Centroidy, drzewa z korzeniem, drzewa BST, drzewa przedziałowe

dr Anna Beata Kwiatkowska, UMK Toruń

Centroid

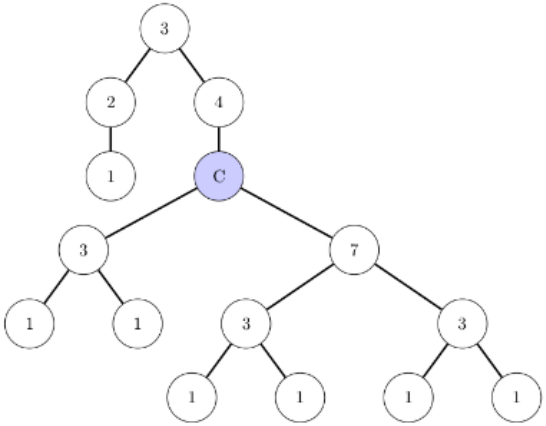
Definicja 5.1

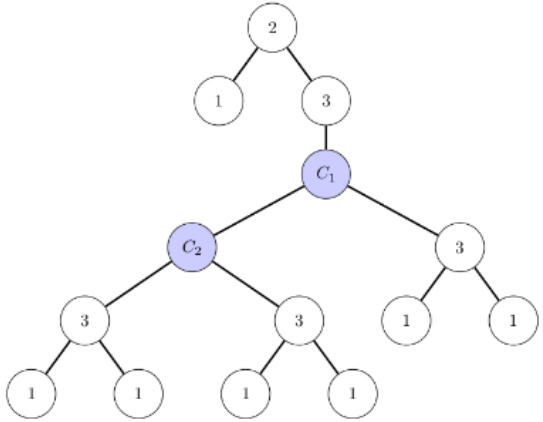
Centroid to wierzchołek, który dzieli drzewo na poddrzewa o liczbie wierzchołków nie większej niż połowa liczby wierzchołków całego drzewa. Każde drzewo ma jeden lub dwa centroidy połączone krawędzią.

Centroidu nie należy mylić z centrum w drzewie.

Centroidy przydają się w algorytmach na drzewach korzystających z techniki dziel i zwyciężaj.

Przykład: Przykładowe drzewa i ich centroidy.





Centroid - algorytm

Dane jest drzewo T o n wierzchołkach. Chcemy znaleźć wierzchołek (wierzchołki), który jest centroidem (centroidami).

Algorytm:

Ukorzeniamy drzewo w dowolnym wierzchołku v. Jeśli poddrzewa nie są odpowiedniej wielkości, przesuwamy się do sąsiedniego wierzchołka w bardziej licznym poddrzewie.

Algorytm ma złożoność O(n).

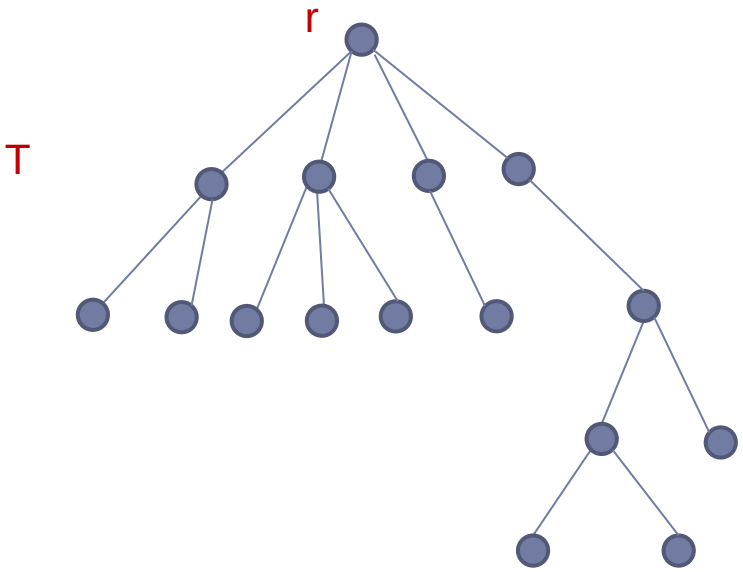
Drzewo z korzeniem

Definicja 5.2

Drzewo T, w którym jeden wierzchołek r jest wyróżniony z wszystkich innych nazywamy **drzewem z korzeniem w wierzchołku r**.

Wierzchołki wiszące nazywamy liśćmi. Wierzchołki, które nie są liśćmi nazywamy wierzchołkami wewnętrznymi. W drzewach ukorzenionych wierzchołki nazywane są węzłami.

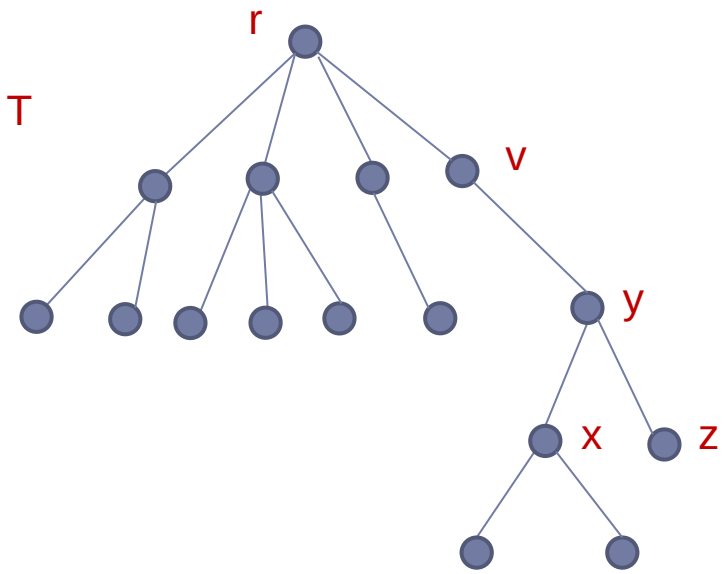
Przykład: Drzewo ukorzenione T o korzeniu r.



