Drzewa Tango
Prawie optymalne dynamiczne drzewa BST

Łukasz Zatorski*

Wrocław, dnia 10 marca 2011 r.

Streszczenie

Problem asymptotycznie najlepszych drzew poszukiwań binarnych wciąż pozostaje otwarty. Praca przedstawia jeden z postulowanych modeli dla wersji online tego problemu oraz definicję konkurencyjności. Prezentuje dowód dolnego ograniczenia dynamicznie optymalnej struktury online oraz opis działania wraz ilustracjami i z analizą złożoności drzew Tango o współczynniku konkurencyjności $O(\lg \lg n)$.

1. Wprowadzenie

Teoria złożoności obliczeniowej to opis ciągłego dążenia ku perfekcji, czyli ku lepszym, szybszym i - jeśli to możliwe - optymalnym strukturom danych i algorytmom. Drzewa wyszukiwań binarnych są z pewnością jednymi z najstarszych i popularniejszych struktur, a mimo to wciąż bada się ich możliwości oraz poddaje usprawnieniom. Drzewa Tango poruszone w tej pracy są świetnym tego przykładem - zbliżyły się do doskonałości na odległość mniejszą niż pozostałe drzewa BST (a przynajmniej w zakresie dowodu konkurencyjności). By jednak móc zacząć mówić o lepszych i gorszych strukturach, niezbędne jest ustalenie co i przy pomocy jakich kryteriów chcielibyśmy ze sobą porównywać.

1.1. Model obliczeń

Rozważmy strukturę danych BST bez operacji dodawania i usuwania elementów, tzn. wspierającą wyłącznie wyszukiwanie, na statycznym ustalonym uniwersum kluczy. Skupimy się wyłącznie na poszukiwaniach kończących się sukcesem, które będziemy nazywać zapytaniami lub dostępami. Danymi wejściowymi takiej struktury jest ciąg zapytań X, składający się z kluczy $x_1, x_2, ..., x_m$. Dodatkowo ograniczymy się do długich ciągów zapytań - takich, dla których $m > cn$ dla odpowiednio dużej stałej c.

W wersji online, w każdym z wierzchołków drzewa takiej struktury, poza kluczem przechowywana jest stała ilość informacji dodatkowych. Strukturę taką definiuje algorytm obsługujący dane zapytanie x_i, nazywany również algorytmem zapytania BST. Algorytm ten dysponuje wskaźnikiem przy pomocy którego wykonuje kolejne operacje na podstawie funkcji klucza, danych dodatkowych i wartości zapytania. W szczególności - na jego decyzje może wpływać

*Student Instytutu Informatyki Uniwersytetu Wrocławskiego, ul. Joliot-Curie 15, 50-383 Wrocław

przeszłość, ale nie dysponuje on wiedzą o przyszłych elementach ciągu zapytań. Na początku każdego zapytania x_i wskaźnik jest z powrotem ustawiany na korzeń drzewa (jedna operacja). Następnie algorytm może wykonywać ciąg składający się z operacji przesuwania wskaźnika do lewego syna, prawego syna lub ojca oraz rotacji na wskaźniku i jego ojcu. Operację w trakcie której wskaźnik jest przesuwany lub ustawiany na dany wierzchołek nazywamy również odwiedzeniem tego wierzchołka. Każdej z operacji przypisuje się koszt jednostkowy, których suma w efekcie daje koszt całego zapytania. Sumę kosztów zapytań dla danego ciągu X nazywamy kosztem obsłużenia tego ciągu przez daną strukturę BST.

Model ten, podobny do tych do wprowadzonego już w [ST83] i stosowany w [Wil89] wydaje się mocno uproszczony, jednak nawet dla niego wiele problemów pozostaje nierozstrzygniętych.

1.2. Definicja konkurencyjności

Dla danego konkretnego ciągu zapytań X, istnieje struktura danych BST która wykonuje go optymalnie, tzn. w najkrótszym czasie. Niech $OPT(X)$ oznacza ilość jednostkowych operacji wykonanych przez najszybszą strukturę BST dla ciągu zapytań X. Ze względu na to że $OPT(X)$ dotyczy konkretnej instancji problemu, możemy traktować go jako równie dobrze jak optymalne drzewo offline, takie, które wie jakie zapytania pojawią się w przyszłości. Możemy również założyć, że taka struktura zaczyna w optymalnym stanie zawierającym n kluczy. Założenie to zmniejsza ilość operacji koniecznych wykonania o conajwyżej $O(n)$, ponieważ każde drzewo BST w czasie liniowym da się przerotować na dowolne inne (na tym samym zbiorze kluczy). Dla $OPT(X) \geqslant m$ i $m > n$ jest to zatem różnica o stały współczynnik. Strukturę danych BST nazywamy dynamicznie optymalną, jeśli wykonuje wszystkie ciągi zapytań X w czasie $O(OPT(X))$. Struktura taka jest $C - konkurencyjna$, jeśli wykonuje te ciągi (odpowiedniej długości) w czasie conajwyżej $C \cdot OPT(X)$. W szczególności struktura dynamicznie optymalna jest $O(1) - konkurencyjna$.

