# Algorytmy i struktury danych

## Lista zadań 6

### Zadanie 1

Jakie informacje przechowujemy w węźle drzewa czerwono-czarnego? Zadeklaruj strukturę `RBTnode` tak, by dziedziczyła z `BSTnode`. Podaj definicję drzewa czerwono czarnego.

```
struct NodeBST
{
    int32_t value;
    NodeBST* left;
    NodeBST* right;
};

struct NodeRBT : public NodeBST
{
    bool isBlack;
};
```

Drzewo czerwono-czarne musi przestrzegać następujące wymagania:

1. Każdy węzeł jest albo czerwony albo czarny.
2. Korzeń jest czarny.
3. Każdy liść (również nullptr) jest czarny.
4. Czerwony węzeł ma czarne dzieci.
5. Każda ścieżka od korzenia do liścia ma tę samą liczbę czarnych węzłów.

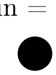
### Zadanie 2

(a) Jaka może być minimalna, a jaka maksymalna ilość kluczy w drzewie czerwono-czarnym o ustalonej czarnej wysokości równej $h_B$?

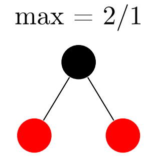
$$\min = 2^{h_B - 1} - 1$$
$$\max = 2^{2(h_B - 1)} - 1$$

(b) Znajdź maksymalną i minimalną wartość stosunku ilości węzłów czerwonych do czarnych w drzewie czerwono-czarnym.

$\min = 0/1$



$\max = 2/1$

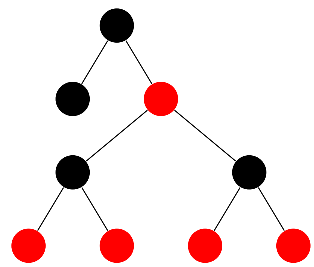


## Zadanie 3

Uzasadnij posługując się rysunkiem i opisem, że operacje wykonywane w trakcie wstawiania do drzewa czerwono-czarnego (rotacja i przekolorowanie) nie zmieniają ilości czarnych węzłów, na żadnej ścieżce od korzenia do liścia.

## Zadanie 4

(a) Narysuj poprawne drzewo czerwono-czarne w którym na lewo od korzenia jest 1 węzeł a na prawo 7 węzłów.



(b) Czy istnieje poprawne drzewo czerwono-czarne, w którym na lewo od korzenia będzie 100 razy mniej węzłów niż na prawo od korzenia?

$$100\left(2^{h_B-1}-1\right) < 2^{2(h_B-1)}-1$$

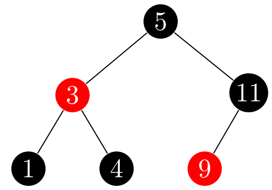
| $h_B - 1$ | **min** | 100 **min** | **max** |
|---|---|---|---|
| 1 | 0 | 0 | 0 |
| 2 | 1 | 100 | 3 |
| 3 | 3 | 300 | 15 |
| 4 | 7 | 700 | 63 |
| 5 | 15 | 1500 | 255 |
| 6 | 31 | 3100 | 1023 |
| 7 | 63 | 6300 | 4095 |
| 8 | 127 | 12700 | 16383 |

Istnieje takie drzewo czerwono-czarne, w którym na lewo od korzenia jest 100 razy mniej węzłów niż na prawo. Dla czarnej wysokości $h_B = 9$ po lewej stronie minimalnie może być 127 węzłów, a po prawej 16383.

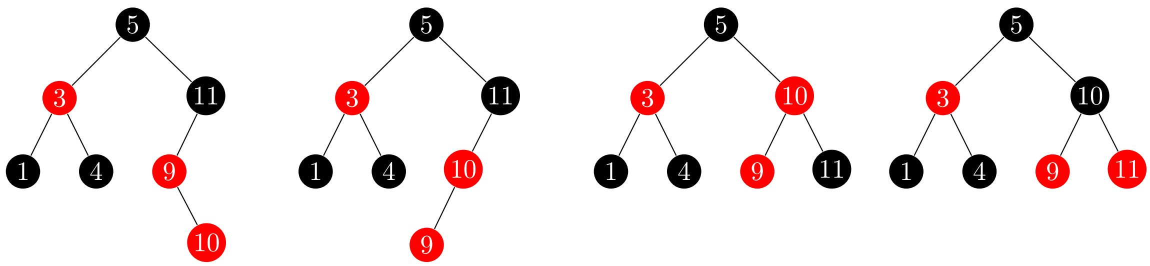
$$100\left(2^{8-1}-1\right) < 2^{2(8-1)}-1$$
$$100\left(2^{7}-1\right) < 2^{2(7)}-1$$
$$100\,(127) < 2^{14}-1$$
$$12700 < 16383$$

