Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wykład 8.

Kolejki priorytetowe

Kolejki priorytetowe

Kolejka priorytetowa - to najbardziej ogólny rodzaj kolejki.

Elementy takiej kolejki pobierane są w kolejności:

1. Wyznaczonej wartością priorytetu elementu w kolejce (jest to dodatkowa cecha elementów wprowadzająca kolejność ich pobierania z kolejki); zakładamy, że pobieramy element o najwyższym priorytecie.

2. Przy równych priorytetach pobierany jest element wcześniej umieszczony w kolejce.

Zwykła kolejka jest szczególnym przypadkiem kolejki priorytetowej, w której wszystkie elementy mają priorytet określony przez czas ich pobytu w kolejce.

Implementacja kolejki jest uzależniona od charakteru zbioru wartości priorytetów.

Rodzaje kolejek priorytetowych:

Kolejki z nieograniczonym zbiorem wartości priorytetów:

- nieuporządkowane,
- uporządkowane,
- o organizacji stogowej.

Kolejki z ograniczonym (niewielkim) zbiorem wartości priorytetów.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

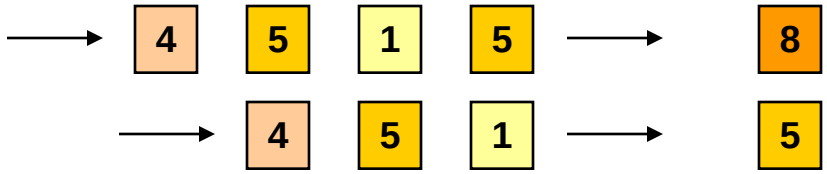
Kolejki priorytetowe z nieograniczonym zbiorem wartości priorytetów

Kolejki nieuporządkowane

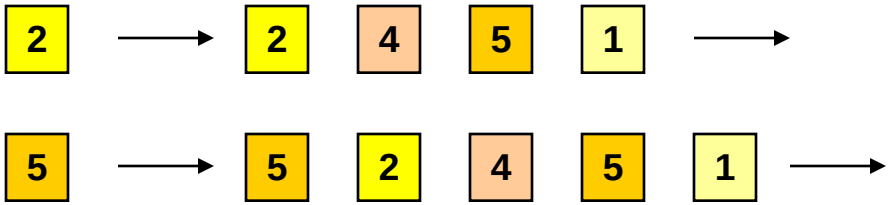
W takich kolejkach:
- wstawianie jest dokonywane na koniec kolejki (złożoność O(1)),
- pobranie i usunięcie elementu o najwyższym priorytecie (pierwszego z równych); wymaga wyszukania tego elementu (złożoność O(N)).

Przykład:
- stan kolejki w danej chwili:

- stan kolejki po kolejnych pobraniach i usunięciach elementu:



- stan kolejki po kolejnych dopisaniach:



Przykładowa implementacja kolejki priorytetowej oparta na liście nieuporządkowanej (na podstawie: [Harris, Ross])

```
package queues;
import lists.LinkedList;
import lists.List;
import sorting.Comparator;

public class UnsortedListPriorityQueue implements Queue {
   private final List _list;
   private final Comparator _comparator;  // do "obsługi" priorytetu

   public UnsortedListPriorityQueue(Comparator comparator) { // konstruktor
       _comparator = comparator;
       _list = new LinkedList();
   }

   public void enqueue(Object value) {
      _list.add(value);
   }

   public Object dequeue() throws EmptyQueueException {
      if (isEmpty()) throw new EmptyQueueException();
      return _list.delete(getIndexOfLargestElement());
   } // c.d.n.
```

Przykładowa implementacja kolejki nieuporządkowanej oparta na liście (na podstawie: [Harris, Ross])

```
// c.d.:

  private int getIndexOfLargestElement() { // pobranie indeksu elementu o najwyższym
                                           // priorytecie
      int result = 0;
      for (int i = 1; i < _list.size(); ++i)
        if (_comparator.compare(_list.get(i), _list.get(result)) > 0)
          result = i;
      return result;
  }

  public void clear() {
      _list.clear();
  }

  public int size() {
      return _list.size();
  }

  public boolean isEmpty() {
      return _list.isEmpty();
  }
}
```

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

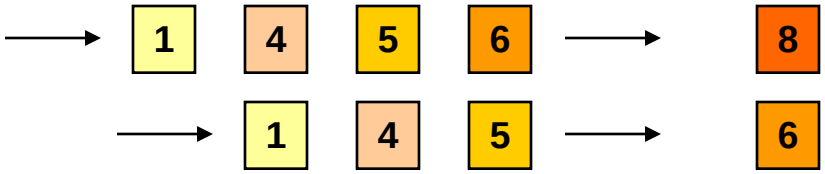
Kolejki priorytetowe z nieograniczonym zbiorem wartości priorytetów

Kolejki uporządkowane

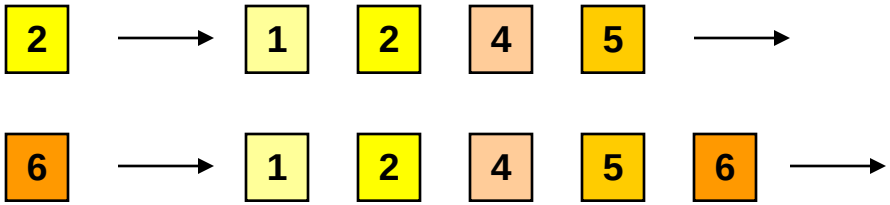
Wykorzystują uporządkowaną tablicę lub listę; wówczas:
- wstawianie jest dokonywane na właściwe miejsce (złożoność O(N)),
- pobranie i usunięcie elementu (o najwyższym priorytecie) z czoła kolejki (złożoność O(1)).

Przykład:
- stan kolejki w danej chwili:

- stan kolejki po kolejnych pobraniach i usunięciach elementu:



- stan kolejki po kolejnych dopisaniach:



Przykładowa implementacja kolejki priorytetowej oparta na liście uporządkowanej (na podstawie: [Harris, Ross])

```
package queues;
import lists.LinkedList;
import lists.List;
import sorting.Comparator;

public class SortedListPriorityQueue implements Queue {
   private final List _list;
   private final Comparator _comparator;

   public SortedListPriorityQueue(Comparator comparator) {
       _comparator = comparator;
       _list = new LinkedList();
   }

   public void enqueue(Object value) {
      int pos = _list.size();
      while (pos > 0 && _comparator.compare(_list.get(pos - 1), value) > 0)
           --pos;
       _list.insert(pos, value);
   }
} // c.d.n.
```

Przykładowa implementacja kolejki priorytetowej oparta na liście uporządkowanej (na podstawie: [Harris, Ross])

```
// c.d.:

  public Object dequeue() throws EmptyQueueException {
      if (isEmpty()) throw new EmptyQueueException();
      return _list.delete(_list.size() - 1);
  }

  public void clear() {
      _list.clear();
  }

  public int size() {
     return _list.size();
  }

  public boolean isEmpty() {
     return _list.isEmpty();
   }
}
```

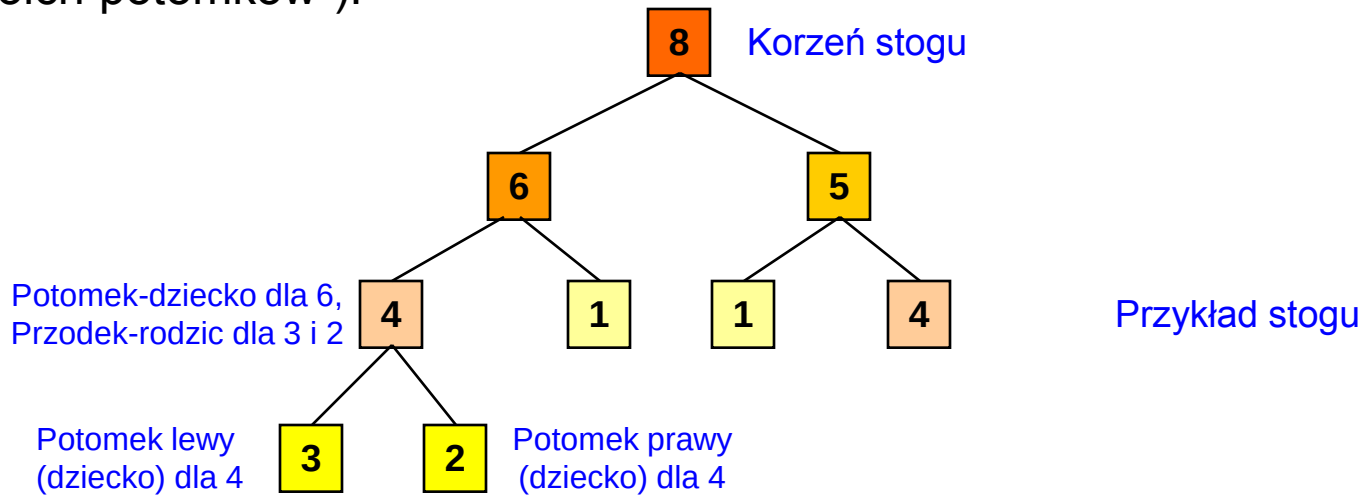
Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Kolejki priorytetowe z nieograniczonym zbiorem wartości priorytetów

Kolejki o organizacji stogowej

Wykorzystują stóg (*ang. Heap*), czasem nazywany też kopcem lub stertą.
Koncepcja stogu:
Stóg jest drzewem binarnym, posiadającym własności:

- kształtu: jest pełnym drzewem, przy czym ostatni poziom jest zapełniany od lewej strony ku prawej,
- uporządkowania (tzw. warunek stogowy): wartość węzła jest zawsze większa od wartości każdego z jego potomków (w skrócie: "węzeł jest zawsze większy od swoich potomków").

