BST (Binary Search Tree) – binarne drzewo poszukiwań

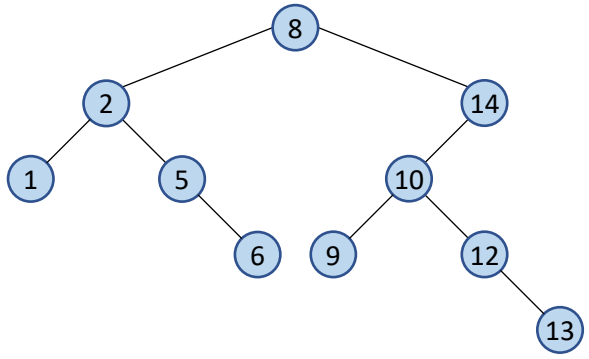
BST jest to drzewo binarne *T*, w którym dla każdego węzła w_i w *T* jest spełniony warunek:

$$klucz_{l(wi)} < klucz(w_i) < klucz_{p(wi)}$$

$l(w_i)$ - lewy potomek węzła w_i, $p(w_i)$ - prawy potomek węzła w_i.

Zbudujmy drzewo BST wczytując do niego kolejno liczby (klucze) z ciągu:

8, 2, 5, 14, 1, 10, 12, 13, 6, 9



BST: wysokość drzewa

Wysokość drzewa to
- liczba poziomów tego drzewa bez wliczania poziomu, na którym jest korzeń

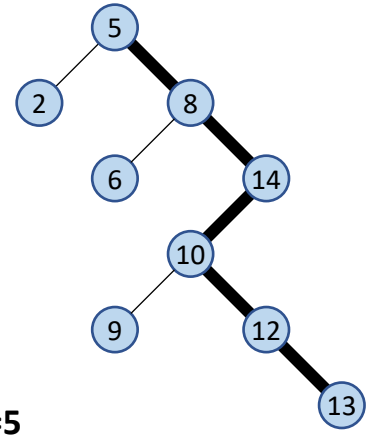
Inaczej mówiąc
- długość najdłuższej ścieżki między korzeniem a liściem w drzewie (długość ścieżki liczona jako liczba krawędzi między dwoma węzłami).

Drzewo, które składa się tylko z korzenia ma wysokość 0.

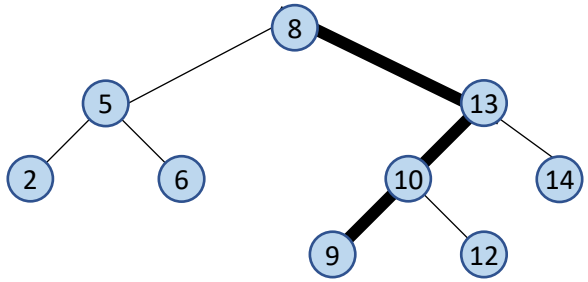
Wysokość drzewa binarnego zależy więc od jego struktury (od liczby poziomów w drzewie):
- dla drzewa zdegenerowanego (winorośli): $h=n-1$
- dla drzewa wyważonego i dokładnie wyważonego: $h=\lfloor(\log_2 n)\rfloor$

gdzie n to liczba elementów (węzłów) w drzewie.

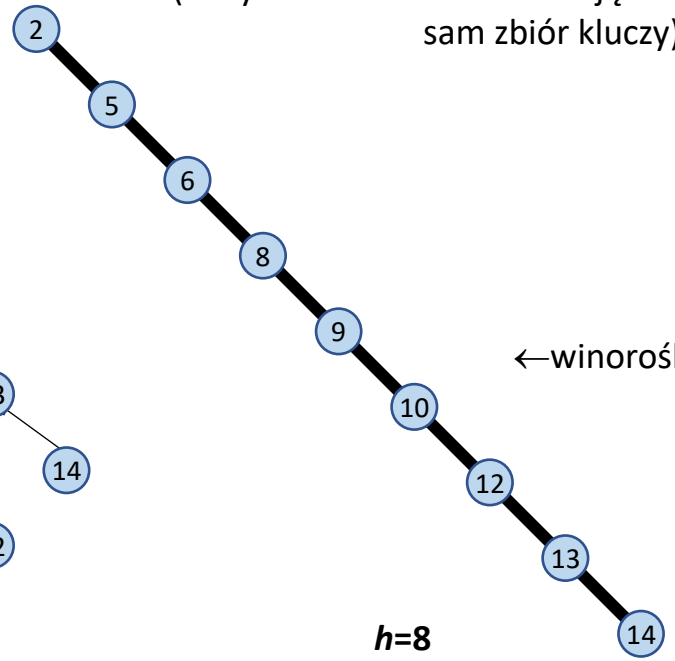
Klucze={8, 2, 5, 14, 10, 12, 13, 6, 9} (wszystkie 3 drzewa zawierają ten sam zbiór kluczy)



