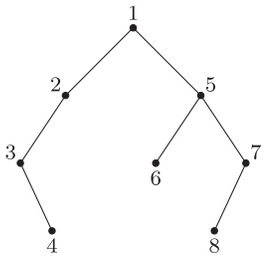
Bardzo oszczędne drzewa (I)

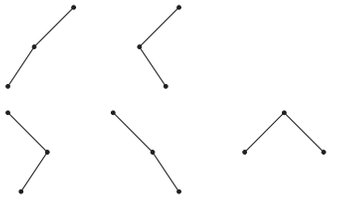
Jakub RADOSZEWSKI



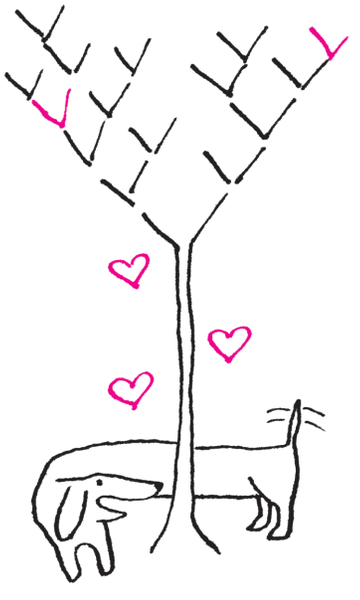
Rys. 1. Przykładowe drzewo binarne o $n = 8$ węzłach.

węzeł	1	2	3	4	5	6	7	8
lsyn	2	3	0	0	6	0	8	0
psyn	5	0	4	0	7	0	0	0
ojciec	0	1	2	3	1	5	5	7

Rys. 2. Reprezentacja drzewa z rysunku 1 za pomocą trzech tablic *lsyn*, *psyn* i *ojciec*. Liczba 0 w tablicy oznacza, że odpowiedni węzeł drzewa nie istnieje.



Rys. 3. Jest $C_3 = 5$ ukorzenionych, nieetykietowanych drzew binarnych o trzech węzłach.



Wiele struktur danych w komputerze można reprezentować w postaci drzewa binarnego, na przykład takiego jak na rysunku 1. Aby przechować takie drzewo w pamięci komputera, należy dla każdego węzła zapamiętać numer jego lewego i prawego syna oraz, jeśli to potrzebne, numer węzła będącego jego ojcem. Wystarczą nam do tego trzy tablice (patrz rys. 2).

Czy jest to najbardziej oszczędna reprezentacja drzewa binarnego? Jeśli rozważane drzewo ma n węzłów, to łączny rozmiar wszystkich tablic wynosi $3n$, a jeśli nie są nam potrzebni ojcowie węzłów, to rozmiar jest równy $2n$. Co więcej, tablice *lsyn* i *psyn* mają łącznie tylko $n-1$ niezerowych komórek. Możemy przeanalizować to jeszcze dokładniej i spojrzeć na liczbę bitów w reprezentacji. Numery węzłów są u nas liczbami całkowitymi z zakresu od 1 do n, więc do reprezentacji każdego z nich w systemie binarnym wystarczy $\lceil \log n \rceil$ cyfr. To oznacza, że cała reprezentacja wymaga z grubsza $cn \log n$ bitów pamięci, gdzie c jest stałą równą 1, 2 lub 3, zależnie od tego, które tablice przechowujemy. Kombinując dalej, można zauważyć, że do przechowywania małych numerów węzłów nie jest potrzebne aż $\lceil \log n \rceil$ bitów, jednak to spostrzeżenie nie pozwoli nam na pewno istotnie zredukować łącznej liczby bitów w reprezentacji.

A czy jest możliwe uzyskanie reprezentacji drzewa binarnego za pomocą istotnie mniej niż $n \log n$ bitów? Aby odpowiedzieć na to pytanie, warto zadać inne: ile jest różnych drzew binarnych (ukorzenionych, nieetykietowanych) o n węzłach? Oznaczmy tę liczbę przez C_n (patrz rys. 3). Spróbujmy ułożyć wzór rekurencyjny na C_{n+1}. Drzewo o $n+1$ węzłach składa się z korzenia i dwóch jego poddrzew. Oznaczmy przez i liczbę węzłów w lewym poddrzewie ($i \in \{0, \ldots, n\}$). Wówczas:

$$C_{n+1} = \sum_{i=0}^{n} C_i C_{n-i}.$$

Dodając do tego wzoru warunek początkowy $C_0 = 1$, otrzymujemy rekurencję definiującą tzw. *liczby Catalana*, o znanym wzorze ogólnym:

$$C_n = \frac{1}{n+1}\binom{2n}{n}.$$

Znajomość wartości C_n pozwala podać teoretyczne oszacowanie dolne na liczbę bitów w reprezentacji drzewa binarnego o n węzłach. Otóż żeby każde drzewo binarne dało się reprezentować za pomocą k bitów, musi zachodzić $2^k \geqslant C_n$, gdyż w przeciwnym razie pewne dwa różne drzewa miałyby tę samą reprezentację. Ponieważ współczynnik dwumianowy występujący we wzorze na C_n majoryzuje pozostałe współczynniki występujące w sumie:

$$\binom{2n}{0} + \binom{2n}{1} + \ldots + \binom{2n}{2n} = 2^{2n},$$

a $n+1$ występujące w mianowniku jest niższego rzędu niż 2^{2n}, więc możemy z niezłą dokładnością asymptotyczną przybliżyć C_n przez 2^{2n}.

Stąd wysnuwamy wniosek, że do reprezentacji drzew binarnych powinno nam wystarczyć mniej więcej $2n$ bitów. Oczywiście, taka reprezentacja istnieje. Wystarczy wszystkie n-węzłowe drzewa binarne ponumerować kolejnymi liczbami naturalnymi. Wówczas reprezentacją danego drzewa będzie jego numer, czyli liczba naturalna o co najwyżej $2n$ bitach. Taka reprezentacja, jakkolwiek niezwykle oszczędna, jest, niestety, dużo mniej wygodna niż nasza początkowa reprezentacja wykorzystująca $n \log n$ bitów. Nie pozwala ona nawigować po drzewie, tj. identyfikować węzłów drzewa i poruszać się po nich w naturalny sposób, czyli w kierunku do synów lub do ojca węzła. Okazuje się jednak, że istnieje inna, sprytna reprezentacja drzew binarnych, która wykorzystuje mniej więcej tyle samo bitów – dokładniej $2n + o(n)$ bitów, czyli więcej tylko o składnik niższego rzędu – i umożliwia łatwą nawigację po drzewie. W dalszej części artykułu przedstawimy taką właśnie *bardzo oszczędną* reprezentację.

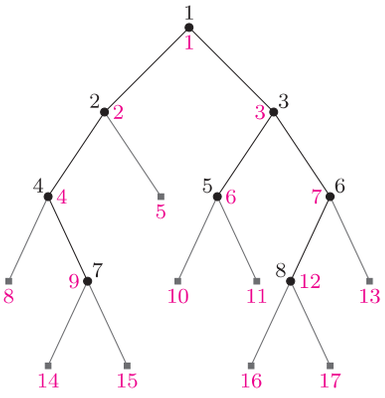
***Rank* i *select*.** Na początek wprowadzimy pomocniczą strukturę danych operującą na ciągach binarnych. Chcielibyśmy umieć obsługiwać dwa typy zapytań dotyczące takich ciągów: wyznaczanie k-tej jedynki (względnie k-tego zera) w ciągu – operacja *select*, oraz sprawdzanie, ile jedynek (względnie ile zer) znajduje się w ciągu do ustalonej pozycji – operacja *rank*. Formalnie, niech $b_1, \ldots, b_n$ będzie ustalonym ciągiem zero-jedynkowym. Wówczas dla $q \in \{0, 1\}$ oraz $k \in \{1, \ldots, n\}$ zapytania mają postać:

$$rank_q(k) = |\{j \leqslant k : b_j = q\}|,$$
$$select_q(k) = \min\{j : rank_q(j) = k\}.$$

Okazuje się, że istnieje struktura danych, która poza ciągiem b zużywa $o(n)$ bitów pamięci i pozwala odpowiadać na określone tu zapytania w czasie stałym. Odtąd będziemy używać tej struktury danych jako czarnej skrzynki (ang. *black box*).

Dla ciągu 0 1 0 0 1 0 1 0 0 1 wyniki kilku przykładowych zapytań to:

$rank_0(8) = 5$, $rank_1(8) = 3$,
$select_0(4) = 6$, $select_1(4) = 10$.



Rys. 4. Uzupełnienie drzewa binarnego z rys. 1 węzłami zewnętrznymi.

