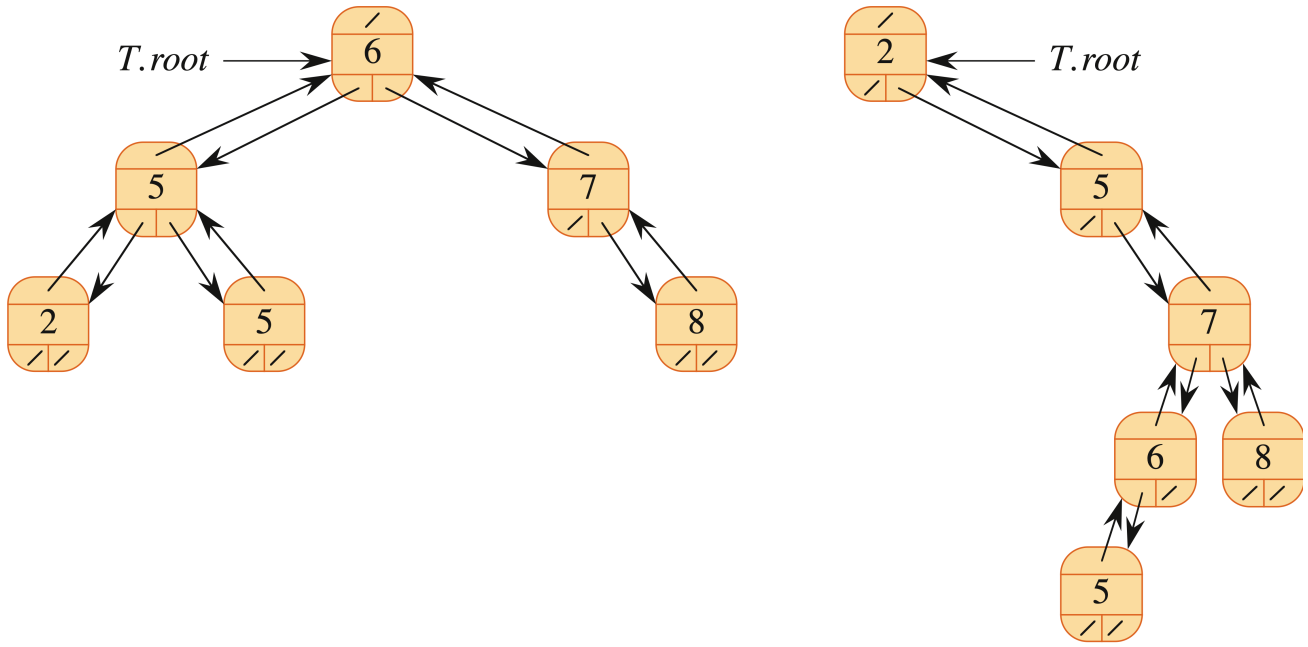
Binarne drzewa przeszukiwań BST

Binarne drzewa przeszukiwań (*Binary Search Trees, BST*) są binarnymi drzewami zbudowanymi jako dynamiczna struktura wskaźnikowa, gdzie każdy węzeł zawiera klucz, dane związane z tym kluczem, oraz wskaźniki do obu potomków, oraz rodzica.

Podstawowa własność **uporządkowania** drzew BST: dla każdego węzła x drzewa, jeśli y jest węzłem w lewym poddrzewie x to $y.key \leq x.key$, a jeśli z jest węzłem w prawym poddrzewie x to $z.key \geq x.key$.

Jak można zauważyć na poniższych przykładach, drzewa BST mogą być zbudowane z różnym stopniem **zrównoważenia**. Oznacza to, że dwa drzewa z tą samą liczbą węzłów mogą mieć znacząco różną wysokość!



Przegląd drzewa

Przeglądem drzewa nazywamy procedurę odwiedzenia wszystkich węzłów drzewa typowo implementowaną rekurencyjnie. Celem odwiedzin każdego węzła jest wykonanie na nim jakichś operacji. Tutaj ograniczymy się do wyświetlenia zawartego w nim klucza.

