Temat: Struktury drzewiste

1. Struktura słownika

Struktura danych = system relacyjny $\langle U, \{r_i\}_{i \in I} \rangle$
U – uniwersum systemu
$\{r_i\}_{i \in I}$ **- zbiór relacji (operacji) na strukturze danych**

Formalna definicja struktury danych składa się z :

- **sygnatury systemu relacyjnego,**
- **sygnatur wszystkich operacji struktury**
- **specyfikacji (opisu) operacji struktury**

Def. Strukturą słownika nazywamy system:

$$\langle E \cup D, search, insert, delete \rangle$$

gdzie:

E- niepusty zbiór możliwych danych
D- niepusty zbiór zwany zbiorem słowników

$$search : E \times D \rightarrow Boolean$$

$$insert : E \times D \rightarrow D$$

$$delete : E \times D \rightarrow D$$

(*Boolean* – typ logiczny o wartościach *true* i *false*)

Specyfikacja operacji słownikowych

1. Operacja *search*(*e*, *d*) –zwraca wartość *true*, gdy element *e* znajduje się w słowniku *d*, a *false* w przypadku przeciwnym.

2. Operacja *insert*(*e*,*d*) – wstawia element *e* do słownika *d*, o ile *search*(*e*, *d*) = *false*.

3. Operacja *delete*(*e*,*d*) – usuwa element *e* ze słownika *d*, o ile *search*(*e*, *d*) = *true*.

2. Implementacje słownika w strukturach liniowych

a) Implementacja tablicowa – tablica nieuporządkowana

- n - rozmiar słownika:
- operacja elementarna - porównania

Koszt pesymistyczny:

$$T_{\max}^{search}(n) = T_{\max}^{insert}(n) = T_{\max}^{delete}(n) = \Theta(n)$$

Koszt średni:

$$T_{śr}^{search}(n) = T_{śr}^{insert}(n) = T_{śr}^{delete}(n) = \Theta(n)$$

b) Implementacja tablicowa – tablica uporządkowana

- n - rozmiar słownika:
- operacja elementarna - porównania

Koszt pesymistyczny:

$$T_{\max}^{search}(n) = T_{\max}^{insert}(n) = T_{\max}^{delete}(n) = \Theta(\log n)$$

Koszt średni:

$$T_{śr}^{search}(n) = T_{śr}^{insert}(n) = T_{śr}^{delete}(n) = \Theta(\log n)$$

Ale !!! Trzeba przyjąć, że operacją elementarną przy realizacji operacji *insert* oraz *delete* są przestawienia elementów tablicy.

Wówczas:

$$T_{\max}^{search}(n) = \Theta(\log n),\ T_{\max}^{insert}(n) = T_{\max}^{delete}(n) = \Theta(n)$$

$$T_{śr}^{search}(n) = \Theta(\log n),\ T_{śr}^{insert}(n) = T_{śr}^{delete}(n) = \Theta(n)$$

c) Implementacja listowa

- operacja elemntarna: porównania

Koszt pesymistyczny:

$$T_{\max}^{search}(n) = T_{\max}^{insert}(n) = T_{\max}^{delete}(n) = \Theta(n)$$

Koszt średni:

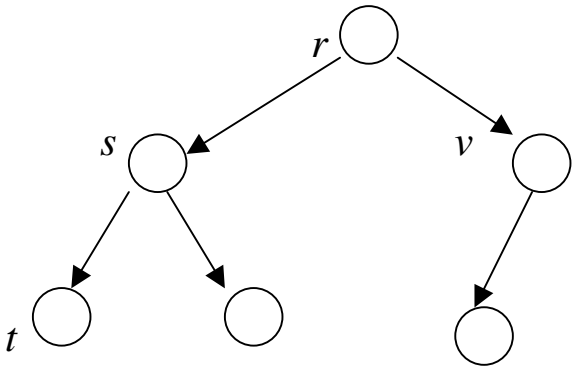
$$T_{śr}^{search}(n) = T_{śr}^{insert}(n) = T_{śr}^{delete}(n) = \Theta(n)$$

3. Drzewo BST (z ang. *Binary Search Tree*)

Def. Drzewo BST to drzewo binarne z relacją porządku symetrycznego

Def. Drzewo binarne to graf skierowany, bez cykli , z wyróżnionym węzłem zwanym korzeniem drzewa. Każdy z węzłów drzewa ma stopień równy co najwyżej dwa. Do każdego z węzłów oprócz korzenia dochodzi dokładnie jedna krawędź.

