Додатак А: Фенвиково стабло

Аутор: Петар Величковић

Опис и анализа структуре:

Фенвиково стабло *(Fenwick Tree / Binary Indexed Tree / BIT)*, је елегантна структура за ефикасно чување и ажурирање кумулативних сума неког низа A величине n. Подржава следеће две операције:

1. $update(x, val)$ – повећавање x-тог елемента у низу за вредност val, тј. $A[x] := A[x] + val$;
2. $read(x)$ – рачунање кумулативне суме на x-тој позицији, тј. $\sum_{\square=l}^{x} A[i]$.

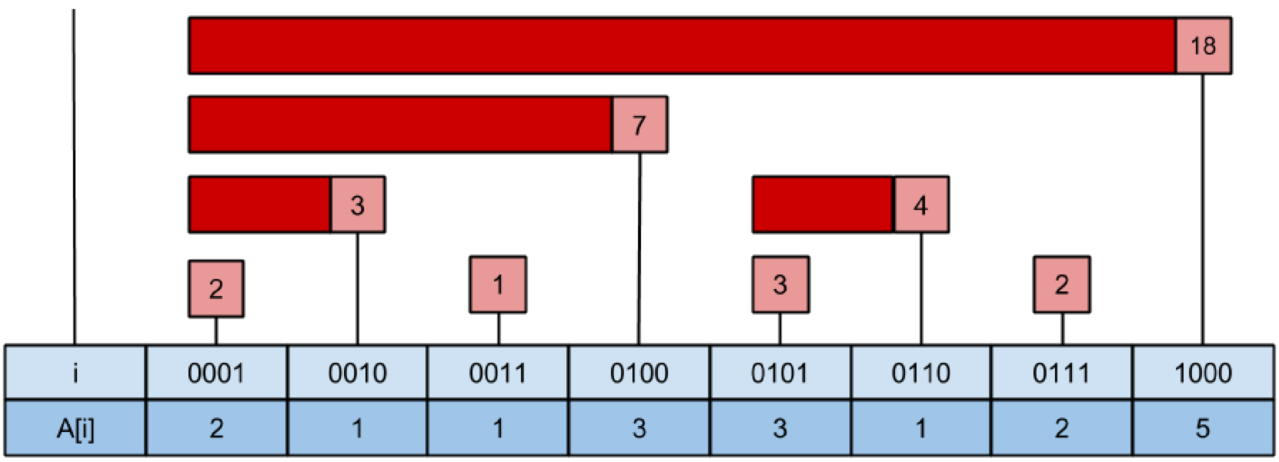
Ова структура користи само један низ, bit, да би процесирала горенаведене упите. Посматраћемо индексе овог низа као бинарне бројеве (нпр, $bit[12] = bit[1100_2]$). Сваки индекс у овом низу ће чувати неку подсуму низа A да би нам олакшао да брзо реконструишемо праву кумулативну суму; конкретно, уколико је са l означимо позицију последње јединице у бинарном запису индекса x, онда важи следеће:

$$bit[x] = \sum_{i=x-2^l+1}^{x} A[i]$$

Доња граница за суму, $x - 2^l + 1$, је еквивалентна томе да се последња јединица у бинарном запису броја x премести на најмање значајну позицију. На пример:

$bit[1100_2] = A[1001_2] + A[1010_2] + A[1011_2] + A[1100_2]$ (приметити како $1100_2 \rightarrow 1001_2$).

Ради илустрације, погледати доњу слику. Најдоњи ред представља низ A, док црвена поља представљају низ bit; "реп" иза сваког црвеног поља представља подниз над којим се рачуна сума.



Слика 1. Илустрација BIT низа за A[] = {2, 1, 1, 3, 3, 1, 2, 5}

Приметимо да се сада операција $read(x)$ може раставити на суму више елемената низа bit, тако да је први индекс једнак броју x, следећи једнак броју који се добије када заменимо последњу јединицу у бинарном запису x нулом, и тако даље. На пример (за горе приказани низ):
$read(7) = read(0111_2) = bit[0111_2] + bit[0110_2] + bit[0100_2] = 2 + 4 + 7 = 13.$

Аналогно, када извршавамо операцију $update(x, val)$, потребно је повећати само оне елементе низа bit који обухватају поље x; исто као што при рачунању кумулативних сума стално одузимамо последњу јединицу, ова операција је обрнута; додајемо 2^l у сваком кораку да бисмо добили следећи индекс низа bit који треба ажурирати (где је l позиција последње јединице у тренутном индексу). Ово радимо све док је индекс унутар низа, тј. док не прекорачи n. Дакле, уколико желимо за низ дат горе да извршимо $update(3,5) = update(0011_2, 5)$, секвенца операција би била следећа:

$bit[0011_2] \coloneqq bit[0011_2] + 5; nextIndex = 0011_2 + 0001_2 = 0100_2;$

$bit[0100_2] \coloneqq bit[0100_2] +\ 5; nextIndex = 0100_2 + 0100_2 = 1000_2;$

$bit[1000_2] \coloneqq bit[1000_2] +\ 5; nextIndex = 1000_2 + 1000_2 = 10000_2 > n; HALT.$

Овиме имамо све што нам је потребно да бисмо имплементирали Фенвиково стабло; само нам треба начин да ефикасно извучемо број који има јединицу само на последњем месту као број x. Ово можемо добити као вредност израза $x\,\&-x$, где & представља операцију битовне конјункције (bitwise AND). Ово функционише зато што се бројеви чувају у рачунарима у форми комплемента двојке; да бисмо добили број супротан неком броју, потребно је обрнути му све битове у бинарном запису и додати му 1.

Сада можемо написати C++ код за операције $read$ и $update$:

```
int bit[MAX_N];

void update(int x, int val)
{
    while (x <= n)
    {
        bit[x] += val;
        x += (x & -x);
    }
}

int read(int x)
{
    int ret = 0;
    while (x > 0)
    {
        ret += bit[x];
        x -= (x & -x);
    }
    return ret;
}
```

Пошто ће обе операције највише да изврше број корака једнак броју цифара у бинарном запису броја n, лако је закључити да су обе операције сложености $O(\log n)$. Фенвиково стабло, као структура која је јако једноставна за кодирање, која нуди ефикасно (и у меморијском и у временском смислу) манипулисање кумулативним сумама (које се јако често појављују као подпроблем у многим задацима), и која се лако генерализује на више димензија, је стога структура коју би сваки такмичар требао да има у свом ”арсеналу”.

Разни задаци:

z-magija (www.z-trening.com), ***It's a Murder!*** (www.spoj.com), ***Matrix Summation*** (www.spoj.com).