***h*=5**



***h*=3**



←winorośl

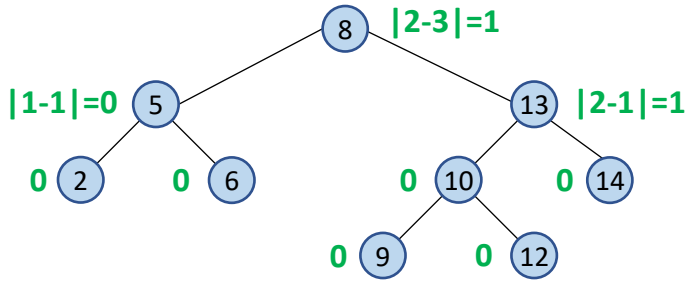
***h*=8**

Drzewo wyważone (zrównoważone, AVL) - drzewo binarne *T*, w którym dla każdego węzła w_i w *T* bezwzględna wartość różnicy między wysokością *h* jego lewego i prawego poddrzewa wynosi co najwyżej 1:

$$|h_{L(wi)} - h_{P(wi)}| \leq 1$$

To drzewo jest wyważone

(bezwzględna różnica wysokości obok każdego klucza)

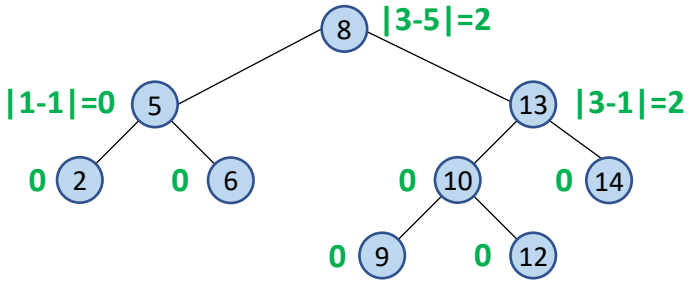


Drzewo dokładnie wyważone (DDW) - drzewo binarne *T*, w którym dla każdego węzła w_i w *T* bezwzględna wartość różnicy między liczbą *N* elementów w jego lewym i prawym poddrzewie wynosi co najwyżej 1:

$$|N_{L(wi)} - N_{P(wi)}| \leq 1$$

… ale nie jest dokładnie wyważone

(bezwzględna różnica liczby elementów w poddrzewach obok każdego klucza)



$L(w_i)$ - lewe poddrzewo węzła w_i

$P(w_i)$ - prawe poddrzewo węzła w_i

BST: wysokość drzewa a liczba elementów

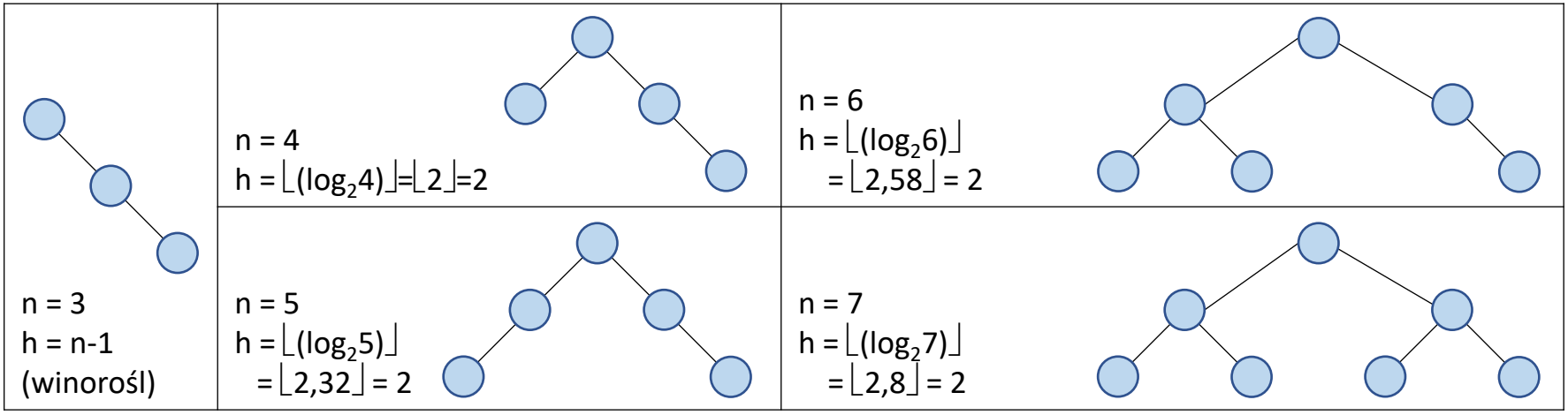
Pytanie: Ile elementów (liczb) *n* potrzeba, aby zbudować drzewo BST o wysokości *h*?

- Jeśli drzewo jest winoroślą, to w drzewie o wysokości *h* możemy mieć $n=h+1$ elementów. Zatem $h+1$ to minimalna liczba elementów, którą potrzebujemy, aby zbudować drzewo o wysokości *h*.

- Jaka jest maksymalna liczba elementów?
  Wynika to z wzoru na wysokość drzewa wyważonego $h=\lfloor(\log_2 n)\rfloor$. Zatem $n=2^{h+1}-1$.