Drzewa z korzeniem

Rozpatrzmy węzeł x drzewa T o korzeniu r. Każdy węzeł na ścieżce z r do x nazywamy przodkiem x. Każdy węzeł na ścieżce w kierunku przeciwnym do korzenia nazywamy potomkiem x.

Potomka bezpośrednio sąsiadującego z x nazywamy synem. Przodka bezpośrednio sąsiadującego z x nazywamy ojcem.



Przykład:

Węzeł y jest ojcem węzłów x i z.

Węzły x i z są synami węzła y.

Węzły r, v, y są przodkami węzła z.

Węzły y, x, z są potomkami węzła v

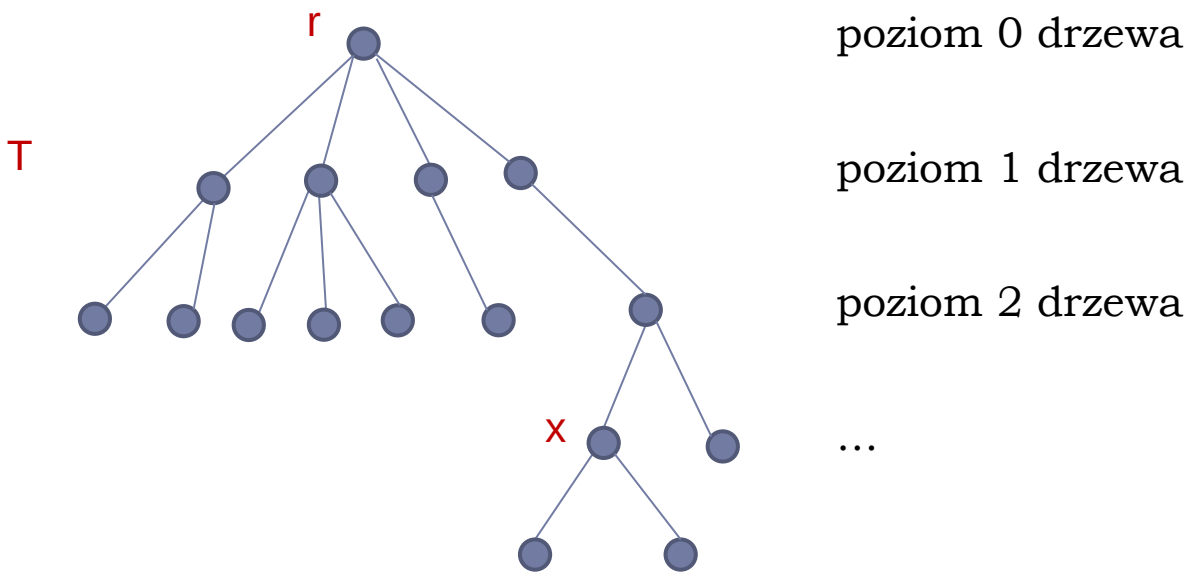
Drzewa z korzeniem

Rozpatrzmy drzewo T z korzeniem r.

Wysokością węzła x w drzewie T nazywamy liczbę krawędzi na ścieżce prostej z x do najdalszego liścia. Wysokością drzewa jest wysokość korzenia r.

Głębokością węzła x w drzewie T nazywamy liczbę krawędzi na ścieżce prostej od korzenia r do węzła x.

Przykład: Drzewo ukorzenione T z korzeniem r o wysokości 4. Wysokość węzła x wynosi 1. Głębokość węzła x wynosi 3.



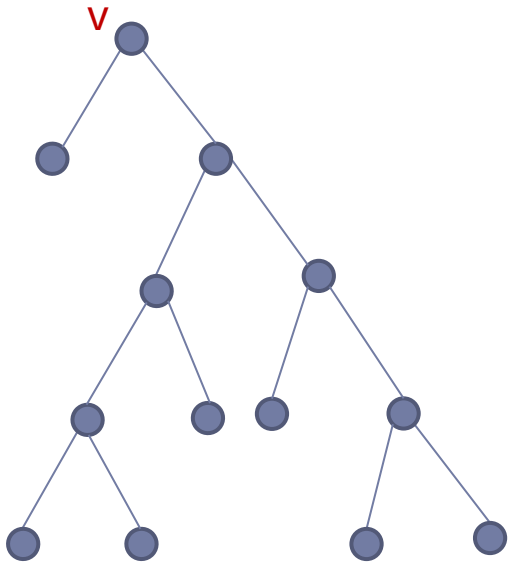
Drzewa binarne

Definicja 5.3

Drzewo binarne, to drzewo w którym dokładnie jeden wierzchołek jest stopnia drugiego, a każdy z pozostałych wierzchołków jest stopnia pierwszego lub trzeciego.

