Dekompozycja pierwiastkowa

adam.szczecin.pl

Wszystko bowiem składa się z wielu części, a jednak nie ma czegoś, co stanowiłoby całość.
Arystoteles

1 Bloki pierwiastkowe

Technika zwana blokami pierwiastkowymi pozwala na obliczanie operacji wykonywanych na tablicy w czasie $O(\sqrt{n})$, poprzez podzielenie tablicy o długości n na $\sqrt{n}$ bloków, o długości $\sqrt{n}$ każdy.

Operacja może być tak naprawdę dowolna, o ile jest ona łączna, jednak aby zrozumieć tę technikę i dla ustalenia uwagi rozważmy następujący przykład.

Przykład 1. Dana jest tablica $A[0..n-1]$. Wykonaj q operacji dwóch typów:
(1) dla danych i oraz x przypisz $A[i] \leftarrow x$,
(2) dla danych l, r znajdź sumę $A[l..r]$.

Jakoby zadanie to nie powinno nastręczyć żadnych trudności. Oczywiście problem ten można rozwiązać naiwnie. Otrzymujemy wtedy złożoność $O(1)$ dla operacji (1) oraz złożoność $O(n)$ dla (2). Istnieje też rozwiązanie korzystające z drzewa przedziałowego osiągające złożoność $O(\log n)$ dla obu typów zapytań.

Jednak pokażemy teraz pozornie gorsze rozwiązanie działające w czasie $O(\sqrt{n})$ dla obu typów zapytań.

Niech k będzie pewną liczbą. Dla uproszczenia załóżmy, że k dzieli n. Podzielmy tablicę A na n/k bloków. Niech $l_j := jk$ oraz $r_j := (j+1)k - 1$. Wtedy j-ty $(0 \leqslant j < n/k)$ blok to podtablica $A[l_j..r_j]$. Oznaczmy $b(i)$ jako indeks bloku, do którego należy i-ty element, tj. $b(i) = i/k$.

Pomysł polega na tym, że dla każdego bloku i będziemy przechowywali wartość $B[i] := \sum A[l_i..r_i]$. Pokażemy teraz jak postępować w przypadku zapytań.

$$\underbrace{a[0], a[1], \ldots, a[k-1]}_{b[0]}, \underbrace{a[k], \ldots, a[2k-1]}_{b[1]}, \ldots, \underbrace{a[(n/k-1)k], \ldots, a[n-1]}_{b[n/k-1]}$$

(1) Niech $j := b(i)$. Zauważmy, że zmienia się tylko $A[i]$ oraz $B[j]$. Zaktualizujemy je więc naiwnie w czasie $O(k)$.

Uwaga 1. Operacja dodawania ma swoją operację odwrotną (odejmowanie). Zapytanie (1) możemy wykonywać więc w czasie stałym. Jednak dla funkcji nieposiadających funkcji odwrotnej, trzeba przeliczać cały blok jeszcze raz.

(2) Wiadomo, że wszystkie bloki między $b(l)+1$ a $b(r)-1$ włącznie zawierają się w podtablicy $A[l..r]$ (nazwijmy te bloki *fragmentem środkowym*). Ponadto bloki $b(l)$ oraz $b(r)$ zawierają się *częściowo* w podtablicy $A[l..r]$ (niech będzie to *fragment krańcowy*). Wszystkie inne bloki nie mają części wspólnej z $A[l..r]$.

Można obliczyć osobno sumę fragmentu krańcowego oraz środkowego.

Wartość sumy fragmentu krańcowego liczymy naiwnie w $O(k)$. Zauważmy, że dzięki tablicy B możemy zrealizować liczenie sumy fragmentu środkowego w czasie $O(n/k)$.

Złożoność tego zapytania wynosi zatem $O(k+n/k)$. Ostateczna złożoność wynosi więc $O(q(k+n/k))$. Pozostaje tylko dobrać k tak, aby zminimalizować $q(k + n/k)$. Oczywiście q jest stałe, więc tak naprawdę minimalizujemy $k + n/k$.

