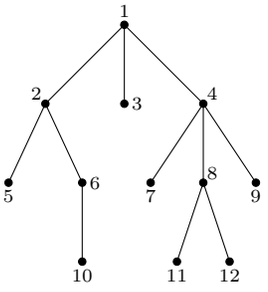
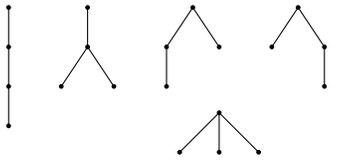
Bardzo oszczędne drzewa (II)

Jakub RADOSZEWSKI

W poprzednim numerze zdefiniowaliśmy operacje *rank* i *select*, które posłużyły nam do zbudowania bardzo oszczędnej (używającej $2n + o(n)$ bitów) reprezentacji dla n-węzłowych drzew binarnych. W drugiej części artykułu uogólniamy naszą konstrukcję i pokazujemy, jak zaimplementować kluczowe operacje *rank* i *select*.

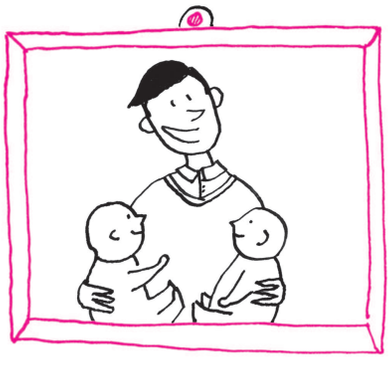


Rys. 1. Przykładowe drzewo ukorzenione o $n = 12$ węzłach. Dla węzła o numerze 2 mamy $ojciec(2) = 1$, $lsyn(2) = 5$, $pbrat(2) = 3$ i $deg(2) = 2$.



Rys. 2. Jest $T_4 = 5$ ukorzenionych, nieetykietowanych drzew o czterech węzłach.

Wzór $T_n = C_{n-1}$ można też uzasadnić, wskazując bijekcję między n-węzłowymi drzewami i $(n-1)$-węzłowymi drzewami binarnymi. Wystarczy zauważyć, że tablice *lsyn* i *pbrat*, potraktowane jako tablice *lsyn* i *psyn*, tworzą drzewo binarne, w którym korzeń ma tylko lewego syna.



Drzewa dowolne. Skoro dotychczas szło nam tak dobrze, spróbujmy pójść za ciosem i zaproponować bardzo oszczędną reprezentację drzew już niekoniecznie binarnych (ale wciąż ukorzenionych). Przykład takiego drzewa można znaleźć na rysunku 1. Tym razem będziemy nawigować po drzewie za pomocą operacji: *ojciec*, *lsyn* oraz *pbrat* (ta ostatnia polega na przejściu do najbliższego po prawej brata danego węzła). Aby operacja *pbrat* miała sens, musimy też umieć dla każdego węzła wyznaczyć jego stopień, czyli liczbę synów (operacja *deg*). Spróbujmy ustalić, jak oszczędną reprezentację mamy tu w ogóle szansę uzyskać. W tym celu, podobnie jak w przypadku binarnym, musimy stwierdzić, ile jest różnych drzew dowolnych o n węzłach. Zliczać będziemy drzewa z nienumerowanymi węzłami i w których porządek synów węzła ma znaczenie. Niech T_n oznacza liczbę takich drzew (patrz rys. 2). Jeśli przez i oznaczymy liczbę węzłów w poddrzewie skrajnie lewego syna korzenia ($1 \leqslant i \leqslant n-1$), otrzymamy następujący wzór rekurencyjny na T_n:

$$T_n = \sum_{i=1}^{n-1} T_i T_{n-i}.$$

Mamy ponadto $T_1 = 1$. Porównując ten wzór z definicją rekurencyjną liczb Catalana, otrzymujemy, że $T_n = C_{n-1}$. Stąd, podobnie jak poprzednio, w oszczędnej reprezentacji drzew dowolnych powinno nam wystarczyć $2n + o(n)$ bitów.

