

КОРЕНОВА ДЕКОМПОЗИЦИЯ НА МАСИВИ

Национална лагер-школа по
информатика, 29 август 2024 г.,
Ямбол

Изготвил: Илиян Йорданов Йорданов

НАЧАЛНА ПОСТАНОВКА

Имаме масив $a[0 \dots N - 1]$ с N числа и онлайн заявки:

- заявка за въпрос $l \ r$ за намиране на сумата $a[l] + a[l + 1] + \dots + a[r]$
- заявка за промяна $p \ v$ на стойност, така че $a[p] += v$

НАЧАЛНА ПОСТАНОВКА

Имаме масив $a[0 \dots N - 1]$ с N числа и онлайн заявки:

- заявка за въпрос $l \ r$ за намиране на сумата $a[l] + a[l + 1] + \dots + a[r]$
- заявка за промяна $p \ v$ на стойност, така че $a[p] += v$

Стандартно решение: сегментно дърво или дърво на Фенуик.

Сложност на заявка: $O(\log_2 N)$

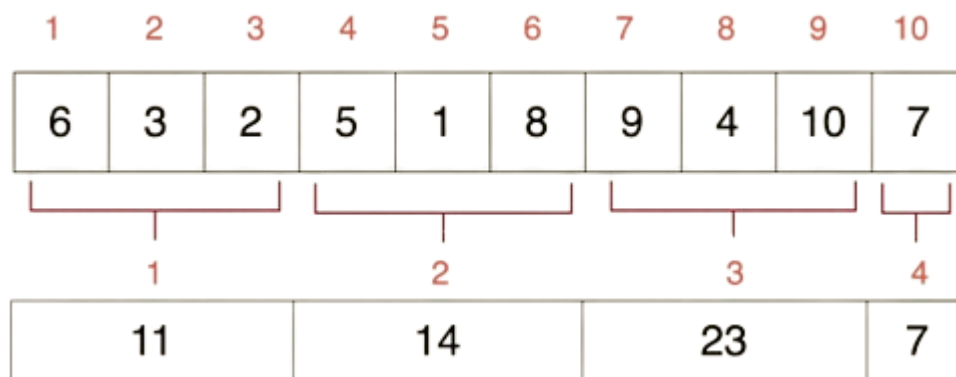
АЛТЕРНАТИВЕН ПОДХОД

Алтернатива е кореновата декомпозиция на масива.

Основната идея е да разделим масива на бъкети (блокове) с равна дължина K (разбира се може да получим един непълен бъкет накрая).

За всеки бъкет пазим сумата на числата в него.

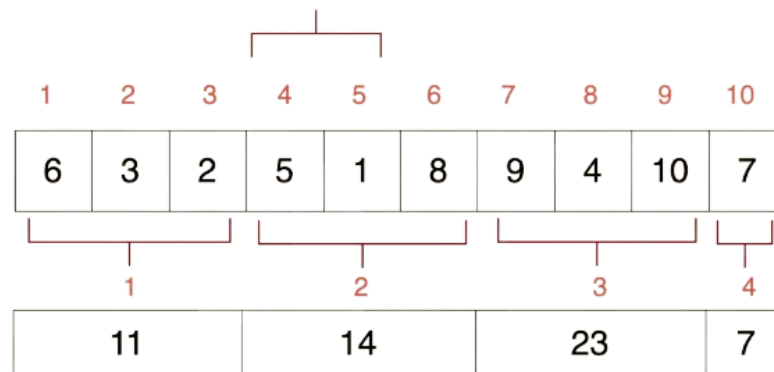
Тогава при заявка за промяна p v трябва да добавим v към стойността на $a[p]$ и към стойността за сумата на бъкета, в който се намира $a[p]$.