Poniższa procedura implementuje przegląd drzewa BST w porządku *inorder* wyświetlający wszystkie klucze drzewa w porządku sortowania:

```
INORDER-TREE-WALK(x)
1  if x ≠ NIL
2        INORDER-TREE-WALK(x.left)
3        print x.key
4        INORDER-TREE-WALK(x.right)
```

Poza porządkiem *inorder* określa się jeszcze porządki przeglądania: *preorder* i *postorder*. Pierwszy wyświetla klucz węzła przed wykonaniem rekurencyjnych przeglądów obu potomków, a drugi rozpoczyna od rekurencyjnego przeglądu potomków, a klucz zawarty w danym węźle wyświetla na końcu.

Przegląd drzewa ma czas działania $\Theta(n)$ gdzie n jest liczbą kluczy w drzewie.

Operacje przeszukiwania drzewa BST

Podstawową operacją wykonywaną na drzewie BST jest **przeszukiwanie** drzewa, którego celem jest zlokalizowanie elementu z daną wartością klucza, lub stwierdzenie, że nie ma takiej w drzewie.

```
TREE-SEARCH(x, k)
1  if x == NIL or k == x.key
2        return x
3  if k < x.key
4        return TREE-SEARCH(x.left, k)
5  else return TREE-SEARCH(x.right, k)

ITERATIVE-TREE-SEARCH(x, k)
1  while x ≠ NIL and k ≠ x.key
2        if k < x.key
3              x = x.left
4        else x = x.right
5  return x
```

Innymi rodzajami przeszukiwania są: znajdowanie minimalnego i maksymalnego klucza całego drzewa, i znajdowanie bezpośredniego poprzednika i następnika danego klucza.

Znajdowanie minimalnego i maksymalnego elementu drzewa BST

Minimalny klucz drzewa BST znajduje się w jego skrajnym lewym elemencie, a maksymalny w skrajnym prawym:

```
TREE-MINIMUM(x)
1  while x.left ≠ NIL
2        x = x.left
3  return x

TREE-MAXIMUM(x)
1  while x.right ≠ NIL
2        x = x.right
3  return x
```

Znajdowanie następnika danego elementu w BST

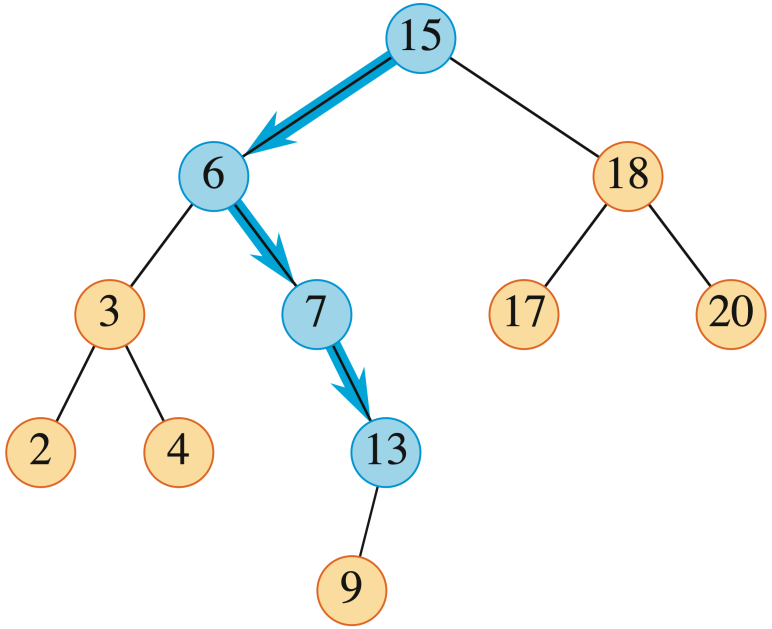
Następnikiem danego elementu w BST nazywamy element odwiedzany bezpośrednio po danym elemencie w przeglądzie *inorder*.

O ile wszystkie klucze w drzewie są różne, to następnik jest również elementem z następnym kluczem w kolejności sortowania. Zwykle będziemy robili założenie o różnowartościowości kluczy, ale definicja następnika podana powyżej jest niezależna od takiego założenia.

