Akademia Górniczo-Hutnicza
Wydział Elektrotechniki, Automatyki,
Informatyki i Inżynierii Biomedycznej

Adrian Horzyk

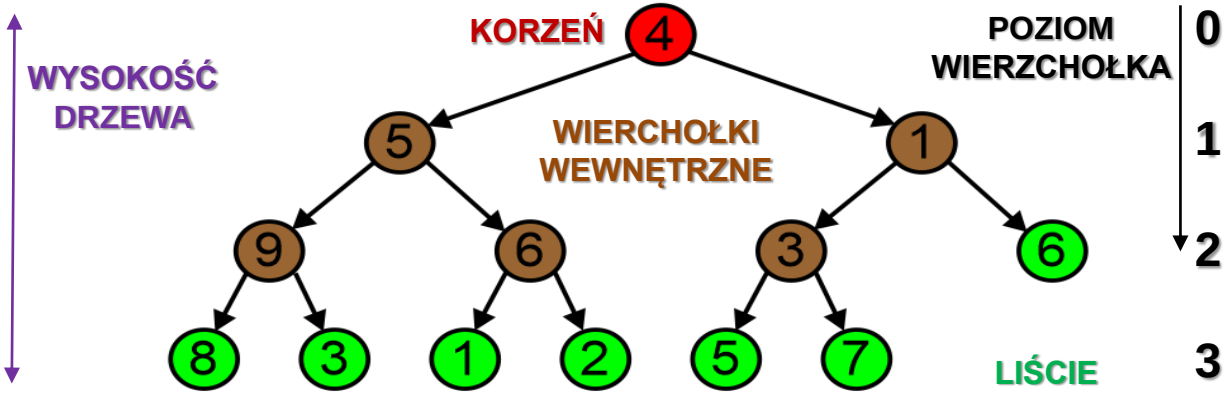
# WSTĘP DO INFORMATYKI

## Drzewa i struktury drzewiaste

# DEFINICJA DRZEWA

**Drzewo** to struktura danych składająca się z **wierzchołków** (węzłów) i **krawędzi**, przy czym krawędzie łączą wierzchołki w taki sposób, iż istnieje zawsze dokładnie jedna droga pomiędzy dowolnymi dwoma wierzchołkami.

Drzewa w informatyce rosną w dół!




Lista przekształcona w drzewo binarne: [4]-[5]-[1]-[9]-[6]-[3]-[6]-[8]-[3]-[1]-[2]-[5]-[7]

Wierzchołki w drzewach przedstawiamy **w postaci warstwowej**,
tzn. każdy wierzchołek w drzewie znajduje się na jakimś **poziomie**.

**Poziom wierzchołka** w drzewie jest równy długości drogi łączącej go z korzeniem. Korzeń drzewa jest na poziomie 0.

**Wysokość drzewa** równa jest maksymalnemu poziomowi drzewa, czyli długości najdłuższej spośród ścieżek prowadzących od korzenia do poszczególnych liści drzewa.

Wierzchołki mogą posiadać **rodzica**, który jest umieszczony na wyższym poziomie oraz **dzieci**,
które są umieszczone na niższym poziomie. Niektóre dzieci nie posiadają własnych dzieci i są **liśćmi**.

Dzieci jednego rodzica nazywamy **rodzeństwem**.

Wierzchołki, które nie posiadają ani jednego dziecka nazywamy **liśćmi**.