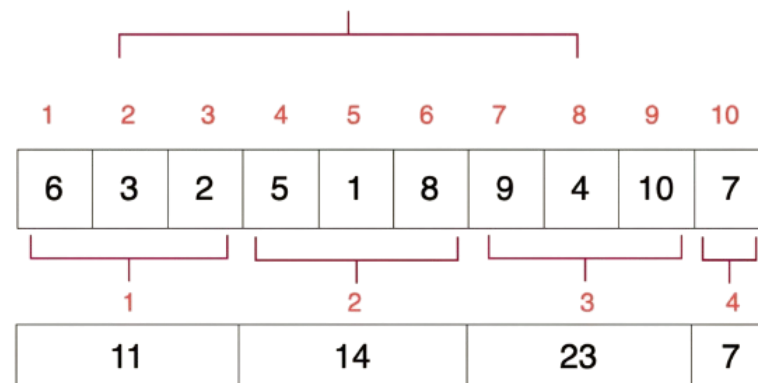
КОРЕНОВА ДЕКОМПОЗИЦИЯ

Заявката за въпрос l r е малко по-сложна. Имаме два случая:

- l и r са в един бъкет



- l и r са в различни бъкети



ИМПЛЕМЕНТАЦИЯ

```
int a[MAXN], n;
int buckets[MAXN/K+1];
void build_buckets () {
    int ind=-1;
    for (int i=0; i<n; i++) {
        if (i%K==0) ind++;
        buckets[ind]+=a[i];
    }
}

void update (int p, int v) {
    a[p]+=v;
    buckets[p/K]+=v;
}
```

```
int query (int l, int r) {
    int lind=l/K, rind=r/K;
    int sum=0;
    if (lind==rind) {
        for (int i=l; i<=r; i++) {
            sum+=a[i];
        }
    }
    else {
        for (int i=l; i<(lind+1)*K; i++) {
            sum+=a[i];
        }
        for (int i=lind+1; i<rind; i++) {
            sum+=buckets[i];
        }
        for (int i=rind*K; i<=r; i++) {
            sum+=a[i];
        }
    }
    return sum;
}
```

АНАЛИЗ НА СЛОЖНОСТТА

Постигнахме следните сложности за заявките:

- $O(1)$ за заявката за промяна
- $O\left(K + \frac{N}{K}\right)$ за заявката за въпрос в най-лошия случай

Математически може да се докаже, че при $K \approx \sqrt{N}$, тази функция достига най-малката си стойност и тогава сложността става $O(\sqrt{N})$.

На практика трябва да се експериментира с най-подходящото K спрямо тестовете!

ПРЕДИМСТВА И НЕДОСТАТЪЦИ

Предимства на кореновата декомпозиция спрямо сегментните дървета:

Недостатъци на кореновата декомпозиция спрямо сегментните дървета:

ПРЕДИМСТВА И НЕДОСТАТЪЦИ

Предимства на кореновата декомпозиция спрямо сегментните дървета:

- по-добра при много заявки за промени
- по-гъвкава и работи за по-широк набор заявки за въпроси и заявки за промени
- само $O(\sqrt{N})$ допълнителна памет спрямо $O(N)$ при сегментните дървета

Недостатъци на кореновата декомпозиция спрямо сегментните дървета:

- значително по-бавни заявки в общия случай

МНОГО ЗАЯВКИ ЗА ВЪПРОС

Можем да обърнем сложностите, така че сложността за отговор да е константна.

Как да го направим?

МНОГО ЗАЯВКИ ЗА ВЪПРОС

Можем да обърнем сложностите, така че сложността за отговор да е константна.

Ще направим префиксен масив за всеки бъкет, както и един префиксен масив за сумите по бъкети.

Така вече при заявка за промяна ще трябва да построим наново префиксния масив на „засегнатия“ бъкет, както и наново да построим префиксния масив за сумите на бъкетите.

ИМПЛЕМЕНТАЦИЯ

```
int a[MAXN], n;
struct bucket {
    int pref[K];
    int find_sum (int l, int r) {
        return pref[r] - ((l==0)?0:pref[l-1]);
    }
};
bucket buckets[MAXN/K+1];
int pref[MAXN/K+1];
void build_buckets () {
    int ind=-1;
    for (int i=0; i<n; i++) {
        int curr=i%K;
        if (curr==0) {
            ind++;
            buckets[ind].pref[0]=a[i];
            if (ind>0) pref[ind]=pref[ind-1];
        }
        else buckets[ind].pref[curr]=buckets[ind].pref[curr-1]+a[i];
        pref[ind]+=a[i];
    }
}
```

```
int query (int l, int r) {
    int lind=l/K, rind=r/K;
    if (lind==rind) return buckets[lind].find_sum(l%K, r%K);
    int sum=0;
    sum+=buckets[lind].find_sum(l%K, K-1);
    sum+=pref[rind-1]-pref[lind];
    sum+=buckets[rind].find_sum(0, r%K);
    return sum;
}
void update (int p, int v) {
    int ind=p/K;
    for (int i=p%K; i<K; i++) {
        buckets[ind].pref[i]+=v;
    }
    for (int i=ind; i<=n/K; i++) {
        pref[i]+=v;
    }
}
```

