Zastosowania binarnych drzew przeszukiwań BST

Binarne drzewa przeszukiwań BST są strukturą niezmiernie przydatną w praktyce, wykorzystywaną w bardzo wielu zastosowaniach. Podstawowym schematem, w którym są wykorzystywane są **tablice symboli**. Tablica symboli jest strukturą, która pozwala powiązać klucze z wartościami. Ogólnie zastosowania tablic symboli obejmują wszelkiego rodzaju słowniki, indeksy, skorowidze, bazy danych, wyszukiwarki, itp.

Największe znaczenie w zastosowaniach takich tablic ma operacja wyszukiwania, która jest wykonywana powtarzalnie, bardzo wiele razy. Nawet dla tablic o umiarkowanych wielkościach, dla których asymptotyczny czas wykonywania operacji niekoniecznie jest kluczowy, liczy się maksymalna sprawność ich obsługi, głównie operacji wyszukiwania.

Ponieważ drzewa BST są uporządkowane, sprawność wyszukiwania kluczy powinna być na poziomie wyszukiwania binarnego w tablicach uporządkowanych, czyli $O(\log n)$. Jednak widzieliśmy, że liczba kroków w przeszukiwaniu drzew BST jest $O(h)$, gdzie h jest wysokością drzewa binarnego, która może wynosić od $\log_2 n$ do n.

Bardzo ważną kwestią jest więc jak zapewnić, by wysokość budowanych drzew BST wynosiła $\log_2 n$, a przynajmniej $O(\log_2 n)$.

Optymalne drzewa BST

Tablice symboli mogą być budowane dynamicznie, ale w szeregu zastosowań tablica jest budowana jednorazowo, i potem tylko wykorzystywana w trybie *read-only* (np. książka telefoniczna). Ponadto, w wielu zastosowaniach struktura jest tylko budowana i potem wykorzystywana — operacja usuwania nie jest w ogóle potrzebna.

W tych przypadkach można wziąć pod uwagę rozkład częstotliwości wyszukiwania kluczy, i zbudować drzewo, które zminimalizuje oczekiwany czas wyszukiwania klucza z tego konkretnego rozkładu. Gdy te najczęściej wyszukiwane klucze znajdą się blisko korzenia drzewa BST, skróceniu ulegnie czas tych najczęściej wykonywanych wyszukiwań, kosztem rzadko wyszukiwanych kluczy, które mogą być zlokalizowane w liściach, nawet w dłuższych gałęziach drzewa.

Nie interesuje nas już zatem najgorszy przypadek wyszukiwania, i nie będziemy próbowali zapewnić by wysokość drzewa była utrzymana w granicach $O(\log_2 n)$. Zamiast tego, globalnej optymalizacji podlega jedynie średni czas wyszukiwania zgodny z przyjętym rozkładem prawdopodobieństwa wyszukiwania kluczy.

Jednak tutaj nie będziemy rozważali metod budowy optymalnych drzew BST zgodnych z tymi założeniami (patrz np. podrozdział 14.p w podręczniku CLRS). Zamiast tego zajmiemy się zagadnieniami budowy prawdziwie dynamicznych drzew binarnych.

Średnia/oczekiwana wysokość drzewa BST

Można udowodnić, że dla danego zbioru kluczy, oczekiwana wysokość drzewa BST zbudowanego podstawową procedurą TREE-INSERT, uśredniona po wszystkich permutacjach tego zbioru kluczy, wynosi $O(\log_2 n)$.

Może to sprawiać wrażenie, że dla celów zastosowań, gdzie istotny jest średni przypadek czasu działania programu, zwykła procedura budowy drzewa BST jest wystarczająca, i nie musimy rozważać przypadku najgorszego. Jednak nie jest to do końca prawda. Jeśli chcemy uśredniać po wszystkich budowanych drzewach to owszem, średnie drzewo będzie miało wysokość $O(\log_2 n)$. Jednak jeśli w konkretnym zastosowaniu, konkretny program zbuduje drzewo zdegenerowane, o wysokości $\Omega(n)$, to w tej aplikacji oczekiwany czas operacji wyszukiwania będzie $\Omega(n)$.

Jeśli więc chcemy mieć pewność, że średni przypadek wyszukiwania w programie budującym dynamicznie drzewo BST będzie działał w czasie $O(\log_2 n)$ to musimy zapewnić, by drzewo w każdym przypadku miało wysokość $O(\log_2 n)$.

