Algorytmy i struktury danych - Drzewa poszukiwań binarnych

Marcin Żurowski

28 maja 2025

Plan zajęć

Drzewo poszukiwań binarnych

Drzewo poszukiwań binarnych (Binary Search Tree, BST) – dynamiczna struktura danych oparta na drzewie binarnym. Drzewo BST może być wykorzystane jako:

- słownik
- kolejka priorytetowa (w przypadku pesymistycznym gorsza złożoność niż przy użyciu kopca binarnego)

Drzewo poszukiwań binarnych

Drzewo poszukiwań binarnych (Binary Search Tree, BST) – dynamiczna struktura danych oparta na drzewie binarnym.Drzewo BST może być wykorzystane jako:

- słownik
- kolejka priorytetowa (w przypadku pesymistycznym gorsza złożoność niż przy użyciu kopca binarnego)

Drzewo poszukiwań binarnych

Drzewo poszukiwań binarnych (Binary Search Tree, BST) – dynamiczna struktura danych oparta na drzewie binarnym.Drzewo BST może być wykorzystane jako:

- słownik
- kolejka priorytetowa (w przypadku pesymistycznym gorsza złożoność niż przy użyciu kopca binarnego)

Drzewo poszukiwań binarnych

Reprezentacja drzewa BST za pomocą struktur wskaźnikowych:

- Drzewo T posiada atrybut $T.root$ wskazujący na korzeń tego drzewa.
- Każdy węzeł x drzewa T jest rekordem posiadającym następujące pola:
  - x.key – klucz (wartość przechowywana przez węzeł x)
  - x.p – wskaźnik do ojca węzła x
  - x.left – wskaźnik do lewego syna węzła x
  - x.right – wskaźnik do prawego syna węzła x

Drzewo poszukiwań binarnych

Reprezentacja drzewa BST za pomocą struktur wskaźnikowych:

- Drzewo T posiada atrybut $T.root$ wskazujący na korzeń tego drzewa.
- Każdy węzeł x drzewa T jest rekordem posiadającym następujące pola:
  - x.key – klucz (wartość przechowywana przez węzeł x)
  - x.p – wskaźnik do ojca węzła x
  - x.left – wskaźnik do lewego syna węzła x
  - x.right – wskaźnik do prawego syna węzła x

Drzewo poszukiwań binarnych

Reprezentacja drzewa BST za pomocą struktur wskaźnikowych:

- Drzewo T posiada atrybut $T.root$ wskazujący na korzeń tego drzewa.
- Każdy węzeł x drzewa T jest rekordem posiadającym następujące pola:
  - x.key – klucz (wartość przechowywana przez węzeł x)
  - x.p – wskaźnik do ojca węzła x
  - x.left – wskaźnik do lewego syna węzła x
  - x.right – wskaźnik do prawego syna węzła x

Drzewo poszukiwań binarnych

Reprezentacja drzewa BST za pomocą struktur wskaźnikowych:

- Drzewo T posiada atrybut $T.root$ wskazujący na korzeń tego drzewa.
- Każdy węzeł x drzewa T jest rekordem posiadającym następujące pola:
  - x.key – klucz (wartość przechowywana przez węzeł x)
  - x.p – wskaźnik do ojca węzła x
  - x.left – wskaźnik do lewego syna węzła x
  - x.right – wskaźnik do prawego syna węzła x

Drzewo poszukiwań binarnych

Reprezentacja drzewa BST za pomocą struktur wskaźnikowych:

- Drzewo T posiada atrybut $T.root$ wskazujący na korzeń tego drzewa.
- Każdy węzeł x drzewa T jest rekordem posiadającym następujące pola:
  - x.key – klucz (wartość przechowywana przez węzeł x)
  - x.p – wskaźnik do ojca węzła x
  - x.left – wskaźnik do lewego syna węzła x
  - x.right – wskaźnik do prawego syna węzła x

Drzewo poszukiwań binarnych

Reprezentacja drzewa BST za pomocą struktur wskaźnikowych:

- Drzewo T posiada atrybut $T.root$ wskazujący na korzeń tego drzewa.
- Każdy węzeł x drzewa T jest rekordem posiadającym następujące pola:
  - x.key – klucz (wartość przechowywana przez węzeł x)
  - x.p – wskaźnik do ojca węzła x
  - x.left – wskaźnik do lewego syna węzła x
  - x.right – wskaźnik do prawego syna węzła x

Drzewo poszukiwań binarnych

Własność drzewa BST:

- Niech x będzie węzłem w drzewie BST. Jeśli y jest węzłem znajdującym się w lewym poddrzewie węzła x, to $y.key \leq x.key$
- a jeśli y jest węzłem znajdującym się w prawym poddrzewie węzła x, to $x.key \leq y.key$

Drzewo poszukiwań binarnych

Własność drzewa BST:

- Niech x będzie węzłem w drzewie BST. Jeśli y jest węzłem znajdującym się w lewym poddrzewie węzła x, to $y.key \leq x.key$
- a jeśli y jest węzłem znajdującym się w prawym poddrzewie węzła x, to $x.key \leq y.key$

Drzewo poszukiwań binarnych

Przechodzenie drzewa BST metodą inorder

Klucz korzenia poddrzewa zostaje wypisany między wartościami z jego lewego poddrzewa a wartościami z jego prawego poddrzewa.

```
procedure INORDER-TREE-WALK(x)
 if x ≠ NIL
   INORDER-TREE-WALK(x.left)
   write(x.key)
   INORDER-TREE-WALK(x.right)
```

Wywołanie INORDER-TREE-WALK(T.root) powoduje wypisanie w porządku niemalejącym kluczy wszystkich elementów drzewa BST o n węzłach, w czasie $\Theta(n)$.