**Przodkami** są rodzice oraz rekurencyjnie rodzice rodziców.

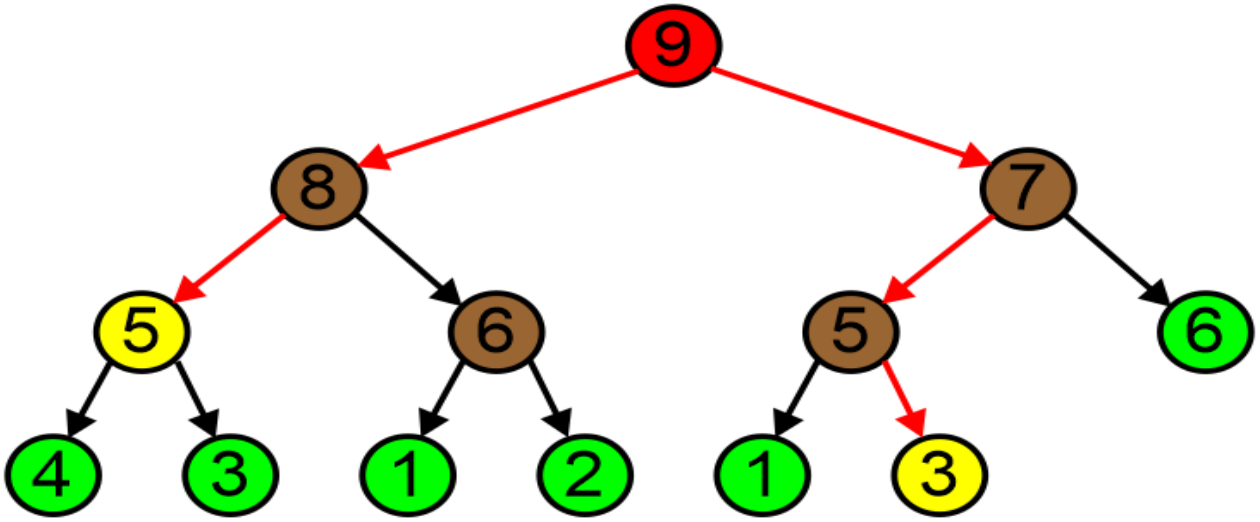
**Potomkami** są dzieci oraz rekurencyjnie dzieci dzieci.

Wierzchołki posiadające zarówno rodzica jak i przynajmniej jedno dziecko nazywamy **wierzchołkami wewnętrznymi**.

Każde drzewo posiada wyróżniony, nie posiadający rodzica wierzchołek, który nazywamy **korzeniem**.

# PARAMETRY DRZEW

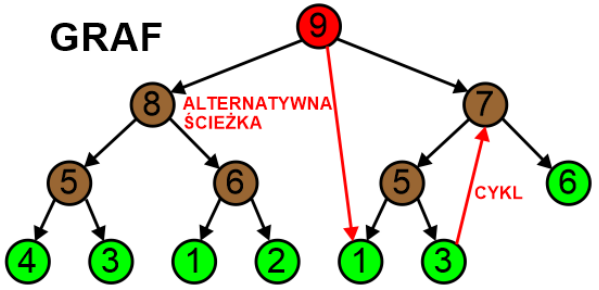
**Drogę (ścieżkę) w drzewie** pomiędzy dwoma wierzchołkami wyznaczamy poprzez wierzchołki pośrednie przechodząc po łączących je krawędziach.




**Długość drogi pomiędzy dwoma wierzchołkami w drzewie** wyznaczamy jako ilość krawędzi, po których trzeba przejść, żeby przejść z jednego wierzchołka do drugiego.

W drzewach **nie istnieją cykle**, czyli nietrywialna droga posiadająca początek i koniec w tym samym wierzchołku. Drzewa **nie posiadają też kilka alternatywnych dróg** pomiędzy tymi samymi wierzchołkami, lecz tylko dokładnie jedną:

Jeśliby do drzewa dodano taką krawędź, która by utworzyła cykl lub alternatywną drogę, wtedy drzewo staje się **grafem**:

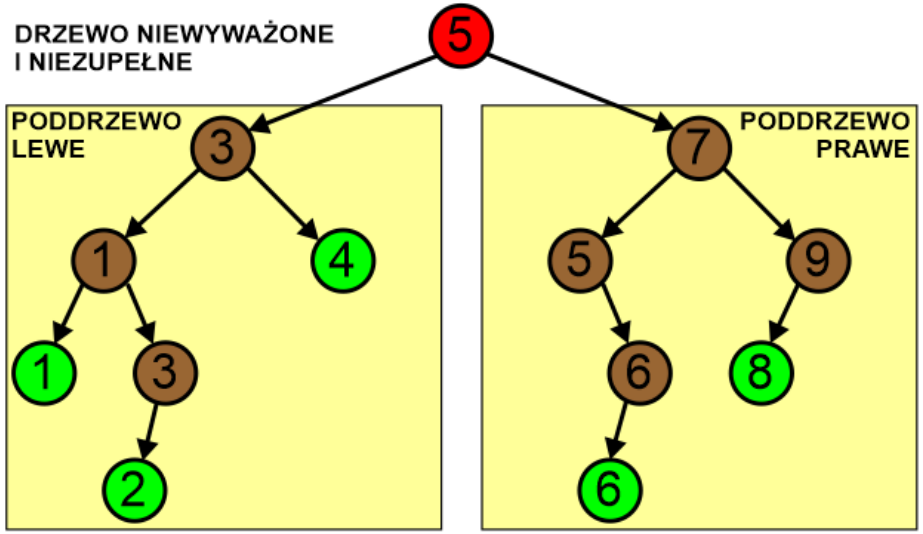


# WYWAŻANIE DRZEW

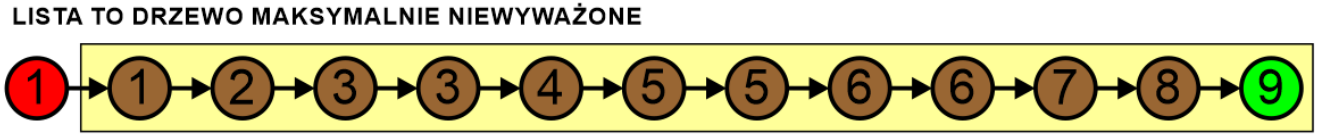
**Drzewo binarne jest wyważone (zrównoważone),** gdy wysokość lewego i prawego poddrzewa każdego jego wierzchołka nie różni się o więcej niż jeden.

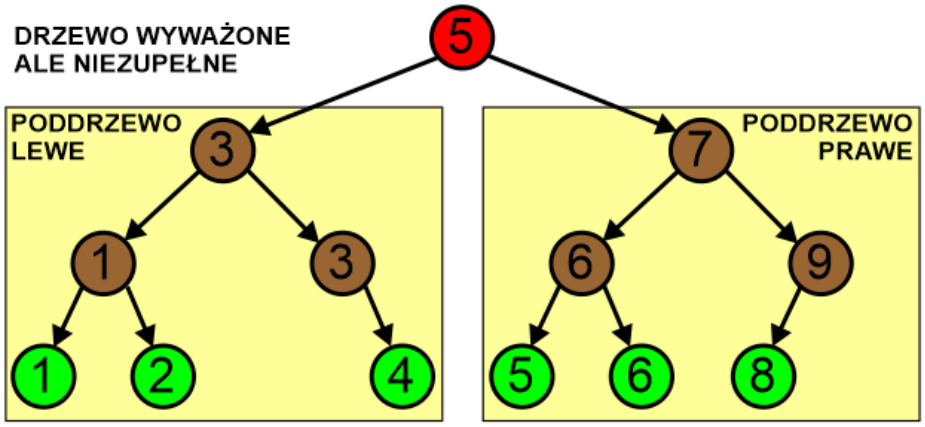
**Drzewo jest doskonale zrównoważone**, gdy dodatkowo wszystkie jego liście znajdują się na maksymalnie dwóch poziomach.

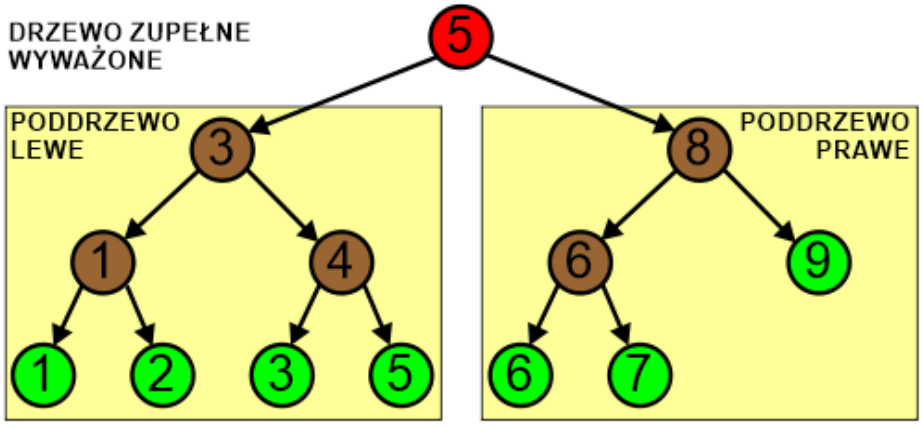
Klasyczne drzewo poszukiwań BST nie posiada wbudowanych mechanizmów równoważenia.




Przykładem drzewa totalnie niezrównoważonego jest lista:



# RODZAJE DRZEW

Drzewa mogą być między innymi:

**Binarne** – jeśli każdy węzeł posiada co najwyżej dwójkę dzieci.

**k-Regularne** – jeśli każdy węzeł posiada co najwyżej k dzieci.

**Nieregularne** – jeśli nie mają określonego maksymalnego stopnia węzła w drzewie.

**Zupełne (kompletne)** – gdy ma wszystkie poziomy z wyjątkiem ostatniego całkowicie zapełnione, a ostatni jest spójnie zapełniony od strony lewej.