Nazewnictwo:



r- korzeń drzewa –węzeł, który nie ma poprzednika (ojca)
s- lewy potomek (syn) węzła *r*
v – prawy potomek (syn) węzła *r*
t-liść-węzeł, który nie ma ani lewego ani prawego potomka (syna)
s- korzeń lewego poddrzewa węzła *r*
v – korzeń prawego poddrzewa węzła *r*
głębokość węzła – numer poziomu, na którym leży węzeł w drzewie. oziom korzenia ma numer 0. Węzły *s* i *v* leżą na poziomie numer 1.

r-s-t – przykład ścieżki w drzewie
długość ścieżki – numer poziomu liścia w ścieżce
wysokość drzewa – numer poziomu liścia w najdłuższej ścieżce.
Wysokość drzewa na rysunku wynosi 2.

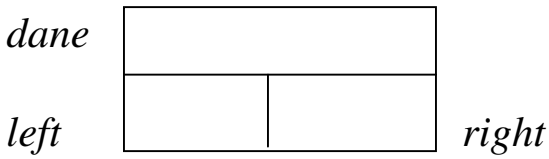
Def. Porządek symetryczny polega na tym, że dla każdego węzła drzewa *x* drzewa jest spełniony następujący warunek:
Jeżeli *y* leży w lewym poddrzewie *x*, to $key(y) \leq key(x)$.
Jeżeli *y* leży w prawym poddrzewie *x*, to $key(x) \leq key(y)$.
Relacja ≤ jest relacją porządku liniowego, a $key(x)$ – jest wartością pola (klucza) danych, na którym relacja ≤ jest określona.

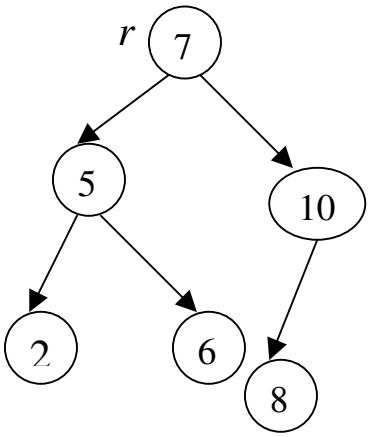
Będziemy zakładali w dalszych rozważaniach, że dane umieszczane w drzewie to liczby całkowite.

Przykład

Węzeł drzewa BST to adres na strukturę (rekord) z polami:

- *dane* – pole do przechowywania danych,
- *left, right* – adresy węzłów potomnych





r – korzeń drzewa

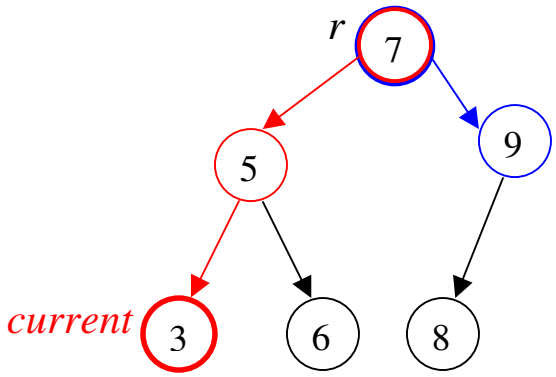
4. Algorytmy operacji słownikowych na drzewie BST

a) Operacja *search*

search(*e*, *r*); *e* – liczba szukana, *r* – korzeń drzewa

search(*e*,*r*)=1, gdy *e* jest w słowniku,
search(*e*,*r*)=0, gdy *e* nie należy do słownika

```
current=r;
while (current!=NULL)
  if (current->dane==e)
    return 1;
  else
    if (current->dane>e)
      current=current->left
    else  current=current->right
 if (current==NULL) return 0;
```