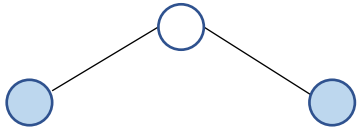
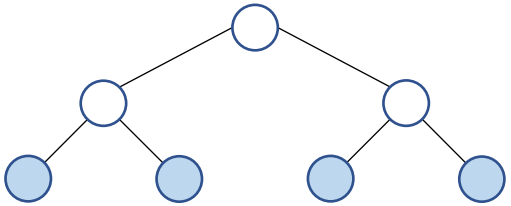
Odpowiedź: Chcąc zbudować drzewo o wysokości *h* potrzebujemy mieć *n* kluczy, gdzie $\boldsymbol{n \in <h+1,\ 2^{h+1}-1>}$.

Przykładowe drzewa o wysokości *h*=2



BST: poziom w drzewie a liczba elementów

Pytanie: Ile maksymalnie elementów (liczb) może się znajdować w drzewie BST na poziomie *L*, tzn. w odległości *L* od korzenia?

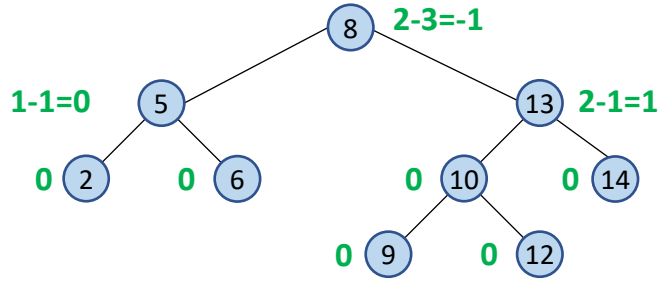
Poziom drzewa	Maksymalna liczba węzłów	
0	$2^0 = 1$ (korzeń)	
1	$2^1 = 2$	
2	$2^2 = 4$	

Odpowiedź: Na poziomie *L* drzewa BST może się znajdować maksymalnie $\mathbf{2}^{L}$ węzłów.

AVL jest to zrównoważone drzewo BST.
W drzewie AVL, każdy węzeł w_i przechowuje klucz k_i oraz wartość współczynnika równowagi bf_i.

Współczynnik równowagi węzła w_i jest równy różnicy między wysokością lewego i prawego poddrzewa tego węzła:
$$bf_i(w_i) = h_{L(wi)} - h_{P(wi)}$$

$L(w_i)$ - lewe poddrzewo węzła w_i
$P(w_i)$ - prawe poddrzewo węzła w_i



To jest drzewo AVL
(współczynnik równowagi obok każdego klucza)

Adel'son-Vel'skii GM, Landis EM (1962) An algorithm for the organization of information. Doklady Akademii Nauk SSSR 146: 263-266.

AVL: budowanie drzewa

Metoda budowania drzewa AVL wykorzystująca **algorytm wyszukiwania połówkowego**:

1. Posortuj rosnąco ciąg kluczy, które wstawisz do drzewa AVL (dowolną metodą sortowania)
2. Wybierz środkowy element ciągu (= mediana) jako element bieżący E
3. Wstaw wybrany element do drzewa BST
4. Jeśli w ciągu po lewej stronie elementu wybranego są jakieś liczby, to wybierz element E_L - środkowy element podciągu leżącego po lewej stronie od elementu bieżącego E i idź do pkt 3. W przeciwnym przypadku koniec.
5. Jeśli w ciągu po prawej stronie elementu wybranego są jakieś liczby, to wybierz element E_P - środkowy element podciągu leżącego po prawej stronie od elementu bieżącego E i idź do pkt 3. W przeciwnym przypadku koniec.