## Zadanie 5

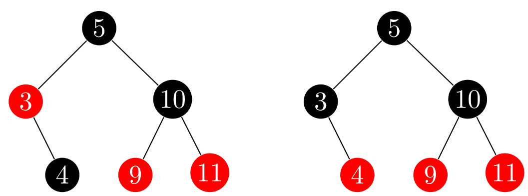
W poniższym drzewie czerwono-czarnym:




- wstaw do niego `10`.



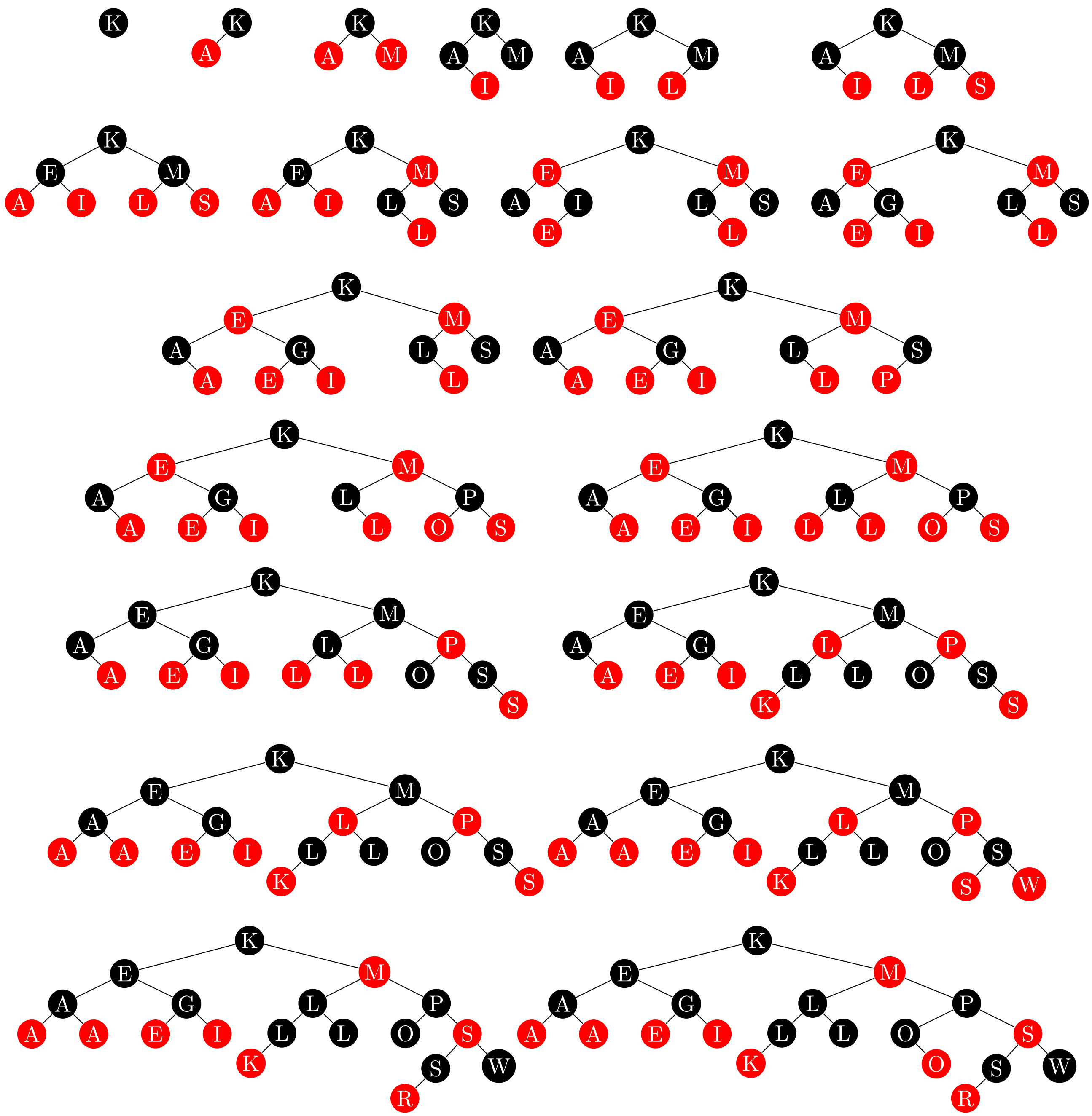

- usuń z wyjściowego drzewa `1`.



