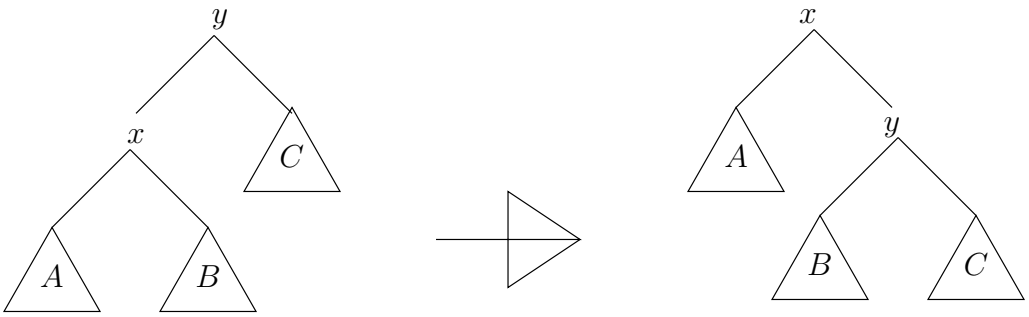
1 Drzewa wyszukiwań binarnych

W tym wykładzie powracamy do problemów słownika i przynależności, będziemy jednak pracować w innym modelu: porównaniowym. Oznacza to, że informacje na temat kluczy przechowywanych w słowniku możemy czerpać jedynie wykonując operację porównania dwóch kluczy. Przypomnijmy, że w takim modelu pesymistyczny czas wyszukiwania elementu w zbiorze n elementów wynosi $\Omega(\log n)$.

Drzewa wyszukiwań binarnych (BST od *Binary Search Trees*) to drzewa binarne, które w swoich węzłach przechowują klucze i to w taki sposób, że dla dowolnego węzła x klucze węzłów w lewym poddrzewie x są mniejsze od klucza x, natomiast klucze węzłów w prawym poddrzewie x są większe od klucza x. Taki porządek kluczy nazywamy *symetrycznym*. Zakładamy, że przy wykonaniu dowolnej operacji dany jest wskaźnik do korzenia drzewa. Zakładamy, że przy wyszukiwaniu elementu w drzewie można korzystać z następujących operacji (przyjmujemy, że zajmują one czas stały):

- przesunięcie wskaźnika do lewego syna, prawego syna lub ojca węzła na który wskazuje aktualny wskaźnik
- rotacja – tzn. zamiana węzła x z jego ojcem y oraz zamiana ich poddrzew tak, żeby zachować porządek symetryczny. Operacja rotacji występuje w dwóch odmianach: lewa (gdy x jest lewym synem y) oraz prawa (gdy prawym). Na rysunku 1 przedstawiono lewą rotację.



Rysunek 1: Lewa rotacja

Przy realizacji pozostałych operacji (wstawianie, usuwanie, łączenie drzew) możemy dodatkowo korzystać z operacji tworzenia nowego węzła i operacji zmiany lewego lub prawego syna.

Jest jasne, że czas wyszukiwania elementu jest proporcjonalny do długości ścieżki od korzenia drzewa do węzła przechowującego ten element. Implikuje to, że pesymistyczny czas wyszukiwania elementu w dowolnym drzewie BST wynosi $\Omega(\log n)$, gdyż takie jest ograniczenie dolne na długość najdłuższej ścieżki w drzewie binarnym. Oczywiście dla danego (statycznego) zbioru kluczy łatwo zbudować drzewo BST, w którym wszystkie ścieżki będą

długości $\lceil \log(n+1) \rceil$: jako korzeń wstawiamy medianę zbioru, oraz rekurencyjnie tworzymy lewe drzewo spośród elementów mniejszych od mediany i prawe z większych. Takie drzewo jest *wyważone*: dla dowolnego wielkość lewego i prawego poddrzewa różni się o nie więcej niż 1. Podobny efekt można uzyskać nawet w sytuacji dynamicznej (gdy umożliwiamy operacje wstawiania i usuwania elementów). Na przykład drzewa AVL utrzymują dla dowolnego węzła różnicę wysokości lewego i prawego poddrzewa nie większą niż 1. Można pokazać, że w takim drzewie najdłuższa ścieżka ma długość $O(\log n)$. W inny sposób ideę utrzymywania ścieżek długości logarytmicznej realizują drzewa czerwono-czarne (opisane w książce Cormen, Leiserson, Rivest „Wprowadzenie do algorytmów"). Obie te metody wymagają jednak przechowywnia w węzłach drzewa dodatkowych informacji: wyważenia drzewa dla drzew AVL (-1 gdy lewe drzewo głębsze o 1 od prawego, 1 gdy odwrotnie, 0 gry równe), lub kolor węzła dla drzew czerwono-czarnych. Są one również dosyć skomplikowane w implemantacji. Za chwilę poznamy inny rodzaj drzew BST, tzw. drzewa splay. Są one proste w implementacji, nie wymagają przechowywania dodatkowych informacji, a złożoność *zamortyzowana* standardowych operacji jest $O(\log n)$.