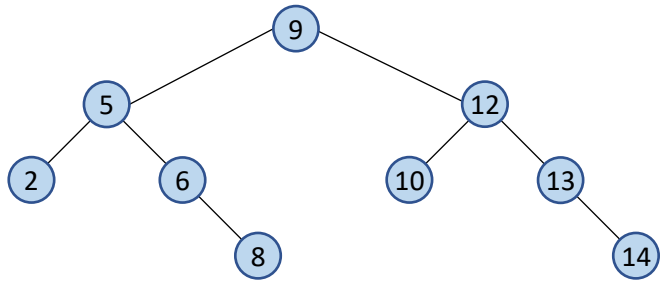
Przykład

Zbiór liczb = {8, 2, 5, 14, 10, 12, 13, 6, 9}

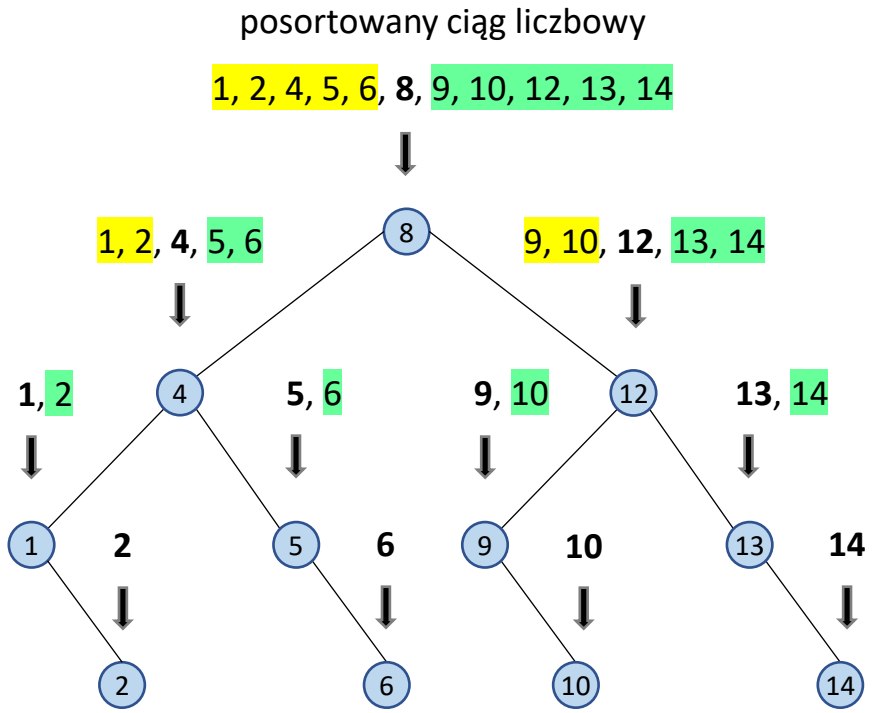
Posortowany ciąg: 2, 5, 6, 8, 9, 10, 12, 13, 14

Pierwszy element drzewa (korzeń): 9

9 jest medianą całego zbioru liczb.



Zbiór liczb: {8, 2, 5, 14, 10, 12, 13, 6, 9, 1, 4}



BST: wyszukiwanie elementu w drzewie

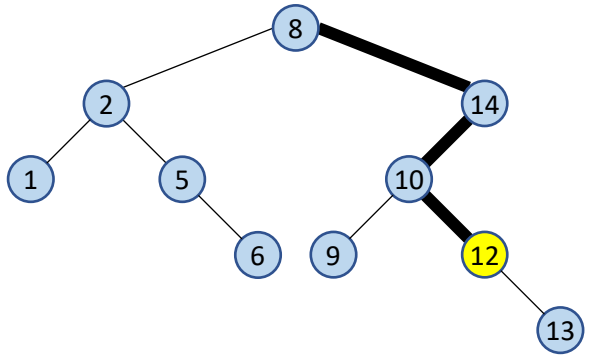
BST to struktura, która **optymalizuje czas wyszukiwania elementów** w zbiorze liczb (które są przechowywane w tej strukturze).

Wyszukanie elementu polega na przejściu drzewa od korzenia w dół dopóki element nie zostanie znaleziony (jeśli jest w drzewie) lub dopóki nie dojdziemy do liścia.

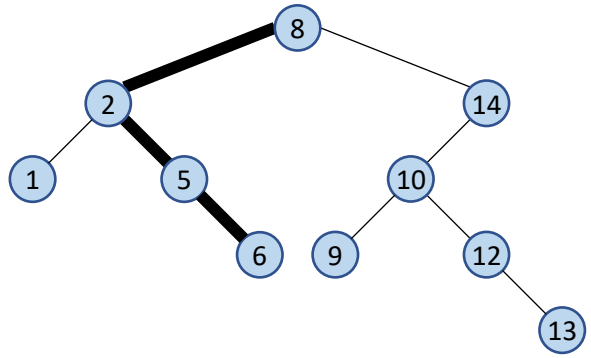
Ścieżka poszukiwań jest zatem co najwyżej tak długa jak h - wysokość drzewa BST. Dlatego ważne jest aby drzewo było zrównoważone (miało jak najmniejszą wysokość).

Procedura wyszukania elementu w drzewie BST ma **złożoność obliczeniową *O*(*h*)**.