Lemat 1. *Funkcja $f(k) = k + n/k$, gdzie $k > 0$ osiąga swoje minimum dla $k = \sqrt{n}$.*

Dowód: Z nierówności między średnią arytmetyczną a geometryczną mamy $f(k) = k + n/k \geqslant 2\sqrt{n} = f(\sqrt{n})$. □

Zatem kiedy wybierzemy $k = \sqrt{n}$ otrzymujemy rozwiązanie działające w żądanym czasie $O(q\sqrt{n})$.

```
1  int k = sqrt(n)+1;
2  int B[n] = {0};
3  void update_internal(int i)
4  { // Aktualizuje blok, do którego należy indeks i.
5      int b = i / k;        // Numer bloku.
6      int l = b * k;        // Początek bloku.
7      int r = min(l+k, n); // Koniec bloku.
8      B[b] = A[i];          // Najpierw skopiuj.
9      for (int i = l; i < r; i++)
10         B[b] = B[b] + A[i]; // Liczy sumę A[l..r].
11 }
12 void update_value(int i, int x)
13 { // Akutalizuje wartość na indeksie i.
14     A[i] = x;
15     update_internal(i);
16 }
17 void init()
18 { // Inicjalizuje tablicę B.
19     for (int i = 0; i < n; i++)
20         update_value(i, A[i]);
21 }
22 int sum_range(int l, int r)
23 { // Zwraca sumę A[l..r]
24     int ans = 0;
25     int i = l;
26     while (i <= r)
27     {
28         if ((i%k == 0) && (i+k-1 <= r))
29         { // Jesteśmy we fragmencie środkowym.
30             ans = ans + B[i/k];
31             i += k; // Skaczemy o k.
32         }
33         else
34         { // Jesteśmy we fragmencie krańcowym.
```

```
35              ans = ans + A[i];
36              i++; // Skaczemy o 1.
37          }
38      }
39      return res;
40  }
```

Ćwiczenie 1. Oblicz tablicę $B[0..2]$ dla tablicy $A = (3, 1, 4, 1, 5, 9, 2, 6, 5)$.

Ćwiczenie 2. Jaś ma tablicę $A[0..15]$. Obliczył dla niej tablicę $B[0..3]$. Okazało się, że aby obliczyć sumę pewnej podtablicy $A[\ell..r]$ potrzebował dokładnie 4 odwołań do pamięci (odczytaie wartości w komórce tablicy A lub B). Jaka mogła być długość przedziału $r - \ell + 1$? Podaj wszystkie możliwości.

W przypadku przykładu 1., jak już wcześniej wspomnieliśmy, można było uzyskać szybsze rozwiązania bez użycia dekompozycji pierwiastkowej. Są jednak zadania, gdzie nie jest to możliwe.

Uwaga 2. Zastanówmy się, co gdyby zapytanie (2) polegało na dodaniu wartości x na pewnym przedziale $A[l..r]$? Można wtedy zastosować leniwą propagację. Będziemy dla każdego bloku j przetrzymywać dodatkową liczbę $S[j]$. Umówimy się, że *prawdziwa* wartość $A[i]$ będzie to $A[i]+S[b(i)]$. Wtedy prawdziwa wartość $B[j]$ będzie to $B[j] + kS[j]$. Zatem przy rozważaniu fragmentów środkowych możemy tylko zmodyfikować $S[j]$, zachowując stałą złożoność dla jednego bloku.

2 Bloki pierwiastkowe na zapytaniach

Tym razem na bloki nie będziemy dzielić tablicy, ale zapytania. Tj. będziemy rozważać zapytania w blokach po $\sqrt{q}$ zapytań.

Rozważmy jeszcze raz przykład 1.

Na chwilę zapomnijmy o zapytaniach typu (1). Oczywiście zadanie staje się wtedy trywialne. Wystarczy tylko obliczyć sumy częściowe tablicy A, aby móc odpowiadać na pytania w czasie stałym.

Co jeśli jednak dostaniemy operację typu (1)? Na razie nie wykonamy żadnej aktualizacji, tylko zapamiętamy, że istotnie taka operacja miała miejsce. Teraz kiedy dostaniemy pytanie typu (2) suma przedziału obliczona z użyciem sum częściowych może być już nieaktualna. Przejdziemy się zatem po wszystkich dotychczasowych operacjach typu (1) i dostosujemy wynik.