МНОГО ЗАЯВКИ ЗА ВЪПРОС

При по-сложни заявки за въпрос подобен подход може да не е приложим.

В общия случай, такъв подход можем да приложим, когато имаме бърз начин да обобщим нужната информация за отговор на въпроса (линеен относно броя елементи).

Как може да постъпим при заявки за минимум?

ПРИМЕРНА ЗАДАЧА - АВЗ. СУМИ ОТ НОИ-3, 2023 ГОДИНА

Дадени са банкноти със стойности x_1, x_2, \dots, x_n и q заявки от вида (l, r, a, b) , за които трябва да се намери колко от сумите със стойности от a до b могат да се представят като сума на част от банкнотите x_l, x_{l+1}, \dots, x_r (може и всичките).

Ограничения: $n \leq 500, q \leq 1.5 \cdot 10^6, s = x_1 + x_2 + \dots + x_n \leq 2.5 \cdot 10^5$

ПРИМЕРНА ЗАДАЧА - АВЗ. СУМИ ОТ НОИ-3, 2023 ГОДИНА

Дадени са банкноти със стойности x_1, x_2, \dots, x_n и q заявки от вида (l, r, a, b) , за които трябва да се намери колко от сумите със стойности от a до b могат да се представят като сума на част от банкнотите x_l, x_{l+1}, \dots, x_r (може и всичките).

Ограничения: $n \leq 500, q \leq 1.5 \cdot 10^6, s = x_1 + x_2 + \dots + x_n \leq 2.5 \cdot 10^5$.

Възползваме се от офлайн отговаряне на заявките и задачата се свежда до заявки за отбелязване на суми, които могат да се постигнат (с добавяне на 1-ца в броячен масив) и въпроси за това кои суми могат да се постигнат в даден интервал. Оказва се, че броят заявки за промяна са $O(ns) \approx 1.25 \cdot 10^8$, а заявките въпрос са само $O(q) \approx 1.5 \cdot 10^6$.

Затова по-бързо е вместо дърво на Фенуик, да се използва коренова декомпозиция върху масива с отбелязаните суми с големина s .

ИНТЕРВАЛНИ ПРОМЕНИ

Нека сега заявката за промяна е $l\ r\ v$ и тя променя стойностите: $a_l += v$, $a_{l+1} += v, \dots, a_r += v$.

Как да се справим с тези заявки?

ИНТЕРВАЛНИ ПРОМЕНИ

Нека сега заявката за промяна е $l\ r\ v$ и тя променя стойностите: $a_l += v$, $a_{l+1} += v, \dots, a_r += v$.

Достатъчно е да подходим подобно на статичното lazy propagation в сегментните дървета – за всеки бъкет ще натрупваме lazy стойност. При заявка за промяна ръчно променяме обхванатите елементи на масива, които са в първия и последен бъкет на заявката, а ако има междинни бъкети, за тях трупаме v само в lazy стойността.

Така реалната стойност на даден елемент е сумата от записаната стойност и lazy стойността на бъкета, в който се намира.

ЗАЯВКИ, КОИТО СТАВАТ САМО С КОРЕНОВА ДЕКОМПОЗИЦИЯ

Задача 13Е. Дупки от codeforces: <https://codeforces.com/problemset/problem/13/E>

Имаме поредица отляво-надясно от N дупки, всяка със сила $a_i \geq 1$. Трябва да се отговори на M заявки от два вида:

- промяна на силата на дупка
- пускане на топче в дадена дупка, след което трябва да преброим броя дупки, през които минава топчето и да намерим последната дупка, която ще достигне топчето преди да излезе (когато топчето попадне в дупка с номер i , то скача в дупка с номер $i + a_i$)

Ограничения: $N, M \leq 10^5$.

