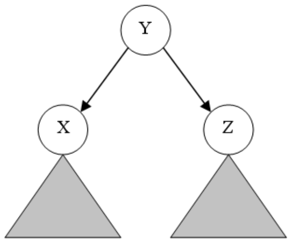
ASD - ćwiczenia VII

Drzewa przeszukiwań binarnych (BST)

- własność porządku drzew BST



  gdzie dla każdej trójki wierzchołków drzewa BST (x, Y, z) porządek etykiet $elem$ jest następujący $x.elem \leq Y.elem \leq z.elem$, gdzie wierzchołki x, z są kolejno elementem poddrzewa o korzeniu X oraz elementem poddrzewa o korzeniu Z,

- definicja wskaźnikowa struktury typu węzeł drzewa BST w pseudokodzie

```
typedef struct TreeNode Tree;

struct TreeNode {
  element elem;
  struct TreeNode left, right;
};
```

- podstawowe operacje dla drzewa binarnego typu BST:

  - $EMPTY\ :\ \mathcal{T} \rightarrow \{TURE, FALSE\}$, sprawdzenie czy struktura jest pusta,
  - $INSERT\ :\ \mathcal{T} \times E \rightarrow \mathcal{T}$, wstawienie elementu do struktury,
  - $DELETE\ :\ \mathcal{T} \times E \rightarrow \mathcal{T}$, usunięcie elementu ze struktury,
  - $MEMBER\ :\ \mathcal{T} \times E \rightarrow \{TURE, FALSE\}$, sprawdzenie, czy dany element jest przechowywany w strukturze,

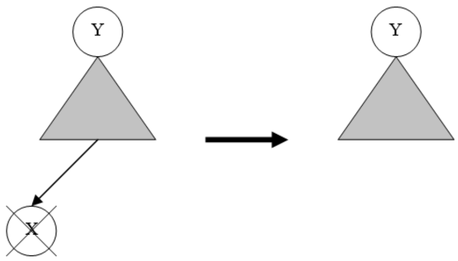
  gdzie $\mathcal{T}$ jest przestrzenią drzew typu BST, E zbiorem etykiet wierzchołków drzewa typu BST,

- złożoność czasowa podstawowych operacji n-elementowego drzewa typu BST:

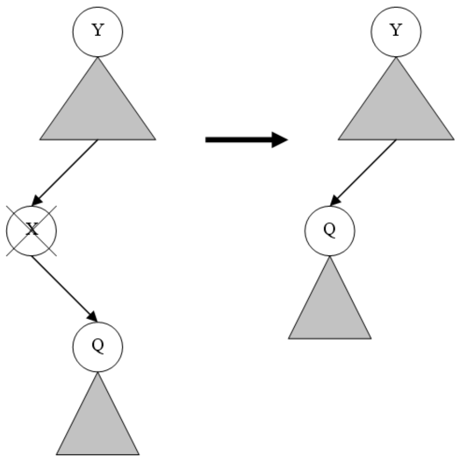
  - $A\left(\texttt{EMPTY}(), n\right) = O\left(1\right)$, $W\left(\texttt{EMPTY}(), n\right) = O\left(1\right)$,
  - $A\left(\texttt{INSERT}(), n\right) = O\left(\log\left(n\right)\right)$, $W\left(\texttt{INSERT}(), n\right) = O\left(n\right)$,
  - $A\left(\texttt{DELETE}(), n\right) = O\left(\log\left(n\right)\right)$, $W\left(\texttt{DELETE}(), n\right) = O\left(n\right)$,
  - $A\left(\texttt{MEMBER}(), n\right) = O\left(\log\left(n\right)\right)$, $W\left(\texttt{MEMBER}(), n\right) = O\left(n\right)$,

- usuwanie z dokładnością do przypadków symetrycznych wierzchołka X z drzewa typu BST – operacja $DELETE$:

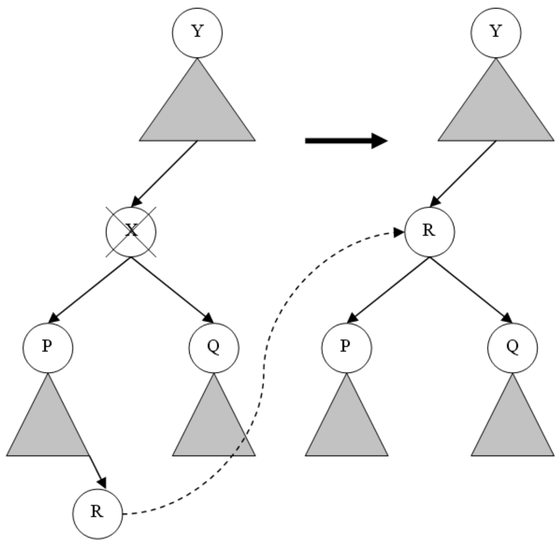
◦ wierzchołek X jest liściem:



◦ wierzchołek X ma dokładnie jednego syna:



◦ wierzchołek X ma dokładnie dwóch synów:



gdzie wierzchołek R jest poprzednikiem wierzchołka X w rozważanym drzewie typu BST.

Zadania

1. Pewne drzewo typu BST przechowuje dowolne liczby z przedziału od 1 do 2000. Sprawdzamy czy liczba 1333 należy do tego drzewa. W tym celu wywołaliśmy funkcję `MEMBER()`, której rezultatami są poniższe ścieżki wędrówki przez rozważaną strukturę danych. Które z nich są poprawne w sensie własności porządku etykiet drzew typu BST:

   (a) 1936, 278, 1347, 621, 1299, 1937, 358, 1333,

   (b) 12, 1252, 1401, 1398, 1330, 1344, 1397, 1333,

   (c) 1924, 1220, 1911, 1244, 1898, 1258, 1362, 1333,

   (d) 1925, 202, 1911, 250, 1612, 245, 1333.