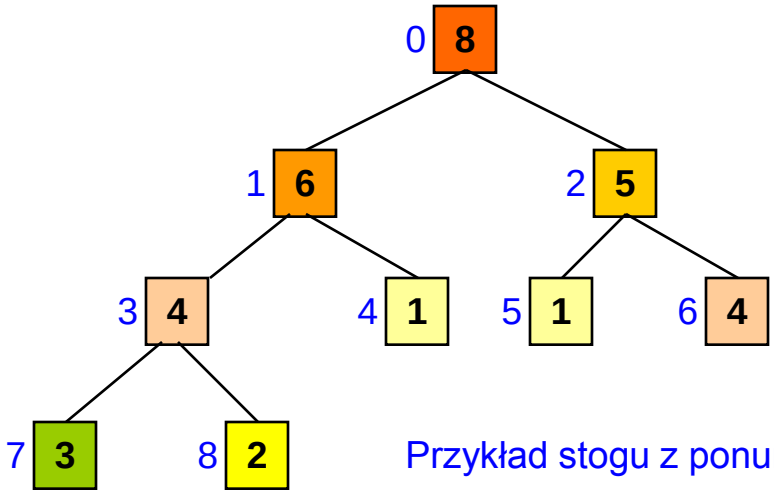


Własność uporządkowania jest przechodnia (w sensie matematycznym), więc korzeń stogu ma wartość największą ("jest największym elementem stogu").
Uwaga: Stogu nie należy utożsamiać z listą posortowaną!

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

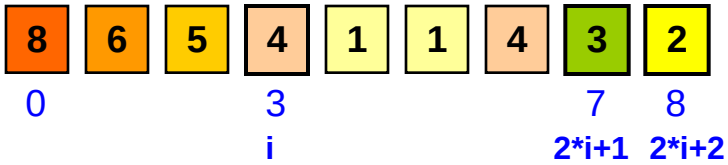
Kolejki priorytetowe o organizacji stogowej

Ze stogu można korzystać tak, jak z listy implementującej interfejs List.
Można ponumerować wierzchołki stogu według zasady:
- korzeń otrzymuje numer 0,
- na kolejnych poziomach (z góry na dół) kolejne wierzchołki (od lewej do prawej) otrzymują kolejne numery: 1, 2, 3, ...



Przykład stogu z ponumerowanymi węzłami

Wówczas lista odwzorowująca stóg ma postać:



Lewy potomek (dziecko) elementu z i-tej pozycji znajduje się na pozycji 2*i+1, prawy – na pozycji 2*i+2, przodek-rodzic elementu i-tego jest na pozycji (i-1)/2 (z zaokrągleniem w dół). Element na pozycji 0 nie posiada przodka-rodzica.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

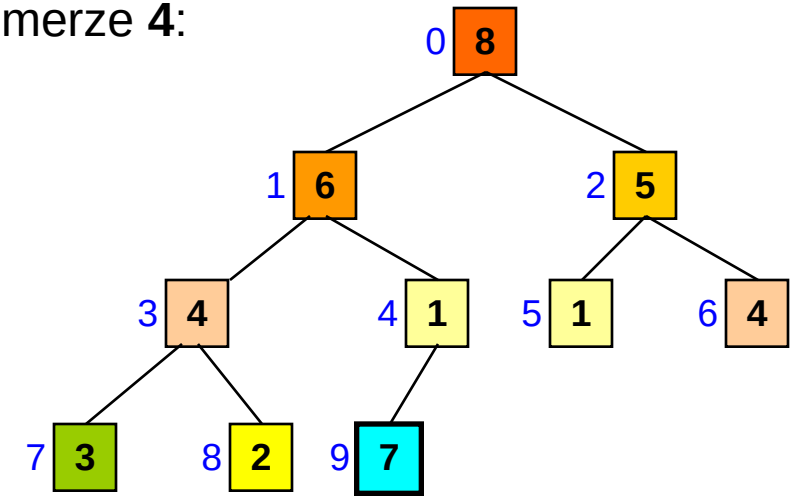
Operacje na stogu

Operacje na stogu (dodanie elementu do stogu oraz pobieranie i usuwanie elementu ze stogu) muszą zapewnić zachowanie warunku stogowego.

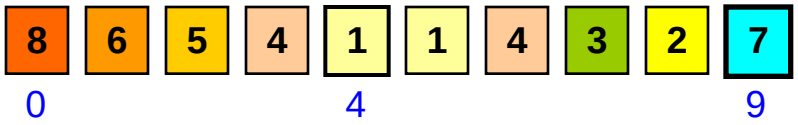
Operacja dodania elementu do stogu obejmuje dwa kroki:

1. **Dołączamy element do pierwszego (w sensie numeracji) węzła, który nie ma "kompletu" potomków-dzieci.**

Przykład: dodawany węzeł o wartości **7** staje się lewym potomkiem węzła o numerze **4**:



a w liście zajmuje ostatnią pozycję (zostaje dołączony na końcu listy):



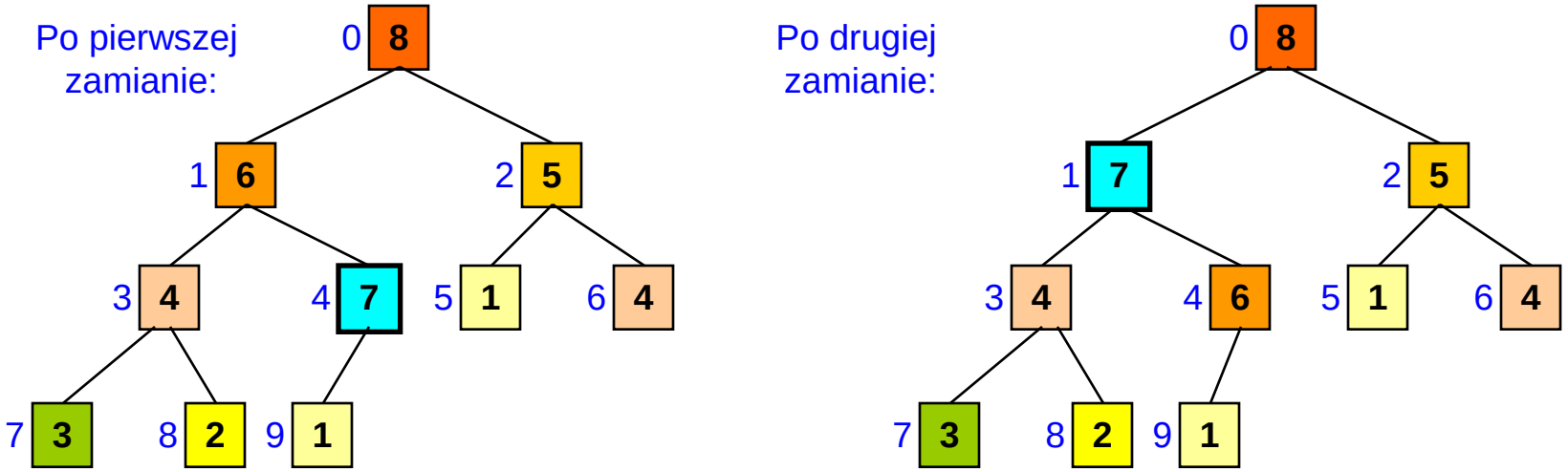
Dodanie tego węzła prowadzi do naruszenia warunku stogowego: węzeł nr 4 (o wartości 1) ma potomka o większej wartości (równej 7).

Przywrócenie warunku stogowego wymaga przeniesienia dopisanego elementu na wyższy poziom, tzw. wynurzenia (wyniesienia) elementu).

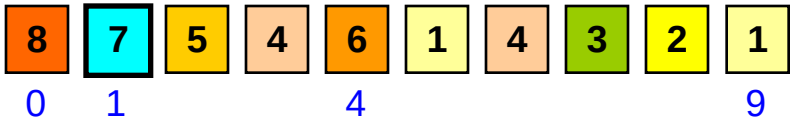
Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Operacje na stogu: dodanie elementu do stogu

2. Realizujemy wynurzenie (wynoszenie) elementu; polega to na sukcesywnej zamianie miejscami potomka-dziecka i jego przodka-rodzica (z dołu do góry stogu), aż do uzyskania sytuacji, w której zostanie spełniony (przywrócony) warunek stogowy.