1.3. Porównanie popularnych struktur

Wiele prac i badań naukowych w zakresie poszukiwań optymalnych drzew BST poświęcono drzewom splay, po raz pierwszy opisanym przez Sleatora i Tarjana [ST85]. Drzewa te są samoorganizującą się strukturą poszukiwań online, korzystającą z prostej heurystyki - klucz, którego dotyczyło ostatnie zapytanie zostaje przesunięty do korzenia drzewa wraz z zachowaniem porządku BST. Postawiona przez samych twórców hipoteza, jakoby drzewa te były $O(1) - konkurencyjne$ nadal pozostaje nierozstrzygnięta. Powstało przez ten czas kilka górnych ograniczeń na pesymistyczną złożoność drzew splay - udowodniono min. że dla niektórych klas zapytań osiągają one złożoność $o(m \lg n)$. Przykładowo, twierdzenie o dostępie sekwencyjnym, w którym zapytania tworzą porządek symetryczny ogranicza ilość operacji wykonanych przez drzewo splay do $m + 4.5n$, niezależnie od układu początkowej struktury[AE04]. Jednocześnie istnieją klasy zapytań, dla których drzewa te również działają w czasie liniowym, lecz najlepsze znane ograniczenie wciąż jest rzędu $O(m \lg n)$. Tym samym nie przedstawiono jak dotąd dowodu na konkurencyjność drzew splay (względej dynamicznie optymalnej struktury BST) mniejszą niż trywialne $O(\lg n)$, charakteryzujące zwykłe drzewa zbalansowane takie jak drzewa AVL, drzewa czerwono-czarne i inne.

Drzewa *Tango* wymyślone przez Erica Demain'e i innych [DHIP04] są pierwszą udowodnioną strukturą BST ściśle przekraczającą tą granicę i osiągającą $O(\lg \lg n)$ - konkurencyjność.

2. Dolne ograniczenie - przeplot

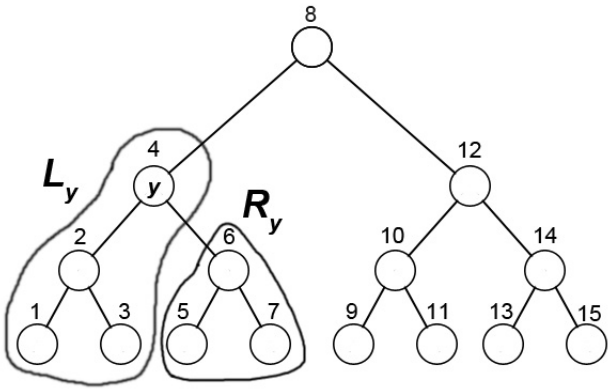
Teoria informacji dostarcza dowodów na to, że każda struktura BST działająca online potrzebuje wykonać $\Omega(m \lg n)$ operacji dla przeciętnego ciągu zapytań, nie stanowi to jednak ścisłego oszacowania dla konkretnej instancji problemu. Wilber przedstawił dwa, jedyne jak dotąd dolne ograniczenia na czas działania $OPT(X)$ z naszego modelu[Wil89]. W swojej pracy dowodzi między innymi, że dla dowolnej struktury BST istnieje taki ciąg zapytań, który wymaga od niej wykonania $\Omega(\lg n)$ operacji dla pojedyńczego zapytania. Tym samym zrównuje on wszystkie takie struktury pod względem pesymistycznej złożoności do $\Omega(m \lg n)$ - w najgorszym wypadku zatem $OPT(X)$ nie jest lepszy od zwykłego drzewa zbalansowanego.

Oba dowody przedstawione przez Wilbera są dosyć skomplikowane i owocują złożonymi funkcjami zależnymi od wejściowego ciągu zapytań. W dalszej części pracy skupimy się na uproszczonym wariancie pierwszego ograniczenia, zaprezentowanym przez Erica Demaine'a i innych [DHIP04], wystarczającym do określenia konkurencyjności drzew Tango.