Ścieżka poszukiwania elementu o kluczu **k=12**

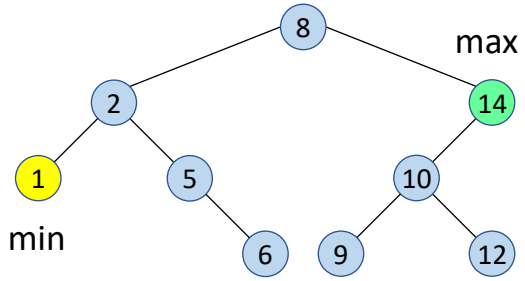


Ścieżka poszukiwania elementu o kluczu **k=7**

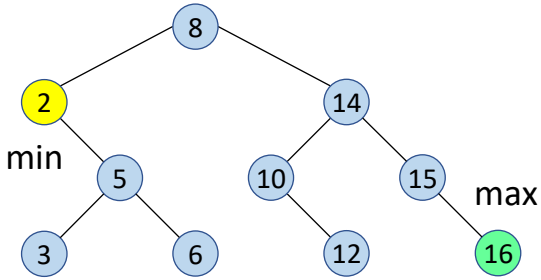


BST: wyszukiwanie min i max w drzewie

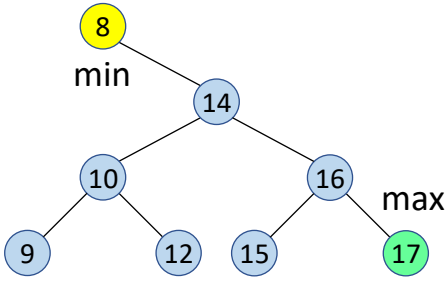
Minimum w drzewie BST
Element o najmniejszym kluczu znajduje się w **skrajnie lewym** węźle drzewa.
Skrajnie lewy element nie musi być liściem (może to być nawet korzeń drzewa). Element ten nie ma lewego potomka.



Maksimum w drzewie BST
Element o największym kluczu znajduje się w **skrajnie prawym** węźle drzewa. Podobnie jak w przypadku minimum, węzeł ten nie musi być liściem.



Poszukiwanie minimum (maksimum) polega na schodzeniu od korzenia w lewo (prawo) dopóki istnieje lewy (prawy) potomek, do którego można zejść.

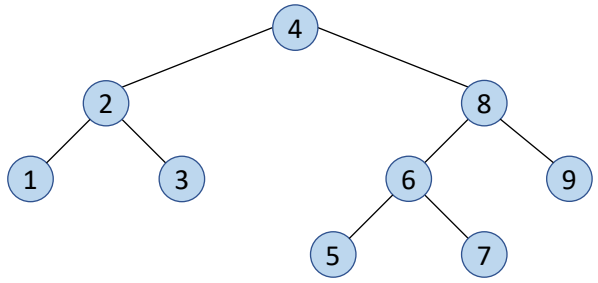


BST: odwiedzanie wszystkich elementów drzewa

Metody przechodzenia wszystkich elementów drzewa binarnego

- **pre-order** (wzdłużna): korzeń, lewe poddrzewo, prawe poddrzewo
- **in-oder** (poprzeczna): lewe poddrzewo, korzeń, prawe poddrzewo
- **post-order** (wsteczna): lewe poddrzewo, prawe poddrzewo, korzeń

Złożoność obliczeniowa przechodzenia drzewa $O(n)$.



- **pre-order** (wzdłużna): korzeń, lewy, prawy

  4, 2, 1, 3, 8, 6, 5, 7, 9

- **in-oder** (poprzeczna): lewy, korzeń, prawy

  1, 2, 3, 4, 5, 6, 7, 8, 9

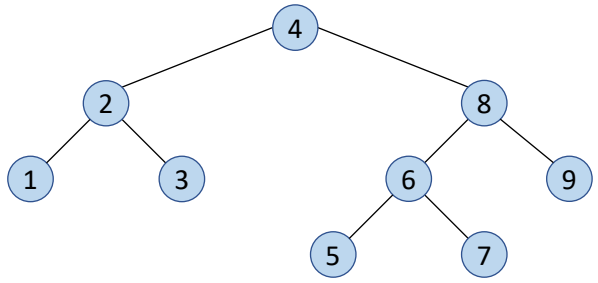
- **post-order** (wsteczna): lewy, prawy, korzeń

  1, 3, 2, 5, 7, 6, 9, 8, 4

BST: odwiedzanie wszystkich elementów drzewa

Metody przechodzenia wszystkich elementów drzewa binarnego

- **pre-order** (wzdłużna): korzeń, lewe poddrzewo, prawe poddrzewo
- **in-oder** (poprzeczna): lewe poddrzewo, korzeń, prawe poddrzewo
- **post-order** (wsteczna): lewe poddrzewo, prawe poddrzewo, korzeń



- **pre-order** (wzdłużna): korzeń, lewy, prawy

  4, 2, 1, 3, 8, 6, 5, 7, 9

  możliwa kolejność dodawania elementów
- **in-oder** (poprzeczna): lewy, korzeń, prawy

  1, 2, 3, 4, 5, 6, 7, 8, 9

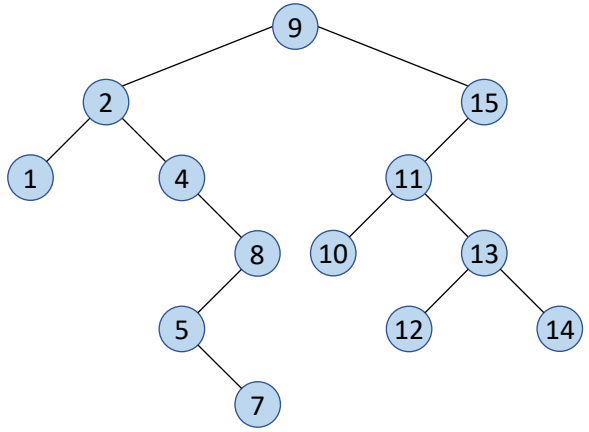
  sortowanie elementów drzewa
- **post-order** (wsteczna): lewy, prawy, korzeń