Każde drzewo binarne jest drzewem z korzeniem. Rolę korzenia pełni wierzchołek stopnia drugiego. Rozpatrujemy drzewa binarne uporządkowane, w których synowie są nazywani od lewej do prawej: lewy i prawy syn.

Przykładowe drzewo binarne:



Drzewa binarne - własności

Własność 5.1

Liczba wierzchołków n w drzewie binarnym jest zawsze nieparzysta.

Dowód: Z definicji drzewa binarnego wynika, że dokładnie jeden wierzchołek jest stopnia parzystego, tzn. korzeń. Jest więc n-1 wierzchołków stopnia nieparzystego. Ponieważ w grafie liczba wierzchołków stopnia nieparzystego jest zawsze parzysta (Twierdzenie 1.2), więc n-1 jest liczbą parzystą. Zatem n jest nieparzyste. □

Własność 5.2.

Jeśli p jest liczbą wierzchołków wiszących w drzewie binarnym T o n wierzchołkach, to $p=(n+1)/2$.

Dowód: W drzewie T liczba wierzchołków stopnia trzeciego jest równa n-p-1. Zatem liczba krawędzi

$$n-1=(p+2+3(n-p-1))/2, \text{ stąd } p=(n+1)/2 \ \square$$

Wniosek:

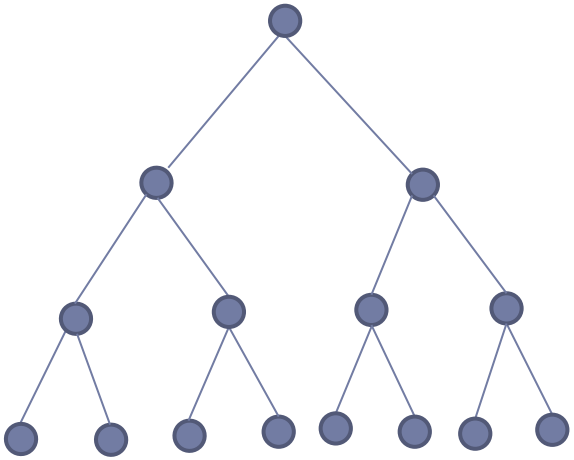
Liczba wierzchołków wewnętrznych w drzewie binarnym jest o jeden mniejsza od liczby wierzchołków wiszących.

Pełne drzewa binarne (wg A. Aho)

Definicja 5.3

Pełne drzewo binarne (wg A. Aho) o wysokości h, to drzewo, w którym każdy wierzchołek o głębokości mniejszej niż h ma lewego i prawego syna oraz każdy wierzchołek o głębokości h jest liściem.

Przykład: Pełne drzewo binarne



Pełne drzewa binarne (wg A. Aho)

Własność 5.3

Pełne drzewo binarne o wysokości h ma $2^{h+1}-1$ wierzchołków.

Dowód:

Sumujemy liczby wierzchołków z kolejnych poziomów

$2^0+2^1+2^2+ \ldots + 2^h = 2^{h+1} -1$ □

Własność 5.4

Wysokość pełnego drzewa binarnego o t liściach jest równa $h=\log_2 t$

Dowód:

Pełne drzewo binarne o t liściach ma $2t-1$ wierzchołków, zatem

$2t-1=2^{h+1}-1$, ale ponieważ $t=2^h$ mamy $h=\log_2 t$ □

Reprezentacja drzew binarnych w programach

Jeśli wysokość `h` nie jest zbyt duża reprezentujemy drzewo w tablicy/liście o rozmiarze 2^h.

- korzeń jest na miejscu `1`.
- dla ojca i lewy syn jest w miejscu `2i`, prawy w miejscu `2i+1`

Drzewa binarne o większej wysokości i `n` węzłach możemy pamiętać w tablicy/liście `n` elementowej.

- dla każdego wierzchołka pamiętamy jedynie jego ojca
- lub dla każdego wierzchołka pamiętamy parę synów (n elementowa tablica par)

Drzewa poszukiwań binarnych (BST)

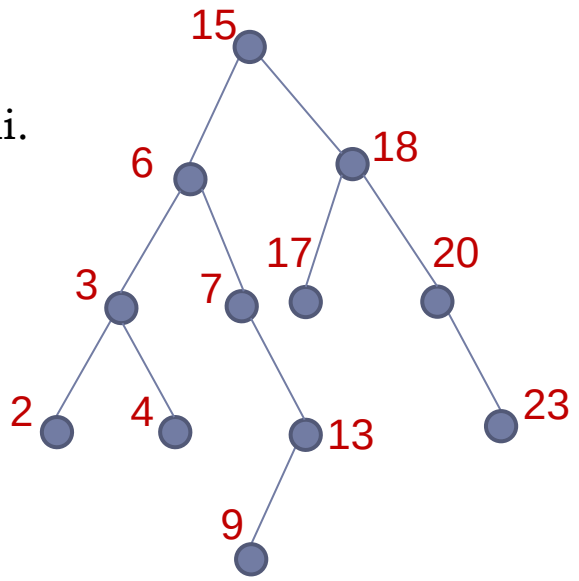
Definicja

Drzewo BST (*ang. binary search trees*) to drzewo binarne, w którym dla każdego wierzchołka wszystkie wartości w lewym poddrzewie są mniejsze (ew. mniejsze równe) od wartości w tym wierzchołku, a wszystkie wartości w prawym poddrzewie są większe (ew. większe równe)

Interesujące nas wartości przechowywane w węzłach drzewa BST nazywamy kluczami.

