Implementacja drzewa AVL w języku Python

Jarosław Madej

Wprowadzenie:

Drzewo AVL, nazywane również drzewem dopuszczalnym – zrównoważone binarne drzewo poszukiwań (BST), w którym wysokość lewego i prawego poddrzewa każdego węzła różni się co najwyżej o jeden.
Drzewo AVL pozostaje drzewem binarnych poszukiwań, co oznacza, że wierzchołki są uporządkowane w określony sposób. Zazwyczaj przyjmuje się, że elementy w lewym poddrzewie są mniejsze od wierzchołka, zaś w prawym – większe. Zrównoważenie drzewa osiąga się, przypisując każdemu węzłowi współczynnik wyważenia, który jest równy różnicy wysokości lewego i prawego poddrzewa. Może wynosić 0, +1 lub -1. Wstawiając lub usuwając elementy drzewa (tak, by zachować własności drzewa BST), modyfikuje się też współczynnik wyważenia, gdy przyjmie on niedozwoloną wartość, wykonuje specjalną operację rotacji węzłów, która przywraca zrównoważenie.
Algorytmy podstawowych operacji na drzewie AVL przypominają te z binarnych drzew poszukiwań, ale są poprzedzane lub następują po nich jedna lub więcej "rotacji". Algorytmy te mogą być realizowane poprzez rekurencję lub, co nawet prostsze, poprzez iteracyjne przechodzenie po krawędziach w górę lub w dół drzewa. Koszt każdej operacji to O(log n).

Współczynnik zrównoważenia:

Każdy węzeł drzewa AVL posiada dodatkowe pole o nazwie bf (ang. balance factor – współczynnik równowagi). Wartość tego pola spełnia równanie:

bf = hL – hR
gdzie: hL i hR to odpowiednio wysokości lewego i prawego poddrzewa.

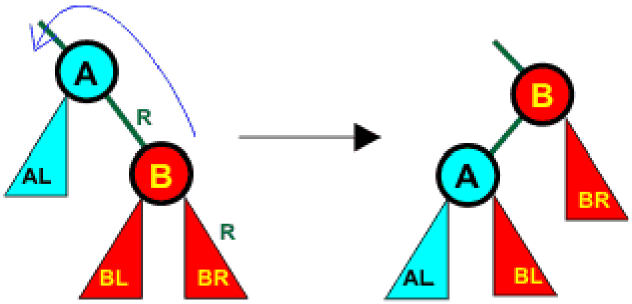
Z powyższego wzoru możemy wysnuć następujące wnioski:

bf = 1 – lewe poddrzewo jest wyższe o jeden poziom od prawego poddrzewa, hL > hR
bf = 0 – oba poddrzewa są równej wysokości, hL = hR
bf = -1 – prawe poddrzewo jest wyższe o jeden poziom od lewego poddrzewa, hL < hR

Innych wartości w drzewie AVL współczynnik bf nie może przyjmować – ograniczenie to nosi nazwę własności drzewa AVL

Rotacja RR:

W rotacji uczestniczą węzły A i B. Węzeł B zajmuje miejsce węzła A, węzeł A staje się lewym synem węzła B. Lewy syn węzła B (BL) staje się prawym synem węzła A.



Algorytm rotacji RR dla drzewa AVL

Wejście:

root – referencja do zmiennej, która przechowuje adres korzenia drzewa AVL

A – wskazanie węzła głównego rotacji.