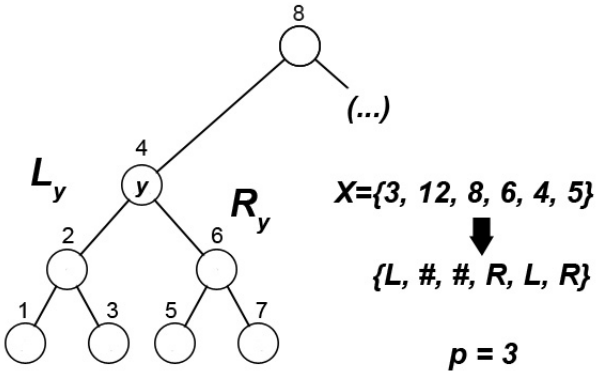
2.1. Drzewo przeplotu

Zdefiniujmy na potrzeby dowodu i przyszłych analiz drzewo przeplotu P dla zbioru kluczy $\{1, 2, 3...n\}$ jako idealne drzewo binarne zawierające owe klucze w swoich wierzchołkach. Przez idealne drzewo rozumiemy takie, w którym każdy wierzchołek poza liściem ma dwoje potomków i wszystkie liście są na tym samym poziomie.

Dla każdego wierzchołka y w drzewie P, niech P_y oznacza poddrzewo wyznaczone przez y, tzn. którego y jest korzeniem. Określamy jego prawy region R_y jako zbiór wierzchołków z prawego poddrzewa y, oraz lewy region, L_y jako zbiór wierzchołków z lewego poddrzewa y wraz z y.



Mając dany wierzchołek y, etykietujemy elementy z ciągu zapytań X w naszym problemie, zapisując sobie czy szukane klucze znajdują się w lewym czy w prawym regionie y (pomijamy te, które nie znajdują się w poddrzewie P_y). Maksymalny, uporządkowany podciąg $x_{i_1}, x_{i_2}, ...x_{i_p}$ zapytań o wierzchołki alterujących między etykietami "lewy", "prawy" nazywamy przeplotem y, a wartość p - jego mocą. Ograniczenie przeplotu $IB(X)$ jest sumą mocy przeplotów po wszystkich wierzchołkach $y \in P$.



Wartość tej funkcji można również wyrazić wzorem:

$$(1) \qquad IB(X) = \sum_{y \in P} |\{0 < i < m : (X_i \in L_y \wedge X_{i+1} \in R_y) \vee (X_i \in R_y \wedge X_{i+1} \in L_y)\}|$$

2.2. Dowód dolnego ograniczenia

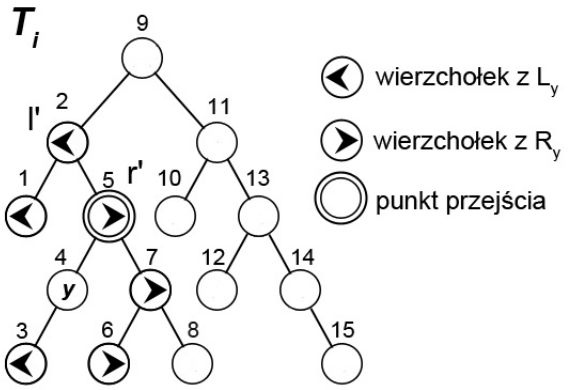
Udowodnimy, że:

Teza 1. $\frac{IB(X)}{2} - n$ *jest dolnym ograniczeniem na* $OPT(X)$*, tzn. koszt optymalnego drzewa offline które obsługuje ciąg zapytań* X*.*

Forma twierdzenia tłumaczy również dlaczego w modelu konieczne było przyjęcie $m > n$.

Niech T będzie dowolnym drzewem BST zgodnym z naszym modelem. Przez T_i będziemy oznaczać stan tego drzewa po wykonaniu zapytań $x_1, x_2, ..., x_{i-1}, x_i$.

Rozważmy przeplot przez wierzchołek y należący do P. Przez *punkt przejścia* dla y w danym momencie i nazywamy wierzchołek z T_i o najmniejszej głębokości, taki, że ścieżka od korzenia T_i do tego wierzchołka zawiera po conajmniej jednym z kluczy z lewego i prawego regionu y. Bezpośrednio z definicji wynika, że znajduje się on w zbiorze R_y albo L_y i jest pierwszym napotkanym kluczem ze swojego regionu.