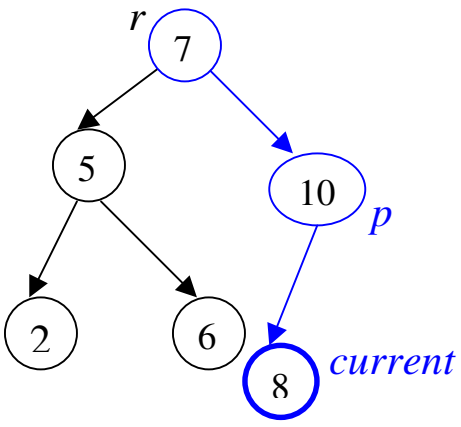


current=NULL

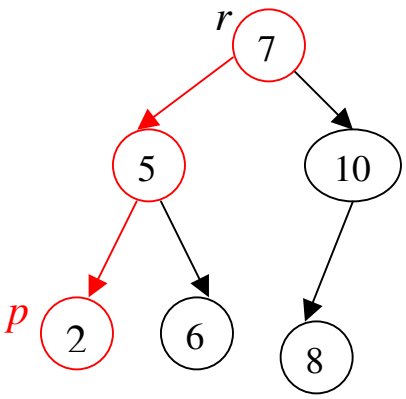
search(3, *r*)=1 *search*(10, *r*)=0

W implementacji operacji *insert* i *delete* niezwykle przydatna okazuje się zmodyfikowana funkcja *search* (*search1*), która zwraca adres węzła, w którym znajduje się poszukiwany element *e* oraz dodatkowo adres *p* – ojca węzła z daną *e*. W szczególności, gdy wynikiem funkcji jest korzeń drzewa *r*, to *p* jest adresem pustym. Gdy danych *e* nie ma w słowniku, to zwracany jest adres pusty, a *p* ustawia się na adresie „potencjalnego ojca" węzła z daną *e*.

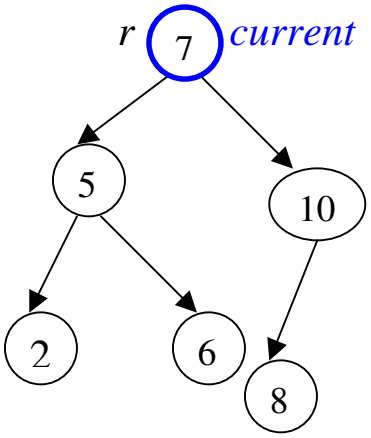
*search*1(*e*, *r*, *p*);



current=*search1*(8, *r*, *p*)



search1(1, *r* , *p*)=NULL



current=*search1*(7, *r*, *p*)

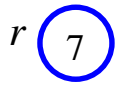
p=NULL

b) Operacja *insert*

insert(*e*, *r*);

Przypadki:

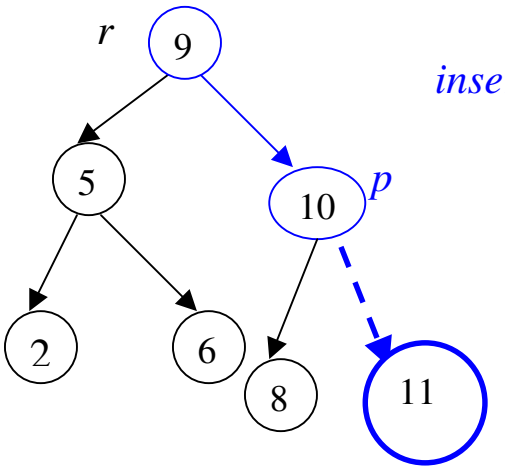
- drzewo jest puste,



insert(7, *r*)

- dana *e* już jest w słowniku – wówczas drzewo się nie zmienia

- dana *e* jeszcze nie znajduje się w słowniku – wówczas nowy węzeł z daną *e* jest "podwiązywany do drzewa " jako liść i odpowiedni (lewy albo prawy) potomek węzła *p* zwróconego w funkcji *search1*.



insert(11,*r*)

c)operacja *delete*

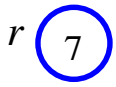
delete (*e*, *r*);

Przypadki:

- Element *e* nie należy do słownika.
  Wywołanie funkcji *search1*(*e*, *r*, *p*) zwraca wówczas adres pusty.
  Słownik w tym przypadku nie zmienia się po wykonaniu operacji *delete*.

- Element *e* znajduje się w korzeniu drzewa i korzeń drzewa jest jedynym węzłem drzewa. Wywołanie funkcji *search1*(*e*, *r*, *p*) zwraca wówczas adres *r* i adresy *left* i *right* w korzeniu *r* są puste.
  Słownik w tym przypadku staje się pusty do wykonaniu operacji *delete*.