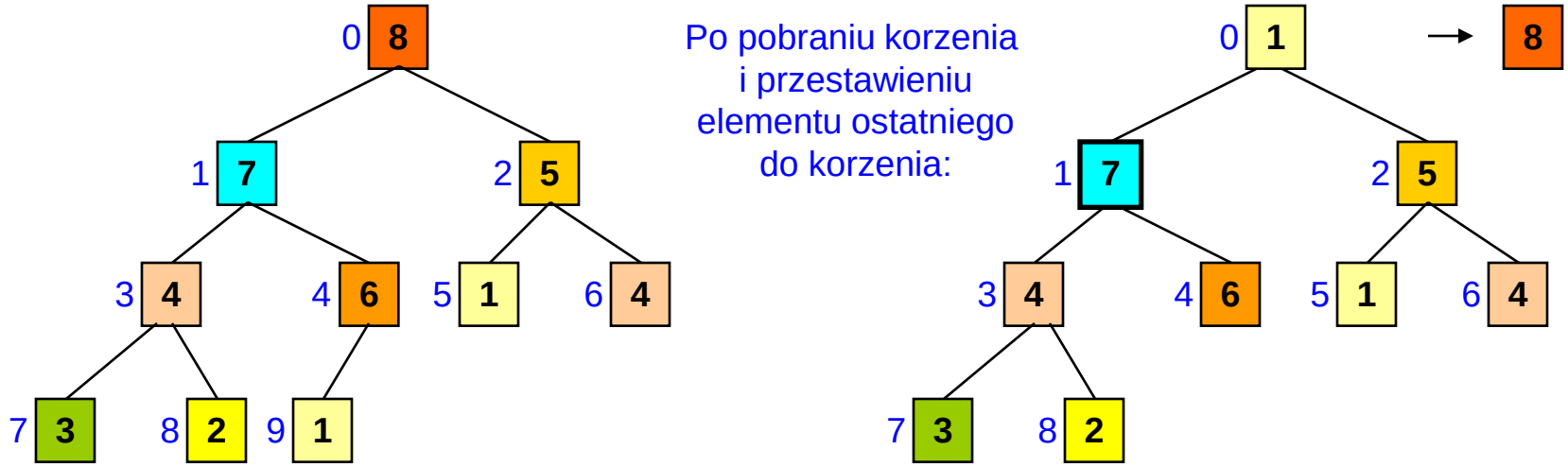
Zamiany miejscami wierzchołków podczas wynurzania elementu dołączanego do stogu – to zamiany miejscami elementów w liście. Postać listy po wynurzeniu elementu:



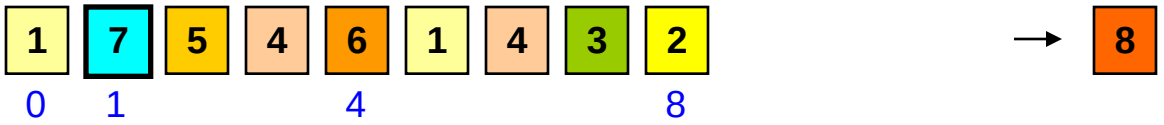
Operacje na stogu: pobieranie i usuwanie elementu ze stogu

Operacja pobrania i usunięcia elementu ze stogu obejmuje dwa kroki:

1. **Pobieramy element z korzenia stogu (jako największy) i przenosimy ostatni element stogu (w jego listowym rozwinięciu) w miejsce pobranego korzenia.**



Rozwinięcie listowe stogu ma teraz postać:

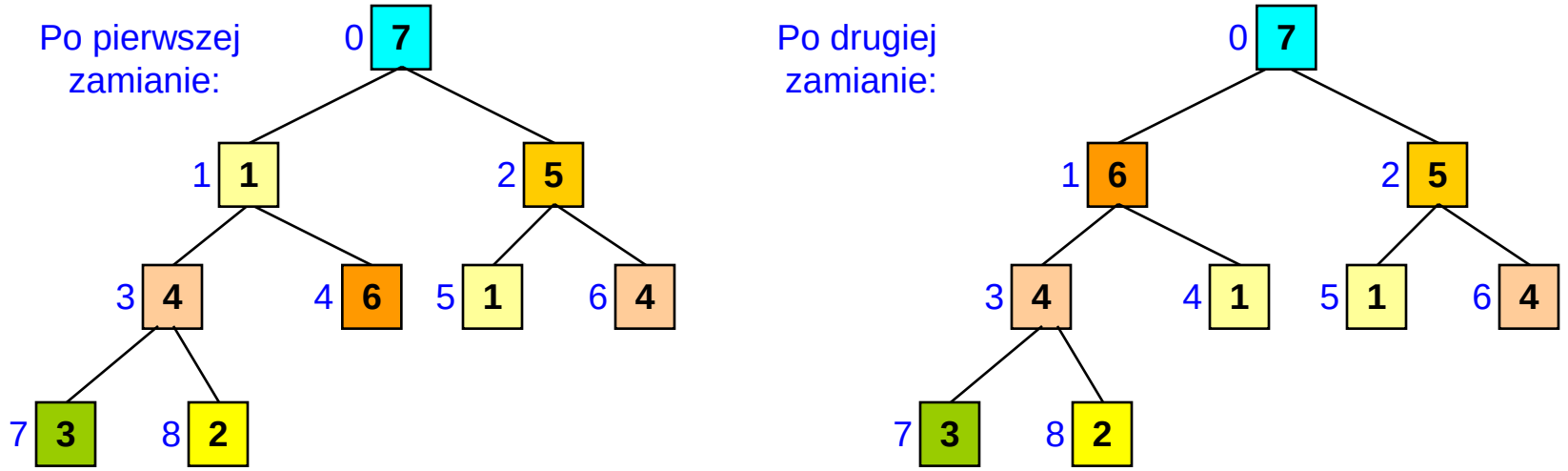


Przestawienie naruszyło warunek stogowy. Element z wierzchołka musi zostać przeniesiony na niższy poziom stogu (zatopiony, opuszczony), by przywrócić warunek stogowy.

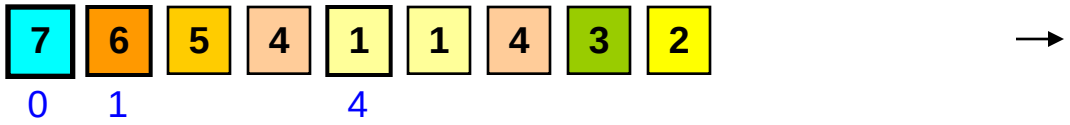
Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Operacje na stogu: pobieranie i usuwanie elementu ze stogu

2. Realizujemy zatapianie (opuszczanie) elementu; jest to sukcesywna zamiana tego elementu z większym z jego potomków-dzieci – aż do spełnienia warunku stogowego.



Zatapianie elementu – to zamiany miejscami elementów w liście (tu: 0<->1, 1<->4).
Postać listy po zatopieniu (opuszczeniu) elementu:



Przedstawione operacje na stogu wykorzystuje implementacja stogowej kolejki priorytetowej.

Przykładowa implementacja stogowej kolejki priorytetowej

```
package queues;
import lists.ArrayList;
import lists.List;
import sorting.Comparator;

public class HeapOrderedListPriorityQueue implements Queue {
    private final List _list;
    private final Comparator _comparator;

    public HeapOrderedListPriorityQueue(Comparator comparator) {
        _comparator = comparator;
        _list = new ArrayList();
    }

    public void enqueue(Object value) { // dołączenie elementu
        _list.add(value);
        swim(_list.size() - 1);                 // wynurzenie (wyniesienie) elementu
    }

// c.d.n.
```

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Przykładowa implementacja stogowej kolejki priorytetowej

```
// c.d.:
 private void swim(int index) { // wynurzanie elementu (wynoszenie elementu w górę)
    int parent;
    while(index != 0 &&
              _comparator.compare(_list.get(index), _list.get(parent= (index - 1) / 2)) > 0)
        { swap(index, parent);
          index=parent; }   //  wersja iteracyjna; w [Harris, Ross] jest wersja rekurencyjna
 }
 private void swap(int index1, int index2) { // zamiana miejscami dwóch elementów
     Object temp = _list.get(index1);
     _list.set(index1, _list.get(index2));
     _list.set(index2, temp);
 }
public Object dequeue() throws EmptyQueueException { // pobranie/usunięcie elementu
    if (isEmpty()) throw new EmptyQueueException();
    Object result = _list.get(0);
    if (_list.size() > 1) {
        _list.set(0, _list.get(_list.size() - 1));
        sink(0);                                          // zatapianie (opuszczanie) elementu
    }
    _list.delete(_list.size() - 1);
    return result;
} // c.d.n.
```

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Przykładowa implementacja stogowej kolejki priorytetowej

```
// c.d.:
  private void sink(int index) {   // zatapianie (opuszczanie) elementu
    boolean isDone=false;
    int child;
    while(!isDone && (child=2*index+ 1 ) < _list.size()) {
      if (child < _list.size()-1 &&
            _comparator.compare(_list.get(child), _list.get(child+1)) < 0)
                ++child;
      if (_comparator.compare(_list.get(index), _list.get(child)) < 0)
                swap(index, child);
        else isDone=true;
      index = child;
    }
  }
  public void clear() {
    _list.clear();
  }
  public int size() {
    return _list.size();
  }
  public boolean isEmpty() {
    return _list.isEmpty();
  }
}
```

Kolejki priorytetowe

Kolejki z ograniczonym, niewielkim zbiorem wartości priorytetów realizuje się z użyciem tablicy (lub listy) zwykłych kolejek. Każda kolejka odpowiada jednej wartości priorytetu i zachowuje kolejność dołączania elementów do kolejki.