2 Opis drzew splay

Drzewa splay opisali w swojej słynnej płacy Sleator i Tarjan [2]. Drzewa splay nie mają żadnej specjalnej struktury (oprócz tego, że są to drzewa BST), tzn. nie jest utrzymywany żaden niezmiennik dotyczący ich kształtu etc. Drzewa splay modyfikowane są jedynie za pomocą operacji splay(x). Po wyszukaniu elementu (w standardowy sposób) zawsze wykonywana jest operacja splay na tym elemencie. Jeśli wyszukiwanie zakończyło się niepowodzeniem wykonujemy splay na ostatnim odwiedzonym węźle. Widać, że bez straty ogólności możemy w dalszym ciagu analizować co dzieje się w sytaucji gdy mamy tylko wyszukiwania zakończone sukcesem (tzw. operacje *dostępu*).

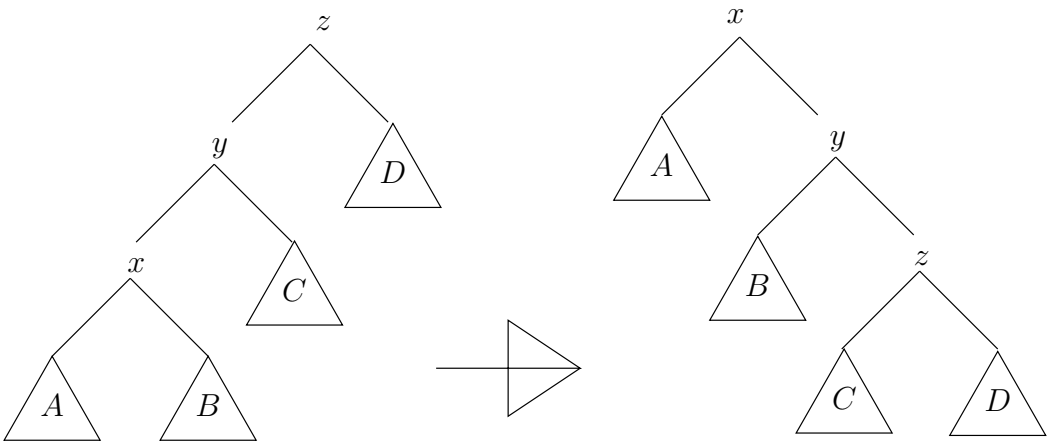
Operacja splay(x)

Operacja splay(x), gdzie x jest węzłem drzewa polega na wykonywaniu, dopóki x nie jest korzeniem, wykonuje następujący krok:

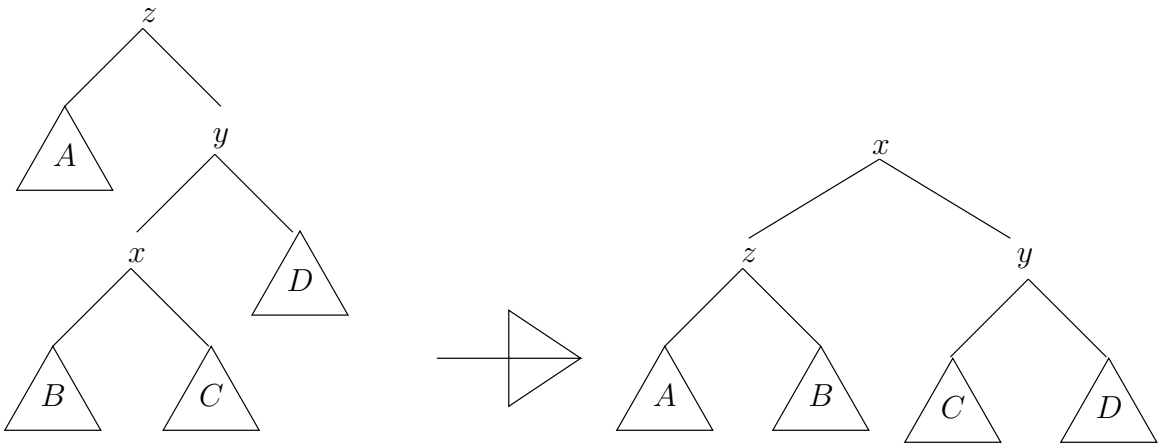
(zyg:) Jeśli $p(x)$ – ojciec węzła x – jest korzeniem, wykonaj pojedyńczą rotację (rysunek 1).