Przed *delete*(7, *r*)

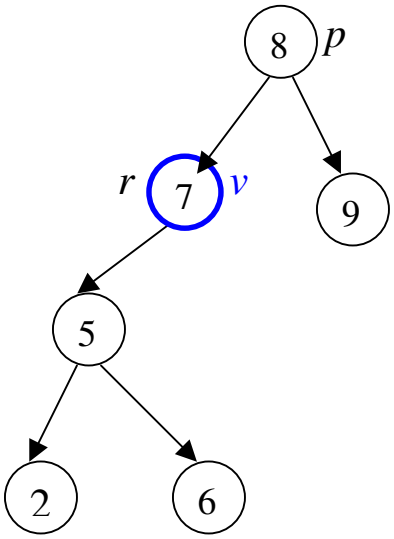


Po *delete*(7, *r*)

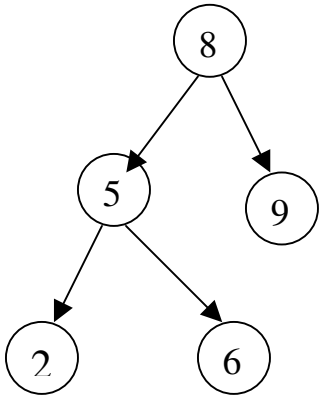
r = NULL

- Element *e* znajduje się w węźle o adresie *v* i węzeł *v* ma puste lewe albo prawe podrzewo. Wywołanie funkcji search1(*e*, *r*, *p*) zwraca wtedy adres niepusty *v* i albo adres *left* albo *right* w węźle *v* jest pusty. Ustalmy, że adres *right* w węźle *v* jest pusty. Wówczas nowym potomkiem węzła *p* będzie lewy potomek usuwanego węzła *v*.
  Może się zdarzyć, że korzeń drzewa musi być uaktualniony w tym przypadku. Zachodzi to wtedy, gdy *v*=*r*.

Przed *delete*(7, *r*)

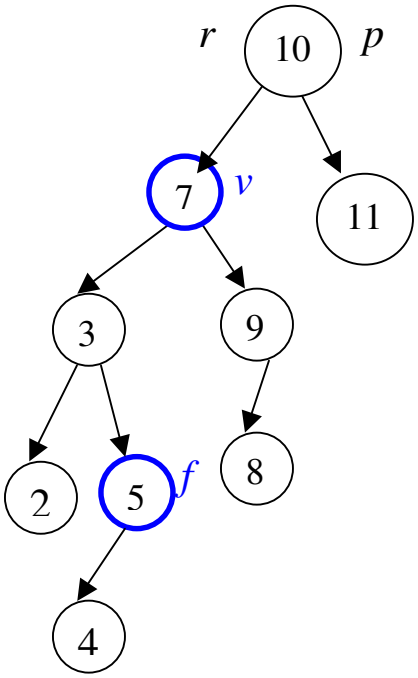


Po *delete*(7, *r*)

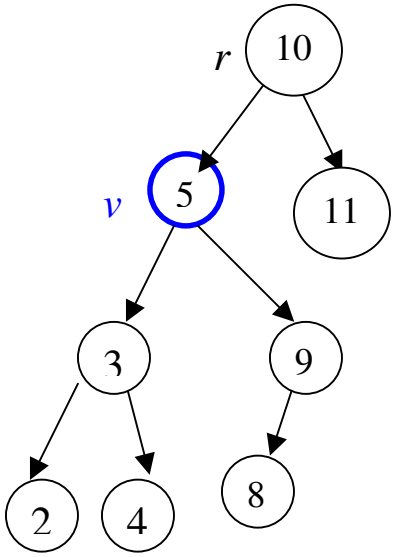


- Element *e* znajduje się w węźle o adresie *v* i węzeł *v* ma niepuste lewe albo prawe podrzewo. Wywołanie funkcji search1(*e*, *r*, *p*) zwraca wtedy adres niepusty *v* adresy *left* i *right* w węźle *v* są niepuste.
  Ustalamy wówczas adres takiego węzła, który zawiera największy element lewego podrzewa węzła *v* albo najmniejszy element prawego poddrzewa węzła *v*. Oznaczmy znaleziony węzeł przez *f*. Dane z węzła *f* kopiujemy do węzła *v*, a sam węzeł *f* usuwamy wg algorytmu z poprzedniego przypadku, gdyż ma on na pewno puste co najmniej jedno z poddrzew.
  Może się zdarzyć, że korzeń drzewa musi być uaktualniony w tym przypadku. Zachodzi to wtedy, gdy *v*=*r*.

Przed *delete*(7, *r*)



Po *delete*(7, *r*)



5. Koszt operacji słownikowych na drzewie BST

Operacja elementarna: porównania między elementem wstawianym, wyszukiwanym lub usuwanym, a elementami słownika.