Każdy algorytm działający na drzewie BST, uzyskując dostęp do jakiegoś elementu z lewego regionu y i elementu z prawego regionu y, zgodnie z intuicją będzie musiał conajmniej raz odwiedzić któryś z punktów przejścia dla y tzn. spełnić warunek przejścia.

Będziemy chcieli oszacować z dołu liczbę wykonanych operacji przez ilość warunków przejścia, które będziemy musieli spełnić. Ich zsumowanie ułatwiają następujące 3 własności:

1) W danym momencie i wierzchołek y ma dokładnie jeden punkt przejścia
2) Punkt przejścia jest stały dla danego wierzchołka aż do czasu jego odwiedzenia (potrzebne do zmiany sposobu sumowania)
3) Nie możemy jedną operacją spełnić kilka warunków przejścia (żaden z wierzchołków nie jest punktem przejścia dla więcej niż jednego innego wierzchołka)

Lemat 2. *Dla danego wierzchołka y w danym momencie i istnieje dokładnie jeden punkt przejścia.*

Dowód. Niech t będzie najniższym wspólnym przodkiem (dalej oznaczanym jako LCA; definicje i własności można znaleźć min. w pracy Harela i Tarjana [HT84]) wszystkich wierzchołków z T_i. Podobnie l - LCA wszystkich wierzchołków z T_i z lewego regionu y oraz r - LCA wszystkich wierzchołków z prawego regionu y. LCA danego zbioru punktów jest jednoznaczy i z definicji należy nadal do przedziału, z którego pochodzą klucze. Wybrane regiony są ciągłymi przedziałami, zatem ich LCA również należą do odpowiadających zbiorów, tzn. $t \in T_i, l \in L_i, r \in R_i$. Dodatkowo t musi być równe l lub r, z uwagi na to, że $T_i = L_i \cup R_i$ Przez symetrię oraz dla ustalenia uwagi załóżmy, że $t \in L_i \rightarrow t = l$, zatem l jest przodkiem r. W takim razie to r jest punktem przejścia dla y - ponieważ ścieżka od korzenia drzewa do niego zawiera po drodze wierzchołek z lewego regionu y (l) oraz jest najkrótszą drogą do jakiegokolwiek wierzchołka z prawego regionu (w tym przypadku r jako ich najniższego wspólnego przodka).

□

Lemat 3. *Dla danego wierzchołka y jego punkt przejścia z nie ulega zmianie, jeśli nie zostanie odwiedzony przez algorytm drzewa BST.*

Dowód. Ustalmy przedział czasowy $[j, k]$, oraz niech l, r będą jak w dowodzie poprzedniego lematu i ponownie dla ustalenia uwagi załóżmy, że r jest punktem przejścia, a l - jego przodkiem (w momencie j). Skoro na tym przedziale czasowym nie odwiedzamy wierzchołka r, to w konsekwencji nie odwiedzimy również żadnego wierzchołka z jego poddrzewa - r wciąż pozostaje $LCA\ (R_y)$ i jednocześnie leży na najkrótszej scieżce od korzenia do dowolnego wierzchołka z tego zbioru. By nadal pozostał punktem przejścia musimy zagwarantować, że na tej ścieżce zawsze znajdzie się jakiś wierzchołek z L_y. Wierzchołek l w momencie j nie znajdował się w poddrzewie r i bez odwiedzania naszego punktu przejścia nie mógł się znaleźć w jego poddrzewie. Zatem $LCA(l,\ r)$ należący do zbioru L_y (patrz poprzedni dowód) i znajdujący się na ścieżce z korzenia T do r gwarantuje, że r nadal pełni rolę punktu przejścia i kończy dowód.

□

I na koniec własność niezbędna do sumowania:

Lemat 4. *W danym momencie i żaden wierzchołek z T_i nie może być punktem przejścia dla dwóch różnych wierzchołków z P.*

Dowód. Rozważmy dwa różne wierzchołki y_1 i y_2 z drzewa P o takim samym punkcie przejścia i określmy dla nich odpowiadajace l_j i r_j jak w dowodzie z Lematu pierwszego. Jeśli żaden z wierzchołków y_j nie jest przodkiem drugiego w drzewie P, to ich poddrzewa są rozłączne, zatem mają różne punkty przejścia (taki punkt należy zawsze do poddrzewa T_{y_j}). W przeciwnym wypadku przez symetrię załóżmy że y_1 jest przodkiem y_2 w drzewie P. Punkt przejścia dla y_2 musi oczywiście należeć do T_{y_2} - jest nim wyższy z wierzchołków l_2 lub r_2. Jednocześnie, tylko niższy z nich może być kandydatem na punkt przejścia dla y_2 - dochodzimy do sprzeczności. □

Tym samym dysponujemy już wszystkim, czego nam trzeba, by ostatecznie udowodnić naszą tezę.