(zyg-zyg:) Jeśli $p(x)$ nie jest korzeniem oraz zarówno x jak i $p(x)$ są lewymi synami swoich ojców (lub obaj prawymi), wykonaj rotację $p(x)$ z ojcem $p(x)$, a następnie rotację x z $p(x)$. (Patrz rysunek 2).

(zyg-zak:) Jeśli $p(x)$ nie jest korzeniem oraz x jest lewym synem a $p(x)$ prawym (lub vice-versa), wykonaj rotację x z $p(x)$, a następnie rotację x z nowym ojcem x. (Patrz rysunek 3).



Rysunek 2: Przypadek zyg-zyg



Rysunek 3: Przypadek zyg-zak

Lemat o dostępie

Widzimy, że złożoność czasowa operacji splay(x) jest taka sama jak złożoność czasowa wyszukania węzła x w drzewie. Choć jedna operacja splay może zająć nawet czas liniowy (np. gdy drzewo jest zdegenerowane do linii), okazuje się że zamortyzowany czas operacji splay w ciągu $\Omega(n)$ operacji jest logarytmiczny. Pokażemy teraz tą, a także wiele innych własności drzew typu splay. Przyjmijmy, że każdy węzeł x ma przypisaną wagę $w(x)$, którą określimy w zależności od potrzeb. Sumę wag wszystkich węzłów będziemy oznaczać przez W. *Rozmiar* $s(x)$ węzła x zdefiniujmy jako sumę wag węzłów w drzewie o korzeniu w x. Natomiast *rzędem* węzła x będziemy nazywać liczbę $r(x) = \log s(x)$. W naszej analizie skorzystamy ze standardowej metody potencjału (patrz Cormen), który określimy jako sumę rzędów wszystkich węzłów drzewa.

Przypomnijmy szybko metodę potencjału: koszt zamortyzowany operacji określamy jako

$$a = t + \Phi' - \Phi$$

gdzie t jest rzeczywistym czasem wykonania operacji, Φ' wartością potencjału po wykonaniu operacji, a Φ wartością potencjału przed wykonaniem operacji. Intuicyjnie, potencjał płaci za długotrwałe operacje. A czym płaci? Ano, jednostkami czasu odłożonymi nań podczas krótkotrwałych operacji. Rzeczywisty czas wykonania ciągu m operacji wynosi wtedy

$$\sum_{j=1}^{m} t_j = \sum_{j=1}^{m} (a_j + \Phi_{j-1} - \Phi_j) = \sum_{j=1}^{m} a_j + \Phi_0 - \Phi_m,$$

gdzie a_j i t_j to zamortyzowany i rzeczywisty czas operacji j-tej, a Φ_k to potencjał po operacji k-tej. Zauważmy, że oznacza to, że jeśli $\Phi_0 - \Phi_m = O(\sum_{j=1}^{m} a_j)$ to całkowity czas rzeczywisty jest asymptotycznie taki sam jak suma czasów zamortyzowanych. W naszym przypadku, ponieważ $w(x) \leq r(x) \leq \log W$, to $\Phi_0 - \Phi_m \leq \sum \log W - \sum \log_w(x_i) = \sum \log(W/w(x_i))$.

Lemat 1 (Lemat o dostępie)**.** *Zamortyzowany czas operacji splay(x) w drzewie o korzeniu t wynosi co najwyżej* $3(r(t) - r(x)) + 1 = O(3\log(s(t)/s(x)))$.

Dowód. Policzmy zamortyzowany czas pojedynczego kroku operacji splay. Pokażemy, że w kroku typu „zyg" wynosi on $1 + 3(r'(x) - r(x))$, a w pozostałych krokach $3(r'(x) - r(x))$. Ponieważ „zyg" wykona się co najwyżej raz całkowity czas zamortyzowany dostaniemy taki jak w lemacie (suma teleskopowa). Niech s,s',r,r' oznaczają funkcje rozmiaru i rzędu przed i po wykonaniu kroku.