Wyjście:

Wykonanie rotacji RR

Dane pomocnicze:

B – wskazanie węzła B

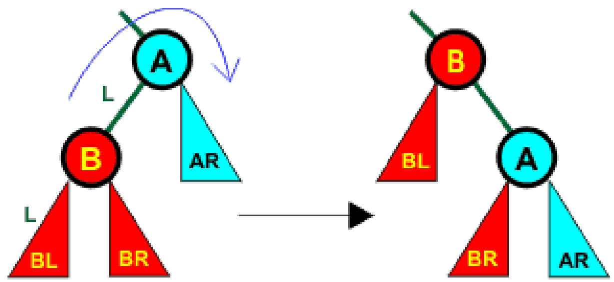
p – wskazanie ojca A

Lista kroków:

K01:	B ← (A→right)	; w B umieszczamy adres prawego syna węzła A
K02:	p ← (A→up)	; w p umieszczamy ojca A
K03:	(A→right) ← (B→left)	; prawym synem A staje się lewy syn B
K04:	Jeśli (A→right) ≠ nil, to (A→right→up) ← A	; jeśli prawy syn istnieje, to jego ojcem jest teraz A
K05:	(B→left) ← A	; lewym synem B staje się A
K06:	(B→up) ← p	; ojcem B jest ojciec węzła A
K07:	(A→up) ← B	; natomiast A zmienia ojca na B
K08:	Jeśli p = nil, to idź do K11	; sprawdzamy, czy węzeł A był korzeniem
K09:	Jeśli (p→left) = A, to (p→left) ← B Inaczej (p→right) ← B	; jeśli nie, to uaktualniamy jego ojca
K10:	Idź do K12	
K11:	root ← B	; jeśli A był korzeniem, to uaktualniamy korzeń
K12:	Jeśli (B→bf) = -1, to (A→bf) ← 0, (B→bf) ← 0 Inaczej (A→bf) ← -1, (B→bf) ← 1	; modyfikujemy współczynniki równowagi
K13:	Zakończ	

Rotacja LL:

Rotacja LL jest lustrzanym odbiciem rotacji RR. W rotacji uczestniczą węzły A i B. Węzeł B zajmuje miejsce węzła A, węzeł A staje się prawym synem węzła B. Prawy syn węzła B (BR) staje się lewym synem węzła A.



Algorytm rotacji LL dla drzewa AVL

Wejście:

root – referencja do zmiennej, która przechowuje adres korzenia drzewa AVL
A – wskazanie węzła głównego rotacji.

Wyjście:

Wykonanie rotacji LL

Dane pomocnicze:

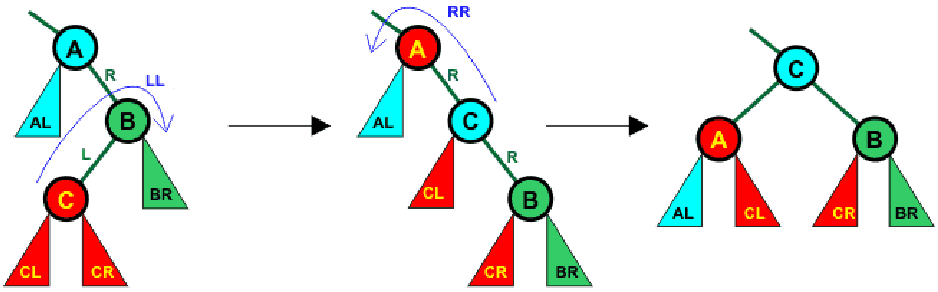
B – wskazanie węzła B
p – wskazanie ojca A

Lista kroków:

K01:	B←(A→left)	; w B umieszczamy adres lewego syna węzła A
K02:	p←(A→up)	; w p umieszczamy ojca A
K03:	(A→left) ← (B→right)	; lewym synem A staje się prawy syn B
K04:	Jeśli(A→left) ≠ nil, to(A→left→up) ←A	; jeśli lewy syn istnieje, to jego ojcem jest teraz A
K05:	(B→right) ←A	; prawym synem B staje się A
K06:	(B→up) ←p	; ojcem B jest ojciec węzła A
K07:	(A→up) ←B	; natomiast A zmienia ojca na B
K08:	Jeślip = nil, to idź do K11	; sprawdzamy, czy węzeł A był korzeniem
K09:	Jeśli(p→left) = A, to(p→left) ←B Inaczej (p→right) ←B	; jeśli nie, to uaktualniamy jego ojca
K10:	Idź do K12	
K11:	root←B	; jeśli A był korzeniem, to uaktualniamy korzeń
K12:	Jeśli(B→bf) = 1, to (A→bf) ← 0, (B→bf) ← 0 Inaczej (A→bf) ← 1, (B→bf) ← -1	; modyfikujemy współczynniki równowagi
K13:	Zakończ	

Rotacja RL:

Rotacja RL jest złożeniem rotacji LL i rotacji RR, dlatego nosi nazwę rotacji podwójnej. Pierwsza rotacja LL sprowadza gałąź drzewa do postaci dogodnej dla następnej rotacji RR.