Przykład:

Drzewo BST o kluczach będących liczbami naturalnymi.



Przechodzenie drzew binarnych - preorder

Przechodzenie drzewa to odwiedzanie wierzchołków drzewa w pewnym porządku.

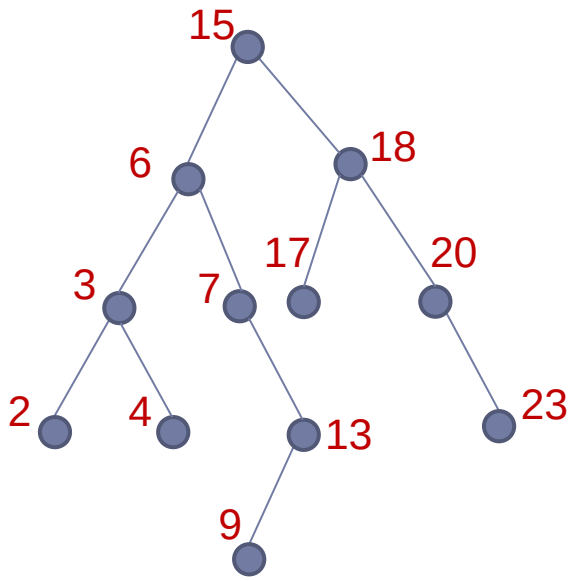
Przechodzenie drzewa binarnego metodą preorder – wypisujemy klucz znajdujący się w korzeniu każdego rozpatrywanego drzewa przed wypisaniem wartości najpierw w lewym, potem w prawym poddrzewie.

Algorytm preorder(r):

1. wypisz klucz znajdujący się w korzeniu
2. wykonaj **preorder** na lewym poddrzewie
3. wykonaj **preorder** na prawym poddrzewie

Przykład: Porządek preorder dla drzewa po prawej

15, 6, 3, 2, 4, 7, 13, 9, 18, 17, 20, 23



Przechodzenie drzew binarnych - postorder

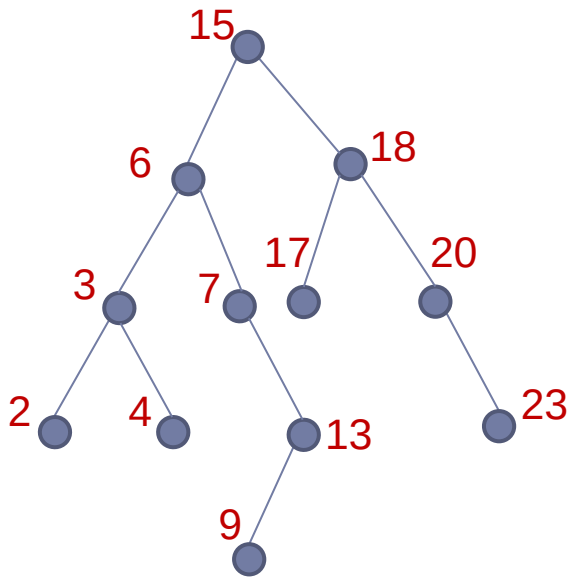
Przechodzenie drzewa binarnego metodą postorder – wypisujemy najpierw klucze lewego, potem prawego poddrzewa, a na końcu wypisujemy klucz znajdujący się w korzeniu.

Algorytm postorder(r):

1. wykonaj **postorder** na lewym poddrzewie
2. wykonaj **postorder** na prawym poddrzewie
3. wypisz klucz znajdujący się w korzeniu

Przykład: Porządek postorder dla drzewa po prawej

2, 4, 3, 9, 13, 7, 6, 17, 23, 20, 18, 15



Przechodzenie drzew binarnych - inorder

Przechodzenie drzewa binarnego metodą inorder – wypisujemy najpierw klucze lewego poddrzewa, potem klucz znajdujący się w korzeniu, a na końcu klucze prawego poddrzewa.

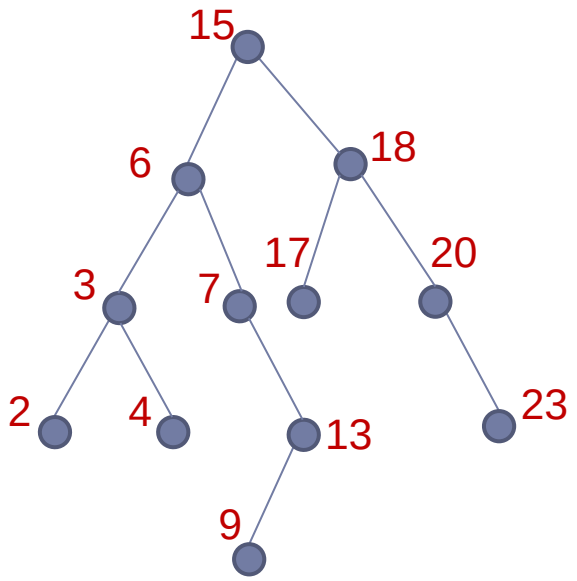
Porządek inorder wypisuje klucze posortowane niemalejąco.