W przypadku „zyg" czas zamortyzowany wynosi:

$$\begin{aligned} & 1 + r'(x) + r'(y) - r(x) - r(y) & \\ \leq\ & 1 + r'(x) - r(x) & \text{bo } r(y) \geq r'(y) \\ \leq\ & 1 + 3(r'(x) - r(x)) & \text{bo } r'(x) \geq r(x). \end{aligned}$$

W przypadku „zyg-zyg" czas zamortyzowany wynosi:

$$\begin{aligned} & 2 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z) & \\ \leq\ & 2 + r'(y) + r'(z) - r(x) - r(y) & \text{bo } r'(x) = r(z) \\ \leq\ & 2 + r'(x) + r'(z) - 2r(x) & \text{bo } r'(y) \leq r'(x) \text{ i } r(y) \geq r(x). \end{aligned}$$

Teraz wystarczy wykazać, że $2 + r'(x) + r'(z) - 2r(x) \leq 3(r'(x) - r(x))$. To jest równoważne $r(x) + r'(z) - 2r'(x) \leq -2$ i dalej $\log(s(x)/s'(x)) + \log(s'(z)/s'(x)) \leq -2$, a to już łatwo widać (jakie maksimum ma $\log x + \log(1 - x)$ w przedziale $[0, 1]$?).

W przypadku „zyg-zak" czas zamortyzowany wynosi:

$$\begin{aligned} & 2 + r'(x) + r'(y) + r'(z) - r(x) - r(y) - r(z) & \\ \leq\ & 2 + r'(y) + r'(z) - 2r(x) & \text{bo } r'(x) = r(z) \\ \leq\ & 2 + r'(x) + r'(z) - 2r(x) & \text{bo } r'(y) \leq r'(x) \text{ i } r(y) \geq r(x). \end{aligned}$$

Podobnie jak powyżej pokazujemy, że ostatnie wyrażenie można oszacować przez $3(r'(x) - r(x)$. □

Widać, że złożoność zamortyzowana ciągu operacji dostępu jest równa złożoności zamortyzowanej ciagu odpowiednich operacji splay.

Twierdzenie 1 (o zrównoważeniu)**.** *Całkowity czas m operacji dostępu w drzewie splay jest $O((m+n)\log n)$.*

Dowód. Wystarczy dobrać wagi. Jak? – Ćwiczenie. □

3 Własności drzew splay

3.1 Statyczna optymalność

Załóżmy, że dla danego ciągu dostępów chcemy zbudować statyczne drzewo BST, żeby zminimalizować całkowity czas wyszukiwania. Można to oczywiście zrobić przez programowanie dynamiczne, w czasie $O(n^3)$ – wystarczy dla wszystkich $O(n^2)$ przedziałów $[i,j]$ wyznaczyć optymalny korzeń $k \in [i,j]$ (zakładamy dla uproszczenia że w drzewie są elementy $1 \dots n$). Knuth [1] pokazał, że można to nawet zrobić w czasie $O(n^2)$. Wskazówka: optymalny korzeń dla $[i,j]$ jest pomiędzy korzeniem dla $[i,j-1]$ i korzeniem dla $[i+1,j]$.

Teraz mała powtórka z teorii informacji. *Bezprefiksowy kod k-arny* to funkcja $\phi : S \to \{0,\dots,k-1\}$ taka, że dla dowolnych $s,p \in S$ jeśli $s \neq p$ to $\phi(s)$ nie jest prefiksem $\phi(p)$.

Zbiór S i rozkład prawdopodobieństwa $p : S \to [0,1]$, $\sum_{s\in S} p(s) = 1$ tworzą *źródło*. *Entropią* źródła (S,p) nazywamy liczbę:

$$H_k(S,p) = \sum_{s\in S} p(S) \cdot \log_k \frac{1}{p(s)}.$$

Oznaczamy zwykle $H(S,p) = H_2(S,p)$. Dla danego kodu ϕ niech $L(\phi)$ będzie średnią długością litery w kodzie, tzn. $L(\phi) = \sum_{s\in S} p(s)|\phi(s)|$. Jeśli za pomocą kodu ϕ kodujemy jakiś tekst długości m i jako $p(x_i)$ przyjmiemy f_i/m, gdzie f_i jest liczbą wystąpień litery x_i w tekście, to zakodowany tekst będzie miał $m \cdot L(\phi)$ bitów (lub znaków $0 \dots k-1$ dla $k > 2$). Jeśli natomiast litery pojawiają się losowo z prawdopodobieństwem p, to $L(\phi)$ jest oczekiwaną średnią długością litery.

Standardowe twierdzenie teorii informacji (Shannona?) mówi, że $L(\phi) \geq H_k(S,p)$.