  1, 3, 2, 5, 7, 6, 9, 8, 4

  usuwanie drzewa element po elemencie

BST: odwiedzanie wszystkich elementów drzewa

Ćwiczenie



- **pre-order** (wzdłużna): korzeń, lewy, prawy

  9, 2, 1, 4, 8, 5, 7, 15, 11, 10, 13, 12, 14

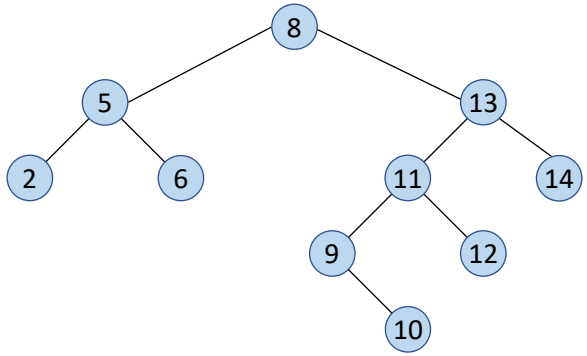
- **in-oder** (poprzeczna): lewy, korzeń, prawy

  1, 2, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14, 15

- **post-order** (wsteczna): lewy, prawy, korzeń

  1, 7, 5, 8, 4, 2, 10, 12, 14, 13, 11, 15, 9

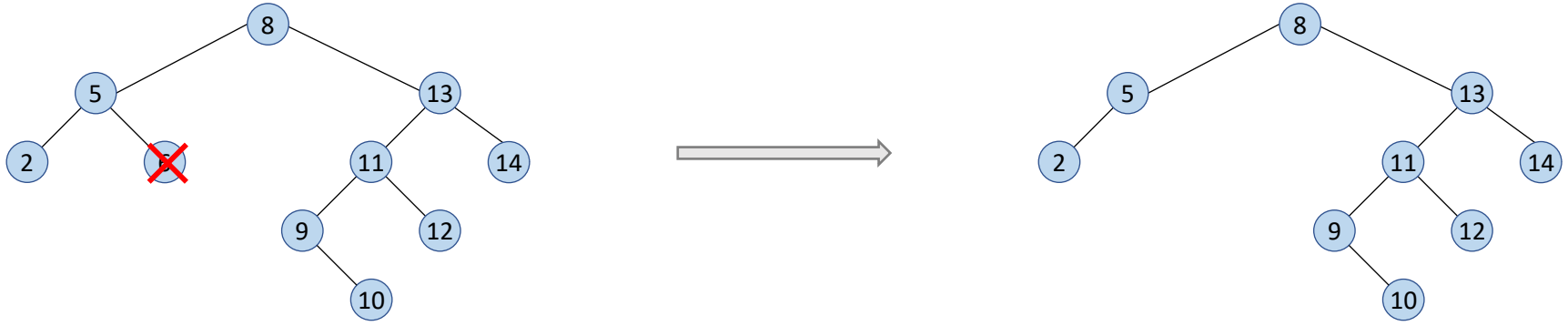
BST: usuwanie elementu z drzewa



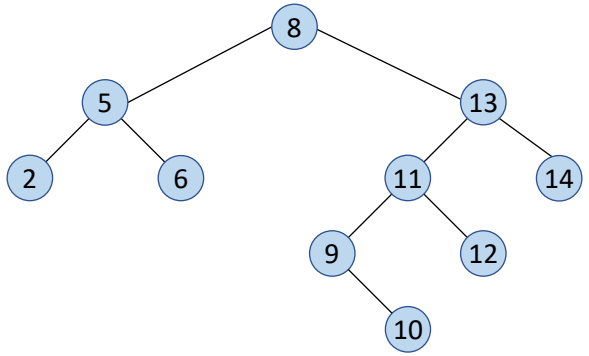
Wyróżniamy 3 przypadki:

1. **usuwanie liścia**
2. usuwanie elementu, który ma jednego potomka
3. usuwanie elementu, który ma dwa węzły potomne