Как да решим задачата?

ЗАЯВКИ, КОИТО СТАВАТ САМО С КОРЕНОВА ДЕКОМПОЗИЦИЯ

Кореновата декомпозиция ни позволява да правим по-бърза симулация за всеки въпрос.

Затова е достатъчно само да поддържаме актуално „големите скокове“, които ни придвижват от дадена дупка към последната дупка преди да напуснем бъкета. Трябва да пазим броя скокове, от които се състои големият скок и разбира се номера на дупката, в която отиваме.

Заявката за промяна е просто да преизчислим всички големи скокове за „засегнатия“ бъкет.

Ако големината на бъкета $K \approx \sqrt{N}$, то сложността и на двете заявки ще е $O(\sqrt{N})$.

ЗАЯВКИ, КОИТО СТАВАТ САМО С КОРЕНОВА ДЕКОМПОЗИЦИЯ

Техниката, която построява наново бързет, за да намери актуалната информация, е една от най-силните страни на кореновата декомпозиция. Още един пример за използването ѝ е в задача В2. Карнавал от Летния турнир, 2022 година.

Започваме с начален масив $x[1 \dots N]$ с N числа и имаме два вида заявки:

- намиране на максималното число в интервал $l \ r$
- промяна на числата с характеристики $l \ r \ a \ d$, така че $x_l += a, x_{l+1} += (a + d), \dots, x_r += (a + (r - l) \cdot d)$ (добавяне на аритметична прогресия с начална стойност a и разлика d към интервала $l \ r$)

Ограничения: $N, Q \leq 10^5, 0 \leq x_i, a, (r - l) \cdot d \leq 10^{12}$.

Ключът на задачата е в подзадача 6, където $l = 1$ и $r = N$ за всички заявки.
Идеи?

ЗАЯВКИ, КОИТО СТАВАТ САМО С КОРЕНОВА ДЕКОМПОЗИЦИЯ

Понеже в подзадача 6 всички заявки обхващат целия масив, то можем да разгледаме всички потенциални стойности на число като права, като x координатата е натрупаната разлика на аритметични прогресии, а y координатата е достигнатата стойност.

Така ако приложим Convex hull trick, можем да знаем кое от числата ще даде максималната стойност в зависимост от натрупаната разлика (началните стойности на аритметичната прогресия се добавят към всички числа, затова те не влияят на максимума и само трябва да смятаме сумата им, за да получим реалните стойности).

ЗАЯВКИ, КОИТО СТАВАТ САМО С КОРЕНОВА ДЕКОМПОЗИЦИЯ

Проблемът идва, когато вече заявките може да обхващат само част от масива. Понеже имаме хубав подход за цял масив, това директно ни позволява да решим цялата задача с коренова декомпозиция!

Когато имаме заявка за промяна, построяваме наново първия и последния бъкети в заявката. Ако има междинни бъкети, за тях просто трупаме разлика и начална стойност на прогресиите (както в подзадача 6).

Когато имаме заявка за въпрос, намираме с просто обхождане максималните стойности в първия и последния бъкети на заявката. Ако има междинни бъкети, в тях намираме максимума като използваме построения Convex hull trick и натрупаната разлика.

TIERED VECTOR – КОРЕНОВА СТРУКТУРА ДАННИ

Кореновата декомпозиция е толкова гъвкава, че дори ни позволява да поддържаме и динамични заявки като добавяне и премахване на елементи.

Така можем да поддържаме масив с $O(1)$ достъп до елемент и $O(\sqrt{N})$ за добавяне и премахване на елементи на дадена позиция.

Главната идея е да поддържаме C на брой бъкети с максимален размер K , като началните бъкети са пълни (съдържат елементите на масива), а накрая има и празни бъкети (за бъдещи числа).

Всеки бъкет ще представим като дек (двустранна опашка), така че константно да можем да премахваме и добавяме числа отпред и отзад.

За повече информация, може да прочетете тази статия:

<https://cs.brown.edu/cgc/jdsl/papers/tiered-vector.pdf>

ПОСЛЕДНА ЗАДАЧА – УСЛОВИЕ

Дадени са два масива A и B . Трябва да се отговорят Q заявки (a, b, c, d) , които питат за сумата на $|A_i - B_j|$, където $a \leq i \leq b$ и $c \leq j \leq d$.