Jaki to ma związek z drzewami BST? Powróćmy do problemu optymalnego (statycznego) drzewa BST. Zauważmy, że drzewo BST możemy traktować jak bezprefiksowy kod tetrarny (3-arny): do każdego wierzchołka wewnętrznego dodajemy nowego syna – liść, oraz wszystkie klucze są przechowywane w liściach. Wtedy średni czas dostępu do elementu jest równy średniej długości litery w odpowiadającym kodzie.

Wniosek 1. *Koszt obsługi ciągu m dostępów do zbioru kluczy S przez dowolne statyczne drzewo BST jest ograniczony z dołu przez $m \cdot H(S,p)/\log 3$, gdzie $p(x)$ jest ilorazem liczby wystąpień elemementu x i liczby m.*

Twierdzenie 2 (o statycznej optymalności)**.** *Niech $q(i)$ będzie liczbą dostępów do elementu i. (Zakładamy, że $f(i) > 0$ dla każdego i). Oznaczmy $p(i) = q(i)/m$. Całkowity czas m operacji dostępu w drzewie splay jest $O(m \sum_{i=1}^{n} p(i) \log \frac{1}{p(i)})$.*

Dowód. Wystarczy dobrać wagi. Jak? – Ćwiczenie. □

3.2 Inne własności

Załóżmy (bez straty ogólności), że drzewo przechowuje elementy $1, \ldots, n$. Niech $x_1, \ldots, x_m$ będzie ciągiem dostępów.

Twierdzenie 3 (o statycznym palcu)**.** *Dla dowolnego ustalonego elemenetu f całkowity czas m operacji dostępu w drzewie splay jest $O(n \log n + \sum_{j=1}^{m} \log |x_j - f|)$.*

Dowód. Wystarczy dobrać wagi. Weźmy $w(i) = \frac{1}{(i-f)^2}$. Ćwiczenie: dokończ dowód. □

Powyższe twierdzenie oznacza, że (dla dostatecznie długiego ciągu dostępów) możemy wskazać palcem dowolny element f i operacje dostępu do elementów bliskich f będą tanie.

Niech $t_i(y)$ będzie liczbą różych elementów (włącznie z y), które wystąpiły w ciągu dostępów od elementu $x_j = y$ (lub od x_1 gdy nie było takiego x_j), dla $j < i$ takiego że $x_k \neq y$ dla $j < k < i$.

Twierdzenie 4 (o zbiorze roboczym)**.** *Całkowity czas m operacji dostępu w drzewie splay jest $O(n \log n + m + \sum_{j=1}^{m} \log t_i(x_i))$.*

Dowód. Wystarczy dobrać wagi. Ale teraz wagi będą się zmieniać w czasie! W chwili i weźmy $w(y) = \frac{1}{(t_i(y))^2}$. Ponownie $s(root) = O(1)$ (dlaczego?). Dostajemy koszt zamortyzowany dostępu do x_i równy $O(\log \frac{O(1)}{t_i(x_i)^{-2}}) = O(\log t_i(x_i))$. Oczywiście to ograniczenie dostajemy przy założeniu, że nowy potencjał jest liczony według starych wag. Aby się dowiedzieć co by się stało, gdybyśmy liczyli według nowych wag załóżmy, że po operacji dostępu wykonywana jest operacja zmiany wag i policzmy jej koszt zamortyzowany. Po wykonaniu operacji, $t_i(x_i)$ zmienia się na 1, a co za tym idzie $w(x_i) = 1$ (czyli byc może rośnie), natomiast wartości t_i pozostałych elementów rosną o 1, czyli ich wagi się zmniejszają. A więc potencjał wzrasta o $O(1)$, czyli „prawdziwy" koszt zamortyzowany dostępu jest $O(1 + \log t_i(x_i))$. □

Powyższe twierdzenie oznacza, że jeśli dostępy koncentrują się w małym podzbiorze całego zbioru kluczy, to czas zamortyzowany dostępu jest bardzo mały – logarytmiczny względem wielkości podzbioru.

Twierdzenie 5 (o dynamicznym palcu)**.** *Zamortyzowany czas dostępu do elementu x_i wynosi w drzewie splay $O(\log(1 + |x_i - x_{i-1}|))$.*

Dowód powyższego twierdzenia jest bardzo skomplikowany. Udowodnił je Cole w roku 2000.