Algorytm inorder(r):

1. wykonaj **inorder** na lewym poddrzewie
2. wypisz klucz znajdujący się w korzeniu
3. wykonaj **inorder** na prawym poddrzewie

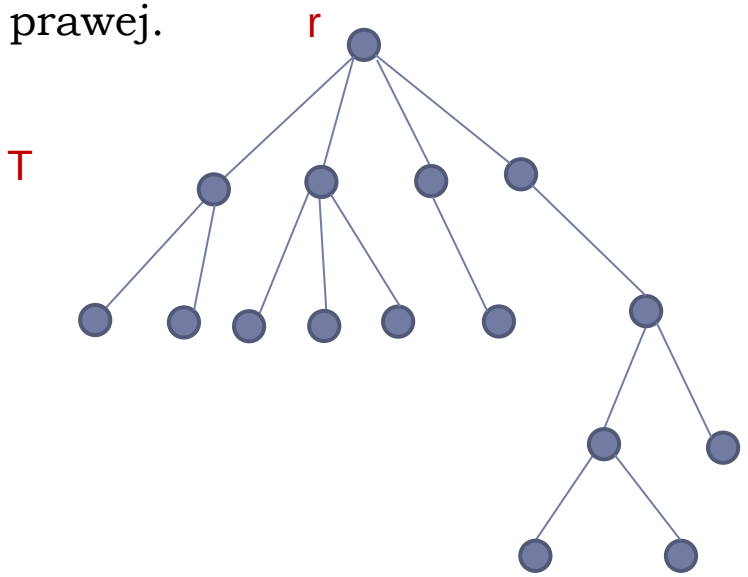
Przykład: Porządek inorder dla drzewa po prawej

2, 3, 4, 6, 7, 9, 13, 15, 17, 18, 20, 23



Przechodzenie dowolnych drzew ukorzenionych

Rozpatrzmy przeszukiwanie drzewa T algorytmem DFS od korzenia r w kolejności sąsiadów od lewej do prawej.



Jeśli wypiszemy klucze w kolejności odwiedzenia wierzchołków algorytmem DFS po raz pierwszy to otrzymujemy porządek **preorder**.

Jeśli wypiszemy klucze w kolejności wycofywania się algorytmu DFS z wierzchołków to otrzymujemy porządek **postorder**.

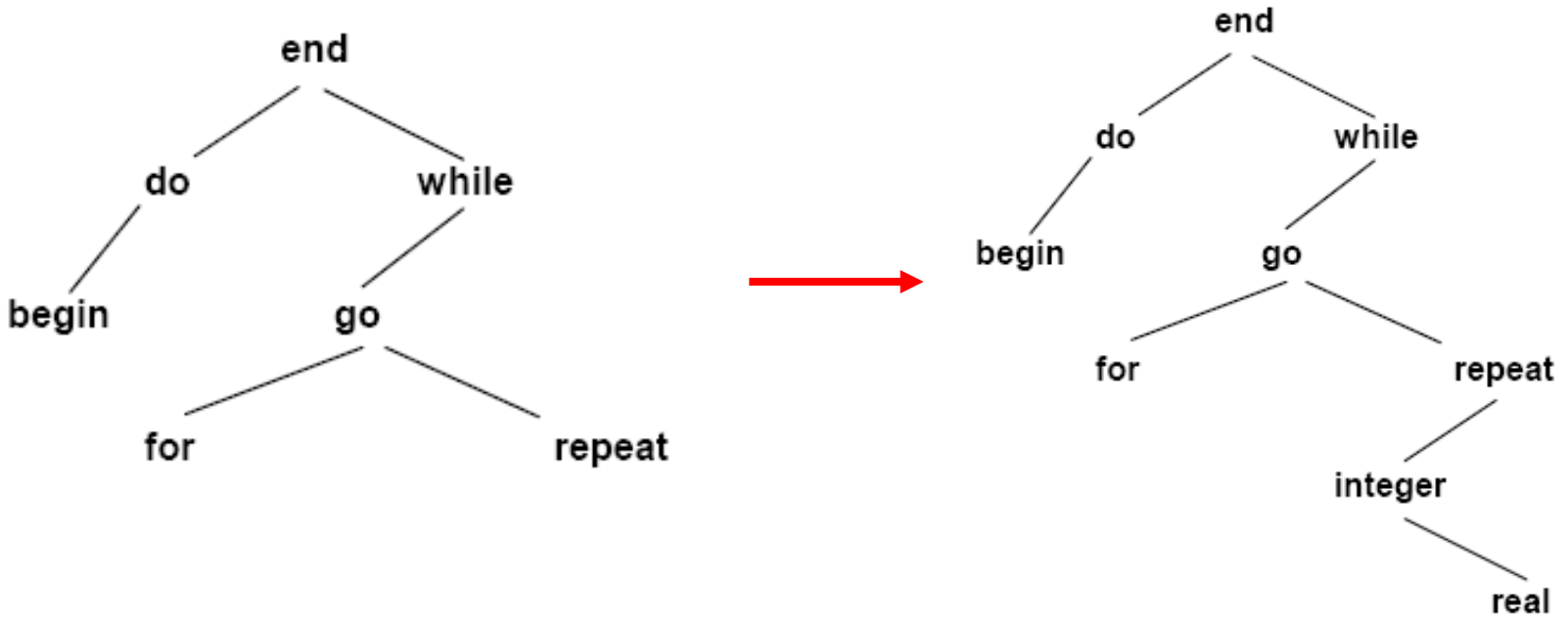
Te porządki wykorzystywane są w wielu algorytmach dotyczących grafów. Kluczami mogą być dowolne obiekty, dla których określony jest porządek.