Drzewa zrównoważone

Drzewo BST zawierające n węzłów o wysokości dokładnie $h = \lceil \log_2(n+1) \rceil - 1$ nazywamy **(w pełni) zrównoważonym**.

n=1	h=0	·
n=2,3	h=1	⋰ ⋱
n=4,5,6,7	h=2	...

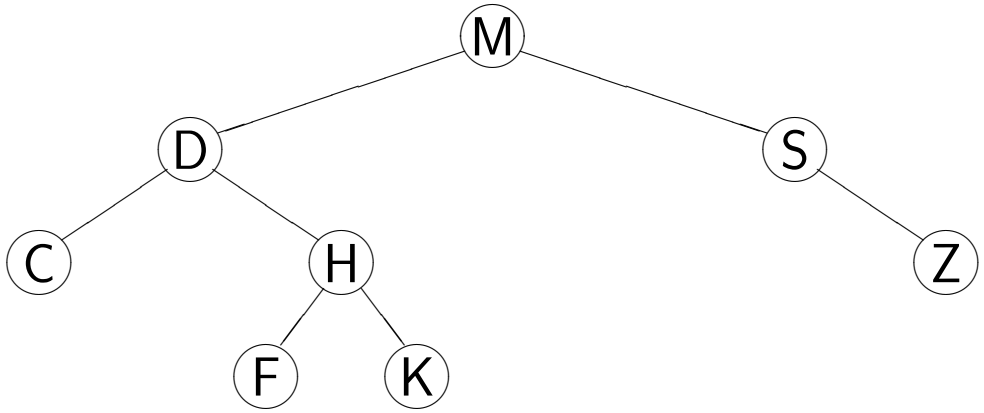
Budowa w pełni zrównoważonych drzew BST jest niepraktyczna. Nie są znane żadne efektywne algorytmy budowy takich drzew. Każdy algorytm jest praktycznie równoważny zbudowaniu wszystkich możliwych drzew z wszystkich permutacji ciągu wejściowego i wybraniu drzewa o minimalnej wysokości!

Zamiast tego, badania koncentrują się nad wynalezieniem efektywnych algorytmów budowy drzew **częściowo zrównoważonych**, to znaczy takich, których wysokość będzie $O(\log_2 n)$.

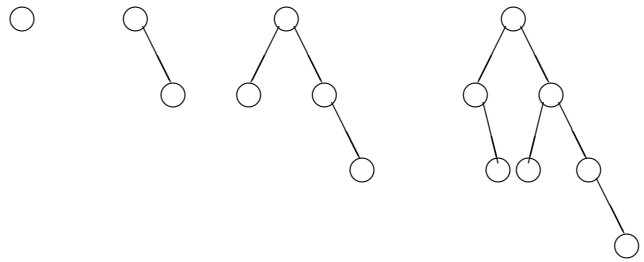
Drzewa AVL

Drzewo BST nazywamy **drzewem AVL** o ile wysokość dwóch poddrzew dowolnego węzła nie różni się o więcej niż 1, w całym drzewie.

Przykładowe drzewo AVL:



„Zdegenerowane" drzewa AVL:



Ponieważ drzewa AVL są pełnoprawnymi drzewami BST, zatem procedury przeszukiwania na drzewach AVL nie różnią się od przeszukiwania na zwykłych drzewach BST.

Jednak jeśli chodzi o kwestię budowania i obsługi (dodawania i usuwania elementów), to pojawiają się jednak dwa istotne pytania:

- Czy przeszukiwanie na drzewach AVL nadal będzie średnio $O(\log n)$?
- Czy istnieje (względnie) łatwa metoda budowy takich drzew?

Najpierw zajmiemy się odpowiedzią na drugie z tych pytań.

Dodawanie/usuwanie węzła do/z drzewa AVL

Procedury dodawania i usuwania węzłów z drzew AVL można zbudować przez pewne modyfikacje procedur dla drzew BST. Istotnym elementem ułatwiającym wykonanie tych operacji jest dodanie do każdego węzła drzewa pola $balance$ o wartości:

-1	—	oznacza, że lewe poddrzewo węzła jest wyższe (o 1)
0	—	oznacza, że oba poddrzewa węzła są w pełnej równowadze
$+1$	—	oznacza, że prawe poddrzewo węzła jest wyższe (o 1)

Przy dodawaniu węzła z do drzewa, z punktu widzenia węzła na danym poziomie drzewa, mogło zdarzyć się, że poddrzewo, do którego węzeł z trafił: zyskało dodatkowy poziom, czyli jego wysokość jest teraz $+1$, lub nie zyskało na wysokości. (Przy usuwaniu węzła, poddrzewo mogło stracić jeden poziom wysokości, lub nie.)