Hipoteza 1 (Traversal Conjecture)**.** *Jeśli mamy dwa drzewa T_1 i T_2 o tych samych kluczach oraz wykonujemy ciag dostępów do T_1 który odpowiada przejściu T_2 w kolejności preorder oraz T_1 jest drzewem splay to ciąg dostępów wykona się w czasie $O(n)$.*

Hipoteza 2 (O dynamicznej optymalności)**.** *Niech OPT będzie optymalnym kosztem wykonania ciągu dostępów $x_1, \ldots, x_m$ w drzewie BST (być może zmieniającym się w czasie). Wtedy drzewo splay wykona ten ciąg dostępów w czasie $O(n + OPT)$.*

Strukturę danych, która wykona taki ciąg dostępów w czasie $O(n + \alpha OPT)$ nazywamy α-kompetytywną. Widać, że drzewa splay są $O(\log n)$-kompetytywne.

Ćwiczenie: jakie implikacje zachodzą między udowodnionymi twierdzeniami?

Ćwiczenie: jak za pomocą operacji splay wykonań następujące operacje:

- Join (T_1, T_2) – złączenie dwóch drzew BST, zakładamy, że dowolny klucz w T_1 jest mniejszy niż dowolny klucz w T_2.
- Split (T, x) – podział na dwa drzewa BST: zawierające elementy $> x$ i $\leq x$.
- Insert (T, x) – wiadomo.
- Delete (T, x) – wiadomo.

3.3 Twierdzenie o skanowaniu

Pokażemy teraz szczególny przypadek hipotezy o dynamicznej optymalności: twierdzenie o skanowaniu. Zobaczymy, że nawet tak szczególny przypadek jest nietrywialny. Jest to rezultat Tarjana [3] z 1985 roku.

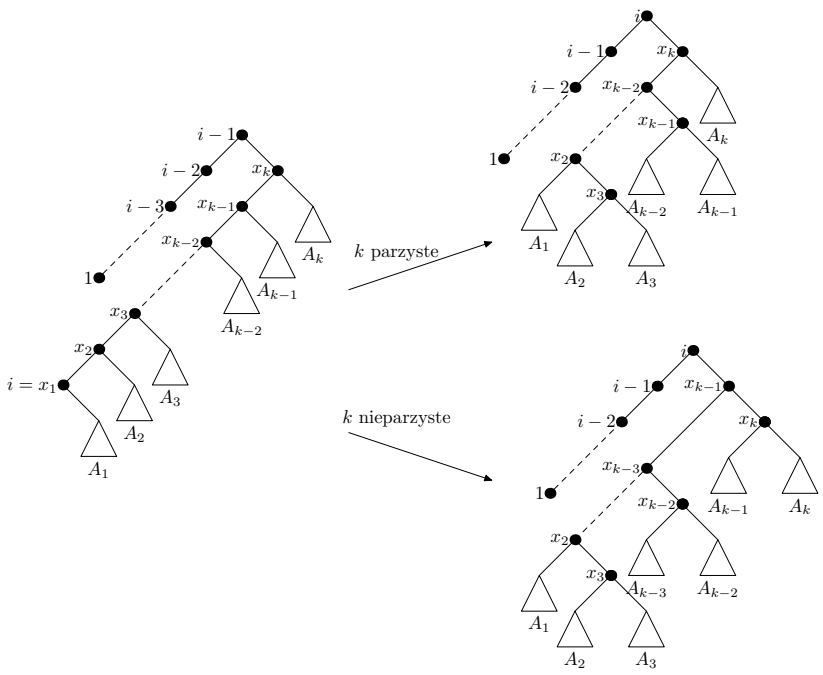
Twierdzenie 6 (o skanowaniu)**.** *Operacji dostępu wykonywane na kolejnych elementach zbioru n kluczy zajmują w drzewie splay czas $O(n)$.*

Dowód. Bez straty ogólności przyjmujemy, że ciąg dostępów do kluczy to $1, 2, 3, \ldots, n$. Na rysunku 4 pokazano efekt operacja splay przy dostępie do elementu i. Widzimy, że skrajnie lewa ścieżka o początku w korzeniu zawsze zawiera elementy, do których wykonywany był już dostęp. Tą ścieżkę będziemy ignorować. Po $i - 1$ dostepach, element i jest na końcu skrajnie lewej ścieżki o początku w prawym synu korzenia. Tą ścieżkę będziemy nazywać *ścieżką dostępu*. Operację splay do ostatniego węzła na ścieżce dostępu będziemy określać jako p-splay (dla naszego ciągu dostępów wykonywane są tylko takie operacje splay).

Powiemy, że klucz a jest *prawym przodkiem* b jeśli a jest przodkiem b w drzewie oraz $a > b$. Oznaczmy przez $A(x)$ zbiór prawych przodków x. Przez $l(x)$ będziemy oznaczać liczbę prawych przodków x na ścieżce dostępu.