Przy konstrukcji ciągu kodowego znów ponumerujemy wszystkie węzły drzewa poziomami, a w ramach poziomów od lewej do prawej (rys. 1). Wypiszmy teraz w jednym ciągu stopnie wszystkich węzłów – dla powyższego przykładu będzie to:

3 2 0 3 0 1 0 2 0 0 0 0

Chcielibyśmy, żeby taki właśnie ciąg był naszym ciągiem kodowym. Niestety, nie jest to możliwe: występuje w nim n liczb, z których każda jest z zakresu od 0 do $n-1$, więc reprezentacja takiego ciągu wymagałaby rzędu $n \log n$ bitów. Aby sobie z tym poradzić, zastosujemy pozornie beznadziejny manewr: zapiszemy wszystkie stopnie *unarnie*, czyli każdy stopień zamienimy na ciąg jedynek odpowiedniej długości zakończony zerem. Ponieważ suma stopni węzłów to zaledwie $n-1$, w ten sposób uzyskamy ciąg złożony z $2n-1$ bitów, np.:

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

Ze względów technicznych wygodnie nam będzie jeszcze dodać do drzewa sztuczny korzeń, którego jedynym synem będzie faktyczny korzeń drzewa. Odpowiada to dopisaniu na początku ciągu jedynki i zera:

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

Taki ciąg kodowy o długości $2n+1$ wraz ze strukturą danych do wykonywania operacji *rank*/*select* będzie naszą bardzo oszczędną reprezentacją drzewa.

Spróbujmy uzasadnić, że podana reprezentacja umożliwia efektywne wykonywanie wszystkich potrzebnych operacji. Zauważmy przede wszystkim, że w ciągu kodowym występuje n jedynek, które w naturalny sposób odpowiadają węzłom drzewa: węzeł o numerze c reprezentuje c-ta jedynka w ciągu, będąca zarazem odpowiednią jedynką w ciągu opisującym stopień jego ojca. Każdemu węzłowi, oprócz już przydzielonego numeru czarnego, przypiszemy znów numer kolorowy. Będzie to pozycja w ciągu kodowym odpowiadającej mu jedynki. Widać natychmiast, że za pomocą operacji *rank* oraz *select* możemy bez problemu przeliczać numery czarne na kolorowe i odwrotnie.

Wiedząc teraz, gdzie w ciągu kodowym znajduje się jedynka odpowiadająca danemu węzłowi, możemy zliczyć zera występujące wcześniej w ciągu i w ten sposób wyznaczyć numer jego ojca. Prawy brat węzła będzie ni mniej, ni więcej jak sąsiednią jedynką w ciągu kodowym. Natomiast ciąg jedynek odpowiadający synom węzła możemy zidentyfikować, znajdując $(c+1)$-sze i c-te zero w ciągu, gdzie c jest numerem czarnym tego węzła. Z tego ciągu łatwo odzyskamy zarówno numer lewego syna węzła, jak i łączną liczbę jego synów, czyli stopień węzła.

Rozwiązanie zadania F 853.
Świeczka gaśnie po wyczerpaniu tlenu z powietrza zawartego pod odwrócona szklanką. Spalanie węglowodorów prowadzi do powstawania cząsteczek CO_2, H_2O i CO. Z każdej cząsteczki tlenu powstają więc jedna lub dwie cząsteczki produktów spalania, a więc w wyniku spalania tlenu wzrasta liczba cząsteczek gazu. Dodatkowo wydzielone ciepło spalania ogrzewa gaz. Oba zjawiska prowadzą do „ucieczki" części gazu spod szklanki. Gdy świeczka zgaśnie, ilość gazu już się nie zmienia, a jego stygnięcie (do temperatury otoczenia T_0) powoduje spadek ciśnienia, co dalej prowadzi do zasysania wody z talerza. Ponieważ wysokość słupa wody $h = 2\,\mathrm{cm}$ odpowiada różnicy ciśnień równej jedynie około 0,2% ciśnienia atmosferycznego, to z dobrym przybliżeniem można przyjąć, że gaz w chwili zgaśnięcia świeczki i po ostygnięciu do temperatury otoczenia znajdował się pod ciśnieniem atmosferycznym. Masa gazu podczas stygnięcia nie zmienia się, a więc stosunek temperatur równy jest stosunkowi objętości – dla szklanki w kształcie walca równego stosunkowi wysokości części wypełnionych gazem

$$\frac{T_k}{T_0} = \frac{H}{H-h} = \frac{10}{8}$$

Otrzymujemy zatem $T_k \approx 369\,\mathrm{K}$.

