Drzewa

Drzewa

- Drzewo (ang. tree) – zbiór węzłów powiązanych wskaźnikami, spójny i bez cykli.
- Drzewo posiada wyróżniony węzeł początkowy nazywany korzeniem (ang. root).
- Drzewo ukorzenione jest strukturą hierarchiczną.
- Korzeń węzła znajduje się na poziomie 0; numer poziomu danego węzła w drzewie jest wyznaczony odległością krawędziową od korzenia.
- Liściem (ang. leaf) w drzewie jest węzeł bez żadnego następnika.
- Węzeł wewnętrzny posiada co najmniej jednego następnika.
- Każdy węzeł oprócz korzenia posiada jednego poprzednika.

Drzewa

- **Wysokość drzewa** – długość najdłuższej ścieżki od korzenia do liścia (liczba węzłów) – inaczej liczba poziomów na których zapisane jest drzewo.
- **Głębokość węzła** – długość ścieżki od korzenia do tego węzła (liczba węzłów) – inaczej numer poziomu, na którym znajduje się węzeł.
- Uwaga: poziomy w drzewie numerujemy od 0; korzeń znajduje się na poziomie 0; głębokość korzenie wynosi 0.

Drzewo

- Drzewo uporządkowane ma ponumerowanych (oznaczonych) następników.
- Drzewo uporządkowane w reprezentacji na-lewo-syn-na-prawo-brat składa się z węzłów, które zawierają następujące pola:
  - value – wartość pamiętana w węźle,
  - parent – wskaźnik na poprzednika,
  - left-child – wskaźnik na lewego syna (lista następników),
  - sibling – wskaźnik na prawego brata.
- Każdy węzeł w drzewie przechowuje jedną wartość.

Drzewo binarne

- Drzewo binarne (ang. binary tree) to drzewo ukorzenione, w którym każdy wierzchołek posiada co najwyżej dwóch rozróżnialnych synów – lewego i prawego.
- Węzeł drzewa binarnego zawiera pole z kluczem `value` oraz wskaźnik na lewe poddrzewo `left` (zakorzenione w jego lewym synu) i wskaźnik na prawe poddrzewo `right` (zakorzenione w jego prawym synu). W drzewie binarnym poruszamy się tylko od korzenia w kierunku liści.
- Jeśli w węźle drzewa binarnego znajduje się wskaźnik na ojca `parent`, to w drzewie takim można się przemieszczać w obu kierunkach – od korzenia w kierunku jakiegoś liścia oraz w drugą stronę w kierunku korzenia. O drzewie takim mówimy, że jest dwukierunkowe.
- Wysokość binarnego drzewa n-elementowego jest nie mniejsza niż log(n) i nie większa niż n.

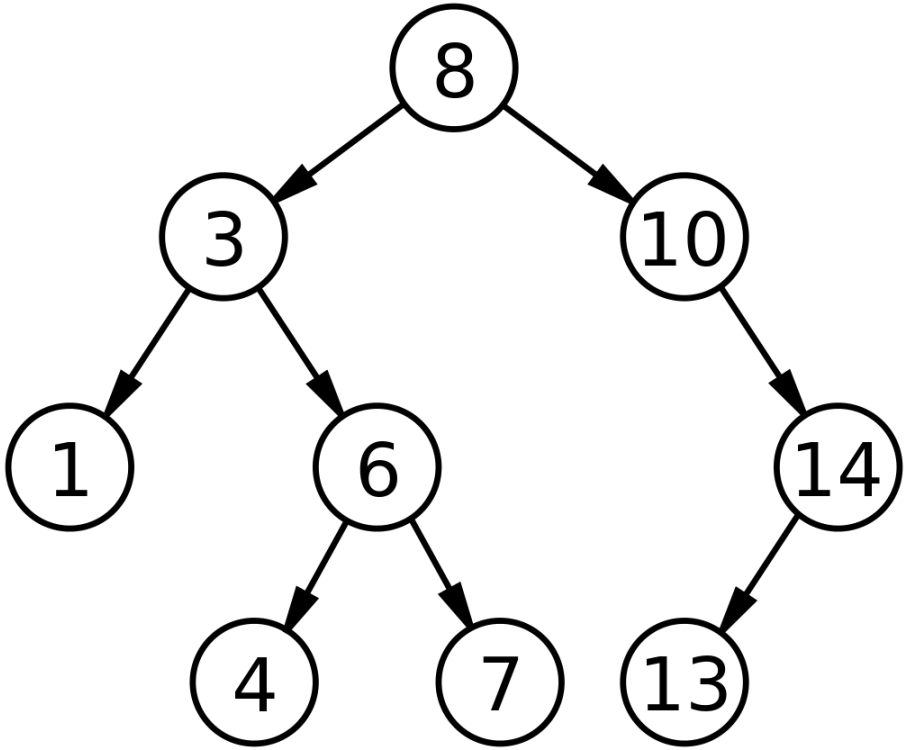
Szczególne drzewa binarne

- Drzewo regularne to drzewo binarne, w którym każdy węzeł wewnętrzny na dwóch synów.
- Drzewo pełne to drzewo regularne, w którym wszystkie liście są na tym samym poziomie.
- Drzewo pełne o wysokości h ma 2^h-1 węzłów.
- Drzewo zupełne to drzewo pełne, z którego usunięto część liści z prawej strony.
- Drzewo zupełne o wysokości h ma od 2^{h-1} do 2^h-1 węzłów.
- Drzewo z wartownikiem – każdy wskaźnik pusty jest ustawiany na wartownika, a w wartowniku mamy wskaźnik do korzenia.

