Wstęp do programowania

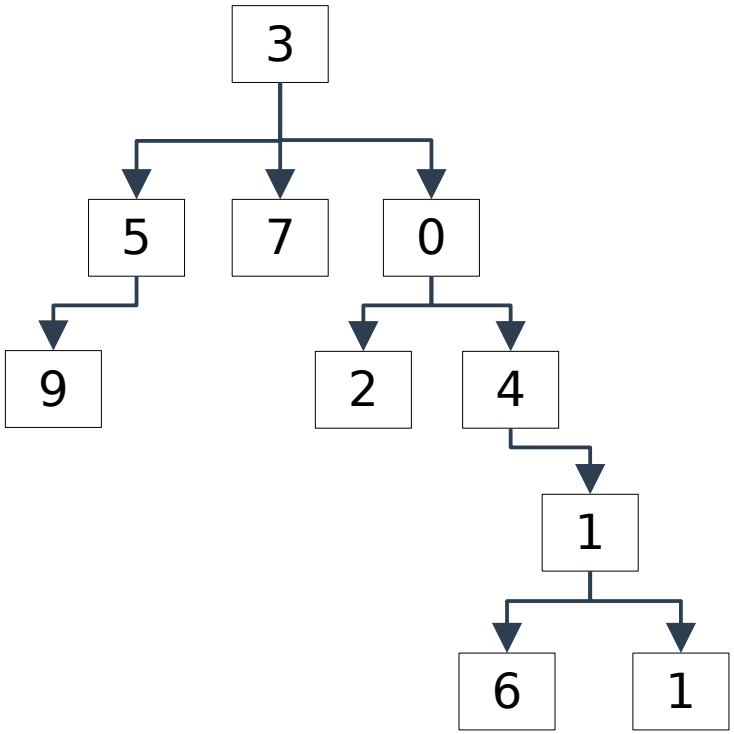
Wykład 6. Drzewa

Plan wykładu

1. Drzewa.
2. Drzewa wyszukiwania binarnego (BST).
- wyszukiwanie,
- wstawianie,
- usuwanie.
3. Drzewa czerwono-czarne.
- rotacje pojedyncze i podwójne,
- wstawianie.
4. Porównanie struktur danych.

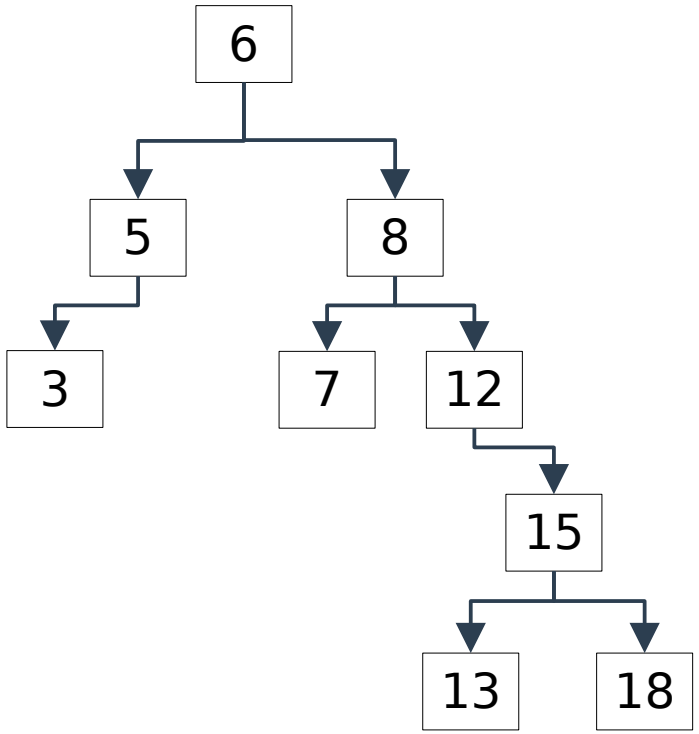
Drzewa

Drzewa pozwalają odwzorować hierarchę elementów. Zaczyna się od korzenia (root). Elementy drzewa mogą posiadać „dzieci”. Elementy bez dzieci to tzw. liście (leafs).



