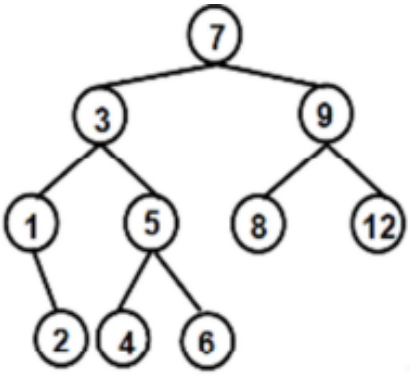
Struktury Danych i Złożoność Obliczeniowa
ĆWICZENIA 3

Drzewo BST:
Drzewo poszukiwań binarnych (Binary Search Tree). Dla każdego węzła (elementu) w tym drzewie zachodzi warunek taki, że:

- Wszystkie elementy lewego poddrzewa są nie większe (mniejsze lub równe) wartości węzła.
- Wszystkie elementy prawego poddrzewa są nie mniejsze (większe lub równe) wartości węzła.




```
struct node
{
    int  key;        klucz (wartość wierzchołka)
    node  *left;     wsk .na lewego potomka
    node  *right;    wsk. na prawego potomka
    node *parent;    wsk. na rodzica
}
```

W przeciwieństwie do kopca w drzewie BST nie ma wymogu zapełnienia wszystkich rzędów. Przez to niemożliwa jest jego tablicowa implementacja. W programie drzewo BST jest widziane tylko za pośrednictwem swojego korzenia.

Operacje na całości drzewa:
Jeżeli chodzi o wykonywanie operacji na każdym wierzchołku drzewa to mamy do dyspozycji 3 podejścia (każde z nich wykorzystujące rekurencję):

- *preorder* - najpierw wykona się operacja dla wierzchołka, w którym jesteśmy, następnie dla kolejno lewego i prawego poddrzewa.

```
void preorder(node *p)
{
   if(p==NULL)  return;
   P(p);
   preorder(p->left);
   preorder(p->right);
}
```

- *inorder* - najpierw wykona się operacja dla lewego poddrzewa, potem na aktualnego wierzchołka, potem dla prawego poddrzewa.

```
void inorder(node *p)
{
  if(p==NULL) return;
  inorder(p->left);
  P(p);
  inorder(p->right);
}
```

- *postorder* - najpierw wykonają się operacje dla lewego i prawego poddrzewa a na końcu dla aktualnego wierzchołka.

```
void postorder(node *p)
{
   if(p==NULL) return;
   postorder(p->left);
   postorder(p->right);
   P(p);
}
```

Wyszukiwanie w BST:
Wyszukiwanie w drzewie BST polega na wyborze odpowiedniego poddrzewa. Jeżeli wiemy, że szukana wartość jest mniejsza od wartości aktualnie rozważanego węzła, to musimy przejść do przeszukiwania jego lewego poddrzewa. Gdy jego większa, to należy przeszukać prawe poddrzewo. Wyszukiwanie kończymy, gdy odnajdziemy wartość lub, gdy dojdziemy do końca drzewa.

Złożoność obliczeniowa drzewa zależy od jego stanu zapełnienia. Gdy wszystkie poziomy są zapełnione, to wyszukwiwanie ma złożoność $O(\log n)$ (przypadek optymistyczny). W przeciwnym wypadku złożoność wynosi $O(n)$.

Znalezienie minimum/maksimum:
Znalezienie wartości największej lub największej w drzewie BST jest dosyć proste. Najmniejszy element to ten, wysunięty najbardziej na lewo. Największy, to ten wysunięty najbardziej na prawo.

Dodanie nowego elementu:
Dodanie elementu zawsze sprowadza się do dodania nowego liścia do naszego drzewa. Algorytm dodawania elementu najpierw szuka dla niego rodzica, a następnie wstawia go w odpowiedniej pozycji w odniesieniu do tego rodzica (lewo/prawo). W wypadku, gdy drzewo jest puste, to nowy element staje się po prostu korzeniem.