Podsumowanie

Kolejki priorytetowe zrealizowane z użyciem listy nieuporządkowanej i listy uporządkowanej są bardzo mało efektywne (dla przeciętnych przypadków) w porównaniu do kolejki stogowej.

Złożoność obliczeniowa kolejki stogowej jest logarytmiczna (jest to cecha charakterystyczna algorytmów opartych na drzewach binarnych).

W najbardziej korzystnym przypadku (dodawanie do kolejki elementów posortowanych w kolejności rosnącej) realizacja kolejki priorytetowej opartej na liście uporządkowanej daje bardzo dobre wyniki. Przyczyna jest oczywista...

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Wykład 9.

Drzewa

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Drzewa – podstawowe pojęcia

Drzewo (*ang. tree*) ukorzenione jest zbiorem węzłów (wierzchołków) takim, że:

- każdy węzeł posiada k węzłów-dzieci (k>=0) i co najwyżej jednego przodka-rodzica,
- dokładnie jeden węzeł nie posiada przodka-rodzica; jest on korzeniem drzewa (*ang. root*),
- dla każdego węzła istnieje jedna droga od korzenia do tego wierzchołka (co oznacza, że w drzewie, traktowanym jak graf, nie występują cykle).

Węzły (wierzchołki) nie posiadające węzłów-dzieci są nazywane liśćmi drzewa.

Jeśli w drzewie, traktowanym jak graf skierowany, istnieje krawędź (gałąź) od węzła X do węzła Y, to X nazywamy przodkiem (rodzicem, ojcem) Y, a Y nazywamy potomkiem (dzieckiem, synem) X.

Liście drzewa są więc węzłami (wierzchołkami) bez potomków.

Węzeł mający co najmniej jednego potomka nazywamy węzłem wewnętrznym. Węzeł reprezentujący elementy nie występujące w drzewie (oznaczony przez NULL) nazywany jest węzłem zewnętrznym**.**

Węzeł X wraz z jego potomkami nazywamy poddrzewem o korzeniu X.

Głębokość (poziom) węzła X jest to długość drogi (wyrażona liczbą krawędzi) od korzenia do węzła X (korzeń ma głębokość 0).

Wysokość drzewa jest to maksymalna długość drogi od korzenia do liści (czyli największa z głębokości liści).

Wysokość drzewa pustego wynosi -1, a drzewa jednowęzłowego 0.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

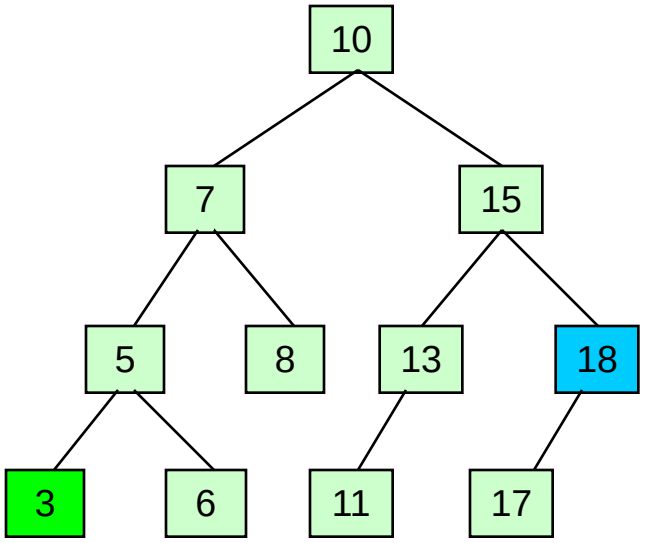
Drzewa uporządkowane, binarne drzewo poszukiwań

Drzewo uporządkowane - drzewo w którym węzły potomne każdego wierzchołka są uporządkowane (na rysunku - od lewej do prawej).

W programowaniu największe znaczenie mają uporządkowane drzewa binarne.

Binarne drzewo poszukiwań (*ang. Binary Search Tree, BST*) - drzewo uporządkowane spełniające warunki :

1. Potomek wierzchołka może być lewy lub prawy.
2. Każdy wierzchołek ma co najwyżej dwa węzły potomne.
3. W lewym poddrzewie znajdują się wartości mniejsze od korzenia, a w prawym większe.
4. Wartości kluczy nie powtarzają się w drzewie.



Minimum – węzeł o najmniejszej wartości klucza jest osiągalny na końcu ścieżki od korzenia poprzez lewych potomków.

Maksimum – węzeł o największej wartości klucza jest osiągalny na końcu ścieżki od korzenia poprzez prawych potomków.

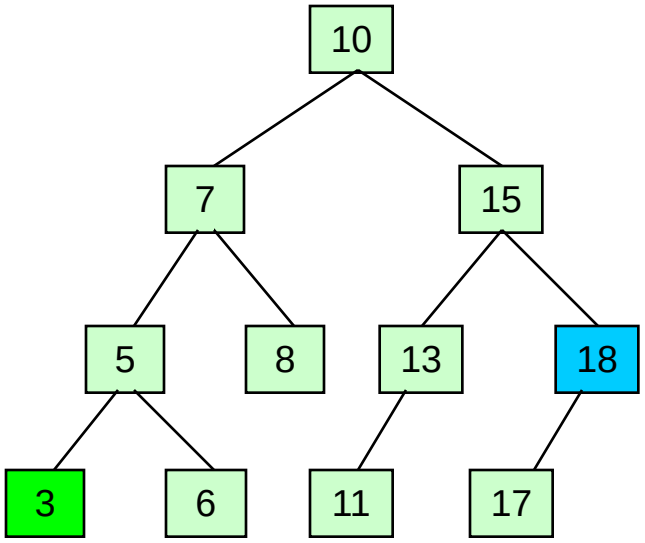
Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Drzewa uporządkowane, binarne drzewo poszukiwań

Następnik (*ang. successor*) danego węzła w drzewie poszukiwań to węzeł o wartości klucza bezpośrednio większej niż wartość klucza danego węzła.

Dwa przypadki:

1. Dla węzła posiadającego prawego potomka następnikiem jest minimum prawego poddrzewa (dla którego korzeniem jest ten prawy potomek).
2. Dla węzła nie posiadającego prawego potomka następnik jest wyszukiwany na ścieżce "w górę", aż do napotkania węzła X, który jest czyimś lewym potomkiem. Wówczas przodek tego węzła X jest szukanym następnikiem.



Przykład dla przypadku 1.:
dla węzła "10" następnikiem jest węzeł "11"

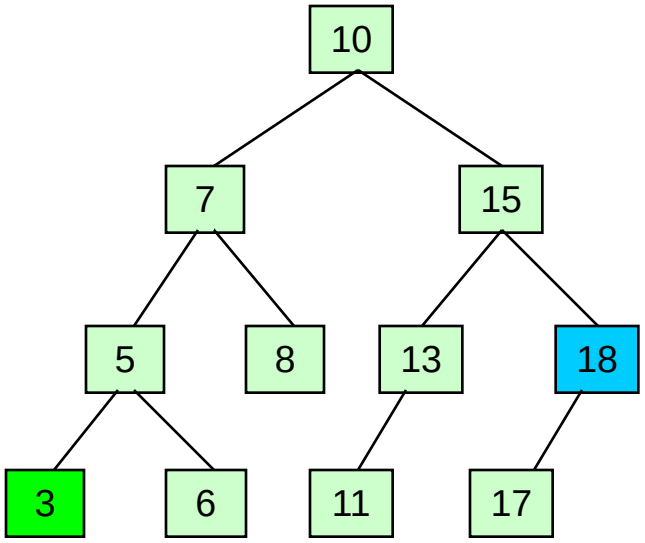
Przykład dla przypadku 2.:
dla węzła "8" następnikiem jest węzeł "10"

Drzewa uporządkowane, binarne drzewo poszukiwań

Poprzednik (*ang. predecessor*) danego węzła w drzewie poszukiwań to węzeł o wartości klucza bezpośrednio mniejszej niż wartość klucza danego węzła.

Dwa przypadki:

1. Dla węzła posiadającego lewego potomka poprzednikiem jest maksimum lewego poddrzewa (dla którego korzeniem jest ten lewy potomek).
2. Dla węzła nie posiadającego lewego potomka poprzednik jest wyszukiwany na ścieżce "w górę", aż do napotkania węzła X, który jest czyimś prawym potomkiem. Wówczas przodek tego węzła X jest szukanym poprzednikiem.