Bardzo oszczędna reprezentacja drzew. Użyjemy teraz naszej pomocniczej struktury danych do konstrukcji bardzo oszczędnej reprezentacji drzew binarnych. Zacznijmy od uzupełnienia drzewa binarnego tzw. węzłami zewnętrznymi, tak aby każdy węzeł wewnętrzny miał dokładnie dwóch synów (rys. 4). Węzły wewnętrzne drzewa ponumerujmy poziomami, a w ramach poziomów od lewej do prawej (czarne numery na rysunku 4). Ponadto w ten sam sposób ponumerujmy wszystkie węzły drzewa (kolorowe numery na rysunku 4).

Obejdźmy teraz wszystkie węzły drzewa w porządku numerów „kolorowych” i dla każdego z nich zapiszmy cyfrę 1, jeśli jest on węzłem wewnętrznym, a 0 w przeciwnym przypadku:

1 1 1 1 0 1 1 0 1 0 0 1 0 0 0 0 0

Tak otrzymany ciąg binarny wraz z powiązaną z nim strukturą danych do wykonywania operacji *rank*/*select* będzie stanowił bardzo oszczędną reprezentację drzewa, tzw. *ciąg kodowy*. Każdy węzeł uzupełnionego drzewa poza korzeniem jest synem jednego z n węzłów wewnętrznych. Stąd łączna liczba węzłów drzewa, a zarazem liczba bitów w ciągu kodowym to $2n + 1$.

Sprawdźmy teraz, na ile użyteczny jest nasz ciąg kodowy. Każdy węzeł wewnętrzny drzewa v ma dwa numery, czarny $c(v)$ i kolorowy $k(v)$. Aby przeliczyć numer czarny na kolorowy, wystarczy znaleźć $c(v)$-tą jedynkę w ciągu kodowym, czyli wykonać operację *select*:

$$k(v) = select_1(c(v)).$$

Dla naszego przykładowego drzewa mamy: $select_1(7) = 9$, a $rank_1(9) = 7$.

Przyporządkowanie odwrotne wykonujemy za pomocą operacji *rank*:

$$c(v) = rank_1(k(v)).$$

Musimy jeszcze opisać sposób poruszania się po drzewie. Jest on zaskakująco prosty:

$$lsyn(c(v)) = 2c(v), \quad psyn(c(v)) = 2c(v) + 1, \quad ojciec(k(v)) = \lfloor k(v)/2 \rfloor.$$

Przykładowo, dla węzła o numerze czarnym 7 i kolorowym 9 mamy:

$lsyn(7) = 14$, $psyn(7) = 15$,
$ojciec(9) = 4$.

Innymi słowy, lewym synem węzła o numerze czarnym $c(v)$ jest węzeł o numerze kolorowym $2c(v)$ i podobnie w przypadku prawego syna; natomiast w przypadku ojca robimy odwrotnie: dla węzła o numerze kolorowym $k(v)$ ojcem jest węzeł o numerze czarnym $\lfloor k(v)/2 \rfloor$.

Podane wzory zasługują na wyjaśnienie. Skoncentrujemy się na pierwszym z nich (dla operacji *lsyn*), pozostałe otrzymuje się analogicznie. Aby go uzasadnić, wystarczy zbadać, ile węzłów uzupełnionego drzewa występuje w porządku kolejnych poziomów przed lewym synem węzła $c(v)$. Każdy taki węzeł jest albo samym korzeniem drzewa, albo synem jednego z węzłów wewnętrznych o czarnych numerach $1, \ldots, c(v) - 1$. Zauważmy, że do tej drugiej grupy zaliczają się tak węzły wewnętrzne (w tym te o numerach czarnych $2, \ldots, c(v)$), jak i zewnętrzne. Wszystkich tych węzłów jest $2c(v) - 1$, więc rzeczywiście numerem kolorowym lewego syna węzła $c(v)$ jest $2c(v)$.

Podsumujmy to, co wiemy o naszej reprezentacji. Ma ona rozmiar $2n + o(n)$ i pozwala identyfikować węzły drzewa i przemieszczać się w górę i w dół drzewa w czasie stałym. Nie wykazaliśmy jeszcze tylko, że jest ona poprawna, czyli że różne drzewa uzyskują różne reprezentacje. To jednakże wynika z faktu, że na podstawie reprezentacji drzewa, nawigując po nim, możemy jednoznacznie odtworzyć jego kształt. Tak więc nasza reprezentacja spełnia wszystkie oczekiwane własności.

O tym, jak zaimplementować strukturę danych z operacjami *rank*/*select*, opowiemy w następnym numerze.