Drzewo poszukiwań binarnych

Przechodzenie drzewa BST metodą inorder
Klucz korzenia poddrzewa zostaje wypisany między wartościami z jego lewego poddrzewa a wartościami z jego prawego poddrzewa.

```
procedure INORDER-TREE-WALK(x)
 if x ≠ NIL
  INORDER-TREE-WALK(x.left)
  write(x.key)
  INORDER-TREE-WALK(x.right)
```

Wywołanie INORDER-TREE-WALK(T.root) powoduje wypisanie w porządku niemalejącym kluczy wszystkich elementów drzewa BST o n węzłach, w czasie $\Theta(n)$.

Drzewo poszukiwań binarnych

Przechodzenie drzewa BST metodą inorder
Klucz korzenia poddrzewa zostaje wypisany między wartościami z jego lewego poddrzewa a wartościami z jego prawego poddrzewa.

```
procedure INORDER-TREE-WALK(x)
 if x ≠ NIL
  INORDER-TREE-WALK(x.left)
  write(x.key)
  INORDER-TREE-WALK(x.right)
```

Wywołanie INORDER-TREE-WALK(T.root) powoduje wypisanie w porządku niemalejącym kluczy wszystkich elementów drzewa BST o n węzłach, w czasie $\Theta(n)$.

Drzewo poszukiwań binarnych

Przechodzenie drzewa BST metodą inorder
Klucz korzenia poddrzewa zostaje wypisany między wartościami z jego lewego poddrzewa a wartościami z jego prawego poddrzewa.

```
procedure INORDER-TREE-WALK(x)
 if x ≠ NIL
  INORDER-TREE-WALK(x.left)
  write(x.key)
  INORDER-TREE-WALK(x.right)
```

Wywołanie INORDER-TREE-WALK(T.root) powoduje wypisanie w porządku niemalejącym kluczy wszystkich elementów drzewa BST o n węzłach, w czasie $\Theta(n)$.

Drzewo poszukiwań binarnych

Inne metody przechodzenia drzewa BST:

- Metoda preorder: klucz korzenia zostaje wypisany przed wartościami znajdującymi się w obu poddrzewach.
- Metoda postorder: klucz korzenia zostaje wypisany po wartościach znajdujących się w obu poddrzewach.

Drzewo poszukiwań binarnych

Inne metody przechodzenia drzewa BST:

- Metoda preorder: klucz korzenia zostaje wypisany przed wartościami znajdującymi się w obu poddrzewach.
- Metoda postorder: klucz korzenia zostaje wypisany po wartościach znajdujących się w obu poddrzewach.

Drzewo poszukiwań binarnych

Wyszukiwanie w drzewie BST – idea

- Wyszukiwanie rozpoczynamy w korzeniu drzewa.
- Kiedy algorytm odwiedza węzeł x, porównujemy jego klucz z szukaną wartością k.
- Jeśli $x.key = k$, to poszukiwanie zakończyło się sukcesem - zwracamy węzeł x.
- Jeśli $k < x.key$, to przechodzimy do lewego syna węzła x (przeszukujemy lewe poddrzewo).
- Jeśli $k > x.key$, to przechodzimy do prawego syna węzła x (przeszukujemy prawe poddrzewo).
- Jeśli $x = NIL$, to wartość k nie występuje w drzewie.
- Algorytm działa w czasie $O(h)$, gdzie h jest wysokością drzewa.

Drzewo poszukiwań binarnych

Wyszukiwanie w drzewie BST – idea

- Wyszukiwanie rozpoczynamy w korzeniu drzewa.
- Kiedy algorytm odwiedza węzeł x, porównujemy jego klucz z szukaną wartością k.
- Jeśli $x.key = k$, to poszukiwanie zakończyło się sukcesem - zwracamy węzeł x.
- Jeśli $k < x.key$, to przechodzimy do lewego syna węzła x (przeszukujemy lewe poddrzewo).
- Jeśli $k > x.key$, to przechodzimy do prawego syna węzła x (przeszukujemy prawe poddrzewo).
- Jeśli $x = NIL$, to wartość k nie występuje w drzewie.
- Algorytm działa w czasie $O(h)$, gdzie h jest wysokością drzewa.

Drzewo poszukiwań binarnych

Wyszukiwanie w drzewie BST – idea

- Wyszukiwanie rozpoczynamy w korzeniu drzewa.
- Kiedy algorytm odwiedza węzeł x, porównujemy jego klucz z szukaną wartością k.
- Jeśli $x.key = k$, to poszukiwanie zakończyło się sukcesem - zwracamy węzeł x.
- Jeśli $k < x.key$, to przechodzimy do lewego syna węzła x (przeszukujemy lewe poddrzewo).
- Jeśli $k > x.key$, to przechodzimy do prawego syna węzła x (przeszukujemy prawe poddrzewo).
- Jeśli $x = NIL$, to wartość k nie występuje w drzewie.
- Algorytm działa w czasie $O(h)$, gdzie h jest wysokością drzewa.

Drzewo poszukiwań binarnych

Wyszukiwanie w drzewie BST – idea

- Wyszukiwanie rozpoczynamy w korzeniu drzewa.
- Kiedy algorytm odwiedza węzeł x, porównujemy jego klucz z szukaną wartością k.
- Jeśli $x.key = k$, to poszukiwanie zakończyło się sukcesem - zwracamy węzeł x.
- Jeśli $k < x.key$, to przechodzimy do lewego syna węzła x (przeszukujemy lewe poddrzewo).
- Jeśli $k > x.key$, to przechodzimy do prawego syna węzła x (przeszukujemy prawe poddrzewo).
- Jeśli $x = NIL$, to wartość k nie występuje w drzewie.
- Algorytm działa w czasie $O(h)$, gdzie h jest wysokością drzewa.