Przykład dla przypadku 1.:
dla węzła "10" poprzednikiem jest węzeł "8"

Przykłady dla przypadku 2.:
dla węzła "8" poprzednikiem jest węzeł "7"
dla węzła "17" poprzednikiem jest węzeł "15"

Uporządkowane drzewa binarne – podstawowe działania

Najprostsza postać węzła drzewa:

```
class Node {
   Object value;   // tu jest przechowana wartość klucza dla węzła
   Node left;      // lewe poddrzewo
   Node right;     // prawe poddrzewo
   Node(Object x) { // konstruktor węzła (wierzchołka) - liścia
      value=x; left = right = null; }
   Node(Object x, Node lewe, Node prawe) { // konstruktor węzła (wierzchołka)
                                           // spinającego (łączącego) poddrzewa
      value=x; left =lewe; right=prawe; // bez sprawdzania różnych warunków
                                        // dla uporządkowanego drzewa binarnego
   }
   // .. stosowne metody dla węzła
}
```

Podstawowe działania na drzewie:

1. Przechodzenie (w określonym porządku/kolejności) przez wszystkie wierzchołki drzewa ("przetwarzanie całego drzewa").
2. Wyszukiwanie elementu w drzewie (sprawdzanie czy w drzewie występuje wierzchołek o zadanej wartości klucza).
3. Wstawianie elementu (dołączanie nowego wierzchołka do drzewa).
4. Usuwanie wierzchołka z drzewa.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Uporządkowane drzewa binarne – podstawowe działania

Przechodzenie przez wszystkie wierzchołki drzewa (o korzeniu x)

Istnieje kilka sposobów przechodzenia (trawersacji) drzewa:

1. Przejście "in-order"

Porządkujące, zwane czasem "porządkiem poprzecznym".
Daje wierzchołki od najmniejszego do największego; najważniejszy sposób przejścia przez drzewo binarne.

przejdź_InOrder(Wierzchołek x) <= korzeń drzewa (poddrzewa)

1. przejdź_InOrder(lewy potomek x);
2. przetwórz(x); // odwiedź (wykorzystaj) węzeł x
3. przejdź_InOrder(prawy potomek x)

Pozostałe sposoby przechodzenia drzewa (2..6):
każda z innych możliwych (sześciu) kombinacji operacji przejdź(...) i przetworz(...)

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Uporządkowane drzewa binarne – podstawowe działania

Przechodzenie przez wszystkie wierzchołki drzewa (o korzeniu x)

```
// przykład metody klasy Node do wyprowadzenia ciągu wartości z drzewa
// w porządku poprzecznym (in order):

 public String toStringInOrder(){
        String b="";
        if (left!=null) b+=left.toStringInOrder();
        b+=" "+value.toString();
        if (right!=null) b += " " + right.toStringInOrder();
        return b;
 }
```

Uporządkowane drzewa binarne – podstawowe działania

Przejście "in-order"

// przykład użycia metody toStringInOrder() dla drzewa

```
import sorting.Comparator;
public class BST {
    private final Comparator _comparator;
    private Node _root;
    public BST(Comparator comparator) {
       _comparator = comparator;
       _root=null;
    }
    public String treeToStringInOrder(){
        String b="In order: ";
        if (_root!=null) b+=_root.toStringInOrder();
            else b+="drzewo puste";
        return b;
   }
   // .. miejsce na pozostałe metody dla drzewa BST ..
}
```

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Uporządkowane drzewa binarne – podstawowe działania

Przechodzenie przez wszystkie wierzchołki drzewa (o korzeniu x)

Inne sposoby przechodzenia (trawersacji) drzewa:

2. Przejście "pre-order"

przejdź_PreOrder(Wierzchołek x) <= korzeń drzewa (poddrzewa)

1. przetwórz(x); // odwiedź (wykorzystaj) węzeł x
2. przejdź_PreOrder(lewy potomek x);
3. przejdź_PreOrder(prawy potomek x)

3. Przejście "post-order"

przejdź_PostOrder(Wierzchołek x) <= korzeń drzewa (poddrzewa)

1. przejdź_PostOrder(lewy potomek x);
2. przejdź_PostOrder(prawy potomek x);
3. przetwórz(x); // odwiedź (wykorzystaj) węzeł x

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Uporządkowane drzewa binarne – podstawowe działania

Wyszukiwanie elementu w drzewie

Oznacza sprawdzanie czy w drzewie występuje element o zadanej wartości V klucza oraz (jeśli jest) pobranie wartości przechowywanej w tym węźle.

wyszukaj(Wartość V):

- Ustal: węzeł = korzeń drzewa.
- Jeśli węzeł == null -> koniec, element o wartości klucza V nie istnieje w drzewie.
- Porównanie wartości klucza dla węzła z wartością V (z użyciem komparatora).
  - jeśli równe -> element znaleziono; zwróć wartość elementu, koniec.
  - jeśli szukana wartość jest mniejsza niż V, ustal węzeł = lewy potomek, przejdź do 2.
  - jeśli szukana wartość jest większa niż V, ustal węzeł = prawy potomek, przejdź do 2.

Przy każdym przejściu do potomka (w głąb drzewa) zostaje wyeliminowana z przeglądania połowa pozostałych jeszcze węzłów drzewa. Jest to więc wyszukiwanie binarne – algorytm o logarytmicznej złożoności obliczeniowej.
O maksymalnej liczbie "kroków" decyduje głębokość drzewa. Jeśli drzewo jest wyważone ("regularne", "symetryczne"), jego głębokość jest określona zależnością logarytmiczną. Może się jednak zdarzyć, że drzewo będzie miało głębokość równą n-1 (gdzie: n – liczba węzłów).

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Uporządkowane drzewa binarne – podstawowe działania

Wyszukiwanie elementu w drzewie

Przykład iteracyjnej realizacji metody wyszukiwania elementu w drzewie:

```
public Object find(Object x) {
   Node t = search(x);
   return t !=null ? t.value : null;
}
private Node search(Object value) { // metoda iteracyjna
   Node node = _root;
   int cmp=0;
   while (node != null &&(cmp = _comparator.compare(value, node.value))!=0)
      node = cmp < 0 ? node.left : node.right;
   return node;
}
```

Realizacja rekurencyjna nie powinna nastręczać trudności. Proszę się o tym przekonać...

Uporządkowane drzewa binarne – podstawowe działania

Wstawianie elementu (dołączanie nowego wierzchołka do drzewa)

Oznacza przeszukanie drzewa i – jeśli nie ma w nim węzła o wartości równej wartości klucza wstawianego elementu – dołączenie go jako liść).
Jeśli taki węzeł jest to zgłaszany jest wyjątek.

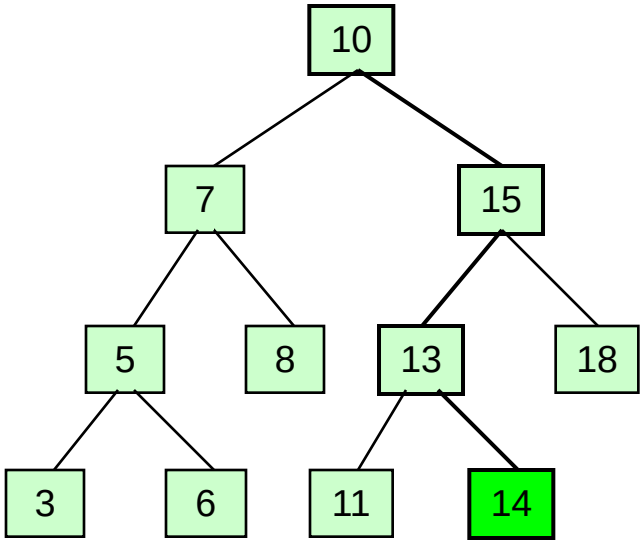
Ustal: węzeł = korzeń drzewa.
wstaw do drzewa (Wartość V):
1. Jeśli węzeł == null, to utwórz węzeł z wartością V. (koniec pomyślny).
2. Porównanie wartości klucza dla węzła z wartością V (z użyciem komparatora).
   - jeśli wartość V jest mniejsza od wartości węzła, wstaw wartość do poddrzewa wskazanego przez lewego potomka (węzeł=lewy potomek, przejdź do 2.)
   - jeśli wartość V jest większa od wartości węzła, wstaw wartość do poddrzewa wskazanego przez prawego potomka (węzeł=prawy potomek, przejdź do 2.)
   - jeśli równe -> element już istnieje; zgłoś wyjątek. (koniec niepomyślny).