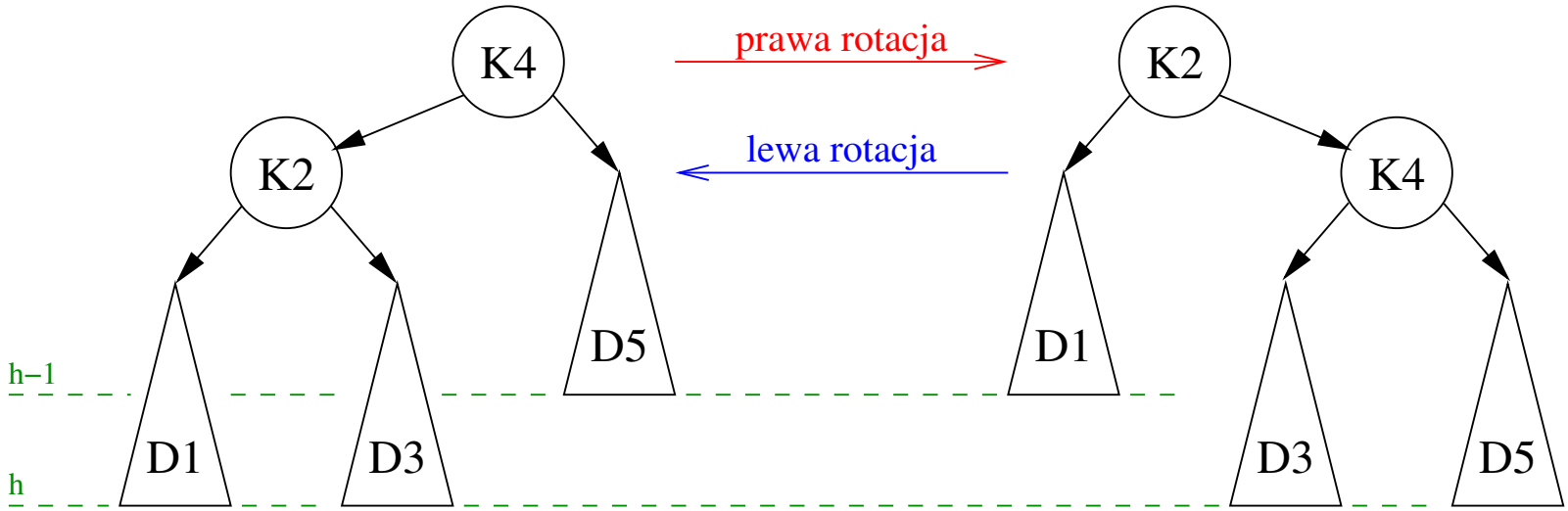
Gdy poddrzewo nie zyskało (lub nie straciło) na wysokości, to z punktu widzenia tego węzła — jak również wszystkich innych węzłów powyżej, aż do samego korzenia — nie ma zmian w zrównoważeniu drzewa (własność AVL), zatem procedura dodawania/usuwania węzła może na tym się zakończyć.

Również gdy poddrzewo zyskało na wysokości, ale wcześniej było niższe niż jego bliźniak (albo straciło, ale wcześniej było wyższe), to również własność AVL pozostaje zachowana (wzwyż aż do korzenia), i trzeba tylko odnotować w tym węźle osiągnięcie pełnej równowagi ($balance = 0$).

Drzewa AVL — rotacje

Jedynym przypadkiem, kiedy drzewo AVL mogło stracić własność AVL jest sytuacja, kiedy z punktu widzenia jakiegoś węzła drzewa, jego poddrzewo do którego został dodany węzeł zyskało na wysokości, a już wcześniej było wyższe (lub straciło, a było niższe). Taka sytuacja wymaga korekty struktury drzewa dla przywrócenia własności AVL.