2. Niech T będzie pewnym drzewem binarnym, zaproponuj funkcję

   `int TREE_DEPTH(Tree T)`,

   która wyznaczy jego głębokość używając:

   (a) techniki rekurencyjnej,

   (b) techniki iteracyjnej wraz z dowolną liniową strukturą danych.

3. Niech T będzie drzewem binarnym, zaproponuj funkcję

   `int VERTEX_DEPTH(Tree T, int level)`,

   która wyznaczy liczbę wierzchołków znajdujących się na poziomie *level* w rozpatrywanym drzewie używając

   (a) techniki rekurencyjnej,

   (b) techniki iteracyjnej wraz z dowolną liniową strukturą danych.

4. Mamy daną tablicę $A[1], A[2], \ldots, A[n]$ składającą się z $n = 2^m - 1$ liczb naturalnych posortowanych w porządku niemalejącym, gdzie $m \in \mathbb{N}$. Elementy tej tablicy są wybierane i umieszczane zgodnie z pewnym algorytmem $\mathcal{A}$ w początkowo pustym drzewie typu BST. Zakładamy, że wynikiem działania tego algorytmu jest pełne drzewo typu BST a jego złożoność jest ograniczona przez $O(n)$. Zaproponuj funkcję

   `Tree BUILD_TREE(int A[], int n)`,

   realizującą algorytm $\mathcal{A}$.

5. Niech F będzie rodziną drzew binarnych lokalnie pełnych[1], A alfabetem etykiet wierzchołków drzew z rodziny F oraz $pre(T)$ słowem otrzymanym przez przejście wierzchołków drzewa $T \in F$ w kolejności preorder. Zaproponuj:

   (a) rekurencyjną funkcję

   `Tree REFLECT(Tree T)`,

[1] Drzewo binarne nazywamy lokalnie pełnym, jeżeli rząd każdego jego wierzchołka jest równy 0 albo 2.

która dla danego drzewa T utworzy drzewo T' będące jego „lustrzanym odbiciem“,

(b) iteracyjną funkcję

```
char [] GET_WORD(Tree T),
```

która dla danego drzewa T odczyta słowo $pre(T)$,

(c) funkcję

```
bool TEST(Tree T),
```

która dla danego drzewa T, stwierdzi czy słowo $pre(T)$ jest palindromem,

(d) funkcję

```
bool IS_SUBWORD(Tree T1, Tree T2),
```

która dla danych drzew $T1$ oraz $T2$, stwierdzi czy słowo $pre(T1)$ jest podsłowem słowa $pre(T2)$,

(e) funkcję

```
char [] INTERSECTION(Tree T1, Tree T2),
```

która dla danych drzew $T1$ oraz $T2$, wyznaczy część wspólną słów $pre(T1)$, $pre(T2)$.

Dla każdej procedury oszacuj złożoność pamięciową i czasową podanego rozwiązania. Dodatkowo przyjmij, że określono funkcję

```
int SIZE(Tree T),
```

która dla dowolnego drzewa $T \in F$ zwraca liczbę jego wierzchołków.

Zadanie o przypuszczalnie skończonych ciągach (do domu)

Niech k będzie pewną liczbą naturalną, wtedy i-ty element, $A_k[i]$ **ciągu gradowego** A_k, dla $i = 1, 2, \ldots$, definiujemy w następujący sposób:

$$A_k[i] = \begin{cases} k & \text{dla} \quad i = 1, \\ \frac{1}{2} \cdot A_k[i-1] & \text{dla} \quad i > 1 \ \wedge \ A_k[i-1] \bmod 2 = 0, \\ 3 \cdot A_k[i-1] + 1 & \text{dla} \quad i > 1 \ \wedge \ A_k[i-1] \bmod 2 = 1. \end{cases}$$

Dalej zakładamy, że dla dowolnego $k \in \mathbb{N}$ ciąg A_k jest okresowy, tj. $\exists x \in \mathbb{N} \setminus \{0\} : A_k[x+i] = A_k[x+p+i]$, dla pewnego $p \in \mathbb{N} \setminus \{0\}$. **Właściwym ciągiem gradowym** B_k nazywamy skończony ciąg elementów ciągu gradowego A_k, będący możliwie najdłuższym ciągiem nieokresowym.

Zaprojektuj możliwie efektywną funkcję

```
int TWINS(int k1, int k2),
```

która dla zadanych argumentów wejściowych $k1$ oraz $k2$ wyznaczy liczbę par $(p1, p2)$, gdzie $p1$ oraz $p2$ są odpowiednio elementami ciągu B_{k1} i B_{k2}, których elementy są liczbami bliźniaczymi. Przypominamy, że liczby $p1$ oraz $p2$ są bliźniacze wttw., gdy $p1$, $p2$ są liczbami pierwszymi oraz $|p1 - p2| = 2$. Zakładamy, że zdefiniowano funkcję

```
bool IS_PRIME(int p),
```

która dla zadanej liczby p zwraca $TRUE$ jeżeli p jest liczbą pierwszą, $FALSE$ w przeciwnym przypadku. W uproszczeniu przyjmij, że złożoność czasowa i pamięciowa funkcji `IS_PRIME()` jest stała.