1 Struktury danych dla liczb całkowitych II

1.1 Sztuczki na bitach

Zanim przejdziemy do kolejnej struktury danych, omówimy krótko tzw. „sztuczki na bitach” czyli dodatkowe operacje, które można wykonać w czasie $O(1)$ w modelu WordRAM, często w bardzo pomysłowy sposób.

Najbardziej/najmniej znaczący bit (MSB/LSB) Tę operację można wykonać w czasie $O(1)$, jest to jednak dość duże i skomplikowane $O(1)$ (kilkadziesiąt operacji). Wrócimy do tej kwestii, gdy już nabierzemy trochę wprawy w tego typu problemach.

Pakowanie liczb W słowie długości w można upakować $\lfloor w/b \rfloor$ liczb długości b, tworząc coś w rodzaju wektora. Nie da się tego oczywiście zrobić w czasie stałym, ale jeśli dysponujemy takimi upakowanymi wektora liczb to możemy wykonywać na nich operacje w czasie stałym! Jest to tzw. $r$ównoległość na poziomie słów (ang. word-level paralellism).

Dodawanie spakowanych wektorów liczb Aby dodać dwa spakowane wektory liczb (współrzędna po współrzędnej) wystarczy dodać ich spakowane reprezentacje. Aby uniknąć problemów z ew. przeniesieniami dodajemy dodatkowe zero pomiędzy każdymi dwoma liczbami, tzn. dodajemy liczby $0(A[1])_2 0(A[2])_2 \dots 0(A[k])_2$ i $0(B[1])_2 0(B[2])_2 \dots 0(B[k])_2$.

Porównywanie spakowanych wektorów liczb Aby porównać dwa spakowane wektory liczb (współrzędna po współrzędnej) wykonujemy $R = 1(A[1])_2 1(A[2])_2 \dots 1(A[k])_2 - 0(B[1])_2 0(B[2])_2 \dots 0(B[k])_2$. Po wykonaniu tego odejmowania bit poprzedzający zapis $A[k]$ jest 1 wtw, gdy $B[k] <= A[k]$.

Replikacja Aby stworzyć upakowaną reprezentację wektora, w którym wszystkie współrzędne są tą samą b-bitową liczbą a, wystarczy pomnożyć a przez liczbę $M = 10^{b-1}10^{b-1}1 \dots 10^{b-1}1$. Jak obliczyć M w czasie stałym?[1]

Poprzednik/następnik w posortowanym upakowanym wektorze Jeśli $A[1] < A[2] < \dots < A[k]$ to przy pomocy upakowanej wersji A można wykonać poprzednika/następnika danej liczby x w A w czasie $O(1)$. W tym celu obliczamy upakowana wersję wektora składającego się z samych x-ów, $X = Mx$. Następnie wykonujemy równoległe porównania A i X, po czym przy pomocy operacji AND zerujemy wszystko poza bitami kontrolnymi. MSB otrzymanej liczby podzielone przez $b + 1$ jest indeksem następnika.

Ranga w nieposortowanym upakowanym wektorze Nawet jeśli wektor A nie jest posortowany, to możemy w czasie stałym obliczyć rangę x w A.

[1] Oczywiście można uznać, że obliczanie M w czasie stałym jest nieistotne, bo M jest stałą, elementem algorytmu. Warto jednak zauważyć, że M zależy od w. Jeśli więc nie wyliczamy M wewnątrz algorytmu, to nasz algorytm staje się „niejednorodny”.

Postępując jak poprzednio, obliczamy $A - X$, i zerujemy wszystko poza bitami kontrolnymi. Liczba jedynek w otrzymanej liczbie B jest równa $\{a \in A : x \leq a\}$. Jak ją znaleźć? Należy pomnożyć B przez $10^b10^b \dots 10^b1$ i odczytać zawartość najbardziej znaczącego „pola” wyniku.

1.2 Stapianie

W poprzednim wykładzie omówiliśmy dwie struktury danych dla problemu poprzednika, które działają szybko dla małego uniwersum (małe w). Tym razem zajmiemy się strukturą danych, która działa szybko dla dużego uniwersum – *d*rzewo stapiające (ang. fusion tree).