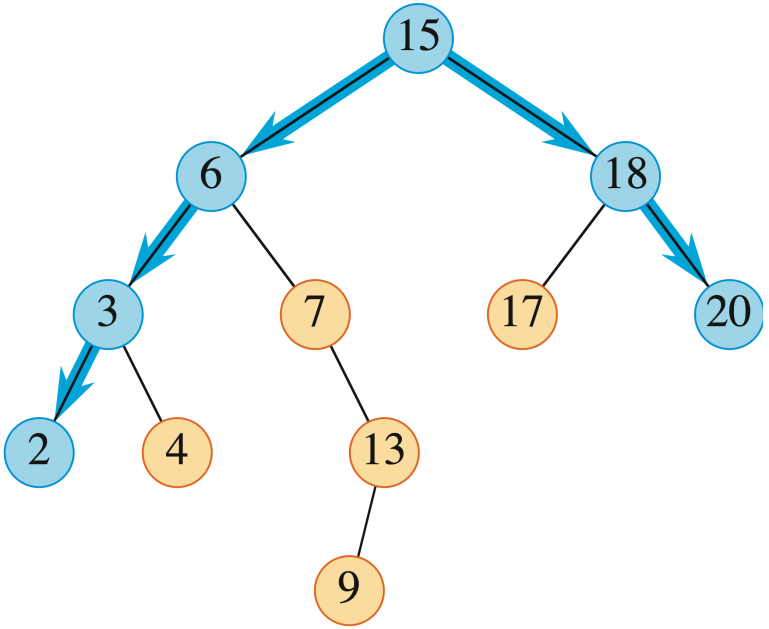
Znajdowanie następnika danego elementu jest łatwe gdy ten element ma niepuste prawe poddrzewo — jest to wtedy minimalny element tego poddrzewa. Jednak gdy prawe poddrzewo elementu jest puste, znalezienie jego następnika jest nieco trudniejsze. Ilustruje to poniższy algorytm, oraz przykłady na następnej stronie:

```
TREE-SUCCESSOR(x)
1  if x.right ≠ NIL
2        return TREE-MINIMUM(x.right) // leftmost node in right subtree
3  else // find the lowest ancestor of x whose left child is an ancestor of x
4        y = x.p
5        while y ≠ NIL and x == y.right
6              x = y
7              y = y.p
8        return y
```