Sprawny czytelnik od razu zauważy, że takie rozwiązanie ma złożoność $O(q^2)$. Jeśli q jest małe działa ono więc dosyć dobrze. Zatem co k zapytań przeliczymy naszą tablicę sum częściowych jeszcze raz, uwzględniając ostatnie k operacji. Wtedy blok k zapytań obsługujemy w czasie $O(k^2 + n)$. Jako, że wszystkich bloków jest q/k, ostatecznie rozwiązanie to działa w czasie $O((k^2 + n)q/k) = O(q(k + n/k))$.

Teraz wystarczy tylko dobrać odpowiednie k. Oczywiście z lematu 1. mamy $k = \sqrt{n}$.

Dostajemy zatem rozwiązanie działające w czasie $O(q\sqrt{n})$.

```
int k = sqrt(n)+1;
int A[n] = {0};
int pref[n] = {0};
vector<pair<int, int>> history;
void rebuild()
{
    for (auto [i, x] : history)
        A[i] += x;
    history.clear();
    pref[0] = A[0];
    for (int i = 1; i < n; i++)
        pref[i] = pref[i-1] + A[i];
}
void update(int i, int x)
{
    history.emplace_back(i, x);
    if (hisotry.size() > k)
        rebuild();
}
int query(int l, int r)
{
    int ans = pref[r] - (l ? pref[l-1] : 0);
    for (auto [i, x] : history)
    {
        if (l <= i && i <= r)
            ans += x;
    }
    return ans;
}
```

W przypadku tego problemu użycie dekompozycji pierwiastkowej nie było konieczne. Jednak może się ona okazać przydatna, kiedy umiemy rozwiązać trudny problem bez aktualizacji.

3 Lekkie/ciężkie

Powiedzmy, że mamy do wykonania pewną liczbę operacji opisanych jakimś parametrem t. Załóżmy, że jeśli t jest małe szybko umiemy wykonać aktualizację. Również jeśli t jest duże umiemy łatwo taką aktualizację przeprowadzić. Aby połączyć te dwa rozwiązania powiemy, że dla pewnego k, dla pytań z $t \leqslant k$ będziemy zachowywać się inaczej niż dla $t > k$. Najczęściej będziemy przyjmować $k = \sqrt{n}$.

W celu zilustrowania, rozważmy następujący przykład.

Przykład 2. Dany jest graf $G = ([n], E)$. Na wierzchołku u zapisana jest liczba $A[u] = u$. Obsłuż q zapytań:

dla danego wierzchołka u, dla każdego v takiego, że $\{u, v\} \in E$, ustaw $A[v] \leftarrow A[u]$.

Podaj $A[1..n]$ po wykonaniu wszystkich zapytań.

Rozwiązanie: Możemy rozwiązać zadanie naiwnie w $O(nq)$, dla każdego zapytania przechodząc się po wszystkich sąsiadach u. Pomimo tej pesymi-

stycznej złożoności, rozwiązanie to jest szybkie, kiedy $\deg u$ jest małe. Zatem dla $\deg u < \sqrt{m}$ po prostu przejdziemy się po sąsiadach u i zaktualizujemy wartości $A[\cdot]$.

Dla $\deg u \geqslant \sqrt{n}$ zwyczajnie wstawimy znacznik w wierzchołku u, że taka aktualizacja miała miejsce.

Jednak, jak już pewnie Czytelnik zauważył, teraz kiedy rozważamy jakiś wierzchołek u, to przed zrobieniem czegokolwiek, musimy najpierw znaleźć prawdziwą wartość $A[u]$. W tym celu iterujemy się po wszystkich sąsiadach u o stopniu co najmniej $\sqrt{m}$ (tylko te wierzchołki mogą mieć jakieś znaczniki). Zauważmy, że takich wierzchołków jest co najwyżej $O(2m/\sqrt{m}) = O(\sqrt{m})$.

Odnotujmy jeszcze, że trzeba przetrzymywać czas ustawienia $A[\cdot]$ oraz czas wstawienia każdego ze znaczników. ◊

Uwaga 3. Najczęściej wybieraliśmy $k = \sqrt{n}$. Nie zawsze jest to optymalny wybór. Zawsze należy się zastanowić, jaka wartość parametru k minimalizuje funkcję złożoności. Nawet kiedy $k = \sqrt{n}$ minimalizuje funkcję złożoności, nie zawsze używa się $k = \lfloor\sqrt{n}\rfloor$. Czasem używa się bardziej *okrągłej* liczby, aby przyspieszyć wykonywanie operacji.