Drzewo poszukiwań binarnych

Wyszukiwanie w drzewie BST – idea

- Wyszukiwanie rozpoczynamy w korzeniu drzewa.
- Kiedy algorytm odwiedza węzeł x, porównujemy jego klucz z szukaną wartością k.
- Jeśli $x.key = k$, to poszukiwanie zakończyło się sukcesem - zwracamy węzeł x.
- Jeśli $k < x.key$, to przechodzimy do lewego syna węzła x (przeszukujemy lewe poddrzewo).
- Jeśli $k > x.key$, to przechodzimy do prawego syna węzła x (przeszukujemy prawe poddrzewo).
- Jeśli $x = NIL$, to wartość k nie występuje w drzewie.
- Algorytm działa w czasie $O(h)$, gdzie h jest wysokością drzewa.

Drzewo poszukiwań binarnych

Wyszukiwanie w drzewie BST – idea

- Wyszukiwanie rozpoczynamy w korzeniu drzewa.
- Kiedy algorytm odwiedza węzeł x, porównujemy jego klucz z szukaną wartością k.
- Jeśli $x.key = k$, to poszukiwanie zakończyło się sukcesem - zwracamy węzeł x.
- Jeśli $k < x.key$, to przechodzimy do lewego syna węzła x (przeszukujemy lewe poddrzewo).
- Jeśli $k > x.key$, to przechodzimy do prawego syna węzła x (przeszukujemy prawe poddrzewo).
- Jeśli $x = NIL$, to wartość k nie występuje w drzewie.
- Algorytm działa w czasie $O(h)$, gdzie h jest wysokością drzewa.

Drzewo poszukiwań binarnych

Wyszukiwanie w drzewie BST – idea

- Wyszukiwanie rozpoczynamy w korzeniu drzewa.
- Kiedy algorytm odwiedza węzeł x, porównujemy jego klucz z szukaną wartością k.
- Jeśli $x.key = k$, to poszukiwanie zakończyło się sukcesem - zwracamy węzeł x.
- Jeśli $k < x.key$, to przechodzimy do lewego syna węzła x (przeszukujemy lewe poddrzewo).
- Jeśli $k > x.key$, to przechodzimy do prawego syna węzła x (przeszukujemy prawe poddrzewo).
- Jeśli $x = NIL$, to wartość k nie występuje w drzewie.
- Algorytm działa w czasie $O(h)$, gdzie h jest wysokością drzewa.

Drzewo poszukiwań binarnych

Wyszukiwanie w drzewie BST – wersja rekurencyjna

```
procedure TREE-SEARCH(x, k)
 if x = NIL or x.key = k
  return x
 if k < x.key
  return TREE-SEARCH(x.left, k)
 else
  return TREE-SEARCH(x.right, k)
```

Wywołanie: TREE-SEARCH(T.root, k)

Drzewo poszukiwań binarnych

Wyszukiwanie w drzewie BST – wersja rekurencyjna

```
procedure TREE-SEARCH(x, k)
 if x = NIL or x.key = k
  return x
 if k < x.key
  return TREE-SEARCH(x.left, k)
 else
  return TREE-SEARCH(x.right, k)
```

Wywołanie: TREE-SEARCH(T.root, k)

Drzewo poszukiwań binarnych

Wyszukiwanie w drzewie BST – wersja rekurencyjna

```
procedure TREE-SEARCH(x, k)
 if x = NIL or x.key = k
  return x
 if k < x.key
  return TREE-SEARCH(x.left, k)
 else
  return TREE-SEARCH(x.right, k)
```

Wywołanie: TREE-SEARCH(T.root, k)

Drzewo poszukiwań binarnych

Wyszukiwanie w drzewie BST – wersja iteracyjna

```
procedure TREE-SEARCH(x, k)
 while x ≠ NIL and x.key ≠ k
  if k < x.key
   x = x.left
  else
   x = x.right
 return x
```

Wywołanie: ITERATIVE-TREE-SEARCH(T.root, k)

Drzewo poszukiwań binarnych

Wyszukiwanie w drzewie BST – wersja iteracyjna

```
procedure TREE-SEARCH(x, k)
 while x ≠ NIL and x.key ≠ k
  if k < x.key
   x = x.left
  else
   x = x.right
 return x
```

Wywołanie: ITERATIVE-TREE-SEARCH(T.root, k)

Drzewo poszukiwań binarnych

Wyszukiwanie w drzewie BST – wersja iteracyjna

```
procedure TREE-SEARCH(x, k)
 while x ≠ NIL and x.key ≠ k
  if k < x.key
   x = x.left
  else
   x = x.right
 return x
```

Wywołanie: ITERATIVE-TREE-SEARCH(T.root, k)

Drzewo poszukiwań binarnych

Wyszukiwanie węzła o najmniejszym i największym kluczu w poddrzewie o korzeniu w węźle x

```
procedure TREE-MINIMUM(x)
 while x.left ≠ NIL
  x = x.left
 return x
procedure TREE-MAXIMUM(x)
 while x.right ≠ NIL
  x = x.right
 return x
```

Złożoność: $O(h)$ dla drzewa o wysokości h.

Drzewo poszukiwań binarnych

Wyszukiwanie węzła o najmniejszym i największym kluczu w poddrzewie o korzeniu w węźle x

```
procedure TREE-MINIMUM(x)
 while x.left ≠ NIL
  x = x.left
 return x
procedure TREE-MAXIMUM(x)
 while x.right ≠ NIL
  x = x.right
 return x
```

Złożoność: $O(h)$ dla drzewa o wysokości h.

