inevitablement, i *partition*() retornarà 1. Així doncs, la condició $index\_pivot - 1 == quantil - 1$ serà certa, i es retornarà sense fer cap crida recursiva

A més, *select_n_divide* també és correcta. Per inducció sobre la mida del subvector $r - l + 1$ i segons el cas:

- $index\_pivot - l == quantil - 1$: el pivot ha resultat ser l'element que es desitjava trobar. S'assigna $index\_pivot - 1$ a r, i es retorna el valor del pivot, fent que es compleixi la post
- $index\_pivot - l < quantil - 1$: no cal seguir buscant a l'esquerra del pivot. Com que inclòs el pivot tenim $index\_pivot - l + 1$ elements que sabem que estan per sota de l'element buscat, caldrà buscar el $quantil - index\_pivot + l - 1$ element més petit a la dreta del pivot. Per això els paràmetres de la crida recursiva són correctes.

  Aquesta crida compleix la precondició $1 \leq quantil' \leq r' - l' + 1$ on la s'usa prima per indicar variables de la nova crida recursiva perquè

  1. A partir de la condició de l'if: $index\_pivot - l < quantil - 1 \Rightarrow quantil - index\_pivot + l - 1 > 0 \Rightarrow quantil' \geq 1$
  2. A partir de la precondició: $quantil \leq r - l + 1 \Rightarrow quantil - index\_pivot - 1 \leq r - l + 1 - index\_pivot - 1 \Rightarrow quantil - index\_pivot - 1 \leq r' - l' + 1 + l \Rightarrow quantil - index\_pivot - 1 + l \leq r' - l' + 1 \Rightarrow quantil' \leq r' - l' + 1$

  Com que es compleixen les precondicions i a més la crida es fa amb un subvector estrictament més petit, per inducció assumirem que la crida satisfarà la seva post, i per tant també es satisfarà la de la crida actual al retornar.

- $index\_pivot - l > quantil - 1$: no cal seguir buscant a la dreta del pivot. Aquí doncs està clar que $quantil' = quantil$ i $l' = l$, i només canviarà el límit dret del vector de cara a la crida recursiva, que posarem just a l'esquerra del pivot.

  Aquesta crida també compleix la precondició $1 \leq quantil' \leq r' - l' + 1$, ja que

  1. A partir de la precondició: $1 \leq quantil \Rightarrow 1 \leq quantil'$
  2. A partir de la condició de l'if: $index\_pivot - l > quantil - 1 \Rightarrow quantil - 1 < index\_pivot - 1 - l + 1 \Rightarrow quantil' - 1 < r' - l' + 1 \Rightarrow quantil' \leq r' - l' + 1$

  De nou es compleix la precondició i la crida es farà amb un subvector estrictament més petit. Suposem doncs que per inducció la crida recursiva serà correcta, fent que se satisfacin les post de la crida actual.

La correcció i terminació de la funció principal *suma2iessims*() és trivial, ja que només acumula la suma dels diferents valors seleccionats.

Cost de l'algorisme

Es fan $|X| = \lfloor \log_2 n \rfloor + 1$ seleccions. La primera selecció és la única que es farà sobre el vector sencer original, i per tant aquesta tindrà cost $\Theta(n)$.

Després d'aquesta primera selecció, la variable r valdrà $2^k - 1$, i per tant treballem amb un vector una mica més petit, i tot i que no sabem exactament quant, sigui $n' = 2^k$ podem dir que $n' = qn$ per una certa q tal que $\frac{1}{2} <= q <= 1$.

Així doncs, en aquesta nomenclatura, podem dir que la segona selecció tindrà cost $\Theta(n')$. La diferència és que ara sabem que, després d'aquesta selecció, r valdrà $2^{k-1} - 1$ i per tant hem dividit el vector a la meitat del seu tamany. Per això la tercera selecció tindrà cost $\Theta(\frac{n'}{2})$. Si apliquem el mateix raonament repetidament veuriem que la quarta tindrà cost $\Theta(\frac{n'}{4})$, la cinquena $\Theta(\frac{n'}{8})$, ...

La suma dels costos de totes les seleccions menys la primera, aleshores, sabem que està acotada superiorment per $\Theta(2n')$, i al ser n' una fracció de n, això és $\Theta(2n)$. La primera selecció també hem dit que tenia cost lineal, en conclusió, fent que el cost de l'algoritme sigui lineal.

Cost: $\Theta(n)$