Fakt 1. *Koszt operacji p-splay w węźle x jest $\leq |A(x)|$*

Fakt 2. *Po operacji p-splay w węźle x, dla dowolnego $y > x$ mamy $|A'(y)| \leq |A(y)| - \lceil l(y)/2 \rceil + 1$ oraz $l'(y) \leq \lfloor l(y)/2 \rfloor$ Dowód:* Przypadki: $x = x_{2i}$, $x = x_{2i-1}$, $x \in A_{2i}$,



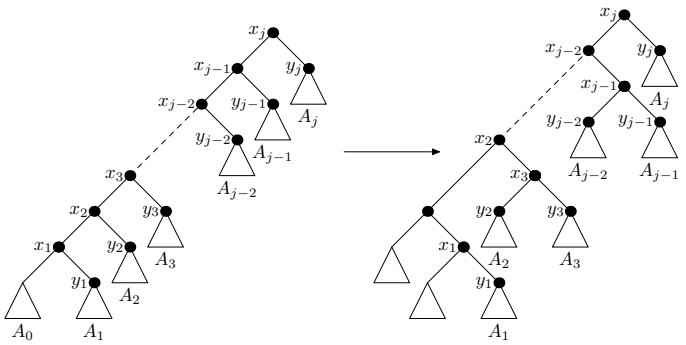
Rysunek 4: Operacja splay przy dostępie do elementu i

$x \in A_{2i-1}$.

Idea dowodu jest następująca. Dowodzimy metodą księgowania. Początkowo, każdy węzeł drzewa ma pewną liczbę kredytów (ograniczoną przez stałą). Tymi kredytami będziemy płacić za wykonanie operacji p-splay (1 kredyt płaci za 1 rotację). Jeśli pokażemy, że z każdego wierzchołka nie weźmiemy więcej kredytów niż mu przydzieliliśmy, da to liniową całkowitą liczbę rotacji.

Które węzły będą płacić za operację splay? Będą to węzły usunięte ze ścieżki dostępu. Musimy być jednak ostrożni, ponieważ węzeł może dowolnie wiele razy wracać na ścieżkę dostępu (i mogłoby mu zabraknąć kredytów). Dlatego płacić będą tylko niektóre węzły ze ścieżki dostępu. Powiemy, że węzeł x jest *głęboki*, gdy $|A(x)| \geq 16$. Pozostałe węzły są *płytkie*. Możemy założyć, że wśród węzłów usuwanych ze ścieżki dostepu co najmniej 1 jest głęboki, bo inaczej ścieżka dostępu ma długość $O(1)$ i koszt operacji p-splay jest stały (nie bierzemy żadnych kredytów). Z faktu 2 wnioskujemy, że $|A(x)|$ nie maleje, a więc podczas dowolny węzeł w swoim życiu co najwyżej raz zmienia swój status z głębokiego na płytki. Pokażemy, że podczas operacji p-splay pewna stała frakcja głębokich węzłów na ścieżce dostępu jest z niej usuwana i powraca w przyszłości jako węzły płytkie. Te węzły zapłacą za tą operację p-splay.

Rozważmy pojedynczą operację p-splay. Wierzchołki usunięte ze ścieżki dostępu są prawymi synami kolejnych węzłów na nowej ścieżce dostępu. Oznaczmy je $y_1, \ldots, y_j$. (Patrz rys. 5. Niech $f(y_i)$ będzie wartością $|A(y_i)|$ w chwili, gdy y_i powróci na ścieżkę dostępu. Niech $F(j)$ będzie najmniejszą liczbą taką że $\sum_{i=1}^{j} f(y_i) \leq F(j)$. Niech $G(j) =$



Rysunek 5: Efekt operacji p-splay na węzłach ścieżki dostępu x_i i ich prawych synach y_i (przypadek parzysty)

$\max_{1\le i\le j} F(i)$. ($G(j)$ definiujemy tak tylko po to, żeby mieć funkcję niemalejącą).