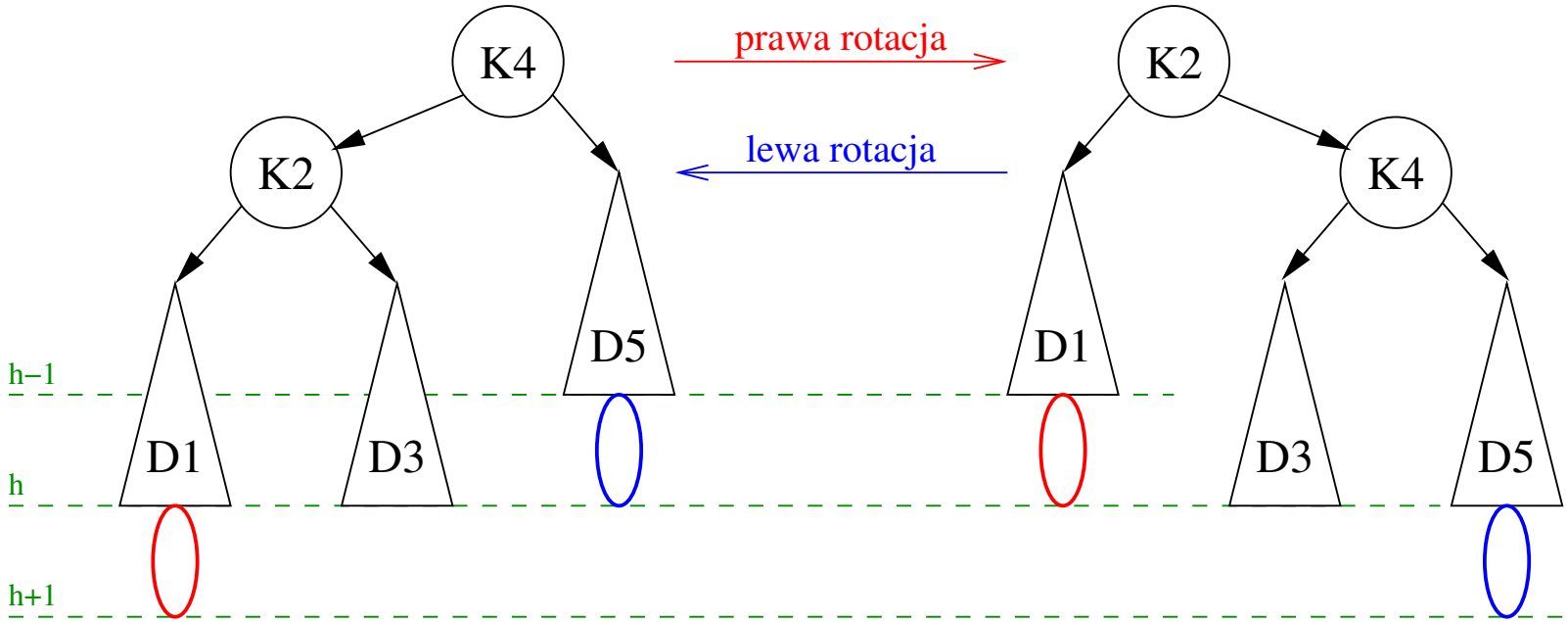
Do przywracania własności AVL przydatna będzie operacja **rotacji**:



Istotną własnością rotacji jest fakt, że z punktu widzenia całej reszty drzewa (powyżej węzłów K2/K4), wysokość drzewa na którym wykonano rotację się nie zmienia. Nic się również nie zmienia w ramach poddrzew D1, D3, i D5.

Drzewa AVL — korekta zrównoważenia: pojedyncza rotacja

Korekty naruszenia własności AVL rozważymy w dwóch przypadkach. Pierwszym przypadkiem jest, kiedy naruszenie nastąpiło w skrajnym poddrzewie drzewa, którego wysokość się zwiększyła:



Patrząc z lewej do prawej, naruszenie nastąpiło w lewym poddrzewie węzła K4, i przyrosło tam lewe poddrzewo D1 tego lewego poddrzewa K2 (dodany został węzeł czerwony). Alternatywnie, patrząc z prawej do lewej, jeśli naruszenie nastąpiło w prawym poddrzewie K4 danego węzła K2, to rozważamy przypadek, kiedy wydarzyło się to w prawym poddrzewie prawego poddrzewa (dodany został węzeł niebieski).