Rozmiar zadania: *n* – ilość danych w słowniku przed realizacją operacji słownikowej.

Koszt operacji słownikowych zasadniczo zależy od kosztu operacji *search* (*search*1):

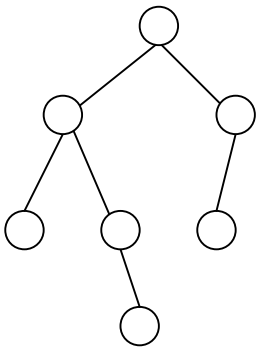
Koszt *search:*

- pesymistyczny $T_{\max}(n) = n = \Theta(n)$ - przypadek drzewa zdegenerowanego
- średni O(*logn*) – określony na postawie wyników eksperymentów, ale nie udowodniony.

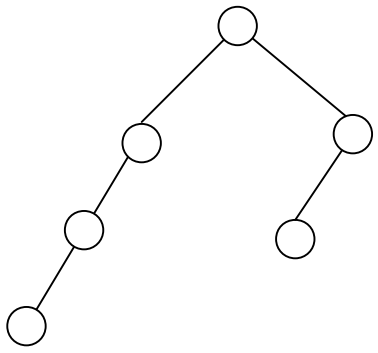
6. Drzewa AVL

Nazwa drzewo AVL pochodzi od nazwisk dwóch specjalistów teorii algorytmów i struktur danych: Adelsona-Velskii i Landisa.

Def. Drzewo AVL to wyważone drzewo BST. Drzewo jest wyważone, kiedy dla każdego wierzchołka, wysokości dwóch jego poddrzew różnią się co najwyżej o jeden.



Drzewo wyważone



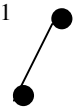
Drzewo, które nie jest wyważone

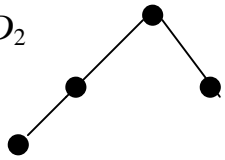
7. Wysokość drzewa AVL

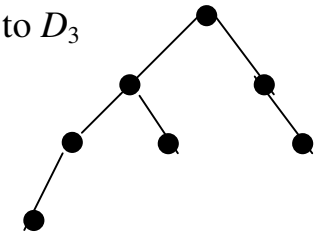
Lemat
Wysokość drzewa AVL o n wierzchołkach $(n \geq 1)$ jest nie większa niż $2log_2 n$.

Dowód:
Niech D_h będzie drzewem AVL o wysokości h, które ma najmniejszą, możliwą liczbę wierzchołków wśród drzew AVL o wysokości h.
Niech n_h będzie liczbą wierzchołków w drzewie D_h.
Gdy:

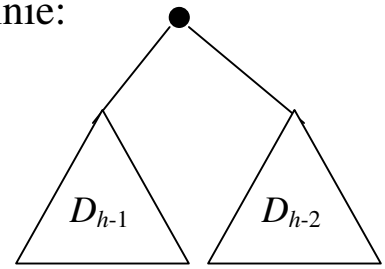
$h = 0$, to D_0  $n_0 = 1$

$h = 1$, to D_1  $n_1 = 2$

$h = 2$, to D_2  $n_2 = 4$

$h = 3$, to D_3  $n_3 = 7$

Ogólnie:



$n_h = n_{h-1} + n_{h-2} + 1$

Udowodnimy przez indukcję, że:

$$n_h \geq 2^{\frac{h}{2}} \quad \text{dla } h > 1$$

Założenie indukcyjne:

$$n_{h-1} \geq 2^{\frac{h-1}{2}}$$

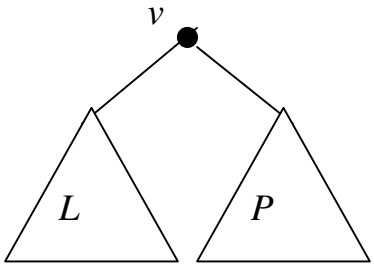
$$n_h = n_{h-1} + n_{h-2} + 1 \geq 2^{\frac{h-1}{2}} + 2^{\frac{h-2}{2}} + 1 = 2^{\frac{h}{2}} \cdot 2^{-\frac{1}{2}} + 2^{\frac{h}{2}} \cdot 2^{-1} + 1 = 2^{\frac{h}{2}} \left(\frac{1}{2} + \frac{\sqrt{2}}{2} \right) + 1 \geq 2^{\frac{h}{2}}$$

Czyli $\log_2 n_h \geq \frac{1}{2} h$

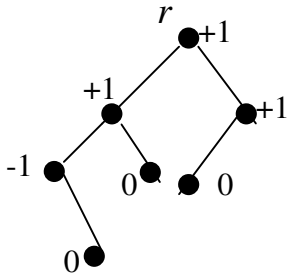
Stąd $h \leq 2 \log_2 n_h$

8. Budowa węzła drzewa AVL

Struktura węzła drzewa AVL jest wzbogacona o wagę *w* taką, że: *w*(*drzewo_puste*)= -1 , *w*(*v*)=*h*(*L*) - *h*(*P*), gdzie *h*(*L*) i *h*(*P*) to odpowiednio wysokość lewego i prawego poddrzewa węzła *v*.