Ćwiczenie 3. Zaimplementuj algorytm 1. oraz wygeneruj do niego wejście dla dużych n, q. Testuj szybkość wykonania programu dla różnych stałych k. Wyszukaj binarnie tą optymalną. Jak ma się do oczekiwanej wartości $\sqrt{n}$?

> *Kluczem do perfekcyjnego wykonania jest równowaga między siłą i lekkością.*
>
> Leonardo da Vinci

> *W wieku piętnastu lat poświęciłem się nauce, w wieku trzydziestu lat stałem się niezależny, w wieku czterdziestu lat nie miałem żadnych wątpliwości, w wieku pięćdziesięciu lat zrozumiałem wolę Niebios, w wieku sześćdziesięciu lat moje uszy stały się posłuszne, w wieku siedemdziesięciu lat mogłem podążać za pragnieniami serca, nie łamiąc zasad.*
>
> Konfucjusz

Proponujemy rozwiązanie kilku zadań.

Zadanie 4. Dana jest tablica $A[1..n]$. Wykonaj q zapytań dwóch typów:

(1) dla danych $1 \leqslant l \leqslant r \leqslant n$ oraz x, znajdź liczbę $i \in [l..r]$ takich, że $A[i] = x$,

(2) dla danych $1 \leqslant l \leqslant r \leqslant n$ oraz y, dla każdego $i \in [l..r]$, przypisz $A[i] \leftarrow A[i] + y$.

Zadanie 5. Dana jest tablica $A[1..n]$. Początkowo dla każdego i mamy $A[i] = 0$. Wykonać q operacji dwóch typów:

(1) dla danych a, b, c dla każdego $k \equiv a \pmod b$ przypisać $A[k] \leftarrow A[k] + c$,

(2) dla danego d znaleźć $A[d]$.

Zadanie 6. Dana jest tablica $A[0..n-1]$. Dla danego s rozważmy następującą procedurę:

Wrzuć piłeczkę do komórki tablicy A pod indeksem s. Następnie, dopóki piłeczka nie wyleci poza tablicę, odbija się ona z komórki pod indeksem i do komórki pod indeksem $i + A[i]$.

Obsłuż q operacji dwóch typów:

(1) dla danych i oraz x przypisz $A[i] \leftarrow x$,

(2) dla danego s wykonaj opisaną procedurę i powiedz, ile razy odbiła się piłeczka.

Zadanie 7 (Kontenery [24OI])**.** Inżynier Bajtazar zarządza rampą przeładunkową dla kontenerów. Rampa składa się z n kolejnych pozycji. Na każdej z nich dźwig może umieścić dowolną liczbę kontenerów ułożonych jeden na drugim. Pozycje numerujemy kolejno od 1 do n.

Niektóre kontenery przechowują niebezpieczne materiały, które wymagają, by takie kontenery nie stały w zbyt dużym skupieniu.

Bajtazar dostał listę k operacji, które wykona dźwig: i-ta z tych operacji ma postać (a_i, ℓ_i, d_i), co oznacza, że dźwig umieści ℓ_i kontenerów – poczynając od pozycji numer a_i, po jednym kontenerze na co d_i-tej pozycji (czyli na pozycjach $a_i, a_i + d_i, a_i + 2d_i, a_i + (\ell_i - 1)d_i$). Bajtazar zastanawia się, jaka będzie liczba kontenerów na każdej z pozycji po wykonaniu wszystkich operacji z listy.

Zadanie 8. Dana jest siatka $n \times m$. Każda komórka ma przypisaną literę. Znajdź dwie komórki o najmniejszej odległości między nimi (w metryce Manhattan), które mają przypisaną tę samą literę.

Zadanie 9. Dana jest siatka $n \times m$. Początkowo każda komórka jest biała, poza jedną czarną. Wykonujemy $nm - 1$ operacji: dla danej białej komórki u oblicz najmniejszą odległość z u do czarnej komórki i następnie zamaluj u na czarno.