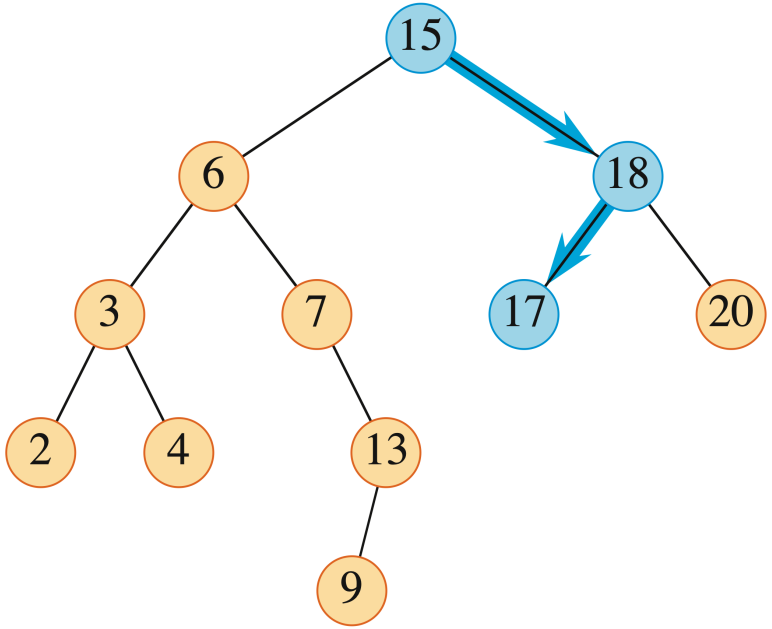
(a) poszukiwanie klucza 13



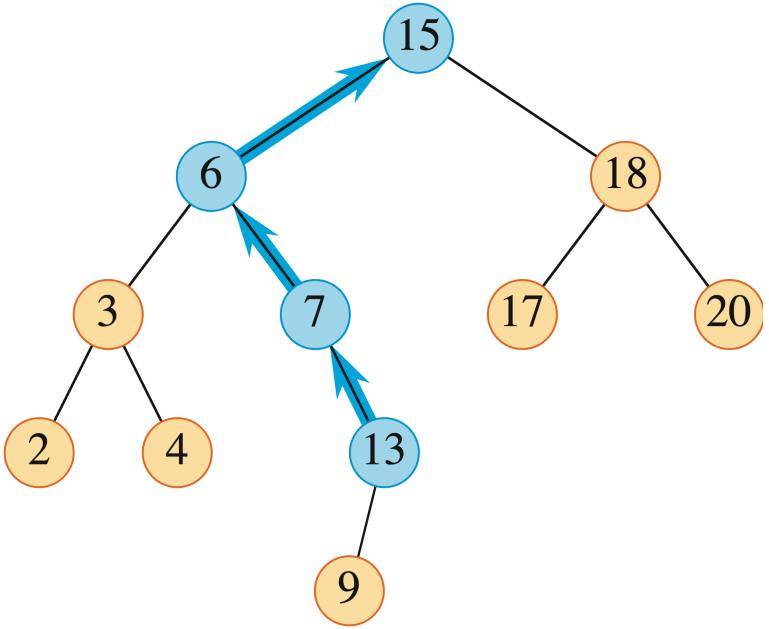
(b) poszukiwanie minimum i maksimum



(c) poszukiwanie następnika klucza 15



(d) poszukiwanie następnika klucza 13



Operacje przeszukiwania drzew BST — czas działania

Operacja znajdowania **poprzednika** (TREE-PREDECESSOR) nie została tu przedstawiona, lecz jest całkowicie symetryczna do powyższej operacji TREE-SUCCESSOR (patrz ćwiczenie).

Można udowodnić, że wszystkie przedstawione operacje przeszukiwania BST: TREE-SEARCH, TREE-MINIMUM, TREE-MAXIMUM, TREE-SUCCESSOR, i TREE-PREDECESSOR działają w czasie $O(h)$ gdzie h jest wysokością drzewa.

Wysokość drzewa BST o n węzłach może się wahać w przybliżeniu od $\log_2 n$ do n.[1] Oznacza to, że czas działania operacji przeszukiwania BST wyrażony w funkcji liczby elementów jest zawarty pomiędzy $O(\log_2 n)$ do $O(n)$.

Ten przedział jest bardzo szeroki, i pojawia się pytanie od czego zależy wysokość konkretnego drzewa z daną liczbą elementów n. Odpowiedź brzmi, że zależy od tego jak udało się dane drzewo zbudować.

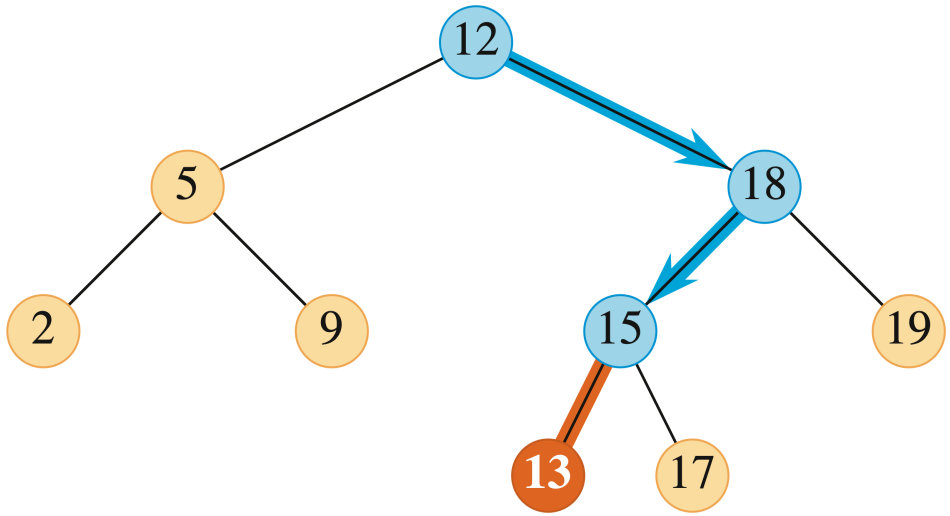
[1] Dokładnie: od $(\lceil \log_2(n+1) \rceil - 1)$ do $(n-1)$.