Algorytm rotacji RL dla drzewa AVL

Wejście:

root – referencja do zmiennej przechowującej adres korzenia drzewa AVL

A – wskazanie węzła głównego rotacji

Wyjście:

Wykonanie rotacji RL

Dane pomocnicze:

B – wskazanie węzła B

C – wskazanie węzła C

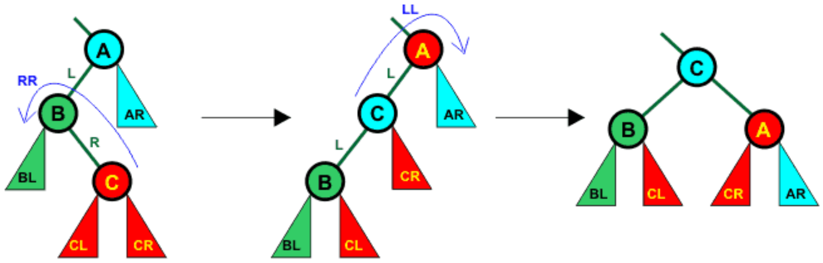
p – wskazanie ojca A

Lista kroków:

K01:	B← (A→right)	; ustawiamy adresy węzłów biorących udział w rotacji
K02:	C← (B→left)	
K03:	p← (A→up)	
K04:	(B→left) ← (C→right)	; lewym synem B staje się prawy syn C
K05:	Jeśli(B→left) ≠ nil, to(B→left→up) ←B	; jeśli lewy syn B istnieje, to B staje się jego ojcem
K06:	(A→right) ← (C→left)	; prawym synem A staje się lewy syn C
K07:	Jeśli(A→right) ≠ nil, to(A→right→up) ←A	; jeśli prawy syn A istnieje, to A staje się jego ojcem
K08:	(C→left) ←A	; lewym synem C staje się A
K09:	(C→right) ←B	; prawym synem C staje się B
K10:	(A→up) ←C	; ojcem A staje się C
K11:	(B→up) ←C	; ojcem B staje się C
K12:	(C→up) ←p	; ojcem C staje się były ojciec A
K13:	Jeśli p = nil, to idź do K16	; sprawdzamy, czy A był korzeniem
K14:	Jeśli(p→left) = A,to(p→left) ←C Inaczej (p→right) ←C	; jeśli nie, to uaktualniamy byłego ojca A, aby był ojcem C
K15:	Idź do K17	
K16:	root←C	; w przeciwnym razie uaktualniamy korzeń
K17:	Jeśli(C→bf) = -1, to(A→bf) ← 1 Inaczej (A→bf) ← 0	; uaktualniamy współczynniki równowagi
K18:	Jeśli(C→bf) = 1, to(B→bf) ← -1 Inaczej (B→bf) ← 0	
K19:	(C→bf) ← 0	
K20:	Zakończ	

Rotacja LR

Podobnie jak w przypadku rotacji pojedynczych RR i LL, rotacja LR jest lustrzanym odbiciem rotacji RL.