Fakt 3. *Dla $j \ge 3$ mamy $G(j) \le \frac{5}{4}G(\lfloor j/2\rfloor + 1) + 5(\lfloor j/2\rfloor + 1)$*

Dowód: Spójrzmy na rysunek 5. Widzimy, że prawie wszystkie węzły y_i parują się: y_{i-1} i y_i są synami (lewym, prawym) węzła x_i. Pozostaną jego synami co najmniej do momentu gdy x_i powróci na ścieżkę dostępu. W tym samym momencie powróci też y_{i-1}. Stąd $f(y_{i-1}) = f(x_i) + 1$. Zauważmy, że co najmniej 2 operacje p-splay pojawią się, zanim y_i powróci na ścieżkę dostępu (p-splay na y_{i-1} lub jego potomku oraz p-splay na x_i). Stąd, i z faktu 2 (użytego dwukrotnie) $f(y_i) \le f(x_i)/4 + 3$. Razem dostajemy $f(y_{i-1}) + f(y_i) \le \frac{5}{4}f(x_i) + 4$. Dzięki temu możemy oszacować $\sum f(y_i)$ przez sumę $f(x_l)$ po tych x_l, które po operacji p-splay stają się prawymi synami węzłów na ścieżce dostępu, a to da nam zależność rekurencyjną. Pozostaje jeszcze rozważyć wierzchołki y_i, które nie zostały sparowane, czyli y_1 i (tylko w przypadku parzystym) y_j. Z Faktu 2 $f(y_1) \le \lfloor j/2\rfloor + 1$, natomiast w przypadku parzystym y_1 pozostaje prawym synem korzenia, czyli $f(y_1) = 0$.

Niech $z_1, \ldots, z_q$ będą tymi spośród węzłów $x_1, \ldots, x_j$, które po operacji p-splay zostały prawymi synami węzłów na ścieżce dostępu. Mamy $q \le \lfloor j/2\rfloor + 1$. Dostajemy:

$$\sum_{i=1}^{j} f(y_i) \le \lfloor j/2\rfloor + 1 + \sum_{i=1}^{q}(\frac{5}{4}f(z_i) + 4) \le \frac{5}{4}G(\lfloor j/2\rfloor + 1) + 5(\lfloor j/2\rfloor + 1).$$

Stąd

$$G(j) \le \frac{5}{4}G(\lfloor j/2\rfloor + 1) + 5(\lfloor j/2\rfloor + 1).$$

(Koniec dowodu faktu 3).

Widzimy, że $G(j)$ jest liniowa. Łatwo dostać przez indukcję ograczenie $G(j) \le 8j$: Dla $j = 1$ mamy $G(j) = 0$, a dla $j = 2$ mamy $G(j) \le 0 + 1 = 1$, a krok indukcyjny z Faktu 3.

Teraz policzmy, ile spośród węzłów $y_1, \ldots, y_j$ wraca na ścieżkę dostępu jako węzły głębokie. Oznaczmy szukaną liczbę przez Deep. Przypomnijmy, że jeśli węzeł x wraca jako

głęboki, to $f(x) \geq 16$. Stąd $\sum_{i=1}^{j} f(y_i) \geq 16\text{Deep}$. Z drugiej strony $\sum_{i=1}^{j} f(y_i) \leq G(j) \leq 8j \leq 8(\lfloor k/2 \rfloor + 1) \leq 4k + 8$. To razem implikuje $\text{Deep} \leq \frac{k}{4} + \frac{1}{2}$.

Teraz policzmy, ile spośród głębokich węzłów na ścieżce dostępu, które zostają usunięte ze ściezki, powraca jako węzły płytkie. Węzłów głębokich na ścieżce jest $k-16$. Usuwanych ze ścieżki jest $\geq \frac{k-16}{2} - 1$. Spośród nich powraca jako węzły płytkie $\geq \frac{k-16}{2} - 1 - \text{Deep}$ węzłów, czyli co najmniej $\frac{k}{4} - 10$. A więc każdemu węzłowi wystarczy przydzielić 4 kredyty. Podczas operacji p-splay, węzły głębokie usuwane ze ścieżki dostępu, które powrócą jako płytkie oddają swoje kredyty, których wystarcza na opłacenie $k - 40$ operacji typu splay. Dostajemy czas zamortyzowany operacji p-splay co najwyżej 44 rotacje. □

Uwaga: W oryginalnej pracy Tarjana stała zamiast 44 wynosi 9.8 (dowód jest analogiczny). W 2003 roku ukazała się praca Amr Elmasry'ego zawierająca dowód ze stałą 4.5 (nie jest istotnie prostszy od powyższego).

Literatura

[1] D. E. Knuth. Optimum binary search trees. *Acta Informatica*, 1:14–25, 1971.

[2] D. D. Sleator and R. E. Tarjan. Self-adjusting binary search trees. *J. A.C.M.*, 32(3):652–686, 1985.

[3] R. E. Tarjan. Sequential access in splay trees. *Combinatorica*, 5(4):367–378, 1985.