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Uporządkowane drzewa binarne – podstawowe działania

Wstawianie elementu (dołączanie nowego wierzchołka do drzewa)

Przykład wstawiania elementu o wartości V=14
1. węzeł=10
2. V>10, wstaw do prawego (15)
1. węzeł=15
2. V<15, wstaw do lewego (13)
1. węzeł=13
2. V>13, wstaw do prawego (13)
1. węzeł==null. Utworz węzeł 14 (jako liść - prawy potomek węzła 13)



Uporządkowane drzewa binarne – podstawowe działania

Wstawianie elementu (dołączanie nowego wierzchołka do drzewa)

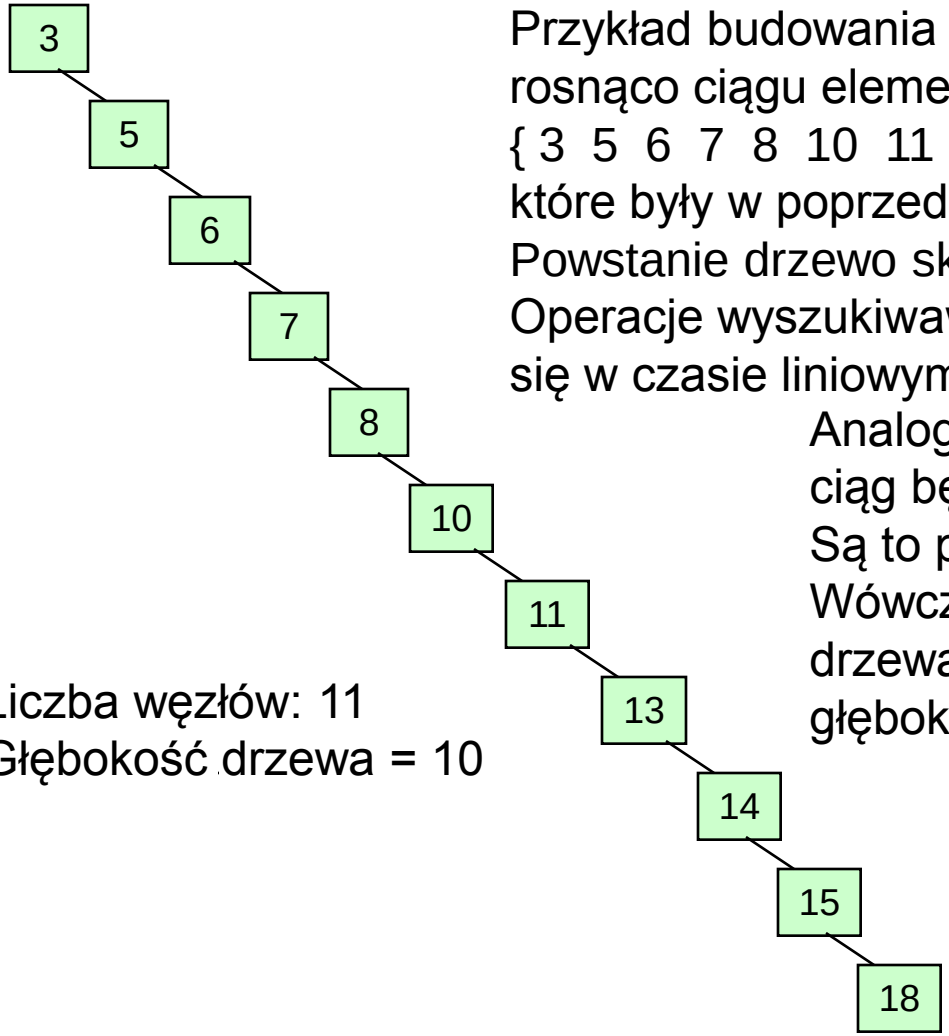
Przykład realizacji metody wstawiania elementu do drzewa:

```
public void insert(Object x) {
    _root= insert(x,_root);
}
protected Node insert(Object x, Node t) {
   if (t== null) t=new Node(x);
     else {
              int cmp=_comparator.compare(x,t.value);
              if(cmp<0) t.left=insert(x, t.left);
                else if(cmp>0) t._right=insert(x, t.right);
                        else throw new DuplicateItemException(x.toString());
            }
     return t;
}
public class DuplicateItemException extends RuntimeException {
     public DuplicateItemException(String message) { super(message); }
}
```

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Uporządkowane drzewa binarne – podstawowe działania

Wstawianie elementu (dołączanie nowego wierzchołka do drzewa)



Przykład budowania drzewa dla uporządkowanego rosnąco ciągu elementów, np.:
{ 3 5 6 7 8 10 11 13 14 15 18 } – czyli tych, które były w poprzednim drzewie.
Powstanie drzewo skrajnie niewyważone.
Operacje wyszukiwawcze w tym drzewie wykonują się w czasie liniowym, zamiast w logarytmicznym!

Analogiczna sytuacja wystąpi, gdy ciąg będzie uporządkowany malejąco.
Są to przykłady degradacji BST.
Wówczas trzeba prowadzić wyważanie drzewa, by zapewnić możliwie najmniejszą głębokość drzewa.

Liczba węzłów: 11
Głębokość drzewa = 10

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Uporządkowane drzewa binarne – podstawowe działania

Usuwanie węzła z drzewa

Należy odszukać węzeł, który należy usunąć. Jeśli węzeł nie istnieje, należy zasygnalizować błąd (zgłosić wyjątek).

Węzeł X, który należy usunąć, może znajdować się w jednym z trzech stanów:

1. jest liściem (wówczas można go po prostu usunąć, bez dodatkowych działań),
2. posiada dokładnie jednego potomka (lewego lub prawego); wówczas potomek zajmuje miejsce tego usuwanego węzła (czyli miejsce węzła zajmuje lewe lub prawe poddrzewo usuwanego węzła),
3. posiada dwóch potomków; należy wówczas:
   - węzeł X zamienić miejscami z jego następnikiem N (co oznacza zamianę wartości elementów przechowywanych w węźle; po tej zamianie w węźle X będzie przechowywana wartość następnika N); w tej chwili może być naruszony warunek drzewa binarnego.
   - usunąć węzeł N, który po tej zamianie znalazł się o w jednym ze stanów: 1 (jest liściem) lub 2 (posiada tylko prawego potomka; nie mógł mieć lewego potomka, bo wówczas nie byłby następnikiem - elementem minimalnym w prawym poddrzewie względem usuwanego elementu X).

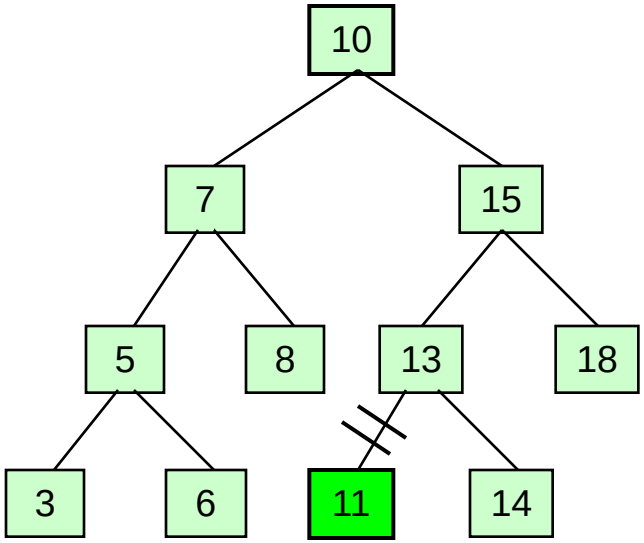
   Węzeł X można zastąpić jego poprzednikiem (elementem maksymalnym, czyli skrajnie prawym w lewym poddrzewie węzła X.

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Uporządkowane drzewa binarne – podstawowe działania

Usuwanie węzła z drzewa

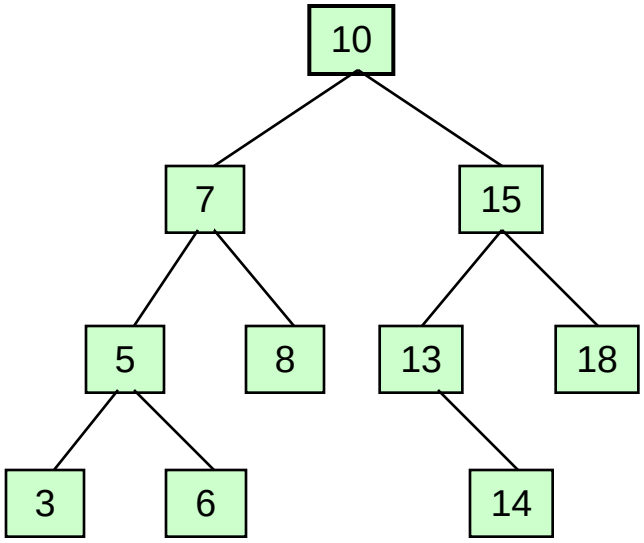
Przykład usuwania elementu o wartości V=11 jest bezproblemowe:
przodek (w tym przypadku "13") traci potomka (w tym przypadku lewego).