Algorytm rotacji LR dla drzewa AVL

Wejście:

root – referencja do zmiennej przechowującej adres korzenia drzewa AVL
A – wskazanie węzła głównego rotacji

Wyjście:

Wykonanie rotacji LR

Dane pomocnicze:

B – wskazanie węzła B
C – wskazanie węzła C
p – wskazanie ojca A

Lista kroków:

K01:	B← (A→left)	; ustawiamy adresy węzłów biorących udział w rotacji
K02:	C← (B→right)	
K03:	p← (A→up)	
K04:	(B→right) ← (C→left)	; prawym synem B staje się lewy syn C
K05:	Jeśli(B→right) ≠ nil, to(B→right→up) ←B	; jeśli prawy syn B istnieje, to B staje się jego ojcem
K06:	(A→left) ← (C→right)	; lewym synem A staje się prawy syn C
K07:	Jeśli(A→left) ≠ nil, to(A→left→up) ←A	; jeśli lewy syn A istnieje, to A staje się jego ojcem
K08:	(C→right) ←A	; prawym synem C staje się A
K09:	(C→left) ←B	; lewym synem C staje się B
K10:	(A→up) ←C	; ojcem A staje się C
K11:	(B→up) ←C	; ojcem B staje się C
K12:	(C→up) ←p	; ojcem C staje się były ojciec A
K13:	Jeśli p = nil,to idź do K16	; sprawdzamy, czy A był korzeniem
K14:	Jeśli(p→left) =A, to(p→left) ←C Inaczej (p→right) ←C	; jeśli nie, to uaktualniamy byłego ojca A, aby był ojcem C
K15:	Idź do K17	
K16:	root←C	; w przeciwnym razie uaktualniamy korzeń
K17:	Jeśli(C→bf) = 1,to(A→bf) ← -1 Inaczej (A→bf) ← 0	; uaktualniamy współczynniki równowagi
K18:	Jeśli(C→bf) = -1,to(B→bf) ← 1 Inaczej (B→bf) ← 0	
K19:	(C→bf) ← 0	
K20:	Zakończ	

Algorytm wstawiania węzła do drzewa AV

Wejście

root – referencja do zmiennej, która przechowuje adres korzenia drzewa AVL
k – klucz wstawianego węzła

Wyjście:

Drzewo AVL z nowym węzłem o kluczu k

Elementy pomocnicze:

w, p, r	– wskazania węzłów
LL(root, x), RR(root, x), LR(root, x), RL(root, x)	– rotacje węzłów, x jest wskazaniem węzła głównego rotacji

Lista kroków:

K01:	Utwórz nowy węzeł	; tworzymy nowy węzeł
K02:	w← adres nowego węzła	
K03:	(w→left) ←nil	; inicjujemy pola węzła
K04:	(w→right) ←nil	
K05:	(w→key) ←k	
K06:	(w→bf) ← 0	
K07:	p←root	
K08:	Jeśli p = nil,to root←w i zakończ	; sprawdzamy, czy drzewo jest puste
K09:	Jeśli k< (p→key), to idź do K13	; porównujemy klucze
K10:	Jeśli(p→right) = nil, to (p→right) ←w Idź do K16	; jeśli prawy syn nie istnieje, to ; nowy węzeł staje się prawym synem ; i wychodzimy z pętli
K11:	p← (p→right)	; inaczej przechodzimy do prawego syna
K12:	Idź do K09	; i kontynuujemy pętlę
K13	Jeśli(p→left) = nil, to: (p→left) ←w Idź do K16	; to samo dla lewego syna
K14:	p← (p→left)	
K15:	Idź do K09	
K16:	(w→up) ←p	; uzupełniamy ojca węzła
K17:	Jeśli(p→bf) = 0,to idź do K20	; modyfikujemy współczynniki równowagi w kierunku korzenia
K18:	(p→bf) ← 0	
K19:	Zakończ	
K20:	Jeśli(p→left) = w, to(p→bf) ← 1 Inaczej (p→bf) ← -1	
K21:	r← (p→up)	; będziemy się przemieszczać w górę drzewa AVL w poszukiwaniu węzła, który ; stracił równowagę w wyniku dodania elementu n.
K22:	Dopóki r ≠ nil:wykonuj kroki K23...K26	; wskazania r i p to ojciec i syn na tej ścieżce
K23:	Jeśli(r→bf) ≠ 0,to idź do K28	; jeśli r.bf = 0, to obie gałęzie r były w równowadze przed dodaniem węzła
K24:	Jeśli(r→left) =p, to(r→bf) ← 1 Inaczej (r→bf) ← -1	; zwiększamy lub zmniejszamy r.bf w zależności od tego, w której gałęzi węzła r został dodany węzeł w. Gałąź ta jest cięższa!