Wstawianie elementów do drzewa BST

Korzystamy z definicji drzewa BST

Przykład: Rozpatrujemy słowa utworzone z małych liter alfabetu angielskiego z porządkiem leksykograficznym.

Do drzewa utworzonego kolejno ze słów: `end`, `while`, `go`, `do`, `begin`, `for`, `repeat` wstawiamy następne słowa `integer` i `real`. Zaczynamy od korzenia przechodząc w lewo, jeśli wstawiane słowo jest wcześniej w porządku, a w prawo w przeciwnym wypadku.



Usuwanie elementów z drzewa BST

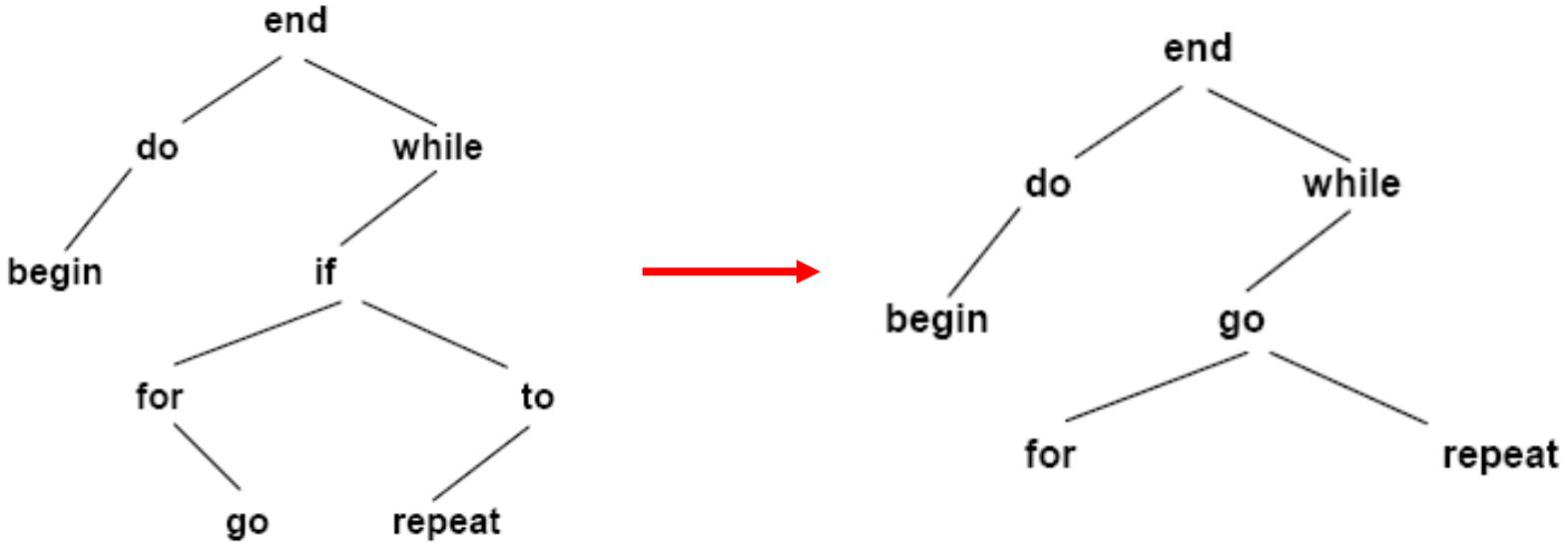
Jeśli chcemy usunąć element z wierzchołka v. Zachodzą trzy przypadki:

1. Jeśli wierzchołek v jest liściem, to usuwamy wierzchołek v z drzewa.
2. Jeśli wierzchołek v ma dokładnie jednego syna w, usuwamy v z drzewa a ojcem wierzchołka w zostaje ojciec wierzchołka v. Jeśli v jest korzeniem, to po usunięciu v korzeniem zostaje jego syn w.
3. Jeśli wierzchołek v ma dwóch synów, to w lewym poddrzewie wierzchołka v ustalamy wierzchołek w o największym kluczu (jest to wierzchołek położony najbardziej na prawo w lewym poddrzewie). Nadajemy wierzchołkowi v klucz wierzchołka w. Rekurencyjnie usuwamy wierzchołek w z poddrzewa

Usuwanie elementów z drzewa BST

Przykład

Z drzewa utworzonego ze słów :`end, while, if, do, for, go, begin, repeat` usuwamy słowo `if` oraz słowo `to`.



Przeszukiwanie drzewa binarnego - złożoność

Złożoność pesymistyczna ma miejsce w przypadku drzew niezrównoważonych, np. gdy drzewo poszukiwań binarnych składa się z pojedynczego łańcucha prawych synów.

Dla każdej z operacji:

- wstawianie elementu,
- usuwanie elementu,
- szukanie elementu o danym kluczu,
- szukanie elementu najmniejszego/największego

złożoność pesymistyczna to O(n).

Twierdzenie 5.1.

Średnia liczba porównań potrzebnych do wstawienia n losowo wybranych elementów w drzewie binarnych poszukiwań, które na początku jest puste wynosi **O(n log n),** dla n≥1.