Drzewa wyszukiwania binarnego

W drzewach wyszukiwania binarnego (BST – binary search tree) każdy element posiada co najwyżej dwójkę dzieci (lewe i prawe). Ponadto elementy myszą spełniać warunek porządku lewe ≤ rodzic ≤ prawe.




```
struct BSTNode{
  double value;
  struct BSTNode *left;
  struct BSTNode *right;
};
```

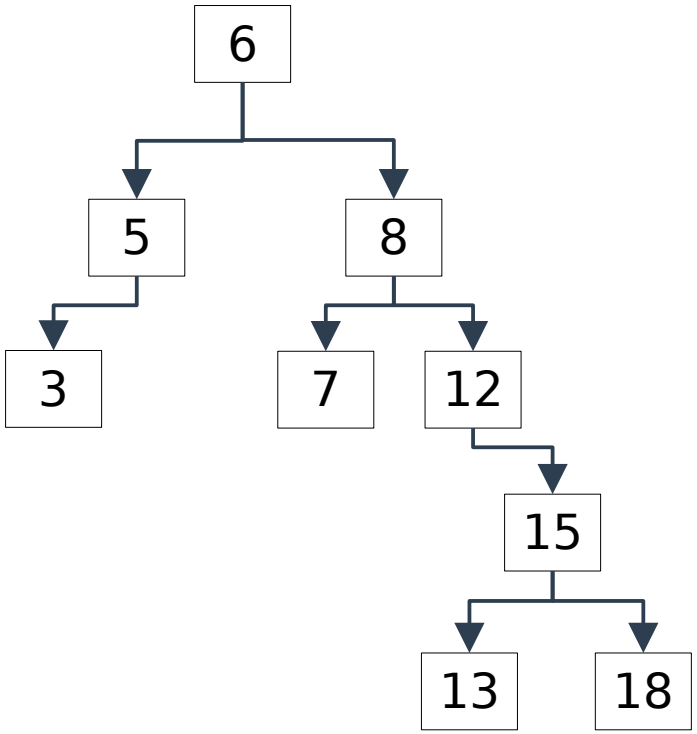
Drzewa wyszukiwania binarnego: szukanie

```
BSTNode *search(BSTNode *root, double v){
  BSTNode *p = root;
  while(p!=NULL && p->value!=v){
    if (p->value < v)
      p = p->right; // szukamy w prawym poddrzewie
    else
      p = p->left; // szukamy w lewym poddrzewie
  };
  return p;
};
```

Wyszukiwanie elementu w drzewie BST działa typowo w czasie O(log n) bo jedno porównanie zmniejsza obszar wyszukiwania o połowę.

Drzewa wyszukiwania binarnego: wypisywanie elementów

```
void print(BSTNode *p){
  if (p!=NULL){
    1
    print(p->left);
    2
    print(p->right);
    3
  }
}
```



W miejsce 1, 2, albo 3 wstawiamy `printf("%d", p->value);` aby uzyskać wypisywanie w porządku preorder, inorder lub postorder.

Który wariant wypisze elementy w kolejności od najmniejszego do największego?

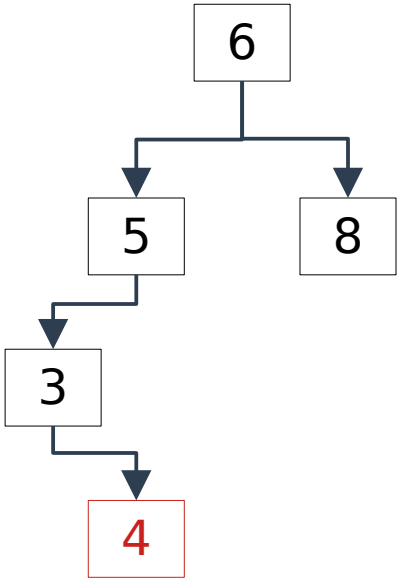
Drzewa wyszukiwania binarnego: wstawianie

```
BSTNode *newNode(double v){
  BSTNode *node = (BSTNode *)malloc(sizeof(BSTNode));
  node->value = v;
  node->left = NULL;
  node->right = NULL;
  return node;
}

BSTNode *add(BSTNode *node, double v){
  if(node==NULL)
    return newNode(v); // alokujemy pamiec
  if (node->value < v) // idziemy na prawo
    node->right = add(node->right, v);
  else // idziemy w lewo
    node->left = add(node->left, v);
  return node; // zwracamy niezmieniny wskaznik
};
```