Uporządkowane drzewa binarne – podstawowe działania

Usuwanie węzła z drzewa

Stan po usunięciu węzła 11

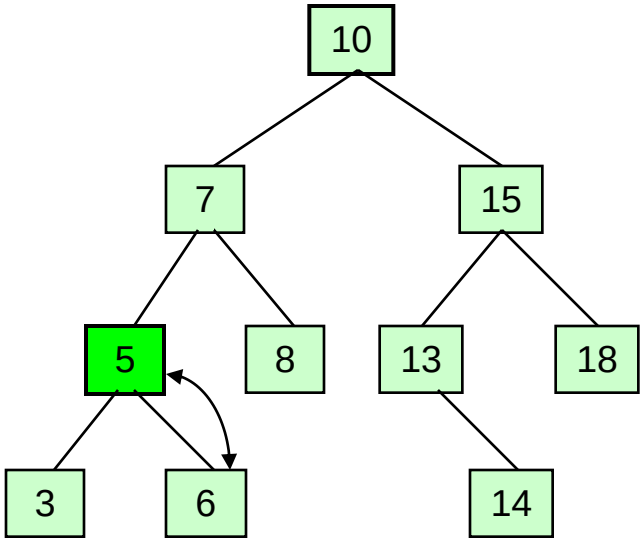


Uporządkowane drzewa binarne – podstawowe działania

Usuwanie węzła z drzewa

Przykład usuwania elementu o wartości V=5:

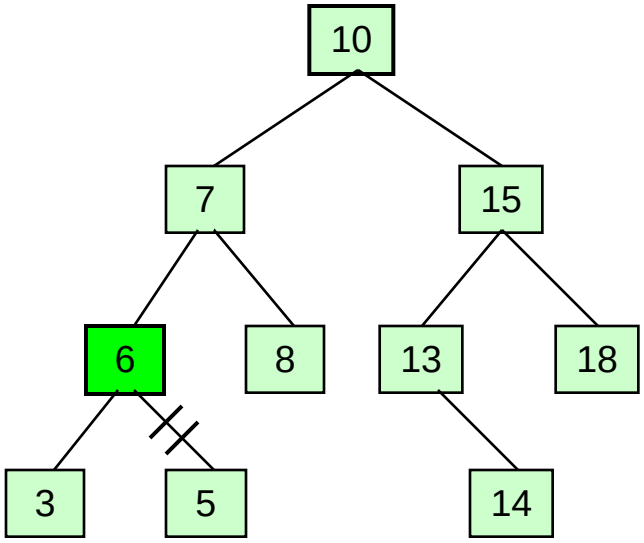
Następnikiem 5 jest 6 (skrajny lewy element jego prawego poddrzewa), więc następuje zamiana wartości 5 i 6 oraz bezproblemowe usunięcie węzła "6", w którym (w wyniku przestawienia) znalazła się usuwana wartość 5.



Uporządkowane drzewa binarne – podstawowe działania

Usuwanie węzła z drzewa

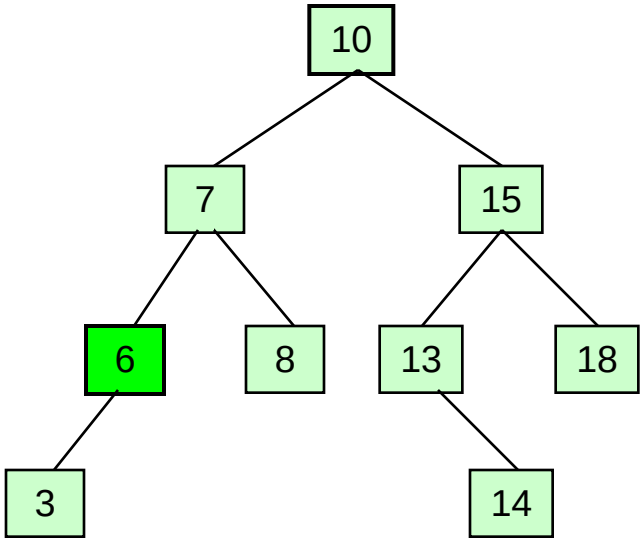
Stan po przestawieniu elementów 5 i 6...



Uporządkowane drzewa binarne – podstawowe działania

Usuwanie węzła z drzewa

Stan po usunięciu elementu 5 (z poprzedniej pozycji elementu 6).



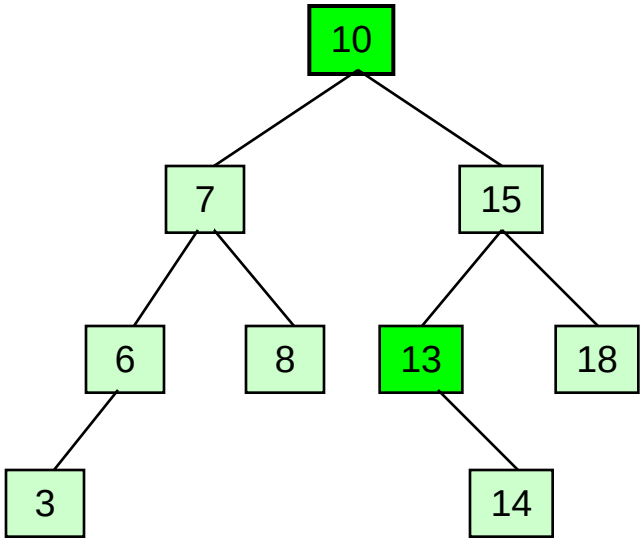
Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Uporządkowane drzewa binarne – podstawowe działania

Usuwanie węzła z drzewa

Jeszcze jedno usunięcie – tym razem korzenia 10

Jego następnikiem jest 13 (lewy skrajny element prawego poddrzewa).



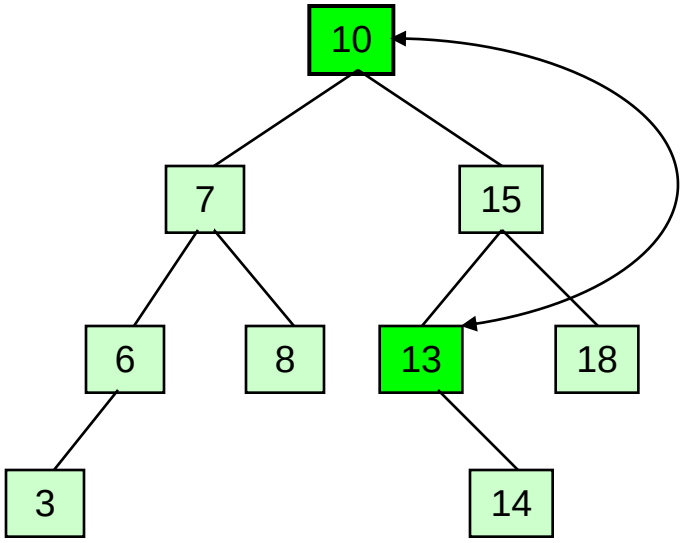
Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Uporządkowane drzewa binarne – podstawowe działania

Usuwanie węzła z drzewa

Jeszcze jedno usunięcie – tym razem korzenia 10

Jego następnikiem jest 13 (lewy skrajny element prawego poddrzewa).
Elementy te należy zamienić miejscami.



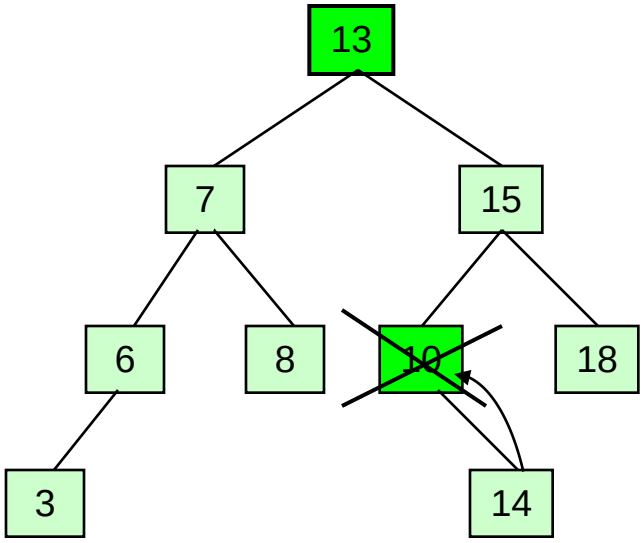
Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Uporządkowane drzewa binarne – podstawowe działania

Usuwanie węzła z drzewa

Jeszcze jedno usunięcie – tym razem korzenia 10

Jego następnikiem jest 13 (lewy skrajny element prawego poddrzewa).
Elementy te należy zamienić miejscami.
Element 10 należy usunąć, zastępując go jego następnikiem (jest nim 14).