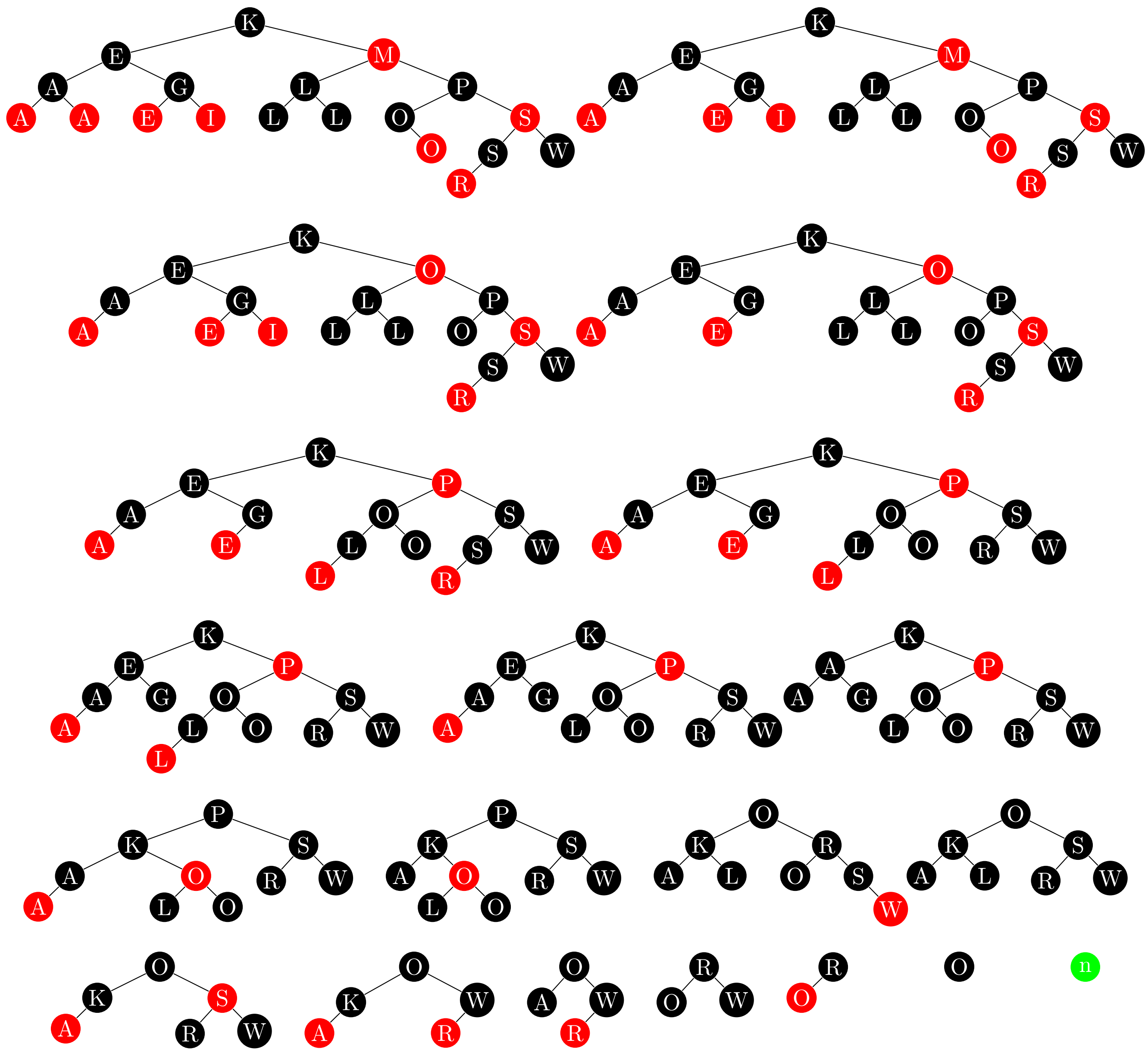
## Zadanie 6

(3 pkt.) Do pustego drzewa czerwono-czarnego wstaw kolejno 20 przypadkowych kluczy. Następnie usuń je w tej samej kolejności w jakiej wstawiałeś. Przypadkowymi kluczami są kolejne litery Twojego nazwiska, imienia i adresu. Zadanie wykonujemy na kartce (lub w pliku) i oddajemy prowadzącemu. Zadanie jest obowiązkowe.

### Wstawianie



## Usuwanie



## Zadanie 7

Analizując kod programu RBT.h udowodnij, że w trakcie wstawiania do drzewa czerwono-czarnego wykonają się co najwyżej dwie rotacje. Czy tak samo jest w przypadku usuwania?

Bezpośrednio w samej procedurze `insert` nie ma żadnych rotacji. Rotacje mogą wystąpić dopiero w procedurze `fix_up`, która jest wywoływana jest co najwyżej raz, zaraz po wstawieniu nowego węzła.

```
...
for (;;)
    if (x < t->key)
    {
        if (t->left)
            t = t->left;
        else
        {
            t->left = new node(x, t);
            fix_up(t->left);
            break;
        }
    }
    else
    {
        if (t->right)
            t = t->right;
        else
        {
            t->right = new node(x, t);
            fix_up(t->right);
            break;
        }
    }
...
```

Wiedząc że procedura `fix_up` wykonuje się co najwyżej jednokrotnie, możemy zatem stwierdzić, że maksymalna liczba rotacji wynosi dwie, co uwidaczniają poniższe fragmenty kodu z procedury `fix_up`.

```
...
else
{
    if (t == p->right)
        rotate_left(p);
    rotate_right(pp);
    return;
}
...
else
{

    if (t == p->left)
        rotate_right(p);
    rotate_left(pp);