Drzewa wyszukiwania binarnego: wstawianie

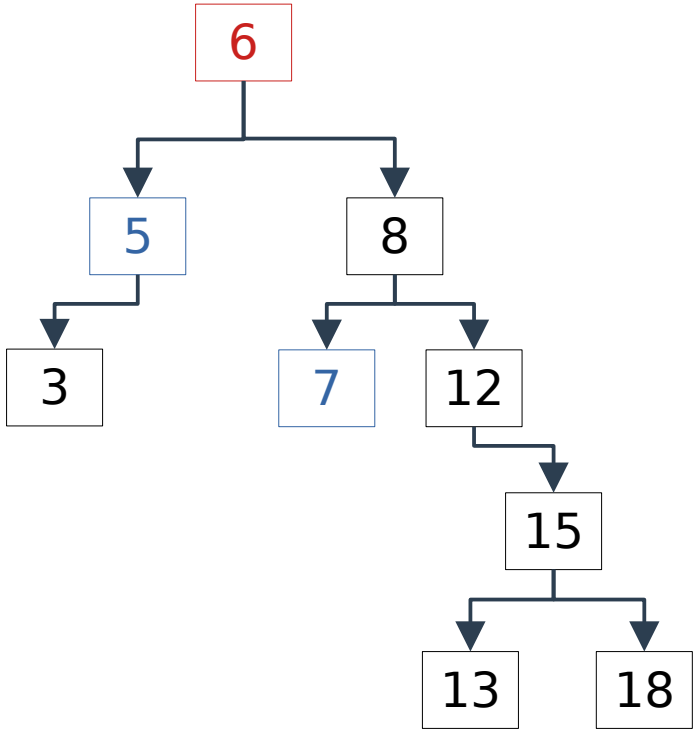
Jak to działa? Przykład wstawiamy do drzewa liczbę 4




```
add(node6, 4):
  add(node5, 4):
    add(node3, 4):
      add(NULL, 4):
        tworzymy nowy element newNode(4);
        zwracamy wskaznik do niego p;
        koniec add(NULL,4)
      node3->right = p;
      zwracamy node3;
      koniec add(node3, 4)
    node5->left = node3;  //nic nie zmienia
    zwracamy node5;
    koniec add(node5, 4)
  node6->left = node5;  //nic nie zmienia
  koniec add(node6, 4)
```

