Pierwiastkowa dekompozycja zapytań

*Krzysztof PIECUCH**

W niniejszym artykule chciałbym przedstawić Czytelnikom technikę, która może okazać się pomocna przy rozwiązywaniu różnych problemów informatycznych. Rozważmy przykładowe zadanie. Dana jest N-elementowa tablica T liczb naturalnych z przedziału $[0, N)$ oraz M zapytań postaci (a, b), gdzie $0 \leqslant a \leqslant b < N$. Dla każdego zapytania należy odpowiedzieć na pytanie ile różnych elementów ma podtablica $T[a..b]$. Można łatwo napisać algorytm, który dla zapytania znajduje odpowiedź w czasie $O(N)$, a więc dla M zapytań działa w czasie $O(NM)$ (kod źródłowy 1).

```
Function zapytanie(a, b)
    answer = 0;
    cnt[N] = {0};
    for (i = a; i <= b; i++) do
        if (cnt[T[i]] == 0) then
            answer++;
        cnt[T[i]]++;
    return answer;
```

Kod źródłowy 1

Rozważmy jednak inne podejście do problemu. Zapamiętajmy wynik poprzedniego zapytania i spróbujmy zmodyfikować go tak, aby pasował do kolejnego (kod źródłowy 2).

```
Function dodaj(i)
    if (cnt[T[i]] == 0) then
        answer++;
    cnt[T[i]]++;
Function usun(i)
    if (cnt[T[i]] == 1) then
        answer--;
    cnt[T[i]]--;
Function zapytanie(a, b)
    while (stare_a > a) do
        dodaj(--stare_a);
    while (stare_a < a) do
        usun(stare_a++);
    while (stare_b < b) do
        dodaj(++stare_b);
    while (stare_b > b) do
        usun(stare_b--);
    return answer;
```

Kod źródłowy 2

Nasz algorytm wciąż rozwiązuje problem w czasie $O(NM)$, gdyż przejście z jednego zapytania do następnego może zająć $O(N)$ czasu. Jednak gdybyśmy zagwarantowali, że kolejne zapytania będą podobne, algorytm mógłby działać szybciej. Rozważmy problem, w którym znamy na wstępie wszystkie zapytania. Możemy wybrać kolejność w jakiej będziemy na nie odpowiadać. Załóżmy dla uproszczenia, że N jest kwadratem liczby naturalnej. Okazuje się, że istnieje kolejność, która gwarantuje, że nasz algorytm wykona jedynie $O((M + N)\sqrt{N})$ operacji.

Podzielmy przedział $[0, N)$ na $\sqrt{N}$ części:

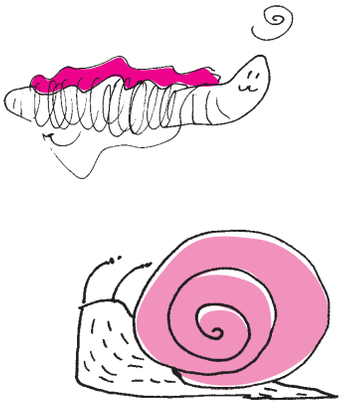
$$[0, \sqrt{N}),\ [\sqrt{N}, 2\sqrt{N}),\ \ldots,\ [(\sqrt{N} - 1)\sqrt{N}, N).$$

Każdy z tych przedziałów jest rozmiaru $\sqrt{N}$. Teraz utwórzmy kubełek dla każdego z nich. Dla każdego zapytania (a, b) patrzymy do którego z tych kubełków wpada a i umieszczamy to zapytanie w odpowiednim kubełku. Następnie w każdym kubełku sortujemy zapytania rosnąco po wartościach b. Przeglądamy teraz kubełki po kolei i odpowiadamy na zapytania zgodnie z ustalonym porządkiem.

Obliczmy złożoność naszego algorytmu. Rozważmy dwie sytuacje. W pierwszej z nich oba zapytania należą do tego samego kubełka. W takiej sytuacji `stare_a` oraz `a` należą do tego samego przedziału. Nie wykonamy zatem więcej niż $O(\sqrt{N})$ operacji `dodaj` i `usuń` w trakcie zmiany `stare_a` na `a`. Jeśli chodzi o wartości `stare_b` oraz `b` to są one w ramach jednego kubełka posortowane rosnąco. Oznacza to, że jeśli dwa zapytania są w jednym kubełku, to będziemy wykonywać jedynie operację `dodaj`. A więc dla ustalonego kubełka sumarycznie wystarczy wykonać jedynie $O(N)$ takich operacji przy zmianach `stare_b` na `b`. Rozważmy teraz sytuację w której zapytania należą do różnych kubełków. Jak powiedzieliśmy wcześniej, w najgorszym przypadku wykonamy $O(N)$ operacji między dwoma zapytaniami. Jednak taka sytuacja zdarzy się co najwyżej $O(\sqrt{N})$ razy, bo tylko tyle mamy kubełków. A więc sumarycznie nasz algorytm zadziała w czasie

$$O(M\sqrt{N} + N\sqrt{N} + N\sqrt{N}) = O((M + N)\sqrt{N}).$$

A teraz dwa ćwiczenia dla Drogiego Czytelnika. Pierwsze jest następujące: mając daną tablicę $T[0..N-1]$ oraz M zapytań (a, b) należy dla każdego zapytania odpowiedzieć ile inwersji znajduje się w podtablicy $T[a..b]$. *Inwersją* w tablicy T nazywamy taką parę liczb (i, j), że $i < j$ oraz $T[i] > T[j]$. Drugie zaś brzmi: mając dany graf $G = (V, E)$ oraz tablicę jego krawędzi T, należy dla każdego zapytania (a, b) odpowiedzieć na pytanie ile spójnych składowych ma graf obcięty do krawędzi $T[a..b]$. Każde z tych zadań można rozwiązać za pomocą opisanej w artykule techniki, nazywanej pierwiastkową dekompozycją zapytań, jednakże wymagają one dodatkowego pomysłu. Mam nadzieję, że rozwiązywanie tych zadań przyniesie Czytelnikom dużo frajdy.



*doktorant, Wydział Matematyki i Informatyki, Uniwersytet Wrocławski