Następnik klucza:
Następnik klucza X to taki element, który jest większy od X (znajduje się w prawym poddrzewie), ale jednocześnie będący najbliższy do X. Jest to po prostu minimalny element prawego poddrzewa X.

W wypadku, gdy X nie ma prawego poddrzewa (jest maksimum swojego poddrzewa), to następnikiem będzie pierwszy węzeł-rodzić, od którego nasz węzeł jest mniejszy (w którego lewym poddrzewie się znajduje).

Oczywiście możliwe jest, że węzeł nie będzie mieć następnika. Dzieje się tak w sytuacji, gdy rozważany węzeł zawiera maksimum drzewa.

Poprzednik klucza:
Poprzednik klucza X to taki element, który jest mniejszy od X, ale jednocześnie jest najbliższy do X. Jest to po prostu maksymalny element lewego poddrzewa.

W wypadku, gdy X nie ma lewego poddrzewa (jest minimum swojego poddrzewa), to poprzednikiem będzie pierwszy węzeł-rodzic, od którego nasz węzeł jest większy (w którego prawym poddrzewie się znajduje).

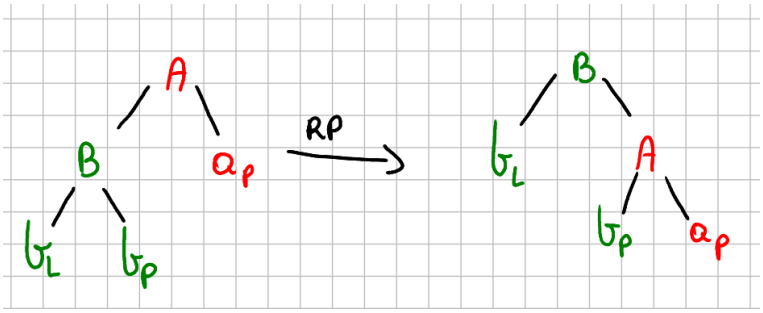
Możliwe jest, że węzeł nie będzie mieć poprzednika. Dzieje się tak w sytuacji, gdy rozważany węzeł zawiera minimum drzewa.

Usuwanie węzła:
Gdy węzeł jest liściem, to jego usunięcie jest trywialne. Po prostu odłączamy go od drzewa i usuwamy.

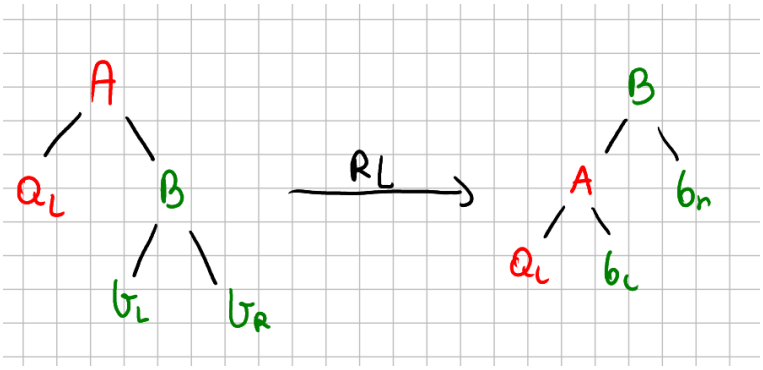
Jeżeli węzeł ma jednego potomka, to po jego usunięciu na jego miejsce wstawiamy tegoż potomka.

Jeżeli węzeł ma więcej potomków, to w miejsce jego wartości wstawiamy wartość jego następnika. Następnie usuwamy jego następnik.