Dowód. Chcemy ograniczyć liczbę operacji potrzebnych do wykonania przez liczbę koniecznych do odwiedzonia punktów przejścia. Dzięki lematowi 3 możemy zsumować te punkty po wierzchołkach y z drzewa P którym odpowiadają. Dla wierzchołka y i jego przeplotu $x_{i_1}, x_{i_2}, ...x_{i_p}$ rozważmy odcinki czasowe $[i_{2j-1}, i_{2j}]$ dla $1 \leqslant j \leqslant \lfloor \frac{p}{2} \rfloor$ oraz zdefiniujmy ponownie l i r. Każde odwiedzenie wierzchołka z lewego regionu wymaga odwiedzenia l, podobnie jak każde odwiedzenie wierzchołka z prawego regionu wymaga odwiedzenia r. By uniknąć kosztu punktu przejścia dla y, gdzieś na tym przedziale czasowym musiałby się on zmienić z r na l lub odwrotnie. Jednakże zgodnie z lematem 2 taka zamiana wymaga odwiedzenia tego punktu. Zatem na każdym z tych przedziałów conajmniej raz musi zostać odwiedzony jakiś punkt przejścia dla y. Sumując po przedziałach wychodzi nam, że algorytm działający na BST musi wykonać conajmniej $\lfloor \frac{p}{2} \rfloor \geqslant \frac{p}{2} - 1$ odwiedzeń. Sumując po wszystkich $y \in P$ otrzymujemy zależność od $IB(X)$.

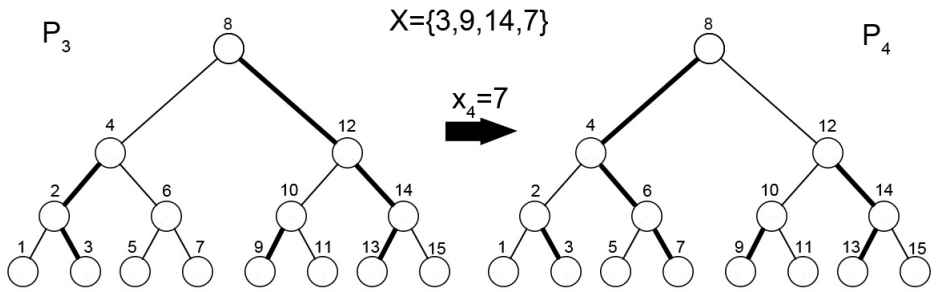
$$\sum_{y \in P} \left(\frac{p_y}{2} - 1 \right) = \frac{\sum_{y \in P} p_y}{2} - n = \frac{IB(X)}{2} - n \tag{2}$$

□

3. Górne ograniczenie - Drzewa Tango

3.1. Opis struktury

Drzewa Tango będą odzwierciedleniem innej, abstrakcyjnej struktury opartej na drzewie przeplotu, wprowadzonym przy dowodzie dolnego ograniczenia na czas działania optymalnego BST. Ponownie rozważmy idealne drzewo binarne P na tym samym zbiorze kluczy $\{1, 2, ...n\}$. Tym razem, dla każdego wierzchołka wewnętrznego $y \in P$ wyróżniamy krawędź do preferowanego syna - lewą, lub prawą, w zależności od tego, czy ostatnie zapytanie dla poddrzewa y dotyczyło odpowiednio lewego lub prawego regionu (w szczególności, zapytanie o y będzie odnosiło się do lewego regionu). Ponadto dla liści i wierzchołków z poddrzew jeszcze nie odwiedzanych nie wyróżniamy preferowanego syna. Tym razem drzewo nie jest zupełnie statyczne. Niech P_i oznacza teraz stan tego drzewa w momencie i, tzn. po ciągu zapytań $x_1, x_2, ..., x_i$. Stan ten jest wyznaczony jednoznacznie i zależny wyłącznie od ciągu zapytań X. Jeśli przyjrzymy się drzewu P to widzimy również, że w danym momencie i podgraf wyróżnionych krawędzi tworzy jego dekompozycję na tzn. preferowane scieżki (spójne składowe tego podgrafu).