Drzewo poszukiwań binarnych

Wyszukiwanie węzła o najmniejszym i największym kluczu w poddrzewie o korzeniu w węźle x

```
procedure TREE-MINIMUM(x)
 while x.left ≠ NIL
  x = x.left
 return x
procedure TREE-MAXIMUM(x)
 while x.right ≠ NIL
  x = x.right
 return x
```

Złożoność: $O(h)$ dla drzewa o wysokości h.

Drzewo poszukiwań binarnych

Wyszukiwanie węzła o najmniejszym i największym kluczu w poddrzewie o korzeniu w węźle x

```
procedure TREE-MINIMUM(x)
 while x.left ≠ NIL
  x = x.left
 return x
procedure TREE-MAXIMUM(x)
 while x.right ≠ NIL
  x = x.right
 return x
```

Złożoność: $O(h)$ dla drzewa o wysokości h.

Drzewo poszukiwań binarnych

- Następnik węzła x w drzewie BST – węzeł, który podczas przechodzenia drzewa w porządku inorder zostanie odwiedzony bezpośrednio po x.
- Jeśli wszystkie klucze w drzewie są różne, to następnikiem węzła x jest węzeł o najmniejszym kluczu większym niż $x.key$.
- Jeśli x zostaje odwiedzony jako ostatni węzeł podczas przechodzenia drzewa w porządku inorder (x ma największy klucz w drzewie, o ile wszystkie klucze są różne), to jego następnikiem jest NIL.
- Poprzednik węzła x w drzewie BST – węzeł, który podczas przechodzenia drzewa w porządku inorder zostanie odwiedzony bezpośrednio przed x.

Drzewo poszukiwań binarnych

- Następnik węzła x w drzewie BST – węzeł, który podczas przechodzenia drzewa w porządku inorder zostanie odwiedzony bezpośrednio po x.
- Jeśli wszystkie klucze w drzewie są różne, to następnikiem węzła x jest węzeł o najmniejszym kluczu większym niż $x.key$.
- Jeśli x zostaje odwiedzony jako ostatni węzeł podczas przechodzenia drzewa w porządku inorder (x ma największy klucz w drzewie, o ile wszystkie klucze są różne), to jego następnikiem jest NIL.
- Poprzednik węzła x w drzewie BST – węzeł, który podczas przechodzenia drzewa w porządku inorder zostanie odwiedzony bezpośrednio przed x.

Drzewo poszukiwań binarnych

- Następnik węzła x w drzewie BST – węzeł, który podczas przechodzenia drzewa w porządku inorder zostanie odwiedzony bezpośrednio po x.
- Jeśli wszystkie klucze w drzewie są różne, to następnikiem węzła x jest węzeł o najmniejszym kluczu większym niż $x.key$.
- Jeśli x zostaje odwiedzony jako ostatni węzeł podczas przechodzenia drzewa w porządku inorder (x ma największy klucz w drzewie, o ile wszystkie klucze są różne), to jego następnikiem jest NIL.
- Poprzednik węzła x w drzewie BST – węzeł, który podczas przechodzenia drzewa w porządku inorder zostanie odwiedzony bezpośrednio przed x.

Drzewo poszukiwań binarnych

- Następnik węzła x w drzewie BST – węzeł, który podczas przechodzenia drzewa w porządku inorder zostanie odwiedzony bezpośrednio po x.
- Jeśli wszystkie klucze w drzewie są różne, to następnikiem węzła x jest węzeł o najmniejszym kluczu większym niż $x.key$.
- Jeśli x zostaje odwiedzony jako ostatni węzeł podczas przechodzenia drzewa w porządku inorder (x ma największy klucz w drzewie, o ile wszystkie klucze są różne), to jego następnikiem jest NIL.
- Poprzednik węzła x w drzewie BST – węzeł, który podczas przechodzenia drzewa w porządku inorder zostanie odwiedzony bezpośrednio przed x.

Drzewo poszukiwań binarnych

Znajdowanie następnika węzła x – idea

- Jeśli prawe poddrzewo węzła x jest niepuste, to następnikiem x jest węzeł położony najbardziej na lewo w tym poddrzewie.
- Jeśli prawe poddrzewo węzła x jest puste, to następnikiem x jest jego najniższy przodek y, którego lewy syn jest także przodkiem x.
- Algorytm opiera się wyłącznie na powyższych własnościach (nie porównuje kluczy węzłów).
- Złożoność: $O(h)$ dla drzewa o wysokości h.

Drzewo poszukiwań binarnych

Znajdowanie następnika węzła x – idea

- Jeśli prawe poddrzewo węzła x jest niepuste, to następnikiem x jest węzeł położony najbardziej na lewo w tym poddrzewie.
- Jeśli prawe poddrzewo węzła x jest puste, to następnikiem x jest jego najniższy przodek y, którego lewy syn jest także przodkiem x.
- Algorytm opiera się wyłącznie na powyższych własnościach (nie porównuje kluczy węzłów).
- Złożoność: $O(h)$ dla drzewa o wysokości h.

Drzewo poszukiwań binarnych

Znajdowanie następnika węzła x – idea

- Jeśli prawe poddrzewo węzła x jest niepuste, to następnikiem x jest węzeł położony najbardziej na lewo w tym poddrzewie.
- Jeśli prawe poddrzewo węzła x jest puste, to następnikiem x jest jego najniższy przodek y, którego lewy syn jest także przodkiem x.
- Algorytm opiera się wyłącznie na powyższych własnościach (nie porównuje kluczy węzłów).
- Złożoność: $O(h)$ dla drzewa o wysokości h.