Dopracowanie szczegółów implementacji operacji pozostawiamy Czytelnikowi, który, wyposażony w strukturę danych *rank*/*select*, wykona to bez większego trudu.

Implementacja operacji *rank* i *select*. Przyszła pora, aby uzupełnić powyższe rozważania opisem implementacji pomocniczej struktury danych. Zajmijmy się najpierw operacją *rank*. Zauważmy, że wystarczy umieć odpowiadać na zapytania dla $q = 1$, gdyż $rank_0(k) = k - rank_1(k)$.

Nasz ciąg bitów b podzielimy na bloki: najpierw na *duże bloki* o długości $l = \log^2 n$, a w drugiej kolejności na *małe bloki* o długości $s = \frac{1}{2}\log n$. W zapisach tych $\log n$ oznacza część całkowitą logarytmu o podstawie dwójkowej z n. Zakładamy dla uproszczenia, że s jest całkowite. Bloki w podziale są rozłączne i składają się z kolejnych elementów ciągu. Ponadto, jeśli ostatni blok okaże się krótszy, uzupełniamy go sztucznymi zerami aż do pełnej długości. Zauważmy, że l jest całkowitą wielokrotnością s i małe bloki stanowią „podpodział" dużych bloków. Bloki każdego typu numerujemy od jedynki.

Dla każdego dużego bloku w tablicy R_L zapamiętamy łączną liczbę jedynek w ciągu znajdujących się przed początkiem tego bloku. Elementy tablicy R_L są mniejsze niż n, więc są liczbami złożonymi z co najwyżej $\log n + 1$ bitów. Łączny rozmiar tej tablicy jest zatem rzędu:

$$\frac{n}{l}\log n = \frac{n}{\log^2 n}\log n = O\left(\frac{n}{\log n}\right).$$

Dalej, dla każdego małego bloku w podobnej tablicy R_S zapamiętamy liczbę jedynek znajdujących się przed początkiem tego bloku, ale *w ramach* dużego bloku zawierającego rozważany mały blok. Tablica R_S ma n/s elementów, jednak każdy jej element jest mniejszy niż l. Tablica ma więc rozmiar rzędu:

$$\frac{n}{s}\log l = \frac{2n}{\log n}\log\left(\log^2 n\right) = O\left(\frac{n\log\log n}{\log n}\right).$$

Obie tablice zajmują zatem $o(n)$ bitów.

Łatwo dostrzec, w jaki sposób możemy użyć podanych tablic do odpowiedzi na zapytanie $rank_1$. Widać też, że potrzebujemy do tego jeszcze jednej informacji: o liczbie jedynek znajdujących się przed danym bitem w ramach jego małego bloku. W tym miejscu wykorzystamy własność, że *różnych co do wartości* małych bloków nie ma zbyt wiele. Dokładniej, jest tylko $2^s = 2^{\log n/2} = O(\sqrt{n})$ takich bloków i możemy je w naturalny sposób ponumerować kolejnymi liczbami naturalnymi – numerem bloku będzie liczba binarna zawarta w tym bloku. Teraz możemy dla każdego z małych bloków wyznaczyć zawczasu wszystkie wyniki. W tablicy T_P dla danego numeru małego bloku i dla każdego indeksu w ramach małego bloku zapamiętamy liczbę jedynek na pozycjach poprzedzających ten indeks (wraz z tym indeksem). Tablica ta będzie miała $O(\sqrt{n}s)$ elementów, każdy nie większy niż s, czyli jej łączny rozmiar to:

$$O(\sqrt{n}s\log s) = O(\sqrt{n}\log n\log\log n).$$

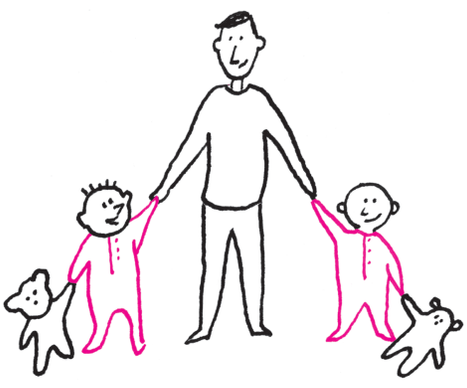
Ostatecznie wszystkie przechowywane tablice mają rozmiar $o(n)$ i pozwalają odpowiedzieć na zapytanie *rank* następująco:

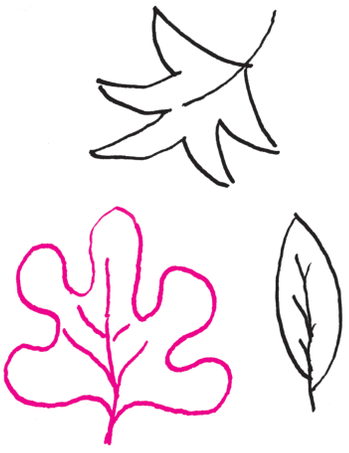
$$rank_1(k) = R_L\left[\lfloor k/l\rfloor\right] + R_S\left[\lfloor k/s\rfloor\right] + \\ + R_P[id\_bloku(b_{\lfloor k/s\rfloor - s+1}, \ldots, b_{\lfloor k/s\rfloor})][(k-1) \bmod s + 1].$$

Odpowiedź na zapytanie uzyskujemy więc w czasie stałym, przy założeniu, że standardowe operacje arytmetyczne i bitowe na liczbach nie większych niż n działają w czasie stałym. W tym celu wystarczy ciąg b reprezentować w tablicy liczb całkowitych odpowiadających paczkom kolejnych bitów ciągu (jeszcze łatwiej sobie wyobrazić, że w tablicy pamiętamy po prostu kolejne identyfikatory małych bloków).

Z operacją *select* można poradzić sobie podobnie, jednak jest to dużo bardziej skomplikowane technicznie. Tym razem bloki nie są stałej długości, lecz wszystkie zawierają tę samą liczbę jedynek. Ponadto są tu potrzebne trzy poziomy podziału na bloki – dokładny opis pomijamy. Natomiast Czytelnika Zainteresowanego pozostawiamy z następującym pytaniem: dlaczego nie dało się

Podobną do opisanej sztuczkę z zapamiętywaniem wszystkich możliwych wyników można znaleźć w artykule *Problem RMQ* w *Delcie* 11/2007.

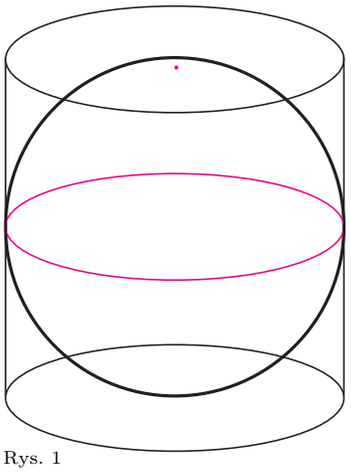




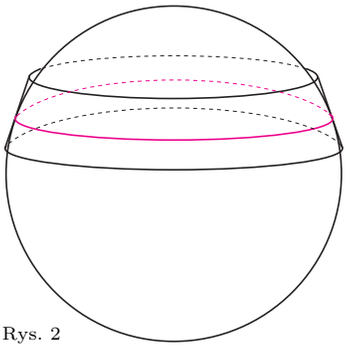
rozwiązać problemu zapytań *rank*, używając tylko jednego rozmiaru bloków – powiedzmy, $\frac{1}{2}\log n$?

Citius, altius, fortius. W tym artykule opisaliśmy bardzo oszczędne reprezentacje drzew binarnych i dowolnych pozwalające efektywnie realizować podstawowe operacje związane z nawigacją po drzewie. Za pomocą bardziej zaawansowanych narzędzi zaprojektowano inne bardzo oszczędne reprezentacje drzew (bazujące na wyrażeniach nawiasowych równoważnych drzewom), które pozwalają wykonywać całe mnóstwo operacji: wyznaczanie rozmiarów poddrzew, wysokości i głębokości węzłów, znajdowanie najniższych wspólnych przodków (LCA) par węzłów i przodków węzłów na określonej głębokości, zliczanie liści w poddrzewach itd. Co więcej, z podanych tutaj pomysłów rozwinęła się cała dziedzina badań zajmująca się bardzo oszczędnymi strukturami danych. Więcej informacji na ten temat Czytelnik znajdzie w Internecie pod hasłem *succinct data structures*.