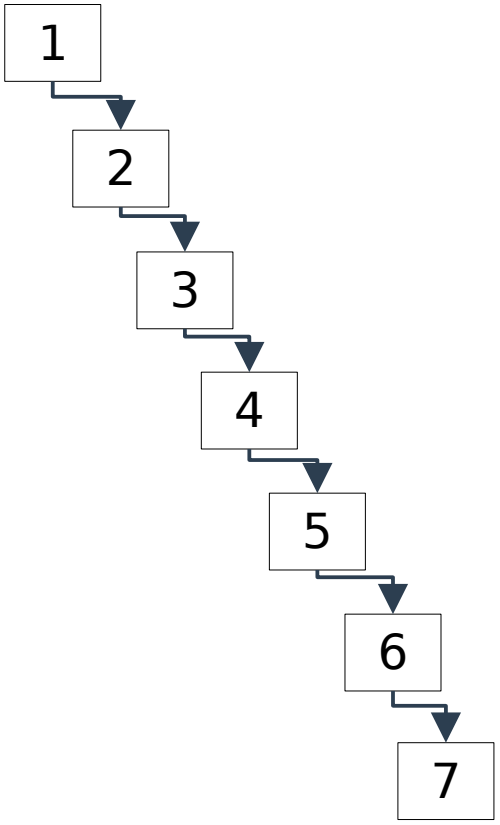
Drzewa wyszukiwania binarnego: usuwanie

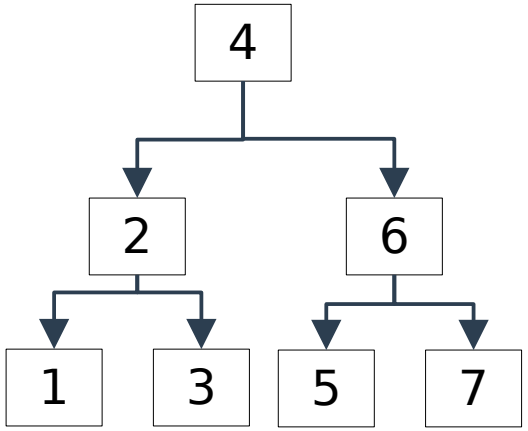
Usuwany element zastępujemy najmniejszym elementem w jego prawym poddrzewie albo największym w jego lewym poddrzewie. Przykład: 6 możemy zastąpić albo 5 albo 7. Jeśli wybierzemy element nie będący liściem (5) to musimy go zastąpić analogicznie jak 6.



Drzewa wyszukiwania binarnego: złożoność obliczeniowa

Złożoność obliczeniowa operacji wyszukiwania, wstawiania i usuwania na drzewie wyszukiwania binarnego zależy od jego budowy i może wahać się od O(log n) do O(n), w zależności od tego, czy drzewo jest zrównoważone, czy nie.

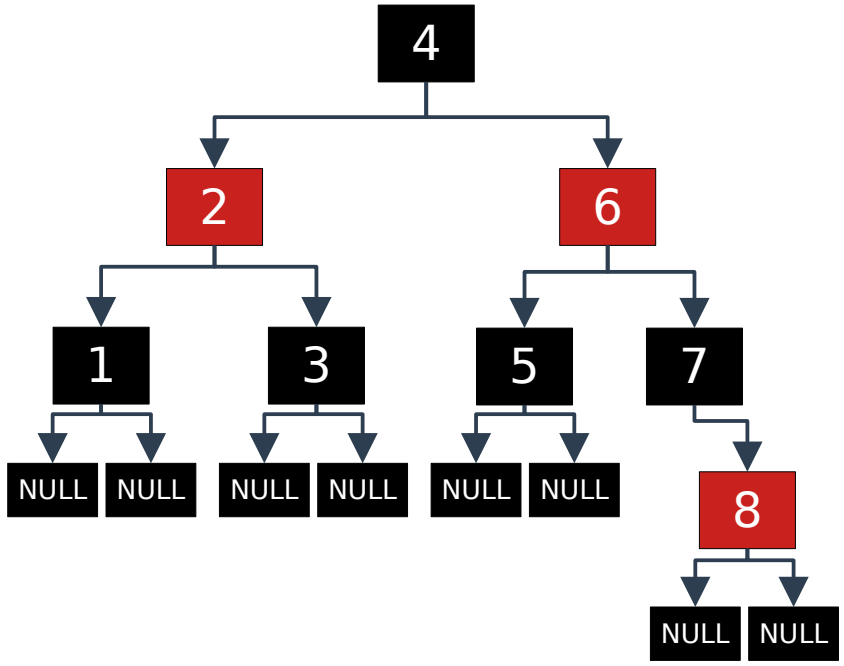




Drzewa czerwono-czarne

Drzewa czerwono czarne to rodzaj samoorganizujących się drzew wyszukiwania binarnego.