Drzewo poszukiwań binarnych

Znajdowanie następnika węzła x – idea

- Jeśli prawe poddrzewo węzła x jest niepuste, to następnikiem x jest węzeł położony najbardziej na lewo w tym poddrzewie.
- Jeśli prawe poddrzewo węzła x jest puste, to następnikiem x jest jego najniższy przodek y, którego lewy syn jest także przodkiem x.
- Algorytm opiera się wyłącznie na powyższych własnościach (nie porównuje kluczy węzłów).
- Złożoność: $O(h)$ dla drzewa o wysokości h.

Drzewo poszukiwań binarnych

Znajdowanie następnika węzła x – idea

- Jeśli prawe poddrzewo węzła x jest niepuste, to następnikiem x jest węzeł położony najbardziej na lewo w tym poddrzewie.
- Jeśli prawe poddrzewo węzła x jest puste, to następnikiem x jest jego najniższy przodek y, którego lewy syn jest także przodkiem x.
- Algorytm opiera się wyłącznie na powyższych własnościach (nie porównuje kluczy węzłów).
- Złożoność: $O(h)$ dla drzewa o wysokości h.

Drzewo poszukiwań binarnych

Znajdowanie następnika węzła x

```
procedure TREE-SUCCESSOR(x)
 if x.right ≠ NIL
  return TREE-MINIMUM(x.right)
 y = x.p
 while y ≠ NIL and x = y.right
  x = y
  y = y.p
 return y
```

Drzewo poszukiwań binarnych

Znajdowanie następnika węzła x

```
procedure TREE-SUCCESSOR(x)
 if x.right ≠ NIL
  return TREE-MINIMUM(x.right)
 y = x.p
 while y ≠ NIL and x = y.right
  x = y
  y = y.p
 return y
```

Drzewo poszukiwań binarnych

Wstawianie elementu do drzewa BST – idea:

- Aby wstawić do drzewa BST klucz v, tworzymy węzeł z, dla którego $z.key = v$ oraz $z.right = z.left = NIL$, a następnie przekazujemy go do procedury wstawiającej węzeł
- Miejsce, w które należy wstawić węzeł z, znajdujemy przechodząc po ścieżce od korzenia w dół drzewa, analogicznie jak w przypadku wyszukiwania klucza v
- Węzeł wstawiamy na końcu przebytej ścieżki (jako nowy liść drzewa)
- Złożoność: $O(h)$ dla drzewa o wysokości h.

Drzewo poszukiwań binarnych

Wstawianie elementu do drzewa BST – idea:

- Aby wstawić do drzewa BST klucz v, tworzymy węzeł z, dla którego $z.key = v$ oraz $z.right = z.left = NIL$, a następnie przekazujemy go do procedury wstawiającej węzeł
- Miejsce, w które należy wstawić węzeł z, znajdujemy przechodząc po ścieżce od korzenia w dół drzewa, analogicznie jak w przypadku wyszukiwania klucza v
- Węzeł wstawiamy na końcu przebytej ścieżki (jako nowy liść drzewa)
- Złożoność: $O(h)$ dla drzewa o wysokości h.

Drzewo poszukiwań binarnych

Wstawianie elementu do drzewa BST – idea:

- Aby wstawić do drzewa BST klucz v, tworzymy węzeł z, dla którego $z.key = v$ oraz $z.right = z.left = NIL$, a następnie przekazujemy go do procedury wstawiającej węzeł
- Miejsce, w które należy wstawić węzeł z, znajdujemy przechodząc po ścieżce od korzenia w dół drzewa, analogicznie jak w przypadku wyszukiwania klucza v
- Węzeł wstawiamy na końcu przebytej ścieżki (jako nowy liść drzewa)
- Złożoność: $O(h)$ dla drzewa o wysokości h.

Drzewo poszukiwań binarnych

Wstawianie elementu do drzewa BST – idea:

- Aby wstawić do drzewa BST klucz v, tworzymy węzeł z, dla którego $z.key = v$ oraz $z.right = z.left = NIL$, a następnie przekazujemy go do procedury wstawiającej węzeł
- Miejsce, w które należy wstawić węzeł z, znajdujemy przechodząc po ścieżce od korzenia w dół drzewa, analogicznie jak w przypadku wyszukiwania klucza v
- Węzeł wstawiamy na końcu przebytej ścieżki (jako nowy liść drzewa)
- Złożoność: $O(h)$ dla drzewa o wysokości h.

Drzewo poszukiwań binarnych

Wstawianie elementu do drzewa BST – idea:

- Aby wstawić do drzewa BST klucz v, tworzymy węzeł z, dla którego $z.key = v$ oraz $z.right = z.left = NIL$, a następnie przekazujemy go do procedury wstawiającej węzeł
- Miejsce, w które należy wstawić węzeł z, znajdujemy przechodząc po ścieżce od korzenia w dół drzewa, analogicznie jak w przypadku wyszukiwania klucza v
- Węzeł wstawiamy na końcu przebytej ścieżki (jako nowy liść drzewa)
- Złożoność: $O(h)$ dla drzewa o wysokości h.

Drzewo poszukiwań binarnych

```
procedure TREE-INSERT(T, z)
 y = NIL
 x = T.root
 while x ≠ NIL
  y = x
  if z.key < x.key
   x = x.left
  else
   x = x.right
 z.p = y
 if y = NIL
  T.root = z
 else if z.key < y.key
  y.left = z
 else
  y.right = z
```

Drzewo poszukiwań binarnych

Sortowanie drzewiaste tablicy $A[1..n]$ – idea:

- Utwórz drzewo BST z elementów $A[1], A[2], \ldots, A[n]$, np. wywołując n razy procedurę TREE-INSERT
- Wypisz klucze elementów drzewa we właściwej kolejności, za pomocą procedury INORDER-TREE-WALK (w czasie $\Theta(n)$)

Z twierdzenia o dolnym ograniczeniu złożoności sortowania za pomocą porównań wynika, że każdy algorytm tworzenia drzewa BST z ciągu n elementów wymaga w przypadku pesymistycznym czasu $\Omega(n \lg n)$.