Struktura drzewa Tango będzie reprezentować właśnie tą dekompozycję. To znaczy, że w danym momencie i abstrakcyjnemu drzewu P_i będzie odpowiadać stan drzewa Tango T_i. Każda z istniejących w danym momencie ścieżek preferowanych będzie reprezentowana przez zbalansowane drzewo BST, zwane drzewem pomocniczym. Na poziomie koncepcyjnym można wyobrażać je sobie zatem jako drzewo drzew - na poziomie strukturalnym drzewo Tango wyglądać będzie jednak jak zwykłe drzewo BST, spełniające założenia z naszego modelu. Ten wymóg oraz statyczność drzewa wyróżniają je od drzew Link-cut Sleatora i Tarjana[Tar83] i sprawiają że niektóre operacje łączenia i dzielenia poddrzew pomocniczych wymagają większej uwagi. Oba rodzaje drzew bazują jednak na tej samej metodzie dekompozycji na ścieżki preferowane.

3.2. Opis algorytmu

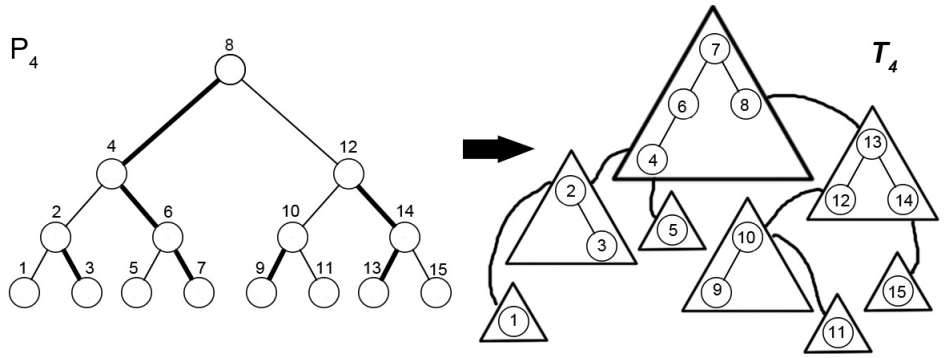
Algorytm Tango będzie wykonywany sekwencyjnie zgodnie z modelem drzewa BST online na ciągu zapytań X. Dla danego zapytania x_i i stanu drzewa T_i algorytm ten ma dwa cele do spełnienia:

1) Odwiedzenie wierzchołka zawierającego klucz z zapytania x_i (warunek konieczny obsłużenia zapytania)

2) Zmiana stanu z T_{i-1} na T_i, odpowiadający abstrakcyjnemu drzewu P_i (warunek zachowania własności struktury)

Wyszukiwanie wierzchołka będzie następowało identycznie jak w zwykłym drzewu BST. Za każdym razem, kiedy będziemy przechodzili przez krawędź niepreferowaną, bedzie to oznaczało przejście do kolejnego drzewa pomocniczego. Niech $S_1, S_2, ...S_{k+1}$ będzie ciągiem odwiedzanych przez nas kolejno ścieżek reprezentowanych przez drzewa pomocnicze. Przy każdym przejściu z S_j do S_{j+1} będziemy musieli dokonać rozcięcia ścieżki S_i na dwie - górną i dolną, względem wierzchołka z którego dokonaliśmy przejścia. Jego efektem będzie odjęcie łączącej obie części krawędzi od zbioru krawędzi preferowanych. Następnie algorytm połączy górną część ścieżki S_j ze ścieżką S_{j+1}, dodając łączącą je krawędź do zbioru preferowanych. Na sam koniec algorytm dokona jeszcze jednej pary operacji cięcia i złączenia jeśli wierzchołek x_i w drzewie nie był liściem i posiadał krawędź preferowaną do prawego syna (zgodnie z naszą definicją, należy ją wówczas zmienić).

Wszystkie te operacje, odnoszące się do scieżek w abstrakcyjnym drzewie P, będą w praktyce wykonywane na drzewach pomocniczych z T. Poniższy rysunek stanowi przykład zależności między takimi ścieżkami i drzewami:



3.3. Drzewa pomocnicze

Drzewo pomocnicze jest rozszerzonym drzewem czerwono-czarnym, przechowującym ścieżkę preferowaną, skierowaną od korzenia w stronę liści w porządku kluczy. Każdy z wierzchołków zawiera również swoją ustaloną i niezmienną głębokość w drzewie P oraz maksymalną i minimalną z wysokości z wierzchołków w swoim poddrzewie (drzewa pomocniczego), oraz informację o tym, czy jest korzeniem drzewa, czy nie. Głębokości tworzą zatem spójny podciąg na przedziale $[0, lg(n+1)]$. Najpłytszy z wierzchołków nazywamy szczytem ścieżki, a najgłębszy - jej dnem. Chcielibyśmy dla tak reprezentowanych ścieżek wykonywać w czasie logarytmicznym od wielkości drzewa operacje:

1) Wyszukiwania elementu

2) Cięcia drzewa pomocniczego na dwa rozłączne drzewa względem danej głębokości d

3) Łączenia dwóch drzew pomocniczych zawierających rozłączne ścieżki preferowane, z których dno jednej jest szczytem drugiej

Nasze drzewa pomocnicze od tych zdefiniowanych w Link-cut trees odróżnia funkcja wagowa - porządek w drzewie wg kluczy a nie wg głębokosci. Zaznaczając które wierzchołki są korzeniami drzew, jesteśmy w stanie reprezentować krawędzie między drzewami traktujemy jak zwykłe krawędzie preferowane. Zaznaczane wierzchołki w trakcie trwania operacji wewnętrzych dla danego drzewa pomocniczego, są ignorowane, dzięki temu operacje te nie wpływają na wygląd innych scieżek.

Na drzewach czerwono-czarnych można zaimplementować funkcje wyszukiwania, rozdzielania i scalania, w czasie $O(\lg k)$ gdzie k to rozmiar drzewa. Pierwsza z tych funkcji odpowiada właśnie wyszukiwaniu elementu w drzewie pomocniczym. Przy pomocy odpowiednio wykonanych operacji rozdzielania i scalania dla drzew czerwono-czarnych możemy zaimplementować równiez cięcie i łączenie dla drzew pomocniczych. Przypomnijmy definicje:

1) Rozdzielenie drzewa czerwono-czarnego w wierzchołku x: przekształcenie drzewa w taki sposób, by x stał się korzeniem, jego lewe poddrzewo stanowi drzewo czerwono-czarne dla kluczy mniejszych od x, a prawe poddrzewo - dla kluczy większych
2) Scalanie dwóch drzew czerwono-czarnych, których korzenie mają wspólnego ojca x: Przekształcenie drzewa w jedno, którego korzeniem jest x

Opis tych funkcji możemy znaleźć na przykład w [CLRS]

Implementacja cięcia drzewa

Załóżmy że chcemy rozciąć drzewo A na głebokości d. Wszystkie wierzchołki spełniające tę własność tworzą przedział kluczy w tym drzewie (bo przechowuje ono ścieżkę z abstrakcyjnego drzewa BST). W czasie $O(\lg k)$ korzystając z zapamiętanej w wierzchołkach informacji o maksymalnej głębokości w danym poddrzewie znajdujemy wierzchołki l r, wyznaczające końce tego przedziału (tzn. wierzchołki o głębokości większej od d o najmniejszym i największym kluczu), oraz l' i r' - ich poprzednika i następnika, tworzących przedział otwarty (l', r'). Nastepnie dokonujemy rozdziału kolejno względem wierzchołków l', r'. Na tym etapie, wierzchołek drzewa D, odpowiadającego wszystkim wierzchołkom na głębokości większej niż d jest lewym synem wierzchołka r'. By wyciąć go z drzewa, wystarczy teraz tylko zaznaczyć go jako korzeń drzewa pomocniczego. Od teraz będzie on ignorowany przez funkcje drzewa A, dlatego możemy je z powrotem scalić kolejno na wierzchołkach r' oraz l'.

Implementacja złączenia drzewa jest procesem dokładnie odwrotnym - wszystkie wymienione czynności powtarzamy od końca. W rezultacie otrzymujemy jedno drzewo czerwono-czarne.

3.4. Analiza złożoności

Lemat 5. *Algorytm Tango obsługuje ciąg zapytań X na zbiorze kluczy $\{1, 2, ..., n\}$ w czasie $O((OPT(X) + n)(1 + \lg \lg n))$*

Dowód. Oszacujmy koszt jednego zapytania x_i przez ilość operacji na drzewach pomocniczych, które musimy wykonać. Niech k oznacza ilość przejść między odwiedzanymi różnymi

drzewami pomocniczymi, czyli dokonanych zmian krawędzi preferowanych. Dla każdego z nich dokonujemy conajwyżej po jednej operacji:

1) Wyszukania (kończącego się sukcesem lub przejściem do innego poddrzewa)

2) cięcia (w celu aktualizacji ścieżki)

3) złączenia (w celu aktualizacji ścieżki)

A w ostatnim drzewie - dodatkowo jednej operacji wyszukania zaznaczonego poprzednika x_i.