Ограничения: $N, Q \leq 10^5, A_i, B_j \leq 10^9$.

ПОСЛЕДНА ЗАДАЧА – РАЗМИСЪЛ

Дадени са два масива A и B . Трябва да се отговорят Q заявки (a, b, c, d) , които питат за сумата на $|A_i - B_j|$, където $a \leq i \leq b$ и $c \leq j \leq d$.

Ограничения: $N, Q \leq 100000, A_i, B_j \leq 10^9$.

Проблем е, че заявката обхваща два масива и сумата, която търсим, изисква сортиране на подмасивите за по-бързо смятане.

За дадена заявка, ако сме сортирали подмасива на A и подмасива на B , то можем да обхождаме двата масива едновременно, подобно на сортиране чрез сливане, за да знаем при фиксирано A_i кои числа B_j са по-малки и кои по-големи.

ПОСЛЕДНА ЗАДАЧА – РАЗМИСЪЛ

В такъв случай, е добре да се пробва решение с коренова декомпозиция. Ще направим коренова декомпозиция на двата масива и с подходящ прекъмпют ще може да отговаряме сравнително бързо заявките и при това онлайн.

Нека помислим какво ще правим като отговаряме заявка. В общия случай подмасивът на A се разпада на два частични бъкета (първия и последния) и няколко пълни междинни бъкета. Имаме само \sqrt{N} възможни бъкета, докато за частичните бъкети възможностите са около $N\sqrt{N}$ (в частния случай, когато подмасивът е вътре в един бъкет, възможностите са най-много).

ПОСЛЕДНА ЗАДАЧА – РАЗМИСЪЛ

Най-трудното е намирането на сумата за числата в частичните бъкети.

Как можем да постъпим?

ПОСЛЕДНА ЗАДАЧА – РЕШЕНИЕ

Ще използваме прекъмпют за сумата, когато комбинираме едно число с бѐкет:

- $prec_A[i][j]$ – търсената сума, която можем да получим, ако комбинираме A_i с бѐкет номер j на масива B
- $prec_B[i][j]$ – търсената сума, която можем да получим, ако комбинираме B_i с бѐкет номер j на масива A

Тези таблици могат лесно да се сметнат като се фиксира някой бѐкет на единия масив и последователно обходим в сортиран ред числата на другия масив.

Също ще се нуждаем от префиксни масиви за таблиците – $pref_A[i][j]$ и $pref_B[i][j]$, за да можем бързо да смятаме сумите за комбиниране на няколко числа с бѐкет.

Сложността на прекъмпюта е $O(N\sqrt{N})$.

ПОСЛЕДНА ЗАДАЧА – РЕШЕНИЕ

Остана да опишем накратко и отговарянето на заявка. Ще разгледаме само общия случай, когато подмасивът на A се разпада на бъкети C_1, C_2, \dots, C_t и подмасивът на B се разпада на бъкети D_1, D_2, \dots, D_s . Тогава за да намерим отговорът гледаме:

- линейно комбинираме C_1 и D_1 , C_1 и D_s , C_t и D_1 , C_t и D_s (ако предварително сме сортирали числата от всеки бъкет)
- използваме $pref_A$ за комбинирането на подмасива на A и D_2, \dots, D_{s-1}
- използваме $pref_B$ за комбинирането на D_1 и C_2, \dots, C_{t-1} , както и D_s и C_2, \dots, C_{t-1}

Трябва да се внимава дали покриваме добре и случая, когато подмасивът на A се съдържа в някой бъкет, но с описания подход този случай не е проблемен.

Така цялата сложност става $O((N + Q)\sqrt{N})$.

ДОМАШНО (ЗАДАЧИ ЗА УПРАЖНЕНИЕ)

- Дорешаване на Задача 13Е. Дупки от codeforces (<https://codeforces.com/problemset/problem/13/E>)
- Задача 455D от codeforces: <https://codeforces.com/contest/455/problem/D> (tiered vector)
- Задача ISQ от BOI 2022 (прекъмпют с коренова декомпозиция)



БЛАГОДАРЯ ЗА ВНИМАНИЕТО!

Изготвил: Илиян Йорданов Йорданов