    return;
}
...
```

W przypadku usuwania, istnieje możliwość wystąpienia więcej niż dwóch rotacji, dlatego że procedura `rem_fix_up` wywołana najwyżej raz zawiera pętlę `while` której ilość iteracji zależna jest od wysokości drzewa, a co za tym idzie dla większych wysokości drzewa, ilość możliwych rotacji będzie wzrastać.

## Zadanie 8

Uzasadnij, że rozmiar stosu ($n = 100$) przyjęty w procedurach `insert` i `remove` w pliku `RBnpnr.h` nigdy nie okaże się za mały.

Wiedząc że na stos przyjęty w procedurach procedurach `insert` i `remove` trafiają kolejne odwiedzone węzły drzewa w trakcie szukania miejsca do wstawienia nowego węzła w przypadku procedury `insert` oraz dodatkowo w trakcie szukania następnika w przypadku procedury `remove` wiadomo że nie trafi tam więcej węzłów niż wynosi wysokość samego drzewa. Przyglądając się ilości węzłów w drzewie o takiej wysokości

$$n_{min} = 2^{h-1} = n_{max} = 2^{99} - 1 = 633825300114114700748351602687$$
$$n_{max} = 2^{h} - 1 = n_{max} = 2^{100} - 1 = 1267650600228229401496703205375$$

możemy zauważyć, że stos nigdy nie będzie za mały, ponieważ gdyby nawet przyjąć że każdy węzeł trzyma jeden bit informacji, to łączna pamięć tego drzewa wynosząca $7.923 \cdot 10^{28}$ bajtów, lub inaczej 79.23 ronnabajtów jest praktycznie nieosiągalna dla współczesnych komputerów. Dla porównania taka pamięć byłaby w stanie pomieścić $8.7 \cdot 10^{11}$ razy szacowany rozmiar deep webu, który wynosi około 91000 terabajtów, czy też $4.7 \cdot 10^{14}$ raza szacowany rozmiar widzialnej sieci internetowej, który wynosi około 170 terabajtów.