Każdą z wyżej wymienionych operacji jesteśmy w stanie wykonać na drzewach pomocniczych w czasie logarytmicznym względem ich wielkości, zachowując dalej ich własności. Najdłuższa ścieżka preferowana, zaczynająca się w korzeniu, ma wysokość $\lg n + 1$. Zatem koszt przetworzenia jednego drzewa możemy z góry ograniczyć przez $c \cdot O(\lg(\lg n + 1)) = O(1 + \lg\lg n)$ dla stałej c, a koszt przetworzenia $k + 1$ takich drzew - $O((k + 1)(1 + \lg\lg n))$.

Koszt przetworzenia ciągu zapytań X możemy również ograniczyć przez $O(K \cdot (1 + \lg\lg n))$ gdzie K to łączna liczba odwiedzeń drzew/ścieżek w naszym drzewu Tango. Zauważmy, że za każdym razem, gdy odwiedzamy nowe drzewo (nie licząc pierwszego), dokonujemy zmiany krawędzi preferowanej z lewej na prawą lub utworzenia nowej. Suma zmian krawędzi preferowanych jest zależna wyłącznie od ciągu zapytań - każda taka zmiana oznacza odwiedzenie wierzchołka z innego regionu niż ostatni dla danego poddrzewa, zatem należy do przeplotu. Nie możemy utworzyć więcej krawędzi niż n, zatem $K = O(IB(X) + n + m)$. W ten sposób koszt algorytmu Tango wynosi $O((IB(X) + n + m)(1 + \lg\lg n))$. Jednocześnie z poprzednich lematów wiemy, że $OPT(X) \geqslant \frac{IB(X)}{2} - n$ oraz $OPT(X) \geqslant m$, zatem czas działania algorytmu Tango wynosi $O((OPT(X) + n)(1 + \lg\lg n))$.

□

Określiliśmy bezpośrednią zależność działania drzew Tango w zależności od dynamicznie optymalnego drzewa BST. W szczególności dla $m = \Omega(n)$ oznacza to, że drzewa te są $O(\lg\lg n)$ - konkurencyjne. Korzystając z dowiedzionego przez nas ograniczenia dolnego nie jesteśmy w stanie poprawić tego rezultatu. Ścieżka preferowana bezpośrednio z korzenia zawiera $lg(n+1)$ wierzchołków, zatem w odpowiadającym jej drzewie zawsze będzie znajdował się wierzchołek na głębokości $\lg\lg n$. Zapytanie o taki wierzchołek może dokonać conajwyżej jednej zmiany krawędzi preferowanej. Dla każdego drzewa Tango istnieje zatem ustalony ciąg zapytań dotyczących zawsze najgłębszego wierzchołka w tym drzewie, w konsekwencji wymuszając wykonanie $\Omega(IB(X) \cdot (\lg\lg n + 1))$ operacji.

Literatura

[AE04] Amr Elmasry. On the sequential access theorem and deque conjecture for splay trees. In *Theoretical Computer Science*, 314(3): 459-466 (2004)

[CLRS] Cormen T., Leiserson C., Rivest R., and Stein C. *Introduction to algorithms.* McGraw-Hill Book Company, Cambridge, London, 2nd edition, 2001.

[DHIP04] Erik D. Demaine, Dion Harmon, John Iacono, and Mihai Patrascu. Dynamic optimality — almost. In *FOCS '04: Proceedings of the 45th Annual IEEE Symposium*

on Foundations of Computer Science (FOCS'04), pages 484–490. IEEE Computer Society, 2004.

[HT84] Dov. Harel; Robert E. Tarjan *Fast algorithms for finding nearest common ancestors*. In SIAM Journal on Computing 13 (2): 338–355, 1984.

[ST83] Daniel D. Sleator and Robert Endre Tarjan. A data structure for dynamic trees. In *Journal of Computer and System Sciences* , 24(3):362–391, June 1983.

[ST85] Daniel D. Sleator and Robert Endre Tarjan. Self-adjusting binary search trees. In *Journal of the ACM*, 32(3):652–686, July 1985.

[Tar83] Robert Endre Tarjan. Linking and cutting trees. In *Data Structures and Network Algorithms*, chapter 5, pages 59–70. Society for Industrial and Applied Mathematics, 1983.

[Wil89] R. Wilber. Lower bounds for accessing binary search trees with rotations. *SIAM J. Comput.*, 18(1):56–67, 1989.