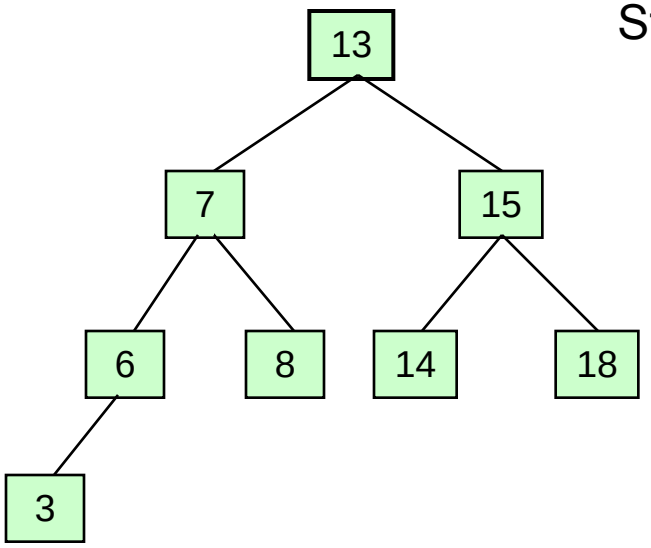
Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Uporządkowane drzewa binarne – podstawowe działania

Usuwanie węzła z drzewa

Jeszcze jedno usunięcie – tym razem korzenia 10

Jego następnikiem jest 13 (lewy skrajny element prawego poddrzewa).
Elementy te należy zamienić miejscami.
Element 10 należy usunąć, zastępując go jego następnikiem (jest nim 14).
Stan po usunięciu elementu 10 z korzenia:



Uporządkowane drzewa binarne – podstawowe działania

Usuwanie węzła z drzewa

Przykład realizacji metody usuwania elementu z drzewa:

```
public void delete(Object x) { _root=delete(x,_root); }

protected Node delete(Object x, Node t) {
    if (t==null) throw new ItemNotFoundException(x.toString());
       else {
                int cmp=_comparator.compare(x,t.value);
                if (cmp<0) t._left=delete(x, t.left);
                 else if(cmp>0) t.right=delete(x,t.right);
                         else if(t.left!=null &&t.right!=null) t.right=detachMin(t.right,t);
                                 else  t = (t.left != null) ? t.left : t.right;
            }
      return t;
    }
//zastąpienie usuwanego elementu jego następnikiem i usunięcie następnika
protected Node detachMin(Node t, Node del) {
  if (t.left!=null) t.left=detachMin(t.left, del);
     else {del.value=t.value; t=t.right;}
 return t;
}
```

Uporządkowane drzewa binarne – podstawowe działania

Wyważanie drzewa

Wstawianie i usuwanie węzłów drzewa binarnego może naruszyć jego wyważenie i znacznie zwiększyć wysokość drzewa przy stosunkowo niewielkiej liczbie węzłów (aż do postaci listy uporządkowanej -> zdegenerowane drzewo binarne), co sprowadza złożoność obliczeniową operacji z logarytmicznej do liniowej.

Dlatego należy dbać na bieżąco o „kształt” drzewa i dokonywać wyważenia (*ang. balancing*) praktycznie po każdej operacji naruszającej to wyważenie.

Wyważanie drzewa jest ciągiem działań polegających na przekształceniu drzewa binarnego przeciążonego (w lewo lub w prawo) do postaci równoważnej o mniejszej wysokości.
Operacje wyważania na bieżąco (po każdej zmianie) dużych drzew są kosztowne.

Jedną z najprostszych metod zaproponowali Adelson-Velskij i Landis (stąd pochodzi akronim AVL dla oznaczania drzew wyważonych tą metodą).

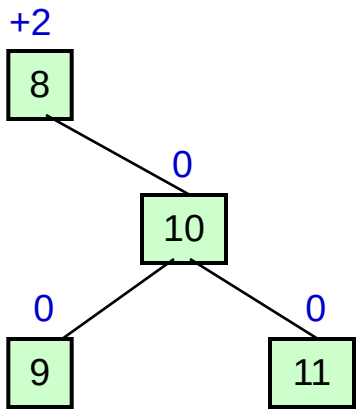
Idea metody:
Śledzenie wysokości obydwu poddrzew każdego z węzłów; jeśli dla każdego z węzłów wysokości poddrzew różnią się nie więcej, niż o 1, to takie drzewo uznawane jest za wyważone (-> drzewo AVL).

Politechnika Wrocławska, Wydział Informatyki i Zarządzania
Zbigniew Szpunar

Uporządkowane drzewa binarne – podstawowe działania

Wyważanie drzewa

Prosty przykład drzewa niewyważonego:



Wyważenie przywraca się poprzez stosowanie ciągu elementarnych operacji (przekształcania drzewa binarnego w drzewo równoważne) zwanych rotacjami.

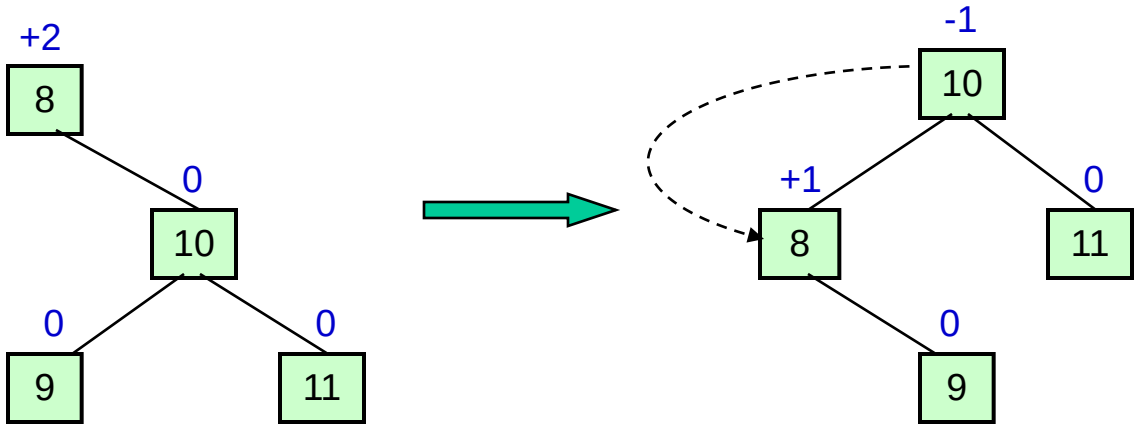
Rotacje wykonuje się rozpoczynając od miejsca wstawienia/usunięcia węzła w kierunku do korzenia drzewa.

Typy rotacji:
- w lewo,
- w prawo,
- w wariancie pojedynczym,
- w wariancie podwójnym.

Uporządkowane drzewa binarne – podstawowe działania

Wyważanie drzewa

Prosty przykład rotacji wyważającej drzewo:



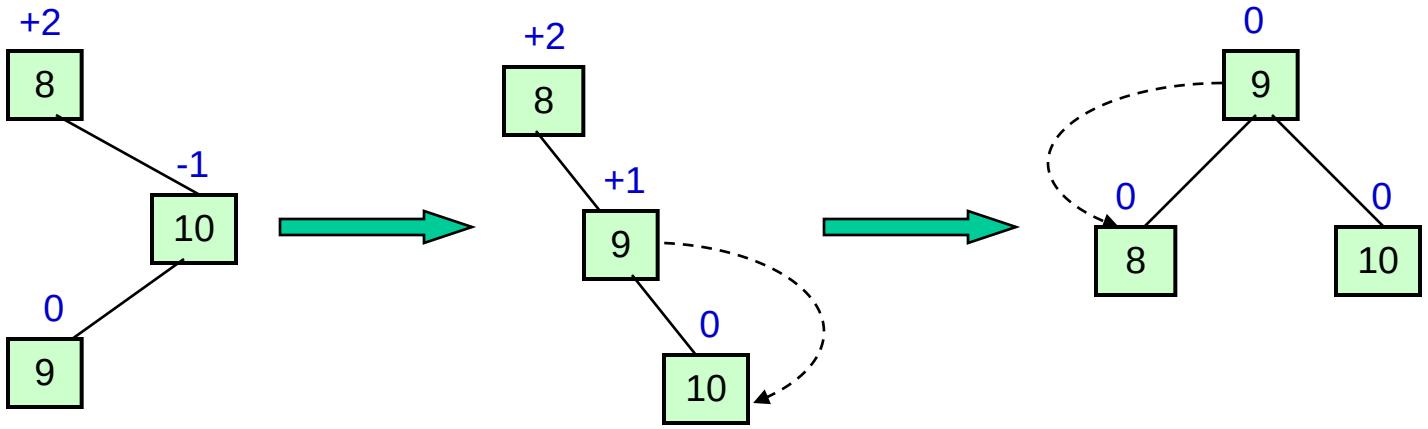
Zastosowana rotacja (pojedyncza, w lewo – ponieważ węzeł 10 był przeciążony w prawo) promuje węzeł 10 i degraduje węzeł 8.

Taka sytuacja "przeciążenia” mogła nastąpić np. po usunięciu (nie pokazanego na rysunku z lewej strony) węzła 7 (lewego potomka węzła 8).

Uporządkowane drzewa binarne – podstawowe działania

Wyważanie drzewa

Inna sytuacja:



Zastosowane zostały dwie rotacje (pierwsza w prawo dla węzła 10, druga w lewo – dla węzła 8). Takie działanie nazywa się rotacją podwójną.

Uporządkowane drzewa binarne – podstawowe działania

Wyważanie drzewa

W zależności od rodzaju przeciążenia wybiera się rodzaj rotacji:

Jeśli występuje przeciążenie w lewo, to:

jeśli potomek zrównoważony lub przeciążony w lewo, to rotacja pojedyncza;
jeśli potomek przeciążony w prawo, to rotacja podwójna.

Jeśli występuje przeciążenie w prawo, to:

jeśli potomek zrównoważony lub przeciążony w prawo, to rotacja pojedyncza;
jeśli potomek przeciążony w lewo, to rotacja podwójna.

Ciekawe wyliczenie dla poparcia tezy o potrzebie wyważania drzewa binarnego:
W drzewie doskonale wyważonym złożonym z 1 000 000 węzłów znalezienie węzła wymaga średnio około log2 (1000000) porównań (czyli ok. 20 porównań).
W drzewie AVL o takich rozmiarach liczba porównań jest tylko o 45% większa (ok. 28). W drzewie zdegenerowanym do listy: 500 000.