Zajmijmy się na początek statycznym problemem poprzednika. Gdybyśmy rozwiązując w-bitowy problem poprzednika dla zbioru n elementowego dysponowali wn-bitowym WordRAMem, to oczywiście moglibyśmy zakodować cały zbiór w jednej liczbie i odpowiadać na pytania o poprzednika w czasie stałym.

Wydaje się, że na w-bitowym WordRAMie sytuacja jest beznadziejna. Okazuje się jednak, że jeśli $n = O(w^{1/5})$, to nie wszystko jest stracone. Nie możemy oczywiście pakować całych liczb, a jedynie ich „najistotniejsze” fragmenty.

Rozważmy zbiór $S = \{x_1, \dots, x_k\}$ i odpowiadające mu drzewo trie. Niech $b_1, \dots, b_r$ będą numerami bitów, którym odpowiadają w tym drzewie rozgałęzienia. Bity te będziemy nazywać *ważnymi bitami*. Ważnych bitów jest $r \leq k$.

Dla dowolnego x (niekoniecznie z S), niech SKETCH(x) będzie ciągiem ważnych bitów x. SKETCH(x) będziemy nazywać *szkieletem* x.

Zauważmy, że $x_i < x_j$ wtw SKETCH(x_i) < SKETCH(x_j). Jeśli $k = O(n^{1/2})$, to możemy upakować wszystkie SKETCH(x_i) w jednym słowie i odpowiadać w czasie stałym na pytania o poprzednika x dla $x \in S$. Wystarczy wyszukać SKETCH(x).

Ćwiczenie 1 Czy dla $x \notin S$ także wystarczy wyszukać SKETCH(x) w zbiorze $\{\text{SKETCH}(x_i) : x \in S\}$?

Oczywiście nie. Okazuje się jednak, że znalezienie dla danego x takiego i, że SKETCH(x_i) < SKETCH(x) ≤ SKETCH(x_{i+1}) znacznie ułatwia znalezienie poprzednika x w S.

Lemat 1. *Niech $S = \{x_1, \dots, x_k\}$, gdzie $x_1 < x_2 < \dots < x_k$. Znając i takie, że SKETCH(x_i) < SKETCH(x) ≤ SKETCH(x_{i+1}) (tzn. poprzednika SKETCH(x) w SKETCH(S)) możemy znaleźć w czasie stałym PRED(x).*

Dowód. Niech pref(a, b) będzie długością wspólnego prefiksu a i b. Przyjmijmy bez straty ogólności, że $p = \text{pref}(x, \text{PRED}(x)) > \text{pref}(x, \text{SUCC}(x))$. Wtedy PRED$(x)$ ma najdłuższy wspólny prefiks z x spośród wszystkich x_i. Pokażemy, że $\max(\text{pref}(x, x_i), \text{pref}(x, x_{i+1})) = p$. Niech x_j będzie tym spośród x_i, x_{i+1}, dla którego $p' = \text{pref}(\text{SKETCH}(x_j), \text{SKETCH}(x))$ jest większe. Wtedy SKETCH(x_j) ma najdłuższy wspólny prefiks z SKETCH(x) spośród wszystkich SKETCH(S). Zauważmy, że prefiks ten musi obejmować co najmniej wszystkie ważne bity prefiksu x i PRED(x). Ale w takim razie także cały prefiks x i PRED(x), bo x_j i PRED(x) po raz pierwszy muszą się różnić na ważnym bicie.

A zatem większy z prefiksów pref(x, x_i),pref(x, x_{i+1}) jest równy prefiksowi pref$(x, \text{PRED}(x))$ lub pref$(x, \text{SUCC}(x))$. Łatwo możemy ten prefiks p znaleźć przy pomocy operacji XOR i MSB. Następnie na podstawie tego, w którą stronę odchodzi x wiemy, czy jest to pref$(x, \text{PRED}(x))$ czy pref$(x, \text{SUCC}(x))$.