Zadanie 10. Dane są tablice $A[1..n]$, $L[1..n]$, $R[1..n]$. Obsłuż q operacji dwóch typów:

(1) dla danych $k \in [n]$ oraz x, przypisz $A[k] \leftarrow x$,

(2) dla danych $1 \leqslant l \leqslant r \leqslant n$, znajdź wartość sumy

$$\sum_{i=l}^{r} \sum_{j=L[i]}^{R[i]} A[j].$$

Zadanie 11. Rozważmy permutację $\pi[1..n]$ liczb od 1 do n. Grafem permutacji $G(\pi)$ nazywamy graf $([n], E)$, gdzie dla każdego i mamy $\{i, \pi[i]\} \in E$. Wykonać q zapytań dwóch typów:

(1) dla danych i, j zamień wartości $\pi[i]$ z $\pi[j]$,

(2) powiedz, ile jest spójnych składowych w grafie $G(\pi)$.

Zadanie 12. Dane jest drzewo $T = ([n], E)$, ukorzenione w wierzchołku 1. Każdy wierzchołek ma kolor: czarny (oznaczany jako 0) lub biały (oznaczany jako 1). Na początku wierzchołek u ma kolor $A[u]$.

Wykonaj q operacji dwóch typów:

(1) dla danego $u \in [n]$, dla każdego wierzchołka v, będącego na najkrótszej ścieżce od 1 do u, zmień kolor v na przeciwny (biały na czarny, a czarny na biały),

(2) dla danego $u \in [n]$, znajdź liczbę par (v, w) takich, że $v < w$, najniższy wspólny przodek v i w to u oraz kolory v i w są białe.

Odpowiedzi

5 Podzielmy tablicę A na $\sqrt{n}$ bloków. Zdefiniujmy dwuwymiarową tablicę $B[\cdot][\cdot]$. Teraz dla każdej operacji typu (1) wykonujemy:

(i) jeśli $b \leqslant \sqrt{n}$ aktualizujemy $B[b][a] \leftarrow B[b][a] + c$ w czasie $O(1)$ lub

(ii) jeśli $b > \sqrt{n}$ nawinie aktualizujemy tablicę A w czasie $O(n/\sqrt{n}) = O(\sqrt{n})$.

Teraz szybko umiemy odpowiadać na zapytania typu (2). Wystarczy zsumować $A[d]$ oraz $B[b][d \bmod b]$ dla każdego $B \in [1, \sqrt{n}]$.

Ostateczna złożoność wynosi zatem $O(q\sqrt{n})$.

6 Oczywiście, gdyby było tylko jedno pytanie typu (2), zadanie można łatwo rozwiązać programowaniem dynamicznym.

Pomysł jest więc taki, by podzielić A na bloki o długości $\sqrt{n}$ i dla każdego indeksu trzymać liczbę odbić, jaką wykona piłeczka, zanim opuści aktualny blok oraz ostatni indeks na jakim się znajdzie przed opuszczeniem bloku (proste DP).

Na (1) odpowiadamy wtedy w $O(\sqrt{n})$. Operację (2) wykonujemy również w $O(\sqrt{n})$, rekalkulując wartości dla całego bloku.

7 Dla każdego $d_i > \sqrt{n}$ po prostu zasymulujemy ustawianie kontenerów. Natomiast dla każdego $d_i \leqslant \sqrt{n}$ przetrzymywać będziemy tablicę, powiedzmy $t[1..n]$, na której końcowo wykonamy sumy prefiksowe co d_i, to znaczy $p[j] = p[j - d_i] + t[j]$, gdzie $p[j]$ to nasza tablica sum prefiksowych. Aby w $p[j]$ znajdowała się liczba kontenerów na j-tej pozycji, przy i-tym zapytaniu musimy zwiększyć $t[a_i]$ o 1 i zmniejszyć $t[a_i + (\ell_i - 1)d_i + 1]$ o 1.

11 Rozważajmy zapytania w blokach po $\sqrt{q}$. Zauważmy, że dla każdego bloku jest tylko $2\sqrt{q}$ wierzchołków, które się zmieniają i mają znaczenie. Możemy więc dla każdego bloku skompresować graf tylko do nich.