**Kopiec** – to drzewo binarne, w którego węzłach znajdują się elementy reprezentowanego multizbioru, w którym dzieci nie są większe od rodziców.

**Decyzyjne** – gdy węzły drzewa zawierają warunki, a liście decyzje.

**BST** to **binarne drzewo poszukiwań** zawierające elementy o unikalnych kluczach, w którym klucze elementów w lewym poddrzewie są mniejsze od kluczy elementów w prawym poddrzewie dla każdego wierzchołka drzewa. BST nie posiada mechanizmów wyważania.

**Czerwono-czarne** – to rodzaj samoorganizującego się binarnego drzewa poszukiwań o wysokości maks. 2 log (n+1), stosowanego np. do implementacji tablic asocjacyjnych.

**AVL** – to **zrównoważone binarne drzewo poszukiwań** (BST). Każdy węzeł drzewa AVL ma przypisaną wartość -1, 0 lub +1 wyrażającą różnicę w wysokości lewego i prawego poddrzewa. Jeśli operacja na drzewie miałaby doprowadzić tą różnicę do wartości niedopuszczalnej (czyli < -1 lub > +1), w drzewie wykonywana jest specjalna operacja rotacji, przywracająca zrównoważenie.

**B-drzewa** – to uporządkowane drzewa zawierające węzły mogące przechowywać k uporządkowanych kluczy, spośród których każdy jest medianą dla wartości kluczy korzeni poddrzew węzła tego klucza, stosowane np. do indeksacji atrybutów w tabelach bazodanowych i dostępu do nich w czasie O(log n).

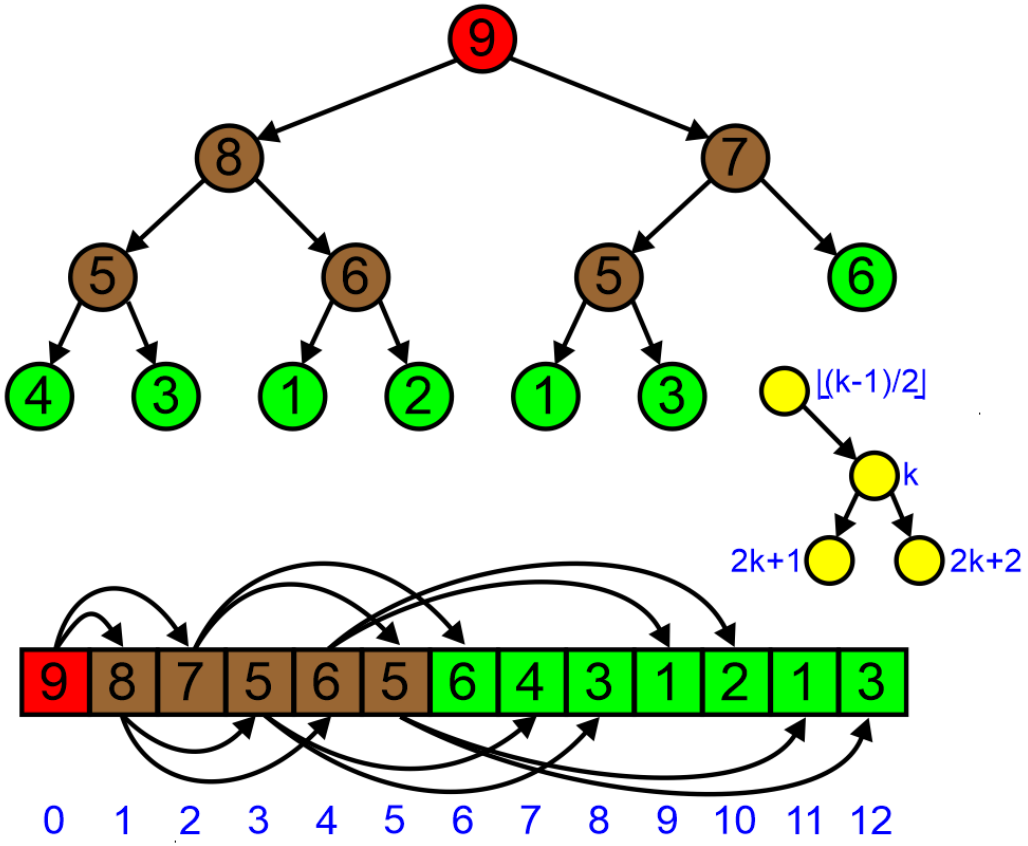
**B+drzewa** – to zmodyfikowane B-drzewa, które przechowują wszystkie obiekty w liściach, duplikując niektóre wartości kluczy w węzłach służących do szybkiego wyszukiwania obiektów w czasie logarytmicznym. Dodatkowo obiekty w liściach są łączone w porządku rosnącym, co tworzy graf.