Przykład wartości wagi *w* dla węzłów drzewa o korzeniu *r*



9. Operacje słownikowe na drzewie AVL

- *search*(*e*, *d*) - operacja realizowana tak samo jak dla drzewa BST

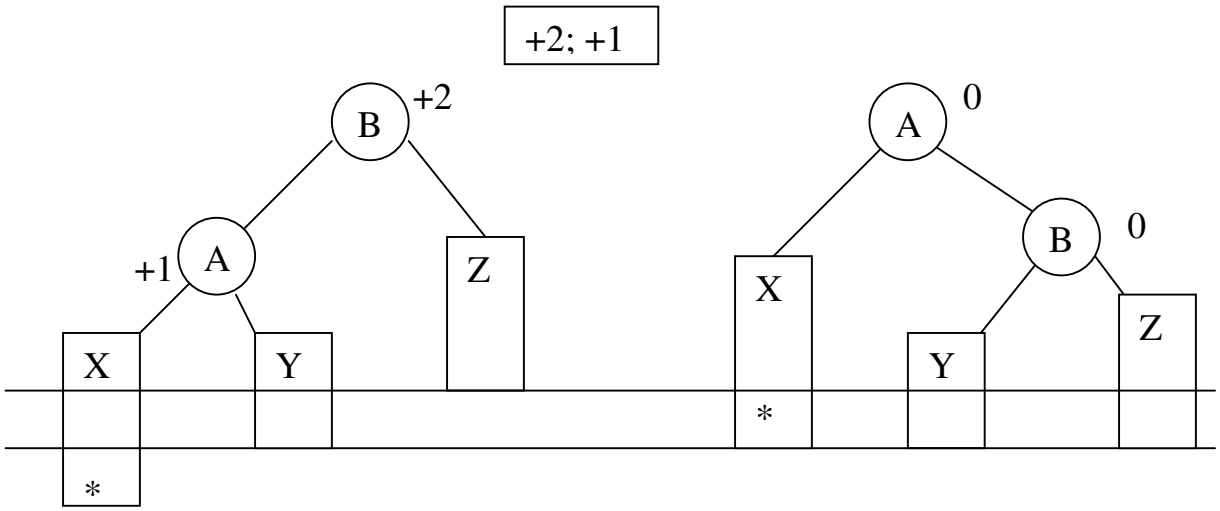
- *insert*(*e*, *d*) - operacja realizowana w pierwszej fazie tak samo jak dla drzewa BST. Druga faza to ewentualne wyważenie powstałego drzewa.

- *delete*(*e*, *d*) - operacja realizowana w pierwszej fazie tak samo jak dla drzewa BST. Druga faza to ewentualne wyważenie powstałego drzewa.