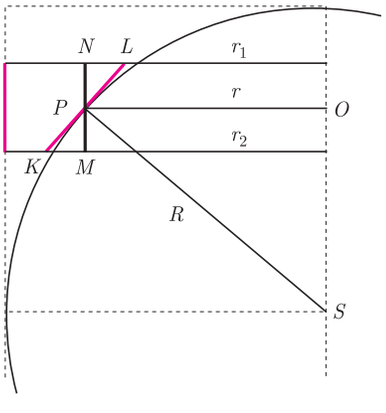
Piłka w puszce



Rys. 1



Rys. 2



Rys. 3

Piłki tenisowe na ogół pakowane są w rurkę po kilka sztuk. Wyobraźmy sobie piłki tak cenne, że pakowane są każda oddzielnie. Takie opakowanie to z matematycznego punktu widzenia walec. Piłka styka się z jego powierzchnią boczną wzdłuż okręgu, a przekrój osiowy tego walca jest kwadratem (rys. 1). Okazuje się, że tak zapuszkowana piłka ma powierzchnię równą powierzchni bocznej swojej puszki, czyli

powierzchnia sfery jest równa powierzchni bocznej opisanego na niej walca.

Jeszcze ciekawszy od samego faktu jest jego starożytny dowód, pochodzący od Archimedesa. Zapoczątkował on bardzo popularny w matematycznej praktyce obyczaj, że gdy chcemy coś udowodnić, to dowodzimy czegoś zupełnie innego.

W tym przypadku udowodnimy, że jeśli wytniemy z piłki (czyli sfery) plasterek płaszczyznami równoległymi do denek puszki (czyli podstaw walca) i w połowie jego wysokości opiszemy na nim płaską obręcz (czyli stożek ścięty, rys. 2), to jej powierzchnia będzie taka sama, jak powierzchnia wyciętego przez te płaszczyzny fragmentu puszki (czyli powierzchnia boczna walca, tylko tym razem niskiego). Nie narysowałem go, bo nic nie byłoby wtedy widać.

Ale gdy narysujemy przekrój opisanej sytuacji płaszczyzną przechodzącą przez środek piłki i prostopadłą do denek puszki, to wszystko stanie się widoczne (rys. 3). Kolorowe kreski to fragment puszki (oznaczmy jego wysokość przez h) i fragment obręczy ($KL =: l$). Narysujmy przez punkt P styczności obręczy i piłki odcinek MN – przesunięty fragment puszki oraz odcinek PS łączący P ze środkiem piłki i odcinek PO, gdzie O jest rzutem prostokątnym P na oś symetrii puszki.

Ponieważ $PL \perp PS$ i $PN \perp PO$, więc trójkąty PLN i PSO są podobne, co daje

$$\frac{\frac{h}{2}}{\frac{l}{2}} = \frac{PN}{PL} = \frac{PO}{PS} = \frac{r}{R}, \text{ czyli } hR = lr, \text{ zatem } 2\pi hR = 2\pi lr = \pi l(r_1 + r_2).$$

Znane wzory na pole powierzchni bocznej walca i stożka ściętego przekonują nas, że dowiedliśmy tego, co chcieliśmy, ale jak to się ma do wyróżnionej kolorem prawidłowości?

Zauważmy, że gdy podzielimy piłkę i puszkę na plasterki, to suma pól obręczy opisanych na plasterkach złoży się na całą powierzchnię boczną puszki, czyli $2\pi \cdot 2R \cdot R = 4\pi R^2$. Dalej Archimedes pisze tak: gdy będziemy rozpatrywali coraz węższe plasterki, to otrzymywane obręcze będą w sumie coraz podobniejsze do powierzchni piłki, aż w końcu (my mówimy: w granicy) będą z tą powierzchnią identyczne. A ponieważ dla każdego podziału suma powierzchni obręczy będzie równa powierzchni bocznej puszki, więc tak będzie i na końcu.

Czy zgodzilibyśmy się na takie rozumowanie?

M.K.