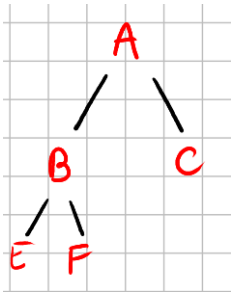
Rotacja w prawo:
Operacja zmieniająca strukturę drzewa, ale nie zmieniająca wartości kluczy. Rotacja w prawo względem węzła X da efekt:



Rotacja w lewo:
Operacja zmieniająca strukturę drzewa, ale nie zmieniająca wartości kluczy. Rotacja w lewo względem węzła X da efekt:



Drzewo zrównoważone:
Drzewo BTS, w którym wysokości lewego i prawego poddrzewa nie różnią się o więcej niż 1.

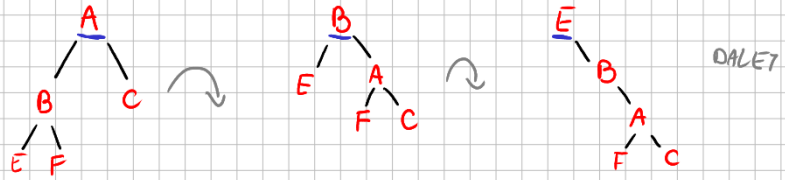


Drzewo doskonale zrównoważone:
Drzewo zrównoważone, w którym wszystkie liście znajdują się maksymalnie na dwóch poziomach.

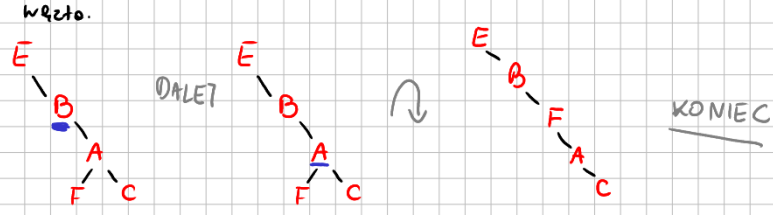
Algorytm równoważenia drzewa (DSW):
Algorytm pozwala na uzyskanie zrównoważonego drzewa BST. Jest to ważne, gdyż do takiego drzewa uzyskamy najmniejszą złożoność dla algorytmu wyszukiwania. W pierwszym etapie algorytmu wykonuje się prostowanie drzewa (sprowadzenie go do postaci liniowej) poprzez szereg rotacji w prawo. W drugim etapie wykorzystuje się szereg rotacji w lewo, w celu utrzymania drzewa zrównoważonego.

PROSTOWANIE

1) Wybieramy korzeń. Wykonujemy szereg rotacji w prawo względem niego, aż nie będzie miał lewego potomka.



2) Przechodzimy do kolejnego węzła. Powtarzamy algorytm, aż do ostatniego węzła.



RÓWNOWAŻENIE:

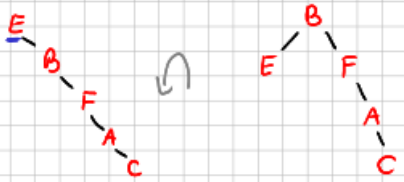
1) Liczymy liczbę m.

$$m = 2^{\lfloor \log_2(n+1) \rfloor} - 1 = 2^{\lfloor \log_2(5+1) \rfloor} = 2^2 = 4$$

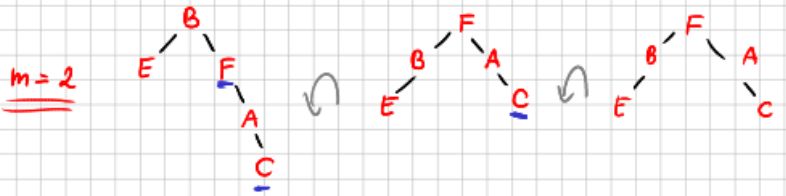
($\lfloor \log_2(n+1) \rfloor$ — ilość węzłów)

2) Wykonaj $(n-m)$ rotacji w lewo startując od korzenia co drugi węzeł.

$n - m = 5 - 4 = 1$



3) Dopóki $(m > 1)$ przypisz do m wartość $\lfloor \frac{m}{2} \rfloor$ i wykonaj m rotacji w lewo co drugi wierzchołek startując od korzenia.



$m = 0$ KONIEC