K25:	p←r	; przemieszczamy się w górę drzewa
K26:	r← (r→up)	
K27:	Zakończ	
K28:	Jeśli(r→bf) = -1, to idź do K32	; sprawdzamy, która z gałęzi r była cięższa
K29:	Jeśli(r→right) =p, to(r→bf) ← 0 i zakończ	; prawa gałąź ze wstawionym elementem równoważy lewą gałąź - kończymy
K30:	Jeśli(p→bf) = -1, to LR(root,r) Inaczej LL(root,r)	
K31:	Zakończ	
K32:	Jeśli(r→left) = p, to(r→bf) ← 0 i zakończ	; lewa gałąź ze wstawionym elementem równoważy prawą gałąź
K33:	Jeśli(p→bf) = 1, to RL(root, r) Inaczej RR(root, r)	
K34:	Zakończ	

Algorytm usuwania węzła z drzewa AVL

Usuwanie węzła z drzewa AVL jest operacją dosyć skomplikowaną, ponieważ wymaga rozważenia dużej liczby przypadków przy przywracaniu równowagi. Przy wstawianiu węzła wystarczyło na ścieżce wiodącej od tego węzła do korzenia wykryć konfigurację węzłów, których rotacja przywracała równowagę w drzewie. Po wykonaniu rotacji drzewo AVL znów było w równowadze i operacja wstawiania mogła się zakończyć. Przy usuwaniu węzła mamy nieco inną sytuację. Tutaj poddrzewo, w którym dokonaliśmy usunięcia węzła, zmniejsza swoją wysokość, co może propagować się w górę aż do samego korzenia drzewa. Jedna rotacja zatem może nie być wystarczająca do przywrócenia równowagi w drzewie - rotacje należy kolejno wykonywać dla coraz wyższych węzłów dotąd, aż zniwelujemy zmniejszenie wysokości poddrzewa. Po drodze należy rozważać wiele różnych przypadków konfiguracji węzłów.

Algorytm usuwania węzła z drzewa AVL

Wejście

root – referencja do zmiennej, która przechowuje adres korzenia drzewa AVL
x – wskazanie węzła do usunięcia

Wyjście:

Wskazanie x oraz drzewo AVL bez węzła x.

Elementy pomocnicze:

t,y,z	–	wskazania węzłów
LL(root, x), RR(root, x) LR(root, x), RL(root, x)	–	rotacje węzłów, x jest wskazaniem węzła głównego rotacji
pred(x)	–	zwraca wskazanie poprzednika węzła x
nest	–	zmienne logiczna, która wskazuje zagnieżdżenie

Lista kroków:

K01:	Jeśli((x→left) = 0) i ((x→right) = 0), to idź do K05	; jeśli węzeł x posiada obu synów, to
K02:	y← removeAVL(root, pred(x))	; rekurencyjnie usuwamy poprzednik x, zapamiętując go w y
K03:	nest← false	; zerujemy zagnieżdżenie
K04:	Idź do K08	
K05:	Jeśli(x→left) ≠ nil, to y← (x→left), (x→left) ←nil Inaczej y← (x→right), (x→right) ←nil	; węzeł posiada jednego syna lub wcale
K06:	(x→bf) ← 0	; x.bf = 0, ponieważ ewentualny syn trafił do y
K07:	nest← true	; ustawiamy zagnieżdżenie
K08:	Jeśli y = nil, to idź do K15	; jeśli y istnieje, to wstawiamy go za x
K09:	(y→up) ← (x→up)	; ojcem y staje się ojciec węzła x
K10:	(y→left) ← (x→left)	; lewy syn x staje się lewym synem y
K11:	Jeśli(y→left) ≠ nil, to(y→left→up) ←y	; jeśli lewy syn istnieje, to jego ojcem staje się y
K12:	(y→right) ← (x→right)	; to samo dla prawego syna
K13:	Jeśli(y→right) ≠ nil, to(y→right→up) ←y	
K14:	(y→bf) ← (x→bf)	