Drzewo BST

- Drzewo binarnych poszukiwań (ang. binary search tree), w skrócie BST, to drzewo binarne, w którym przechowujemy elementy z pewnego uniwersum z porządkiem liniowym i w drzewie tym jest zachowany porządek symetryczny.
- W drzewie binarnym jest zachowany porządek symetryczny, gdy elementy mniejsze od korzenia znajdują się w lewym poddrzewie, elementy większe od korzenia w prawym poddrzewie oraz w lewym i w prawym poddrzewie też jest zachowany porządek symetryczny.

Drzewo BST



Drzewo BST

- Wyszukiwanie w drzewie BST – działa podobnie do poszukiwania binarnego:

```
serach (węzeł *w, x) -> boolean
{
        if (w == null) return false;
        if (x < w.value) return search(w.left, x);
        else if (w.value < x) return search(w.right, x);
        else return true;
}
```

Drzewo BST

- Element minimalny znajduje się najbardziej po lewej stronie w drzewie BST – od korzenia poruszamy się ciągle w lewo.
- Element maksymalny znajduje się najbardziej po prawej stronie w drzewie BST – od korzenia poruszamy się ciągle w prawo.

Drzewo BST

- Wstawienie nowego elementu do drzewa BST – znajdujemy mu miejsce tak jak w wyszukiwaniu binarnym (aż dojdziemy do wskaźnika pustego):

```
insert (węzeł *w, x) -> node*
{
        if (w == null) return new node(x);
        if (x < w.value) w.left := insert(w.left, x);
        else if (w.value < x) w.right := insert(w.right, x);
        else w.value := x;
        return w;
}
```

Drzewo BST

- Usunięcie elementu z drzewa BST – znajdujemy miejsce tego elementu tak jak w wyszukiwaniu binarnym (jak dojdziemy do wskaźnika pustego to elementu nie ma w drzewie):
  - gdy element jest w liściu, to odcinam liść;
  - gdy element jest w węźle z jednym następnikiem, to usuwam ze ścieżki ten węzeł;
  - gdy element jest w węźle z dwoma synami, to z prawego poddrzewa usuwamy minimum (albo z lewego poddrzewa usuwamy maksimum) i przenosimy je do tego węzła.

Drzewo BST

- Usunięcie elementu z drzewa BST:

```
remove (węzeł *w, x) -> node*
{
    if (w == null) return null;
    if (x < w.value) w.left := remove(w.left, x);
    else if (w.value < x) w.right := remove(w.right, x);
    else if (w jest liściem) { delete w; return null; }
    else if (w me tylko lewego syna)
        { s := w.left; delete w; return s; }
    else if (w me tylko prawego syna)
        { s := w.right; delete w; return s; }
    else { w.value := max(w.left); w.left := remove-max(w.left); }
    return w;
}
```

Przeglądanie drzew BST

- In-order – najpierw jest przeglądane i przetwarzane lewe poddrzewo w porządku in-order, potem korzeń a na końcu prawe poddrzewo w porządku in-order.
- Przeglądanie drzewa BST w porządku in-order gwarantuje, że węzły tego drzewa zostaną przetworzone w kolejności od najmniejszej do największej wartości.
- Przeglądanie in-order można wykorzystać do sortowania danych.

Przeglądanie drzew BST

- Pre-order – najpierw jest przeglądany i przetwarzany korzeń drzewa, potem lewe poddrzewo w porządku pre-order a na końcu prawe poddrzewo w porządku pre-order.
  - Przeglądanie drzewa BST w porządku pre-order gwarantuje, że na początku będą przetworzone węzły z lewej ścieżki tego drzewa w kolejności od korzenia do węzła o najmniejszej wartości.
- Post-order – najpierw jest przeglądane i przetwarzane lewe poddrzewo w porządku post-order, potem prawe poddrzewo w porządku post-order a na końcu korzeń drzewa.
  - Przeglądanie drzewa BST w porządku post-order gwarantuje, że na końcu będą przetworzone węzły z prawej ścieżki tego drzewa w kolejności od węzła o największej wartości do korzenia.

Drzewo BST

- Przetwarzanie drzewa binarnego w porządku in-order):

```
inorder (węzeł *w) -> void
{
        if (w == null) return;
        inorder(w.left);
        przetworzenie w.value;
        inodrer(w.right);
}
```

Problemy

- Jak przeglądnąć drzewo BST iteracyjnie w porządku in-order? (średnio trudne)
- Jak przeglądnąć drzewo binarne iteracyjnie w porządku in-order? (bardzo trudne)