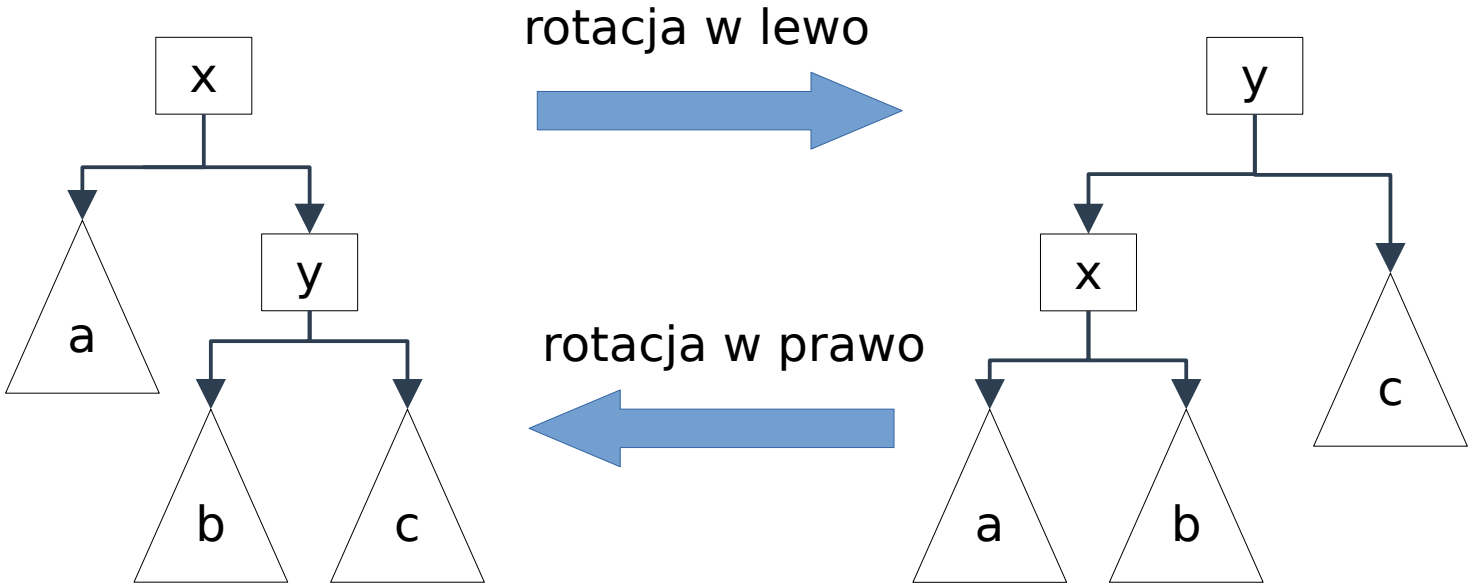
- każdy węzeł jest czerwony lub czarny,
- korzeń jest czarny,
- liście (NULL) są czarne,
- jeśli węzeł jest czerwony to jego dzieci muszą być czarne,
- każda ścieżka z ustalonego węzła do każdego z liści liczy tyle samo węzłów czarnych.



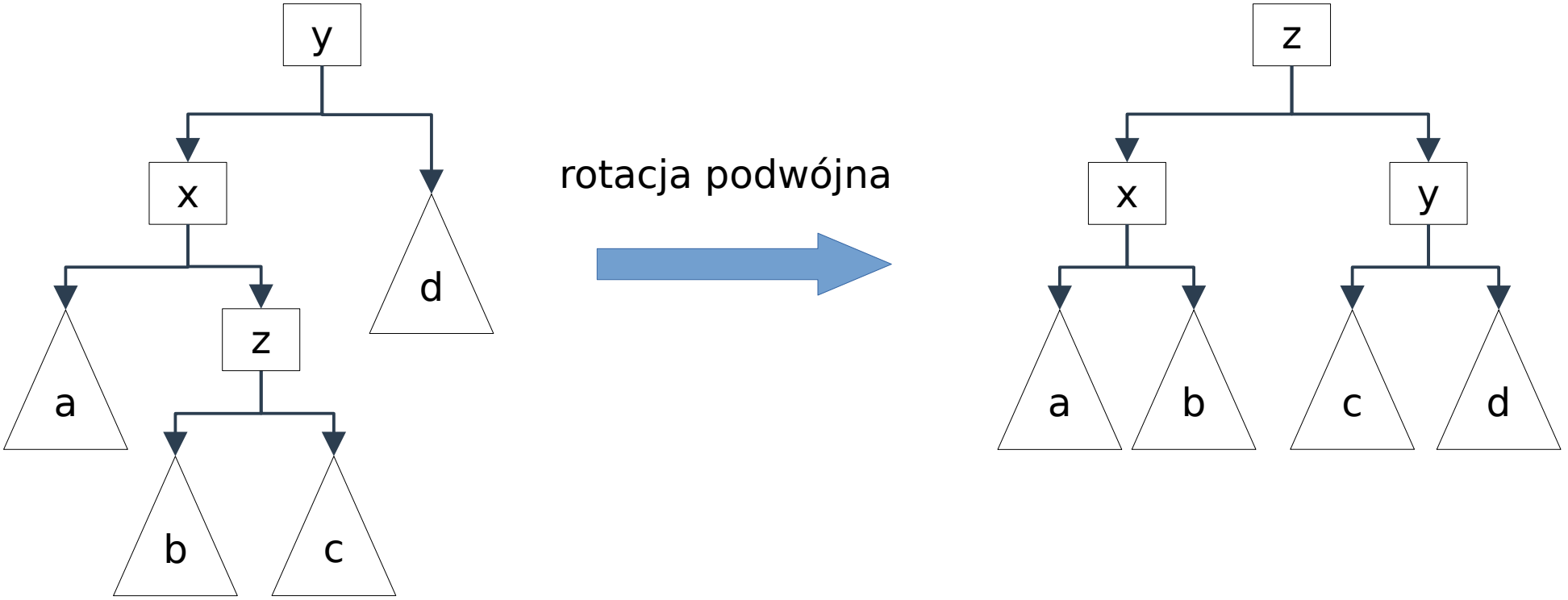
Wymagania te gwarantują, że najdłuższa ścieżka od korzenia do liścia będzie co najwyżej dwukrotnie dłuższa niż najkrótsza.