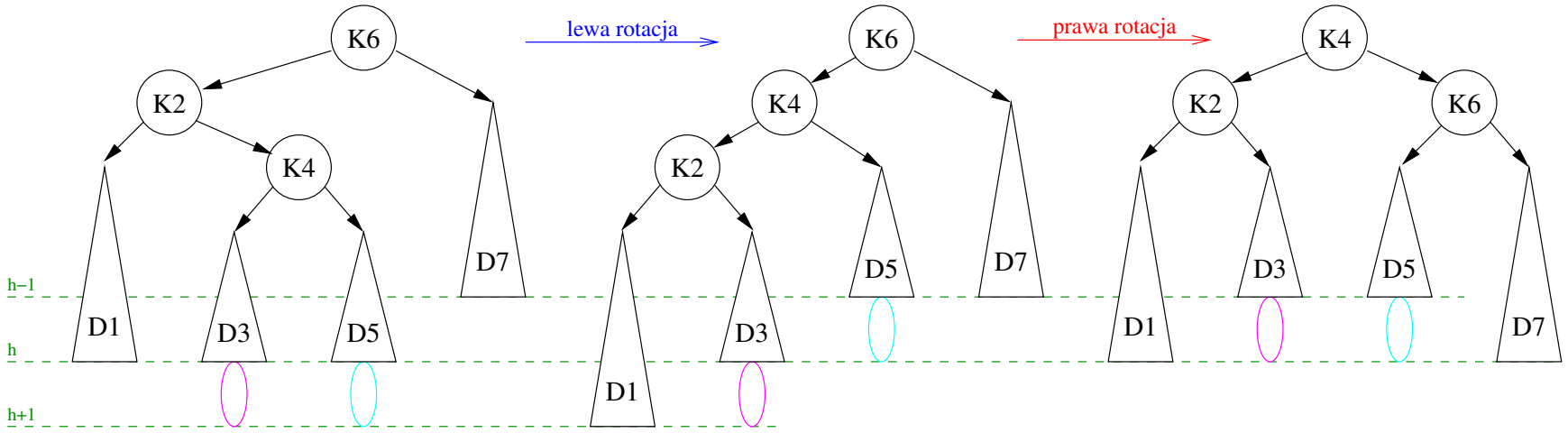
Jak widać, po wykonaniu rotacji równowaga całego (pod)drzewa została przywrócona.

Drzewa AVL — korekta zrównoważenia: podwójna rotacja

Drugim przypadkiem jest sytuacja, kiedy — z punktu widzenia danego węzła drzewa — nastąpiło naruszenie własności AVL w jednym z jego poddrzew, ale w tym poddrzewie wydarzyło się to w jego poddrzewie „wewnętrznym".

Inaczej mówiąc, jeśli z punktu widzenia danego węzła naruszenie nastąpiło w poddrzewie lewym, to w tym lewym poddrzewie miało to miejsce w jego poddrzewie prawym. A jeśli naruszenie nastąpiło w poddrzewie prawym, to w tym poddrzewie było to w jego poddrzewie prawym.

Poniższy rysunek ilustruje tylko pierwszą z tych sytuacji (naruszenie w prawym poddrzewie lewego poddrzewa danego węzła). Drugą sytuację można otrzymać przez lustrzane odbicie rysunku.



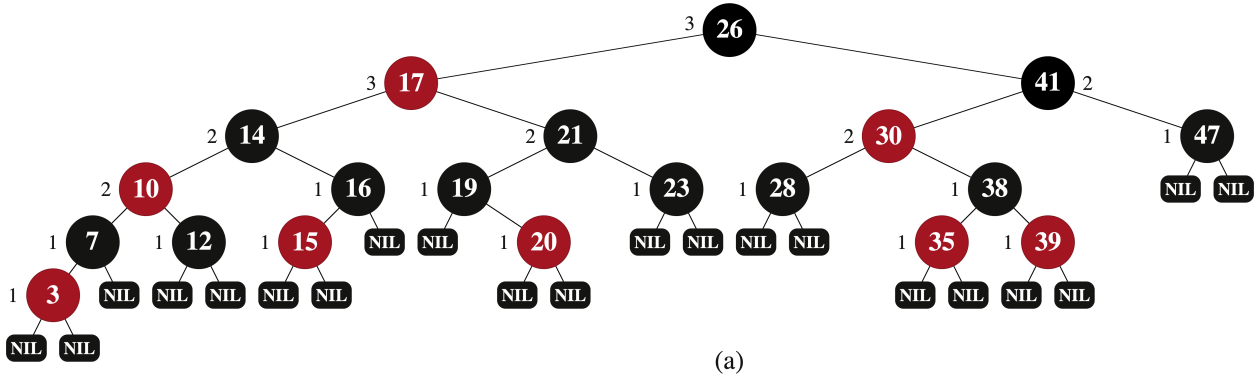
W tym przypadku konieczne są dwie rotacje: lewa+prawa, dla przywrócenia równowagi.