# DRZEWA REGULARNE

**Drzewa regularne** można w łatwy sposób reprezentować w **tablicy**.

Jeśli drzewa te są dodatkowo **zupełne**, wtedy reprezentacja ta jest **optymalnie oszczędna** z punktu widzenia wykorzystania pamięci, gdyż taka tablica jest spójnie wypełniona od strony lewej:

Przykład reprezentacji w tablicy drzewa binarnego oraz zależność umożliwiająca wyznaczenie indeksów dzieci oraz rodzica dla węzła o indeksie k:




Dla drzew regularnych o większej ilości dzieci zależność tą odpowiednio modyfikujemy:

3k+1, 3k+2, 3k+3 i $\lfloor (k-1)/3 \rfloor$

4k+1, 4k+2 , 4k+3 , 4k+4 i $\lfloor (k-1)/4 \rfloor$ **itp.**

# KOPIEC

**Kopiec** to drzewo binarne, w którego węzłach znajdują się elementy reprezentowanego multizbioru, które spełniają **warunek kopca**.

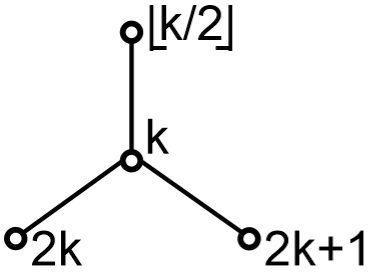
**Warunek kopca** mówi, iż jeśli węzeł y jest następnikiem węzła x, to element w węźle y jest nie większy niż element w węźle x.

Drzewo ma **uporządkowanie kopcowe**, gdy wszystkie jego elementy zachowują porządek kopcowy, tzn. elementy na ścieżkach w drzewie od korzenia do liścia uporządkowane są w sposób nierosnący.

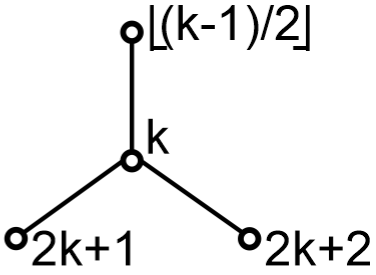
**Kopiec zupełny** to zupełne drzewo binarne o uporządkowaniu kopcowym, czyli takie, którego wszystkie poziomy są wypełnione całkowicie za wyjątkiem co najwyżej ostatniego, który jest spójnie wypełniony od strony lewej.

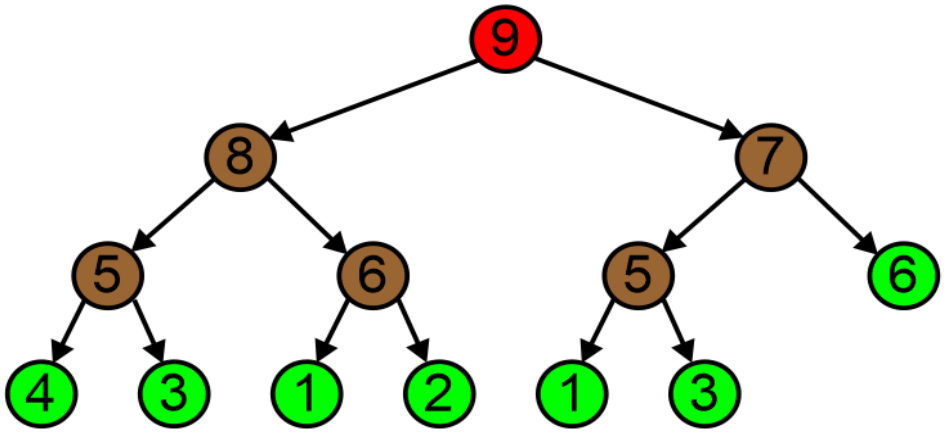
Każde drzewo binarne można w łatwy sposób reprezentować w tablicy, a indeksy określone są przy pomocy następującej zależności:

PRZY INDEKSACJI OD 1




PRZY INDEKSACJI OD 0



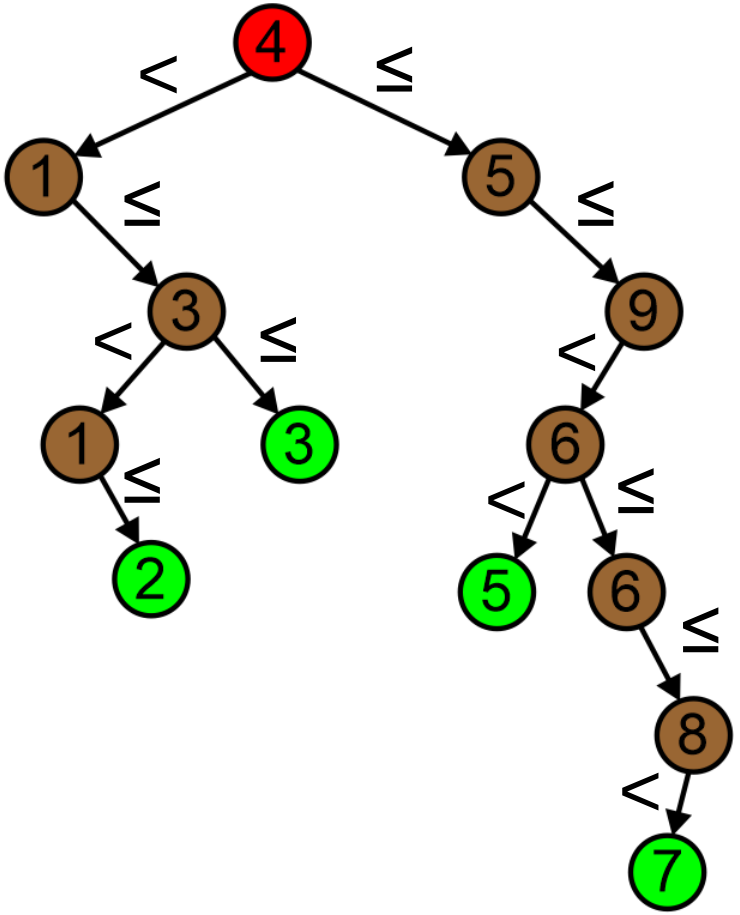
# BINARNE DRZEWA POSZUKIWAŃ

**Binarne drzewo poszukiwań (BST – binary search tree)** to binarne drzewo reprezentujące elementy multizbioru w taki sposób, iż każdy wierzchołek ma po lewej elementy mniejsze, a po prawej większe od reprezentowanej wartości klucza przez ten wierzchołek (https://www.cs.usfca.edu/~galles/visualization/BST.html).

Odnajdywanie miejsca wstawienia nowego elementu polega na przechodzeniu od korzenia po węzłach drzewa w taki sposób, iż jeśli wartość jest mniejsza, wtedy idzie się w lewo, a jeśli większa lub równa w prawo.

Przechodzenie po drzewie jest dokonywane tak długo, dopóki nie natrafimy na sytuację, gdy węzłowi brak węzła lub liścia w pożądaną stronę.
Tam jest dodawany nowy element.

Drzewa BST nie posiadają żadnego mechanizmu wyważania, więc w pesymistycznymi przypadku może zostać zbudowana lista – czyli trywialna postać drzewa.