Drzewo poszukiwań binarnych

Sortowanie drzewiaste tablicy $A[1..n]$ – idea:

- Utwórz drzewo BST z elementów $A[1], A[2], \ldots, A[n]$, np. wywołując n razy procedurę TREE-INSERT
- Wypisz klucze elementów drzewa we właściwej kolejności, za pomocą procedury INORDER-TREE-WALK (w czasie $\Theta(n)$)

Z twierdzenia o dolnym ograniczeniu złożoności sortowania za pomocą porównań wynika, że każdy algorytm tworzenia drzewa BST z ciągu n elementów wymaga w przypadku pesymistycznym czasu $\Omega(n \lg n)$.

Drzewo poszukiwań binarnych

Sortowanie drzewiaste tablicy $A[1..n]$ – idea:

- Utwórz drzewo BST z elementów $A[1], A[2], \ldots, A[n]$, np. wywołując n razy procedurę TREE-INSERT
- Wypisz klucze elementów drzewa we właściwej kolejności, za pomocą procedury INORDER-TREE-WALK (w czasie $|\Theta(n))$

Z twierdzenia o dolnym ograniczeniu złożoności sortowania za pomocą porównań wynika, że każdy algorytm tworzenia drzewa BST z ciągu n elementów wymaga w przypadku pesymistycznym czasu $\Omega(n \lg n)$.

Drzewo poszukiwań binarnych

Sortowanie drzewiaste tablicy $A[1..n]$ – idea:

- Utwórz drzewo BST z elementów $A[1], A[2], \ldots, A[n]$, np. wywołując n razy procedurę TREE-INSERT
- Wypisz klucze elementów drzewa we właściwej kolejności, za pomocą procedury INORDER-TREE-WALK (w czasie $\Theta(n)$)

Z twierdzenia o dolnym ograniczeniu złożoności sortowania za pomocą porównań wynika, że każdy algorytm tworzenia drzewa BST z ciągu n elementów wymaga w przypadku pesymistycznym czasu $\Omega(n \lg n)$.

Drzewo poszukiwań binarnych

Sortowanie drzewiaste tablicy $A[1..n]$ – idea:

- Utwórz drzewo BST z elementów $A[1], A[2], \ldots, A[n]$, np. wywołując n razy procedurę TREE-INSERT
- Wypisz klucze elementów drzewa we właściwej kolejności, za pomocą procedury INORDER-TREE-WALK (w czasie $\Theta(n)$)

Z twierdzenia o dolnym ograniczeniu złożoności sortowania za pomocą porównań wynika, że każdy algorytm tworzenia drzewa BST z ciągu n elementów wymaga w przypadku pesymistycznym czasu $\Omega(n \lg n)$.

Drzewo poszukiwań binarnych

Sortowanie drzewiaste – złożoność:

- W przypadku optymistycznym powstanie drzewo o wysokości $h = \Theta(\lg n)$. Algorytm sortowania drzewiastego będzie działać w czasie $\Theta(n \lg n)$.
- W przypadku pesymistycznym powstanie łańcuch, czyli drzewo o wysokości $h = n - 1$. Algorytm sortowania drzewiastego będzie działać w czasie $\Theta(n^2)$

Drzewo poszukiwań binarnych

Sortowanie drzewiaste – złożoność:

- W przypadku optymistycznym powstanie drzewo o wysokości $h = \Theta(\lg n)$. Algorytm sortowania drzewiastego będzie działać w czasie $\Theta(n \lg n)$.
- W przypadku pesymistycznym powstanie łańcuch, czyli drzewo o wysokości $h = n - 1$. Algorytm sortowania drzewiastego będzie działać w czasie $\Theta(n^2)$

Drzewo poszukiwań binarnych

Sortowanie drzewiaste – złożoność:

- W przypadku optymistycznym powstanie drzewo o wysokości $h = \Theta(\lg n)$. Algorytm sortowania drzewiastego będzie działać w czasie $\Theta(n \lg n)$.
- W przypadku pesymistycznym powstanie łańcuch, czyli drzewo o wysokości $h = n - 1$. Algorytm sortowania drzewiastego będzie działać w czasie $\Theta(n^2)$

Drzewo poszukiwań binarnych

Usuwanie elementu z drzewa BST – idea:

- Do procedury przekazujemy wskaźnik do usuwanego węzła z (jeśli chcemy usunąć pewien klucz, trzeba najpierw znaleźć zawierający go węzeł)
- Jeśli z jest liściem, można go usunąć, zastępując wskaźnik do z w jego ojcu $z.p$ stałą NIL
- Jeśli z ma jednego syna, to staje się on synem węzła $z.p$
- Jeśli z ma dwóch synów, to zamiast węzła z usuwamy z drzewa jego następnik y, po czym zastępujemy zawartość (klucz) węzła z zawartością (kluczem) węzła y
- Złożoność: $O(h)$ dla drzewa o wysokości h

Drzewo poszukiwań binarnych

Usuwanie elementu z drzewa BST – idea:

- Do procedury przekazujemy wskaźnik do usuwanego węzła z (jeśli chcemy usunąć pewien klucz, trzeba najpierw znaleźć zawierający go węzeł)
- Jeśli z jest liściem, można go usunąć, zastępując wskaźnik do z w jego ojcu $z.p$ stałą NIL
- Jeśli z ma jednego syna, to staje się on synem węzła $z.p$
- Jeśli z ma dwóch synów, to zamiast węzła z usuwamy z drzewa jego następnik y, po czym zastępujemy zawartość (klucz) węzła z zawartością (kluczem) węzła y
- Złożoność: $O(h)$ dla drzewa o wysokości h

Drzewo poszukiwań binarnych

Usuwanie elementu z drzewa BST – idea:

- Do procedury przekazujemy wskaźnik do usuwanego węzła z (jeśli chcemy usunąć pewien klucz, trzeba najpierw znaleźć zawierający go węzeł)
- Jeśli z jest liściem, można go usunąć, zastępując wskaźnik do z w jego ojcu $z.p$ stałą NIL
- Jeśli z ma jednego syna, to staje się on synem węzła $z.p$
- Jeśli z ma dwóch synów, to zamiast węzła z usuwamy z drzewa jego następnik y, po czym zastępujemy zawartość (klucz) węzła z zawartością (kluczem) węzła y
- Złożoność: $O(h)$ dla drzewa o wysokości h

Drzewo poszukiwań binarnych

Usuwanie elementu z drzewa BST – idea:

- Do procedury przekazujemy wskaźnik do usuwanego węzła z (jeśli chcemy usunąć pewien klucz, trzeba najpierw znaleźć zawierający go węzeł)
- Jeśli z jest liściem, można go usunąć, zastępując wskaźnik do z w jego ojcu $z.p$ stałą NIL
- Jeśli z ma jednego syna, to staje się on synem węzła $z.p$
- Jeśli z ma dwóch synów, to zamiast węzła z usuwamy z drzewa jego następnik y, po czym zastępujemy zawartość (klucz) węzła z zawartością (kluczem) węzła y
- Złożoność: $O(h)$ dla drzewa o wysokości h

Drzewo poszukiwań binarnych

Usuwanie elementu z drzewa BST – idea:

- Do procedury przekazujemy wskaźnik do usuwanego węzła z (jeśli chcemy usunąć pewien klucz, trzeba najpierw znaleźć zawierający go węzeł)
- Jeśli z jest liściem, można go usunąć, zastępując wskaźnik do z w jego ojcu $z.p$ stałą NIL
- Jeśli z ma jednego syna, to staje się on synem węzła $z.p$
- Jeśli z ma dwóch synów, to zamiast węzła z usuwamy z drzewa jego następnik y, po czym zastępujemy zawartość (klucz) węzła z zawartością (kluczem) węzła y
- Złożoność: $O(h)$ dla drzewa o wysokości h

Drzewo poszukiwań binarnych

Usuwanie elementu z drzewa BST – idea:

- Do procedury przekazujemy wskaźnik do usuwanego węzła z (jeśli chcemy usunąć pewien klucz, trzeba najpierw znaleźć zawierający go węzeł)
- Jeśli z jest liściem, można go usunąć, zastępując wskaźnik do z w jego ojcu $z.p$ stałą NIL
- Jeśli z ma jednego syna, to staje się on synem węzła $z.p$
- Jeśli z ma dwóch synów, to zamiast węzła z usuwamy z drzewa jego następnik y, po czym zastępujemy zawartość (klucz) węzła z zawartością (kluczem) węzła y
- Złożoność: $O(h)$ dla drzewa o wysokości h

Drzewo poszukiwań binarnych

```
procedure TREE-DELETE(T, z)
 if z.left = NIL or z.right = NIL
 y = z
 else y = TREE-SUCCESSOR(z)
 if y.left ≠ NIL
  x = y.left
 else x = y.right
 if x ≠ NIL
  x.p = y.p
 if y.p = NIL
  T.root = x
 else if y = y.p.left
  y.p.left = x
 else y.p.right = x
 if y ≠ z
  z.key = y.key // skopiuj zawartość dodatkowych pól z y do z
```

Drzewo poszukiwań binarnych

Zrównoważone drzewa binarne:

- Drzewo binarne T nazywamy zrównoważonym, jeśli dla dowolnego węzła x tego drzewa wysokości jego lewego i prawego poddrzewa różnią się co najwyżej o 1
- Zrównoważone drzewo binarne T nazywamy doskonale zrównoważonym, jeśli wszystkie jego liście znajdują się na co najwyżej dwóch poziomach
- Wysokość zrównoważonego drzewa binarnego o n węzłach jest rzędu $\Theta(\lg n)$

Drzewo poszukiwań binarnych

Zrównoważone drzewa binarne:

- Drzewo binarne T nazywamy zrównoważonym, jeśli dla dowolnego węzła x tego drzewa wysokości jego lewego i prawego poddrzewa różnią się co najwyżej o 1
- Zrównoważone drzewo binarne T nazywamy doskonale zrównoważonym, jeśli wszystkie jego liście znajdują się na co najwyżej dwóch poziomach
- Wysokość zrównoważonego drzewa binarnego o n węzłach jest rzędu $\Theta(\lg n)$

Drzewo poszukiwań binarnych

Zrównoważone drzewa binarne:

- Drzewo binarne T nazywamy zrównoważonym, jeśli dla dowolnego węzła x tego drzewa wysokości jego lewego i prawego poddrzewa różnią się co najwyżej o 1
- Zrównoważone drzewo binarne T nazywamy doskonale zrównoważonym, jeśli wszystkie jego liście znajdują się na co najwyżej dwóch poziomach
- Wysokość zrównoważonego drzewa binarnego o n węzłach jest rzędu $\Theta(\lg n)$

Drzewo poszukiwań binarnych

Zrównoważone drzewa binarne:

- Drzewo binarne T nazywamy zrównoważonym, jeśli dla dowolnego węzła x tego drzewa wysokości jego lewego i prawego poddrzewa różnią się co najwyżej o 1
- Zrównoważone drzewo binarne T nazywamy doskonale zrównoważonym, jeśli wszystkie jego liście znajdują się na co najwyżej dwóch poziomach
- Wysokość zrównoważonego drzewa binarnego o n węzłach jest rzędu $\Theta(\lg n)$

Drzewo poszukiwań binarnych

Złożoność operacji na drzewie BST o n węzłach:

- Podstawowe operacje na drzewie BST (różne rodzaje wyszukiwania, wstawianie, usuwanie) działają w czasie $O(h)$, gdzie h jest wysokością drzewa
- Wysokość h drzewa BST o n węzłach spełnia zależność $\lfloor \lg n \rfloor \leq h \leq n - 1$
- W ogólnym przypadku podstawowe operacje na n-elementowym drzewie BST działają w czasie $O(n)$
- Podstawowe operacje na zrównoważonym n-elementowym drzewie BST działają w czasie $O(\lg n)$
- W zastosowaniach, w których ważna jest złożoność operacji na drzewie BST, wykorzystuje się opartą na nim strukturę zdanych zapewniającą zrównoważenie drzewa (np. drzewa AVL, drzewa czerwono-czarne)

Drzewo poszukiwań binarnych

Złożoność operacji na drzewie BST o n węzłach:

- Podstawowe operacje na drzewie BST (różne rodzaje wyszukiwania, wstawianie, usuwanie) działają w czasie $O(h)$, gdzie h jest wysokością drzewa
- Wysokość h drzewa BST o n węzłach spełnia zależność $\lfloor \lg n \rfloor \leq h \leq n - 1$
- W ogólnym przypadku podstawowe operacje na n-elementowym drzewie BST działają w czasie $O(n)$
- Podstawowe operacje na zrównoważonym n-elementowym drzewie BST działają w czasie $O(\lg n)$
- W zastosowaniach, w których ważna jest złożoność operacji na drzewie BST, wykorzystuje się oparte na nim struktury danych zapewniające zrównoważenie drzewa (np. drzewa AVL, drzewa czerwono-czarne)

Drzewo poszukiwań binarnych

Złożoność operacji na drzewie BST o n węzłach:

- Podstawowe operacje na drzewie BST (różne rodzaje wyszukiwania, wstawianie, usuwanie) działają w czasie $O(h)$, gdzie h jest wysokością drzewa
- Wysokość h drzewa BST o n węzłach spełnia zależność $\lfloor \lg n \rfloor \leq h \leq n - 1$
- W ogólnym przypadku podstawowe operacje na n-elementowym drzewie BST działają w czasie $O(n)$
- Podstawowe operacje na zrównoważonym n-elementowym drzewie BST działają w czasie $O(\lg n)$
- W zastosowaniach, w których ważna jest złożoność operacji na drzewie BST, wykorzystuje się oparte na nim struktury danych zapewniające zrównoważenie drzewa (np. drzewa AVL, drzewa czerwono-czarne)

Drzewo poszukiwań binarnych

Złożoność operacji na drzewie BST o n węzłach:

- Podstawowe operacje na drzewie BST (różne rodzaje wyszukiwania, wstawianie, usuwanie) działają w czasie $O(h)$, gdzie h jest wysokością drzewa
- Wysokość h drzewa BST o n węzłach spełnia zależność $\lfloor \lg n \rfloor \leq h \leq n - 1$
- W ogólnym przypadku podstawowe operacje na n-elementowym drzewie BST działają w czasie $O(n)$
- Podstawowe operacje na zrównoważonym n-elementowym drzewie BST działają w czasie $O(\lg n)$
- W zastosowaniach, w których ważna jest złożoność operacji na drzewie BST, wykorzystuje się oparte na nim struktury danych zapewniające zrównoważenie drzewa (np. drzewa AVL, drzewa czerwono-czarne)

Drzewo poszukiwań binarnych

Złożoność operacji na drzewie BST o n węzłach:

- Podstawowe operacje na drzewie BST (różne rodzaje wyszukiwania, wstawianie, usuwanie) działają w czasie $O(h)$, gdzie h jest wysokością drzewa
- Wysokość h drzewa BST o n węzłach spełnia zależność $\lfloor \lg n \rfloor \leq h \leq n - 1$
- W ogólnym przypadku podstawowe operacje na n-elementowym drzewie BST działają w czasie $O(n)$
- Podstawowe operacje na zrównoważonym n-elementowym drzewie BST działają w czasie $O(\lg n)$
- W zastosowaniach, w których ważna jest złożoność operacji na drzewie BST, wykorzystuje się oparte na nim struktury danych zapewniające zrównoważenie drzewa (np. drzewa AVL, drzewa czerwono-czarne)

Drzewo poszukiwań binarnych

Złożoność operacji na drzewie BST o n węzłach:

- Podstawowe operacje na drzewie BST (różne rodzaje wyszukiwania, wstawianie, usuwanie) działają w czasie $O(h)$, gdzie h jest wysokością drzewa
- Wysokość h drzewa BST o n węzłach spełnia zależność $\lfloor \lg n \rfloor \leq h \leq n - 1$
- W ogólnym przypadku podstawowe operacje na n-elementowym drzewie BST działają w czasie $O(n)$
- Podstawowe operacje na zrównoważonym n-elementowym drzewie BST działają w czasie $O(\lg n)$
- W zastosowaniach, w których ważna jest złożoność operacji na drzewie BST, wykorzystuje się oparte na nim struktury danych zapewniające zrównoważenie drzewa (np. drzewa AVL, drzewa czerwono-czarne)