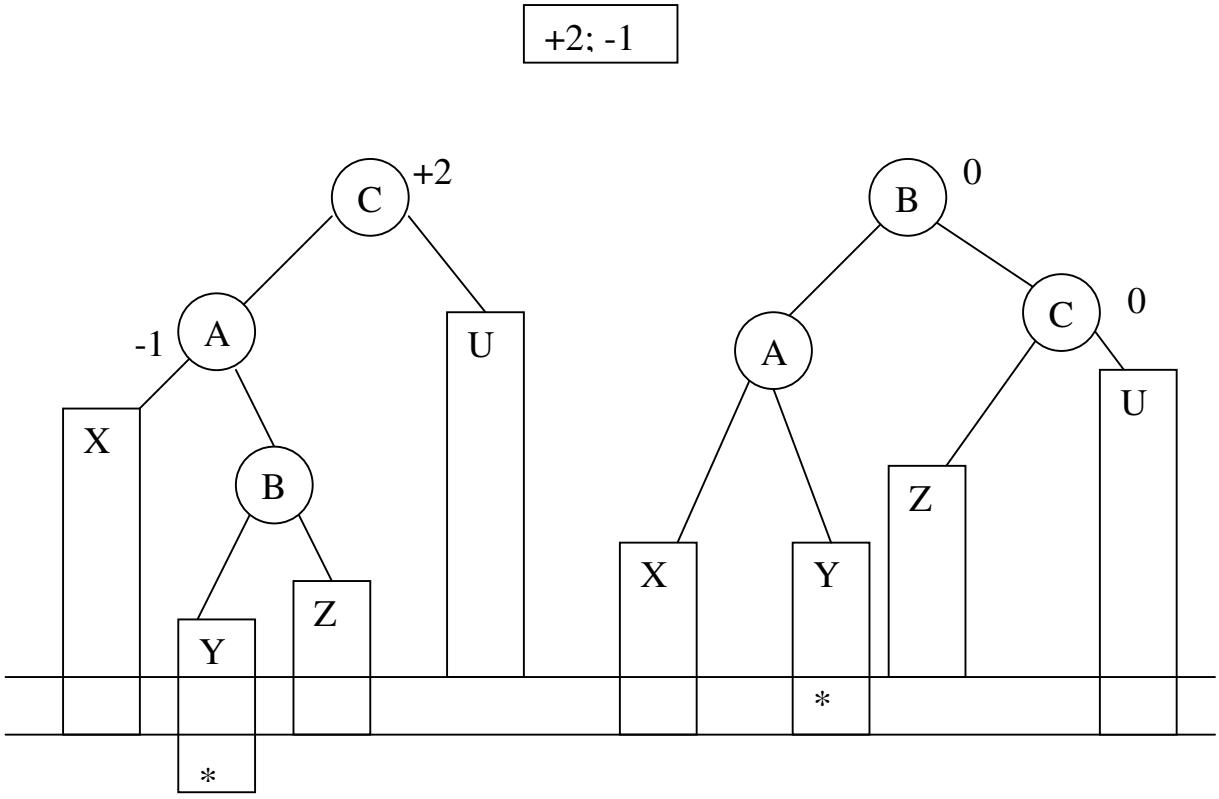
Podczas realizacji pierwszej fazy operacji *insert* i *delete* należy na stosie odłożyć adresy węzłów ze ścieżki, którą „odwiedza" wywołanie funkcji *search1*. Stos jest niezbędny do zrealizowania drugiej fazy algorytmu, tj. sprawdzenia warunku wyważenia na podstawie wag węzłów ze stosu. Zatem funkcja *search1* zwracać powinna, w przypadku drzew AVL, adres wierzchołka takiego stosu, zamiast adresu *p*.