Usuń węzeł o kluczu **k=6**



BST: usuwanie elementu z drzewa

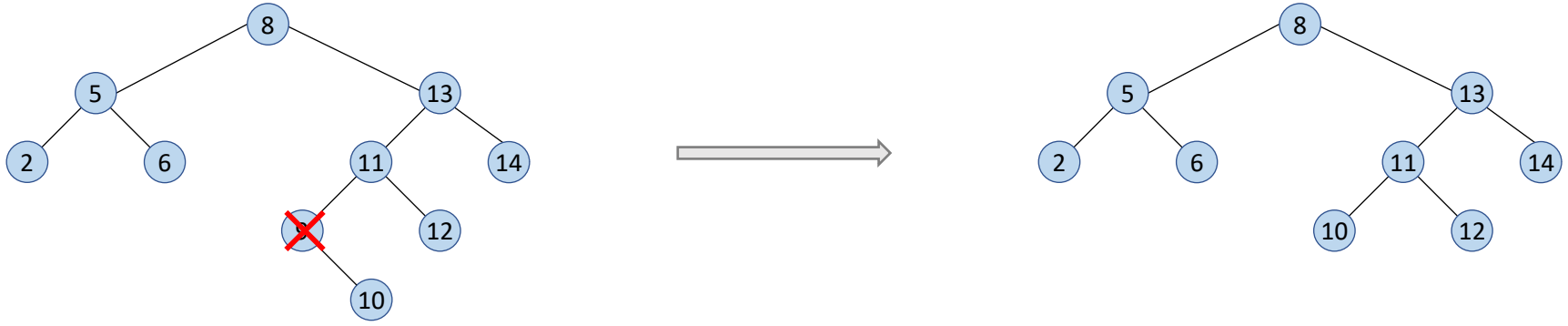


Wyróżniamy 3 przypadki:

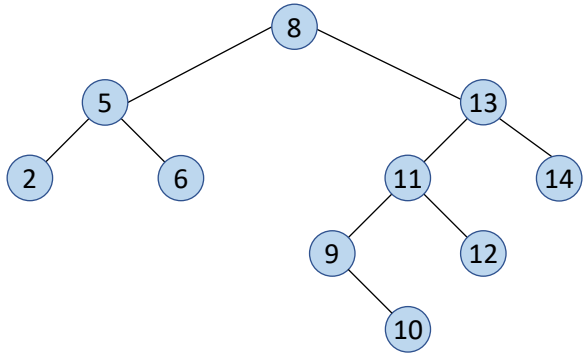
1. usuwanie liścia
2. **usuwanie elementu, który ma jednego potomka**
3. usuwanie elementu, który ma dwa węzły potomne

Usuń węzeł o kluczu **k=9**

Po usunięciu wybranego węzła jego poddrzewa zostają w strukturze.



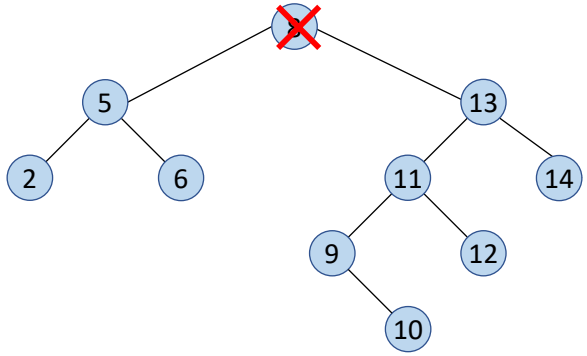
BST: usuwanie elementu z drzewa



Wyróżniamy 3 przypadki:

1. usuwanie liścia
2. usuwanie elementu, który ma jednego potomka
3. **usuwanie elementu, który ma dwa węzły potomne**

Usuń węzeł o kluczu **k=8**

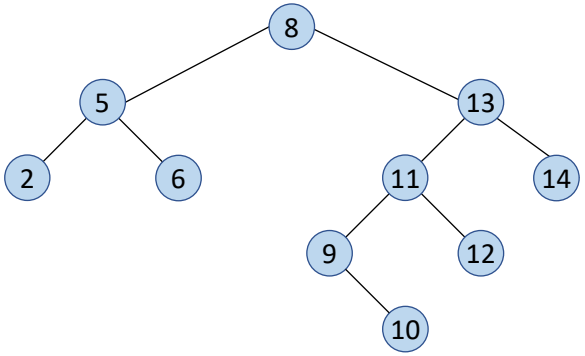


W miejsce usuwanego węzła wstawiamy klucz „sąsiedni”.

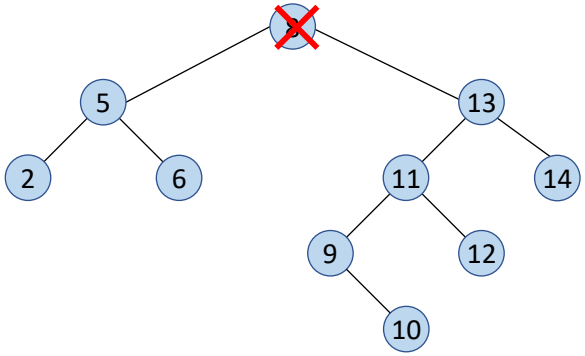
Ciąg dalszy na następnej stronie…

BST: usuwanie z drzewa elementu, który ma dwa węzły potomne

Dane jest drzewo BST:



Chcemy usunąć z niego węzeł o kluczu **k=8**

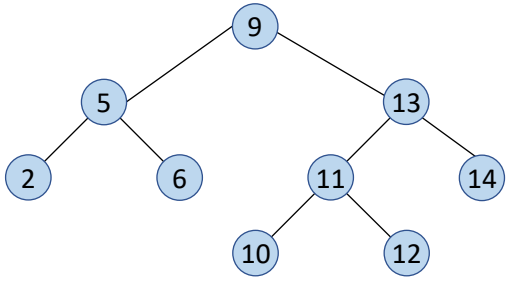


Jeśli posortujemy klucze tego drzewa (np. metodą in-order), dostaniemy ciąg:
2, 5, **6**, **8**, **9**, 10, 11, 12, 13, 14.