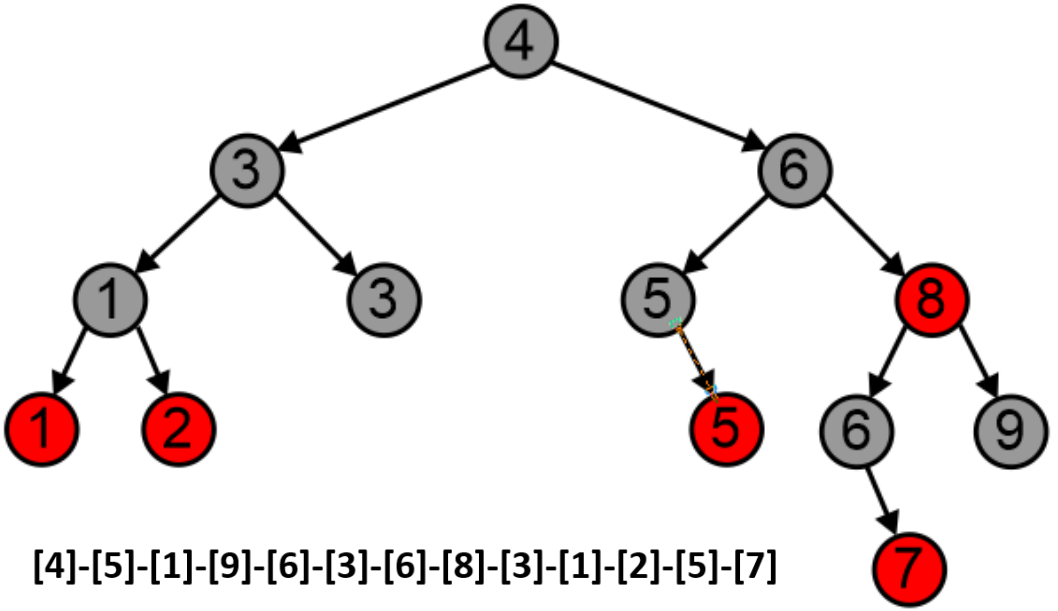
# DRZEWA CZERWONO-CZARNE

**Czerwono-czarne drzewo** to binarne drzewo poszukiwań, w którego węzłach znajdują się elementy reprezentowanego multizbioru, pokolorowane na kolor czerwony lub czarny (https://www.cs.usfca.edu/~galles/visualization/RedBlack.html), przy czym:

- Korzeń drzewa jest zawsze czarny.
- Dzieci czerwonego rodzica muszą być czarni.
- Każda droga prowadząca z ustalonego węzła do dowolnego liścia liczy tyle samo czarnych.

Po odnalezieniu miejsca algorytmem poszukiwań binarnych i wstawienia nowego elementu, zostaje on pokolorowany na czerwono, a następnie jeśli nie są spełnione w/w warunki, dokonywane są operacje rotacji w lewo lub w prawo oraz odpowiednio zmieniany jest kolor.

Drzewa te są jednak mniej efektywne niż drzewa AVL oraz B-drzewa, więc szczegółowy algorytm działania nie będzie rozważany.

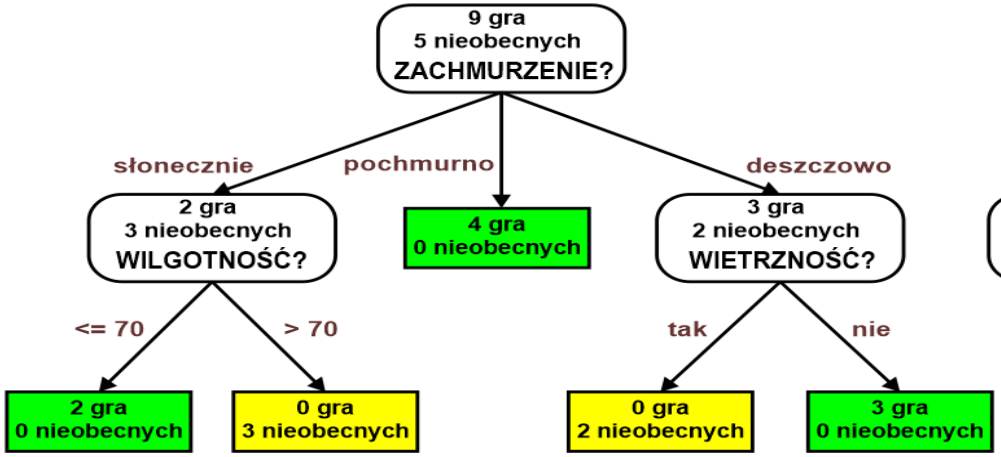



[4]-[5]-[1]-[9]-[6]-[3]-[6]-[8]-[3]-[1]-[2]-[5]-[7]

# DRZEWA I DIAGRAMY DECYZYJNE

**Drzewa decyzyjne** to specjalny rodzaj drzew, który w węzłach drzewa podejmuje decyzje na podstawie zdefiniowanych warunków i dostępnych opcji wyboru:

## DRZEWO DECYZYJNE




## DIAGRAM DECYZYJNY

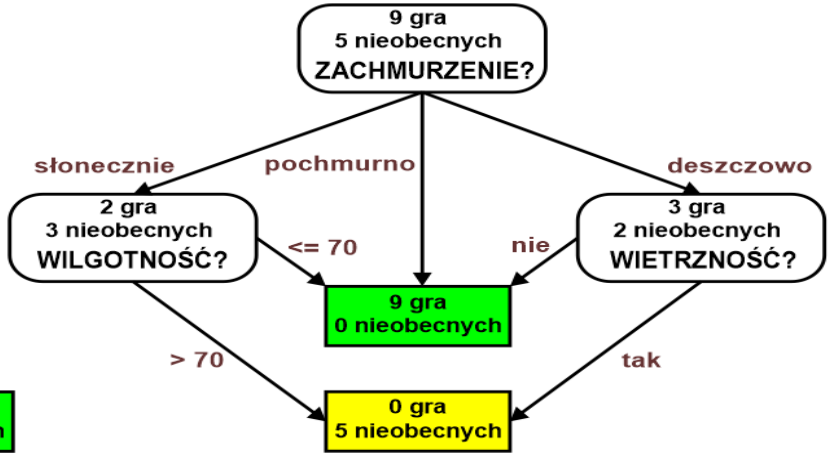



**Diagram decyzyjny** powstaje poprzez redukcję drzewa decyzyjnego na skutek:

- łączenia (agregacji) liści o tych samych wartościach,
- następnie poprzez redukcję węzłów, których wszystkie połączenia prowadzą do tego samego następnika.