Operacje budowy drzew BST — dodawanie węzła

Idea dodawania węzła z daną wartością klucza do drzewa BST polega na tym, że najpierw próbujemy znaleźć ten klucz w drzewie. Przy założeniu, że klucze nie mogą się powtarzać, wynikiem tego szukania będzie, o ile drzewo jest niepuste, węzeł z pustym poddrzewem (lewym lub prawym), w którym powinien znaleźć się wstawiany element. Wtedy jest od podczepiany w tym miejscu:

```
TREE-INSERT(T, z)
 1  x = T.root                  // node being compared with z
 2  y = NIL                     // y will be parent of z
 3  while x ≠ NIL               // descend until reaching a leaf
 4        y = x
 5        if z.key < x.key
 6              x = x.left
 7        else x = x.right
 8  z.p = y                     // found the location—insert z with parent y
 9  if y == NIL
10        T.root = z            // tree T was empty
11  elseif z.key < y.key
12        y.left = z
13  else y.right = z
```

Przykład dodawania węzła z kluczem 13 do drzewa BST:

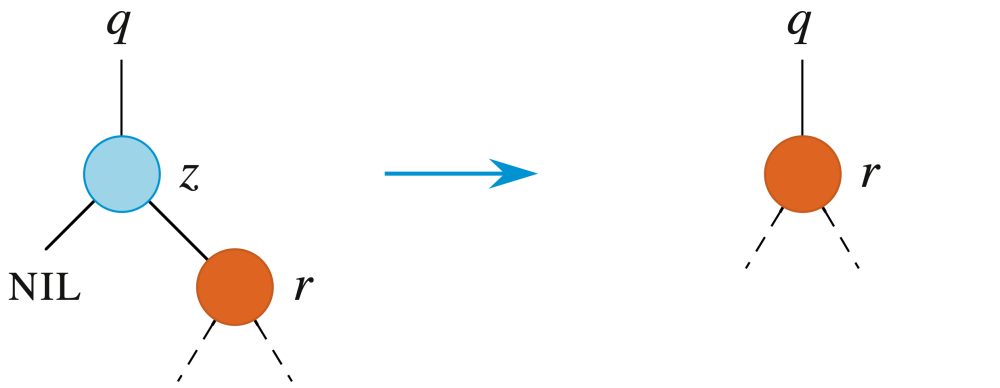


Podobnie jak operacje przeszukiwania drzewa BST, dodawanie węzła działa w czasie $O(h)$

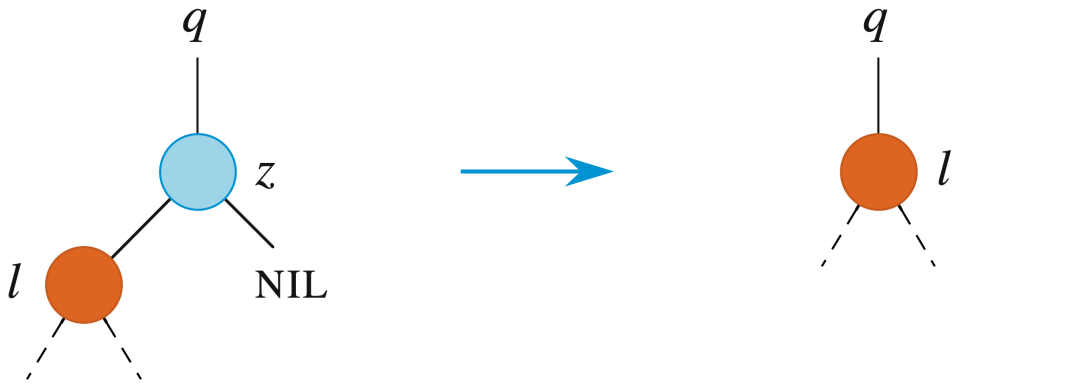
Operacje budowy drzew BST — usuwanie węzła

Operacja usuwania węzła jest nieco bardziej skomplikowana. Usunięcie węzła jest trywialne gdy jest on liściem drzewa — wtedy po prostu jest on zastępowany wartością NIL. Dość proste są również przypadki, gdy usuwany węzeł nie jest liściem, ale ma tylko jedno poddrzewo (lewe lub prawe) — wtedy to poddrzewo jest w całości „podnoszone" do miejsca usuwanego węzła. Jednak gdy usuwany węzeł jest węzłem wewnętrznym w drzewie, to po jego usunięciu musi się zmienić struktura drzewa!