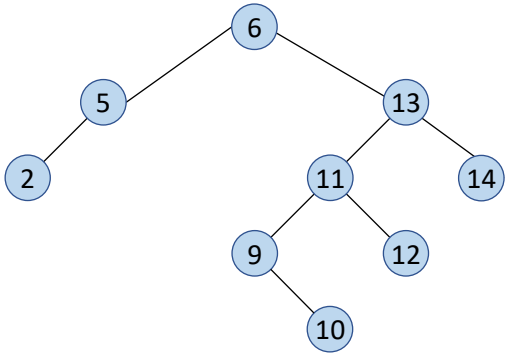
Usunięcie „8” polega na:

1. Wpisaniu w miejsce „8” wartości sąsiedniej z ciągu posortowanego. Są dwie możliwości, tutaj: 6 lub 9.
2. Uruchomieniu procedury usunięcia z drzewa węzła, z którego klucz przepisaliśmy.

w miejsce usuwanego trafia 9



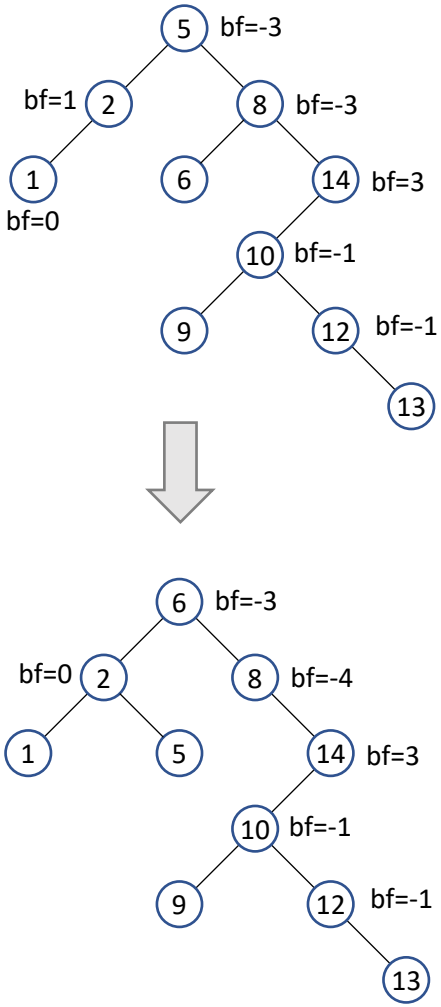
w miejsce usuwanego trafia 6



Metody równoważenia drzewa (złożoność O(n)):

- **Algorytm Martina-Nessa**
  - Posortowana lista kluczy trafia na stos, poszukiwanie mediany (przeszukiwanie połówkowe) i wstawianie do drzewa [Martin WA, Ness DN (1972) Optimal Binary Trees Grown with a Sorting Algorithm. Communication of the ACM 15(2):88-93]
- **Algorytm Day'a**
  - Przejście drzewa w porządku in-order, utworzenie listy posortowanej, seria rotacji w lewo [Day AC (1976) Balancing a Binary Tree. Computer Journal XIX: 360-361]
- **Algorytm Day'a-Stouta-Warrena** (DSW)
  - Modyfikacja algorytmu Day'a: utworzenie winorośli poprzez rotacje w prawo, seria rotacji w lewo [Stout QF, Warren BL (1986) Tree Rebalancing in Optimal Time and Space. Communication of the ACM 29(9):902-908]
- **Algorytm Changa-Iyengara**
  - Przejście drzewa w porządku in-order i odbudowanie drzewa z wykorzystaniem wyszukiwania połówkowego [Chang H, Iyengar SS (1984) Efficient Algorithms To Globally Balance a Binary Search Tree. Communication of the ACM 27(8):695-702]
- **Algorytm równoważenia przez usuwanie korzeni niezbalansowanych poddrzew**
  - Iteracyjne usuwanie węzłów o bezwzględnym współczynniku równowagi >1 i wstawianie ich kluczy z powrotem do drzewa

BST: równoważenie drzewa przez usuwanie korzenia



Równoważenie przez usuwanie korzenia:

1. Przechodząc po drzewie od korzenia w kierunku liści metodą BFS (poziomami), znajdź pierwszy węzeł w_i, dla którego $|bf_i|>1$, gdzie bf_i to współczynnik równowagi węzła w_i. Jeśli nie ma takiego węzła to zakończ (drzewo jest zrównoważone). W przeciwnym razie idź do pkt 2.
2. Usuń węzeł w_i z drzewa (jeśli w_i ma dwa poddrzewa, to zastąp w_i węzłem z poddrzewa o większej wysokości). Następnie wstaw klucz k_i z powrotem do drzewa (k_i to klucz usuniętego węzła w_i).
3. Zaktualizuj współczynniki równowagi w całym drzewie. Wróć do pkt 1.