K15:	Jeśli(x→up) = nil,to idź do K18	; jeśli x posiada ojca, to
K16:	Jeśli(x→up→left) =x, to (x→up→left) ←y Inaczej (x→up→right) ←y	; synem tego ojca staje się y
K17:	Idź do K19	
K18:	root←y	; inaczej y wstawiamy do korzenia
K19:	Jeśli nest= false,to idź do K43	; sprawdzamy zagnieżdżenie
K20:	z←y	; ustawiamy wskaźniki, będziemy szli w kierunku korzenia drzewa
K21:	y← (x→up)	
K22:	Dopókiy ≠ nil,wykonuj kroki K23...K42	; w pętli sprawdzamy różne przypadki
K23:	Jeśli(y→bf) ≠ 0,to idź do K26	
K24:	Jeśli(y→left) = z, to(y→bf) ← -1	
K25	Idź do K43	; przerywamy pętlę
K26:	Jeśli(((y→bf) ≠ 1)∨((y→left) ≠z))∧ (((y→bf) ≠ -1)∨((y→right) ≠z)), to idź do K31	
K27:	(y→bf) ← 0	
K28:	z←y	; przechodzimy na wyższy poziom drzewa
K29:	y← (y→up)	
K30:	Następny obieg pętli K22	
K31:	Jeśli(y→left) =z, to t← (y→right) Inaczej t← (y→left)	; t wskazuje syna y przeciwnego do z
K32:	Jeśli(t→bf) = 0, to idź do K35	
K33:	Jeśli(y→bf) = 1, to LL(root, y) Inaczej RR(root, y)	
K34:	Idź do K43	; przerywamy pętlę
K35:	Jeśli(y→bf) ≠ (t→bf), to idź do K40	
K36:	Jeśli(y→bf) = 1, to LL(root, y) Inaczej RR(root, y)	
K37:	z←t	; idziemy na wyższy poziom
K38:	y← (t→up)	
K39:	Następny obieg pętli K22	
K40:	Jeśli(y→bf) = 1,to LR(root,y) Inaczej RL(root, y)	
K41:	z← (y→up)	; idziemy na wyższy poziom i kontynuujemy pętlę
K42:	y← (z→up)	
K43:	Zakończ z wynikiem x	

Dodatkowe informacje

Opis plików:
- *avlTree.py* – plik zawiera implementację drzewa AVL
- *avlTreeTest.py* – plik testujący podstawowe operacje w zaimplementowanym drzewie AVL

Uruchamianie programu i testów
Testy uruchamiamy w terminalu komendą "avlTreeTest.py"
Główny program uruchamiamy w terminalu komendą "avlTree.py"

Alternatywne działanie programu
Aby włączyć obsługę drzewa AVL za pomocą terminala, należy odkomentować kod poniżej napisu "Prosta obsluga drzewa AVL poprzez terminal" w linii 323.

Polecenia:
1. Wstawianie węzła do drzewa AVL.
2. Usuwanie węzła z drzewa AVL.
3. Wyświetlanie zawartości drzewa AVL.
4. Wyście z programu.

Testy
Plik avlTreeTest.py, zawiera przetestowane podstawowe operacje drzewa AVL takie jak wstawianie, szukanie, usuwanie węzła. Wyniki zostały przyrównane do innej implementacji tego algorytmu na stronei https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

Bibliografia

Program jest zmodyfikowaną wersją projektu, który wykonałem na przedmiot Algorytmy i Struktury Danych w języku C++

Źródła:

- https://www.cs.usfca.edu/~galles/visualization/AVLtree.html
- https://pl.wikipedia.org/wiki/Drzewo_AVL
- http://eduinf.waw.pl/inf/alg/001_search/0119.php