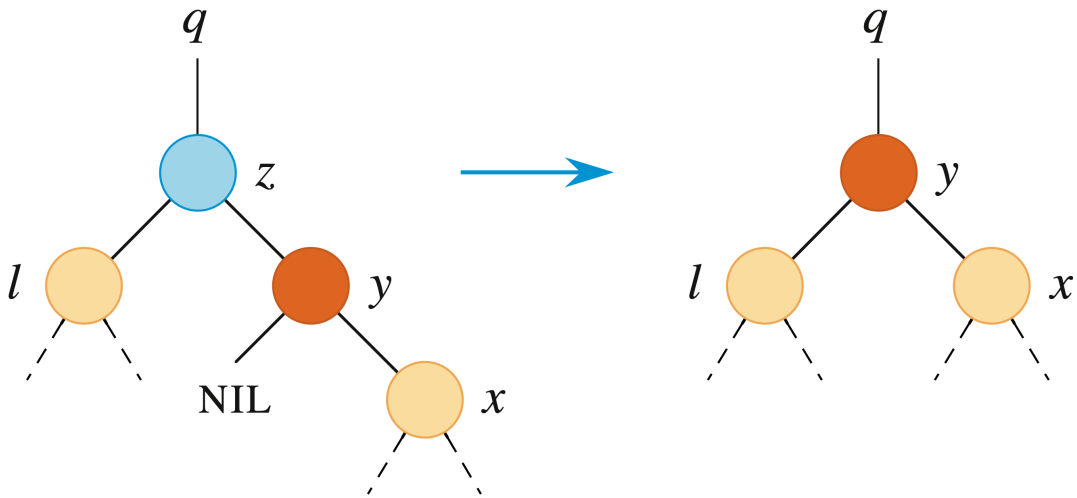
(a) jeśli z nie ma lewego potomka to prawy potomek (razem z całym swoim poddrzewem) podnoszony jest na miejsce z; ten przypadek obejmuje również sytuację, gdy $r = \text{NIL}$, czyli z jest liściem



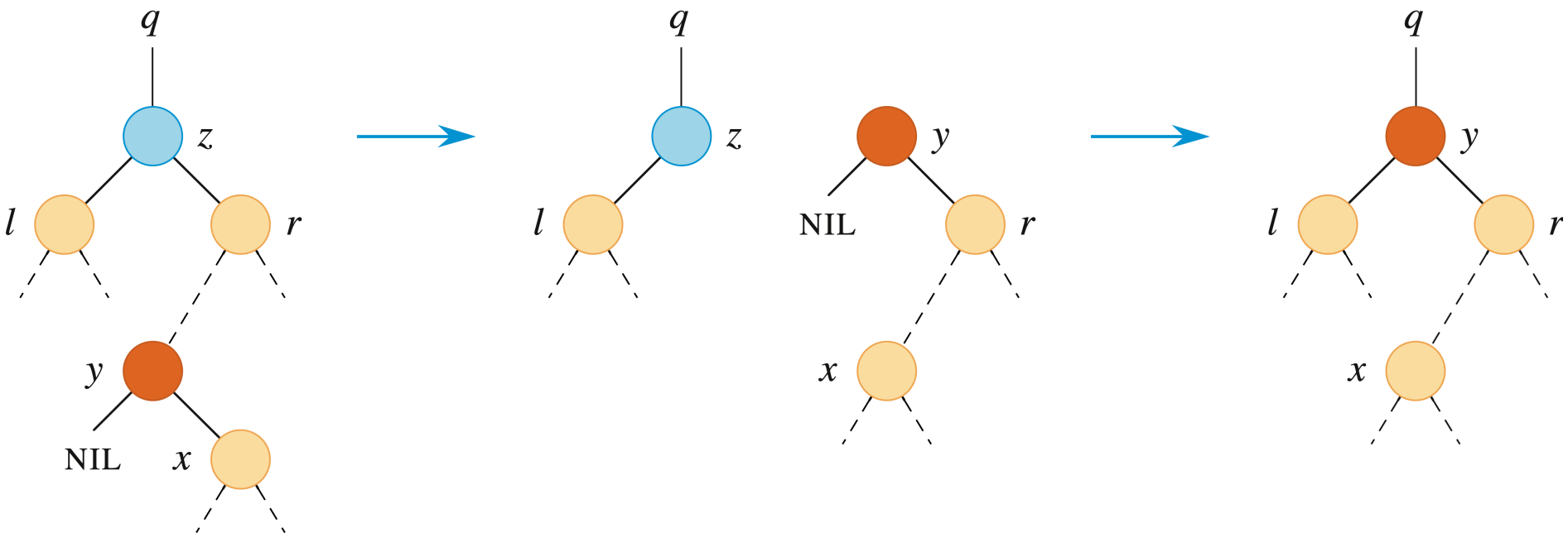
(b) wpw (z ma lewego potomka), jeśli z nie ma prawego potomka to lewy potomek (razem z całym poddrzewem) podnoszony jest na miejsce z