10. Rotacje prowadzące do wyważenia drzewa po usunięciu albo wstawieniu elementu

Rotacje w prawo

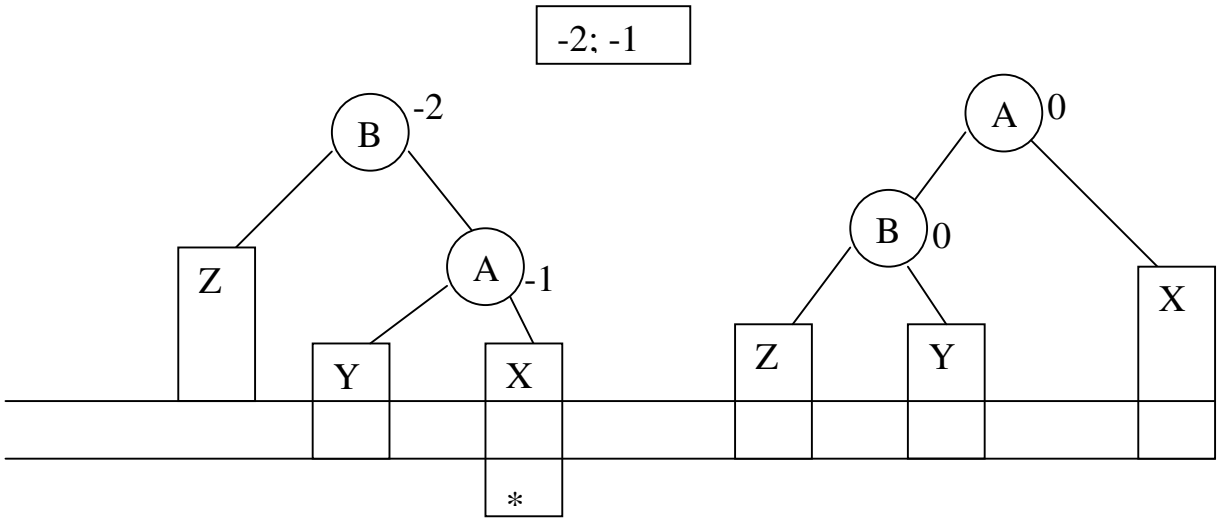


pojedyncza rotacja w prawo względem B

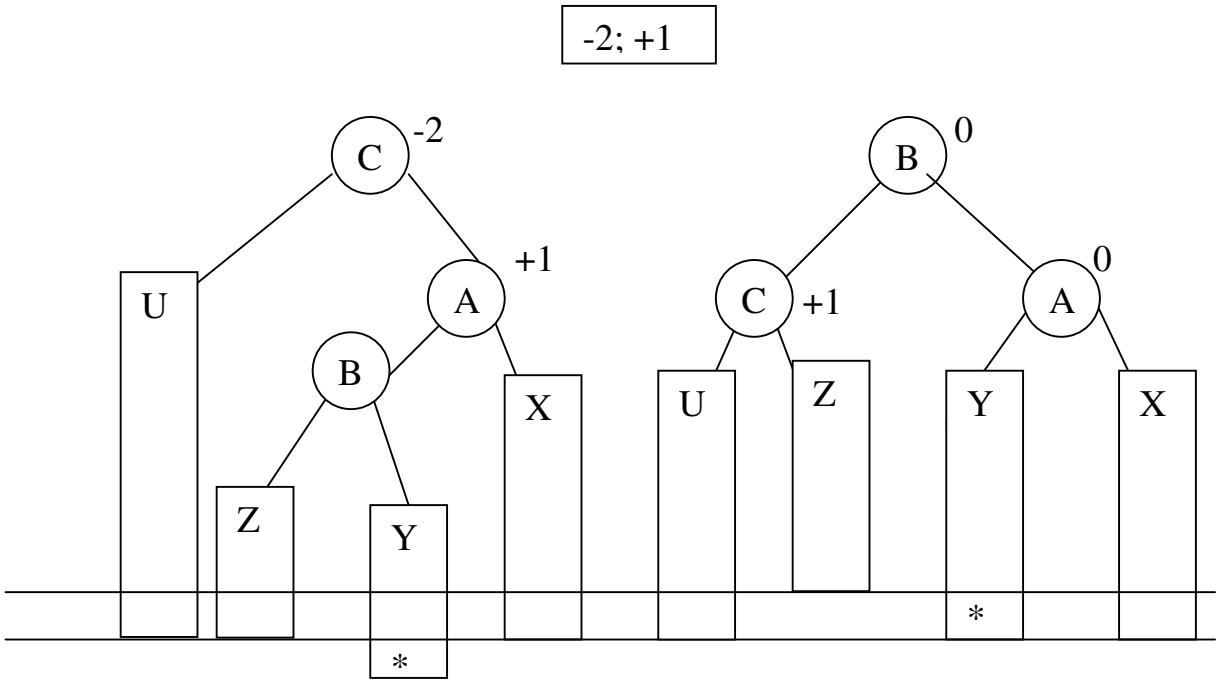


podwójna rotacja w prawo: rotacja pojedyncza w lewo względem A i pojedyncza w prawo względem C

Rotacje w lewo



pojedyncza rotacja w lewo względem B



rotacja podwójna w lewo: pojedyncza w prawo względem A
i pojedyncza w lewo względem C

Uwaga

Rotacje w procesie wyważania wykonywane są tylko na jednej ścieżce w drzewie.

- Operacja *delete* realizowana na drzewie AVL wymaga wykonania co najwyżej $2log_2 n$ rotacji.
- Operacja *insert* realizowana na drzewie AVL wymaga wykonania co najwyżej jednej rotacji.
- Może się zdarzyć, że po zrealizowaniu pierwszej fazy operacji *delete* zajdzie konieczność wykonania rotacji o układzie wag +2;0 albo –2;0. Wykonujemy wówczas odpowiednio pojedyncza rotacje w lewo albo prawo względem węzła z wagą odpowiednio +2 albo -2.

Wniosek

Ponieważ drzewo AVL ma pesymistyczną wysokość O(*logn*), a ewentulane rotacje wykonywane podczas wstawiania albo usuwania elementów realizowane są kosztem O(*logn*), to operacje słownikowe zrealizowane na drzewie AVL mają koszt pesymistyczny O(*logn*).