Dodawania węzła do drzewa AVL — pseudokod

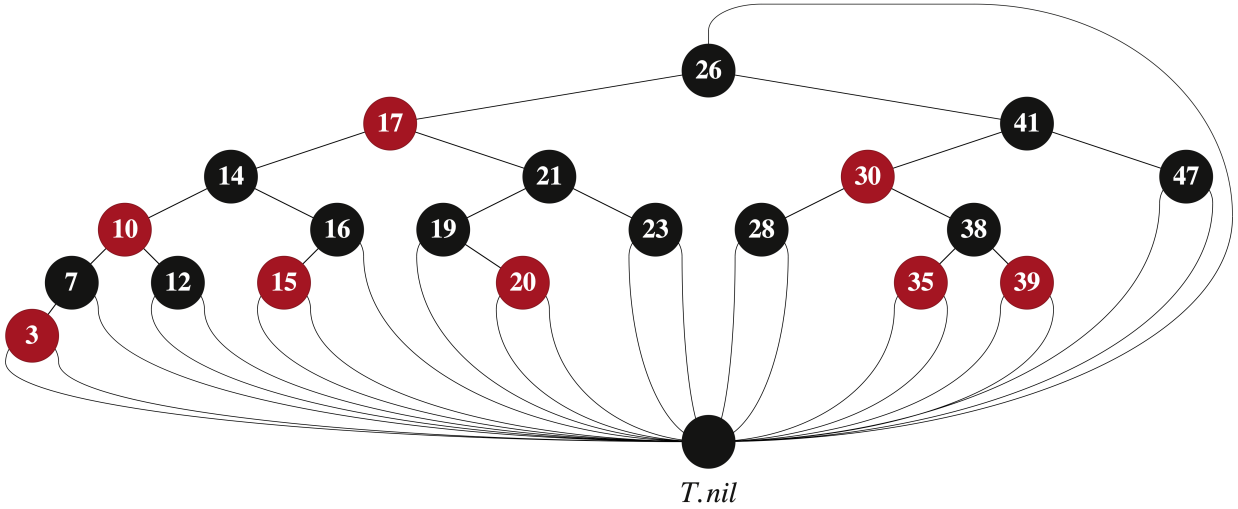
Podsumowanie — drzewa częściowo zrównoważone

Drzewa:	Wysokość drzewa w przypadku:		
	najgorszym	najlepszym	średnim
binarne BST:			
w pełni zrównoważ.	$\log_2(N+1)$	$\log_2(N+1)$	$\log_2(N+1)$
niezrównoważone	N	$\log_2(N+1)$	$1.39 \times \log_2(N)$
drzewa AVL	$1.44 \times \log_2(N)$	$\log_2(N+1)$	$\log_2(N)+0.25$
drzewa WAVL			
czerwono-czarne	$2 \times \log_2(N)$		
niebinarne ST:			
drzewa 2-3			
B-drzewa	$\log_{D+1}(\frac{N+1}{2})+1$	$\log_{2D+1}(N+1)$	$\log_{1.38D+1}(\frac{N}{D+1})+1$

Drzewa czerwono-czarne

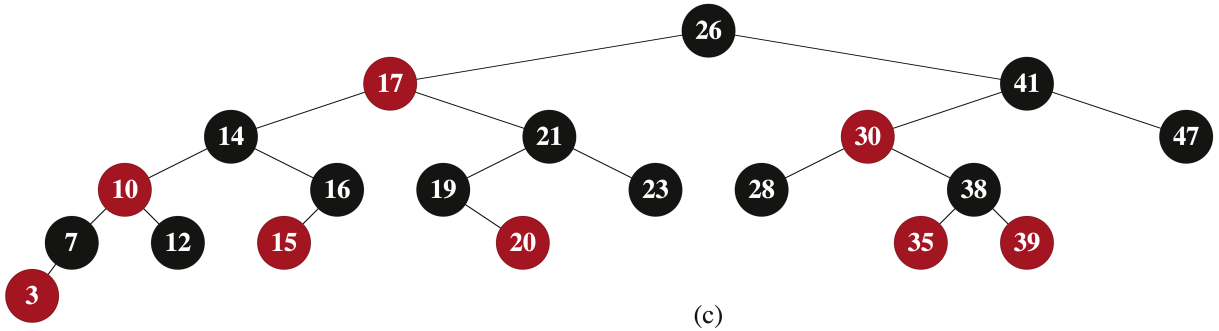


(a)



T.nil

(b)



(c)