(c) wpw (z ma obu potomków), znajdź jego następnik y, i jeśli y jest bezpośrednim potomkiem z (prawym), to y nie ma lewego potomka, i wtedy y może być również podniesiony na miejsce z



(d) wpw (z ma obu potomków, i jego następnik y jest jego dalszym potomkiem, w poddrzewie zakorzenionym w prawym potomku z-a nazwanym r), to y nadal nie ma lewego potomka, i można przenieść go w miejsce z instalując poddrzewo r jako prawego potomka z — jest to operacja **transplantacji**



Jako uzasadnienie poprawności transplantacji zauważmy, że klucz y jest mniejszy od wszystkich innych kluczy w poddrzewie r, wszystkie klucze zawarte w poddrzewie x są mniejsze od klucza r, a całe prawe poddrzewo klucza r nie uległo zmianie.

Operacje budowy drzew BST — transplantacja węzła

W trakcie usuwania węzła drzewa, poddrzewa są przenoszone wewnątrz drzewa BST. Dotyczy to przypadku (d) powyżej, ale również (c) w nieco prostszy sposób, oraz przypadki (a) i (b) są również przypadkami takiego przenoszenia poddrzew w inne miejsce w drzewie. Poniższa procedura TRANSPLANT zastępuje węzeł u (razem ze swoim poddrzewem, jeśli takie ma) jako potomka swojego rodzica, poddrzewem zakorzenionym w węźle v. W wyniku tej transplantacji rodzic węzła u staje się rodzicem węzła v, a v staje się jego potomkiem po odpowiedniej stronie. Procedura dopuszcza by v był pustym poddrzewem NIL:

```
TRANSPLANT(T, u, v)
1  if u.p == NIL
2        T.root = v
3  elseif u == u.p.left
4        u.p.left = v
5  else u.p.right = v
6  if v ≠ NIL
7        v.p = u.p
```

TRANSPLANT nie sprawdza uporządkowania kluczy, ani nie aktualizuje potomków węzła v. Obie te czynności pozostają w gestii procedury wywołującej.

Operacje budowy drzew BST — usuwanie węzła: pseudokod

Teraz możemy zapisać pseudokod usuwania węzła w czterech przypadkach z użyciem procedury TRANSPLANT:

```
TREE-DELETE(T, z)
 1  if z.left == NIL
 2       TRANSPLANT(T, z, z.right)          // replace z by its right child
 3  elseif z.right == NIL
 4       TRANSPLANT(T, z, z.left)           // replace z by its left child
 5  else y = TREE-MINIMUM(z, right)         // y is z's successor
 6       if y ≠ z.right                     // is y farther down the tree?
 7            TRANSPLANT(T, y, y.right)     // replace y by its right child
 8            y.right = z.right             // z's right child becomes
 9            y.right.p = y                 //       y's right child
10       TRANSPLANT(T, z, y)                // replace z by its successor y
11       y.left = z.left                    // and give z's left child to y,
12       y.left.p = y                       //       which had no left child
```

Ponieważ procedura TRANSPLANT działa w czasie stałym $O(1)$, podobnie jak wszystkie kroki powyżej, z wyjątkiem wywołania TREE-MINIMUM, to TREE-DELETE działa w czasie $O(h)$ gdzie h jest wysokością drzewa.