Powyższe twierdzenie odnosi się do wszystkich instrukcji. Źródłem złej efektywności algorytmów na drzewach jest niezrównoważenie drzew.

Drzewo przedziałowe

Drzewo przedziałowe jest to przykład drzewa binarnego zrównoważonego. Innymi drzewami zrównoważonymi są m.in. drzewa czerwono-czarne, drzewa AVL i 2-3 drzewa.

Zastosowanie

Drzewa przedziałowe możemy stosować przy rozwiazywaniu problemów dotyczących własności obiektów z danego przedziału: np. suma, max i min, liczba wyróżnionych znaków itp., które możemy odczytać przechodząc po wysokości drzewa, z przedziałów bazowych.

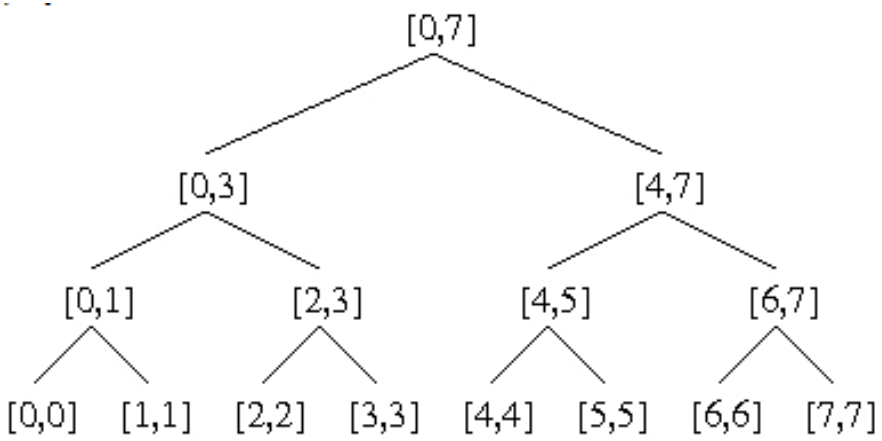
Stosując drzewa przedziałowe przy k zapytaniach dla M kluczy (np. liczb) otrzymujemy złożoność obliczeniową O(k*logM) zamiast złożoności O(k*M).

Dla ułatwienia w dalszych rozważaniach zakładamy, że liczba kluczy $M=2^k$. Gdy tak nie jest uzupełniamy klucze do tego rozmiaru wartościami niewpływającymi na wynik. Klucze numerujemy od zera, czyli od 0 do M-1.

Drzewo przedziałowe - konstrukcja

Rozważamy podprzedziały przedziału [0, M-1]. W drzewie przedziałowym liście reprezentują pojedyncze klucze, a węzły wewnętrzne – przedziały.

Np. M=8



Przedziały w węzłach nazywamy bazowymi.

Pytania dla M elementów:

Ile jest dokładnie węzłów i liści w drzewie przedziałowym?

Odp.: $2^k+2^{k-1}+\ldots + 2^0= 2^{k+1}-1=2*M-1$, O(M)

Jakiego rzędu jest głębokość drzewa przedziałowego?

Odp.: O(log(M))

Numerujemy węzły poziomami poczynając od korzenia, któremu dajemy numer 1, jaki numer mają synowie i ojciec węzła o numerze v?

Odp.: syn lewy: 2*v, syn prawy 2*v+1, ojciec v/2

W jakiej strukturze możemy pamiętać drzewo przedziałowe?

Odp.: Tablica/lista statyczna o odpowiednim rozmiarze. Jakim?

Ostatnich M elementów struktury to kolejne klucze. Przechodząc ja od końca obliczamy wartość dla ojca o indeksie i, odwołując się do synów o indeksach 2i i 2i+1.

Spacer od liścia do korzenia

Jaki numer w tablicy/liście przechowującej drzewo przedziałowe ma liść reprezentujący klucz numer x?

Odp.: M+x

Spacer po drzewie przedziałowym od liścia do korzenia:

```
int v = M + x;
while (v != 1)
        {
        // Zrób coś.
        v /= 2;
        }
```

Obliczanie wartości w węzłach drzewa przedziałowego

Jeśli jesteśmy w węźle v, **W[v]** oznacza wartość, który chcemy pamiętać dla elementu oznaczonego przez ten węzeł.

Na przykład, jeśli zapytania będą dotyczyły sumy w przedziale to korzystamy ze wzoru:

$$\mathbf{W}[v]=\mathbf{W}[2*v]+\mathbf{W}[2*v+1]$$

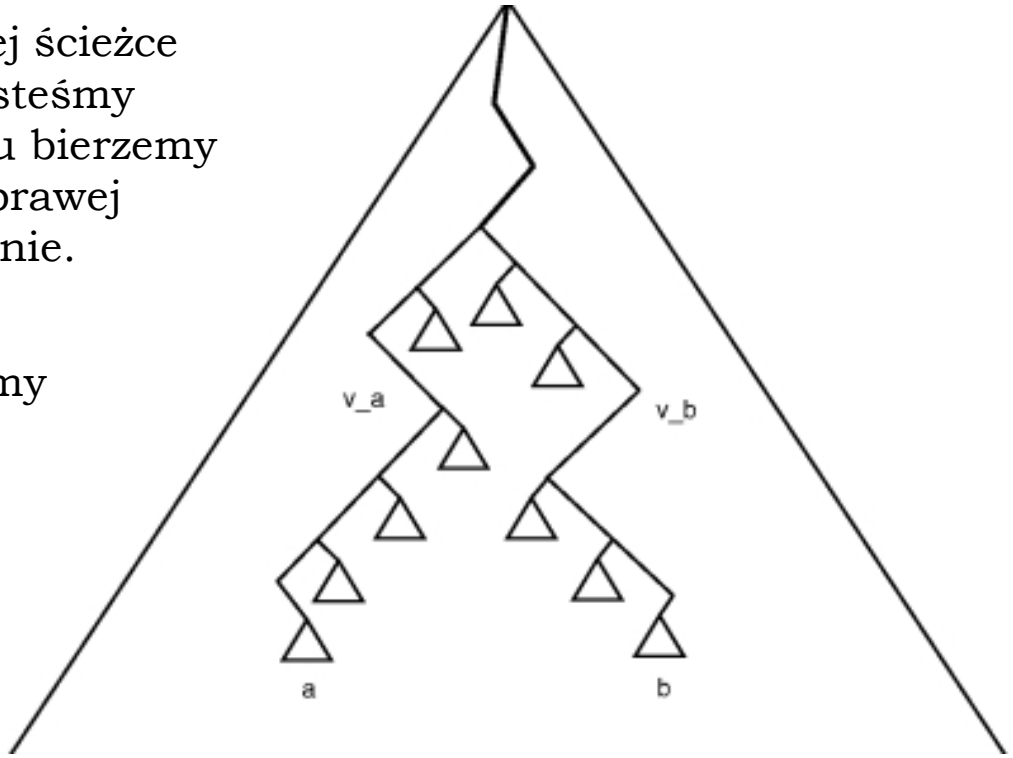
Jak realizować zapytania o przedział?

Szukamy odpowiedzi dla zapytania o przedział [a, b].

Startujemy od liści [a, a] i [b, b] i idziemy od nich w górę ku korzeniowi, aż ścieżki się nie spotkają.

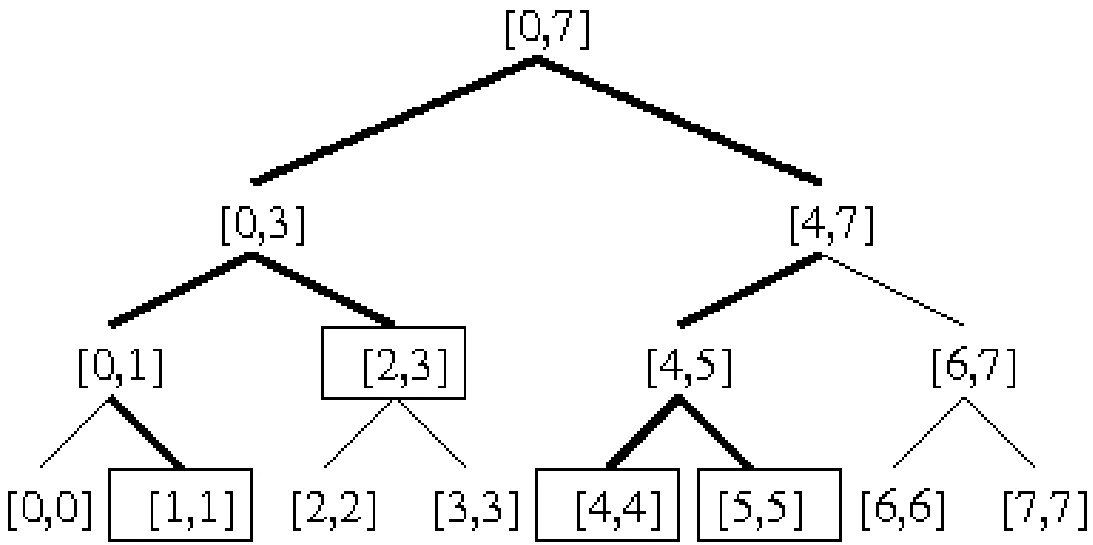
Za każdym razem, jeżeli na lewej ścieżce (dla *a*) idziemy w górę tak, że jesteśmy lewym synem ojca, to do wyniku bierzemy wartość dla prawego brata, na prawej ścieżce po stępujemy symetrycznie.

Wynik dla zapytania otrzymujemy w czasie **O(log(M).**



Przykładowe zapytanie

Pytamy o przedział [1,5]. W prostokąty ujęto przedziały, które bierzemy pod uwagę.



Implementacja zapytania

```
int query(int a, int b)
{
        int va = M + a,
        vb = M + b;
                                        /* Skrajne przedziały do rozkładu. */
        int wyn = w[va];
        if (va != vb) wyn += w[vb];
                                        /* Spacer aż do momentu spotkania. */
        while (va / 2 != vb / 2)
        {
                if (va % 2 == 0) wyn += w[va + 1];
                                        /* prawy przedział na lewej ścieżce */
                if (vb % 2 == 1) wyn += w[vb - 1];
                                        /* lewy przedział na prawej ścieżce */
                va /= 2;
                vb /= 2;
        }
        return wyn;
}
```

Dziękuję z uwagę

dr Anna Beata Kwiatkowska

aba@mat.umk.pl

tel. 602 184 813