Bez straty ogólności możemy przyjąć, że $p = \text{pref}(x, \text{PRED}(x))$. W jaki sposób znaleźć PRED(x)? Wystarczy znaleźć PRED(SKETCH(y)) w SKETCH(S) dla $y = \text{pref}(x, \text{PRED}(x))011\ldots111$. □

1.3 Niedokładne szkielety

Jak dotąd zignorowaliśmy jeden poważny problem. Żeby móc w czasie $O(1)$ odpowiadać na pytania PRED(x), musimy być w stanie w czasie $O(1)$ obliczać SKETCH(x), a to niestety nie jest możliwe (?). Na szczęście nie jest to potrzebne. Wystarczy obliczać liczby, które zawierają wszystkie ważne bity w prawidłowej kolejności na z góry ustalonych pozycjach, ale oprócz tego mogą one zawierać dodatkowe zera pomiędzy tymi bitami. Będziemy takie szkielety nazywać *n*iedokładnymi. Pokażemy, że można w czasie $O(1)$ obliczać niedokładne szkielety rozmiaru $O(r^4)$, gdzie r jest liczbą ważnych bitów. Dla $k = |S| = O(w^{1/5})$ możemy upakować k takich szkieletów w słowie, a zatem w tym przypadku nasz algorytm będzie działał w czasie stałym.

Lemat 2. *Niedokładne szkielety rozmiaru $O(r^4)$ można obliczać w czasie $O(1)$.*

Dowód. Obliczenie szkieletu x będzie wyglądało następująco:

$$\text{SKETCH}(x) = (x \text{ AND } \sum_{i=1}^{r} 2^{b_i}) * M \text{ AND } \sum_{i=1}^{r} 2^{c_i},$$

tzn. zerujemy nieważne bity, mnożymy przez chytrze dobraną liczbę M, po czym znów zerujemy nieważne bity, przy czym za drugim razem ważne bity znajdują się na pozycjach $c_1 < c_2 < \ldots < c_r$, gdzie $c_r - c_1 = O(r^4)$.

Pokażemy, że istnieje takie M i $\{c_i\}$, że opisany wzór poprawnie wylicza niedokładny szkielet x. Niech $M = \sum_{i=1}^{r} 2_i^m$. Wystarczy pokazać, że istnieją m_i takie, że:

- $m_i = O(w)$ dla $i \in \{1, \ldots, r\}$,
- liczby postaci $m_i + b_j$ są parami różne dla $i, j \in \{1, \ldots, r\}$,
- ciąg $c_i = m_i + b_i$ jest rosnący,
- $c_r - c_1 = O(r^4)$.

Ćwiczenie 2 Pokaż, że takie liczby faktycznie istnieją.
Wskazówka Pokaż przez indukcję, że można wybrac m_i tak aby liczby postaci $m_i + b_j$ były parami różne mod r^3. Zmodyfikuj te liczby tak, aby dodatkowo ciąg $m_i + b_i$ był rosnący. □

1.4 Drzewa stapiające

W jaki sposób wykorzystać technikę stapiania? Rozważmy B-drzewa ze stopniem rozgałęzienia $B = O(w^{1/5})$. Jesteśmy w stanie zaimplementować węzeł takiego drzewa tak, aby w czasie stałym decydował on którą gałąź należy wybrać. W takim drzewie możemy szukać poprzednika w czasie $O(\frac{\log n}{\log w})$.

Ćwiczenie 3 Co jeśli chcemy mieć strukturę dynamiczną?

Rozwiązanie Stosujemy technikę indirection. Modyfikowanie drzewa zajmuje czas $O(\log nw^{O(1)})$ i tyle właśnie wierzchołków trzymamy w drzewach na dole. To daje czas $O(\log n / \log w + \log w)$.

Ćwiczenie 4 Pokaż, że można zawsze uzyskać czas $O(\sqrt{\log n})$.

Wskazówka Wybieramy lepszą strukturę z drzew van Emde Boasa i drzew stapiających. Problem pojawia się, gdy w jest duże, ponieważ w czasach dla obu struktur mamy składnik $\log w$.

Należy zauważyć, że aktualizacja wierzchołka zajmuje czas $O(k^{O(1)})$, a nie $O(w^{O(1)})$. Dla dużych w po prostu nie wykorzystujemy całego potencjału w.

Ćwiczenie 5 Pokaż, że MSB/LSB można faktycznie znajdować w czasie stałym.

Wskazówka Sprowadź problem dla liczb w-bitowym do 2 problemów $\sqrt{w}$-bitowym (rozmiar słowa pozostaje ten sam!).