Warunek niepowtarzalności kluczy

Większość procedur operujących na drzewach BST zakłada, że klucze elementów zapisanych na drzewie są unikalne. A nawet procedury, które dopuszczają wielokrotne wystąpienie danego klucza, przy wykonywaniu operacji na elemencie z daną wartością klucza, zwykle nie pozwalają na wybranie instancji, na której ta operacja będzie wykonywana (na przykład przy usuwaniu węzła).

Implementując operacje na BST musimy przyjąć albo odrzucić to założenie o niepowtarzalności kluczy. Gdy je przyjmiemy, to, na przykład, przy dodawaniu klucza do drzewa, gdy zamiast pustego miejsca w liściu, odpowiadającego pozycji danego klucza, znajdziemy tam już istniejący element o tej samej wartości, możemy:

- uznać to za błąd (naruszenie założenia) i zgłosić wyjątek, lub
- uznać to za wariant procedury dodawania elementu, i nadpisać istniejący element nową wartością.

Krótkie podsumowanie — pytania sprawdzające

1. Rozważ wyszukiwanie klucza 363 w drzewie BST zawierającym liczby pomiędzy 1 i 1000. Które z poniższych sekwencji nie mogą być sekwencją sprawdzanych kluczy:
   a. 2, 252, 401, 398, 330, 344, 397, 363.
   b. 924, 220, 911, 244, 898, 258, 362, 363.
   c. 925, 202, 911, 240, 912, 245, 363.
   d. 2, 399, 387, 219, 266, 382, 381, 278, 363.
   e. 935, 278, 347, 621, 299, 392, 358, 363.

2. Wzorując się na przedstawionej na stronie 5 funkcji TREE-SUCCESSOR napisz pseudokod funkcji znajdowania poprzednika TREE-PREDECESSOR.

3. Dla zbioru kluczy $\{1, 4, 5, 10, 16, 17, 21\}$ skonstruuj jego permutacje, takie, że gdy procedura TREE-INSERT zostanie wywołana po kolei na kluczach w danej permutacji, zbuduje drzewa BST o wysokości: 2, 3, 4, 5, i 6.

4. Profesor Kilmer twierdzi, że odkrył szczególną własność drzew BST. Załóżmy, że poszukiwanie klucza k w drzewie BST kończy się w liściu. Rozważ trzy zbiory: A — zbiór kluczy na lewo od ścieżki przeszukiwania, B — zbiór kluczy na ścieżce przeszukiwania, i C — zbiór kluczy na prawo od ścieżki przeszukiwania. Profesor twierdzi, że dla dowolnych trzech kluczy $a \in A, b \in B$ i $c \in C$ spełnione jest $a \leq b \leq c$. Znajdź najmniejszy możliwy kontrprzykład dla twierdzenia profesora.

5. Udowodnij, że jeśli węzeł drzewa BST ma dwóch niepustych potomków, to jego następnik nie ma lewego potomka, a jego poprzednik nie ma prawego potomka.

6. DIa drzewa BST z wszystkimi różnymi kluczami, niech x będzie jego liściem, a y rodzicem x. Pokaż, że $y.key$ jest albo najmniejszym kluczem w T większym niż $x.key$, lub największym kluczem w T mniejszym niż $x.key$.

7. Rozważ operacje usuwania dwóch różnych elementów z drzewa BST. Czy postać drzewa po tych dwóch operacjach będzie identyczna, niezależnie od kolejności ich wykonania? Uzasadnij, że tak będzie, lub podaj kontrprzykład.

Literatura i materiały pomocnicze

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L Rivest, Clifford Stein: Wprowadzenie do algorytmów, PWN, 2024, rozdział 12.