Drzewa binarne: rotacja pojedyncza

Rotacje pozwalają na reorganizację drzew wyszukiwania binarnego bez zaburzenia ich struktury.



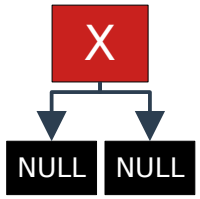
Drzewa binarne: rotacja podwójna



Analogiczna rotacja jest możliwa w przypadku symetrycznym. Rotacje (pojedyncze i podwójne) są wykonywane w czasie stałym O(1) – niezależnym od rozmiaru drzewa.

Drzewa czerwono-czarne: wstawianie

1. Umieść nowy element w drzewie tak samo jak w drzewach wyszukiwania binarnego.



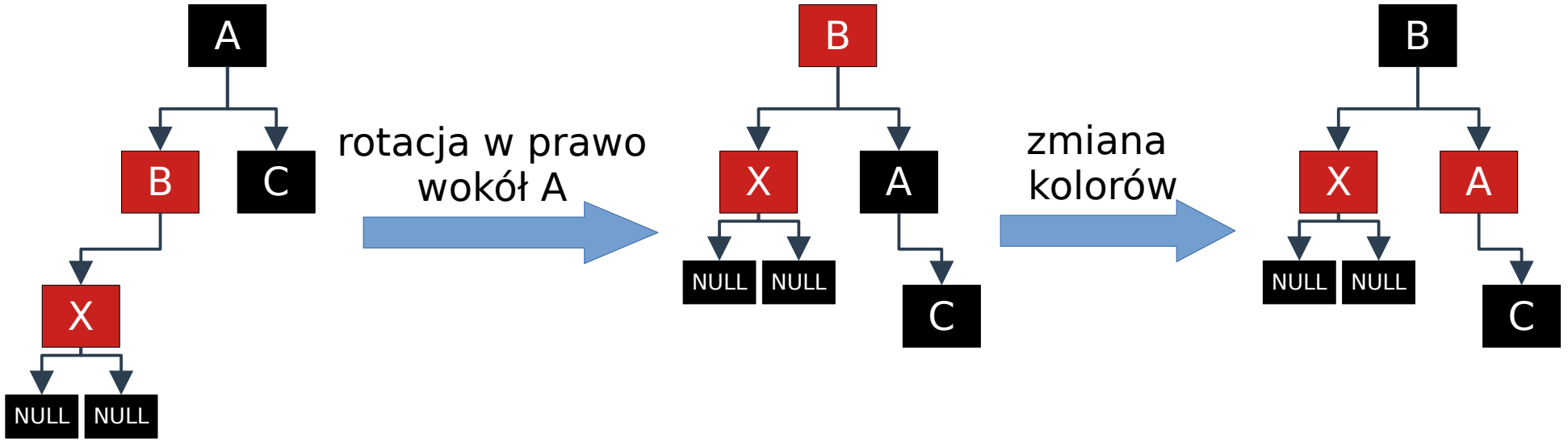
2. Pokoloruj go na czerwono

3. Przywróć własności drzewa czerwono-czarnego.

Przypadek 1

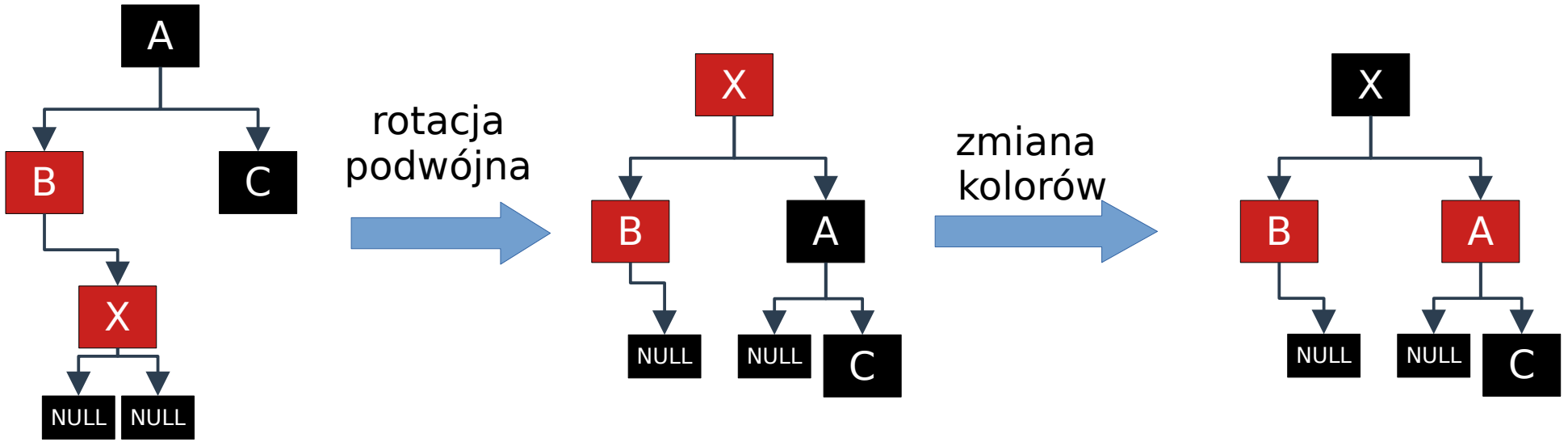
Drzewa czerwono-czarne: wstawianie

Przypadek 2



Drzewa czerwono-czarne: wstawianie

Przypadek 3



Usuwanie elementów z drzew czerwono-czarnych jest analogiczne jak w drzewach wyszukiwania binarnego, ale w związku z kolorowaniem trzeba rozpatrzyć więcej przypadków. Ogólnie strukturę drzew przywracamy korzystając z rotacji. Dodatkowe informacje: https://eduinf.waw.pl/inf/alg/001_search/0121.php

Drzewa binarne

Istnieją także inne drzewa binarne zapewniające koszt operacji wstawiania wyszukiwania i usuwania elementów w czasie O(log n). Najpopularniejsze z nich to drzewa AVL oraz drzewa Splay (są opisane np. w Wikipedii).

	Tablica	Tablica posortowana	Lista	Drzewo BST	Drzewo czerwono-czarne
wstawianie	O(1)	O(n)	O(1)	O(log n) - O(n)	O(log n)
wyszukiwanie	O(n)	O(log n)	O(n)	O(log n) - O(n)	O(log n)
usuwanie	O(1)	O(n)	O(1)	O(log n) - O(n)	O(log n)

Sortowanie

Dziękuję za uwagę