**Drzewo decyzyjne** jest już **grafem**, gdyż zawiera alternatywne drogi pomiędzy węzłami oraz cykle, lecz jest oszczędniejszy niż analogiczne drzewo decyzyjne.

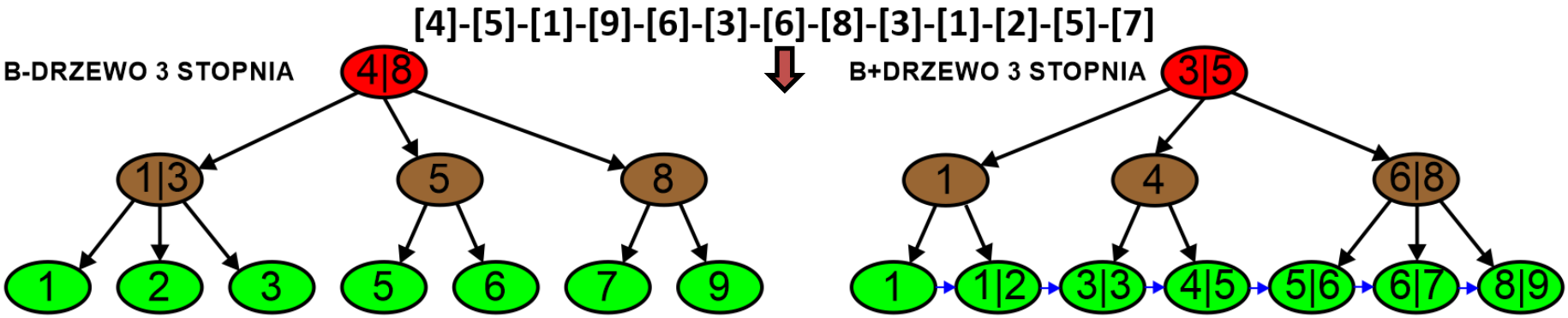
Jeśli każdy węzeł drzewa decyzyjnego posiada dokładnie dwa następniki, wtedy w wyniku takiego przekształcenia otrzymujemy **binarny diagram decyzyjny**.

# B-DRZEWA i B+DRZEWA

**B-drzewa i B+drzewa stopnia k** to specjalne k-regularne drzewa pozwalające reprezentować klucze obiektów w uporządkowany sposób, zapewniając w czasie logarytmicznym dodawanie obiektów i ich usuwanie, ponieważ drzewa te posiadają mechanizm automatycznego wyważania węzłów, nie kluczy. Drzewa te mogą przechowywać k-1 kluczy/obiektów w jednym węźle oraz posiadać k dzieci (k ≥ 3). Są to drzewa samoorganizujące się, a więc wykonujące operacje podziału i łączenia, przesuwania i tworzenia i usuwania wierzchołków, które przywracają odpowiednią strukturę drzewa oraz porządek po dodaniu lub usunięciu obiektu o określonym kluczu do i z tych drzew.

**B-drzewa 3 stopnia** są rozszerzeniem idei binarnych drzew poszukiwań, gdyż każde poddrzewo zawiera elementy odpowiednio mniejsze, pomiędzy pewnymi wartościami lub większe od wartości kluczy przechowywanych w poszczególnych węzłach. Jest to struktura bardzo efektywna, gdyż zapewnia dodawanie, usuwanie i odnajdywanie dowolnego elementu w czasie O($log_k n$).

**B+drzewa** są nieco redundantne, bo przechowują duplikaty niektórych kluczy w węzłach decyzyjnych, a same obiekty przechowują w liściach na tym samym poziomie. Ponadto wzbogacają strukturę drzewa o krawędzie pomiędzy sąsiednimi liśćmi, które zapewniają bezpośredni dostęp do kolejnych posortowanych elementów. Program w Pythonie: http://python-wiki.appspot.com/?p=5846034605408256



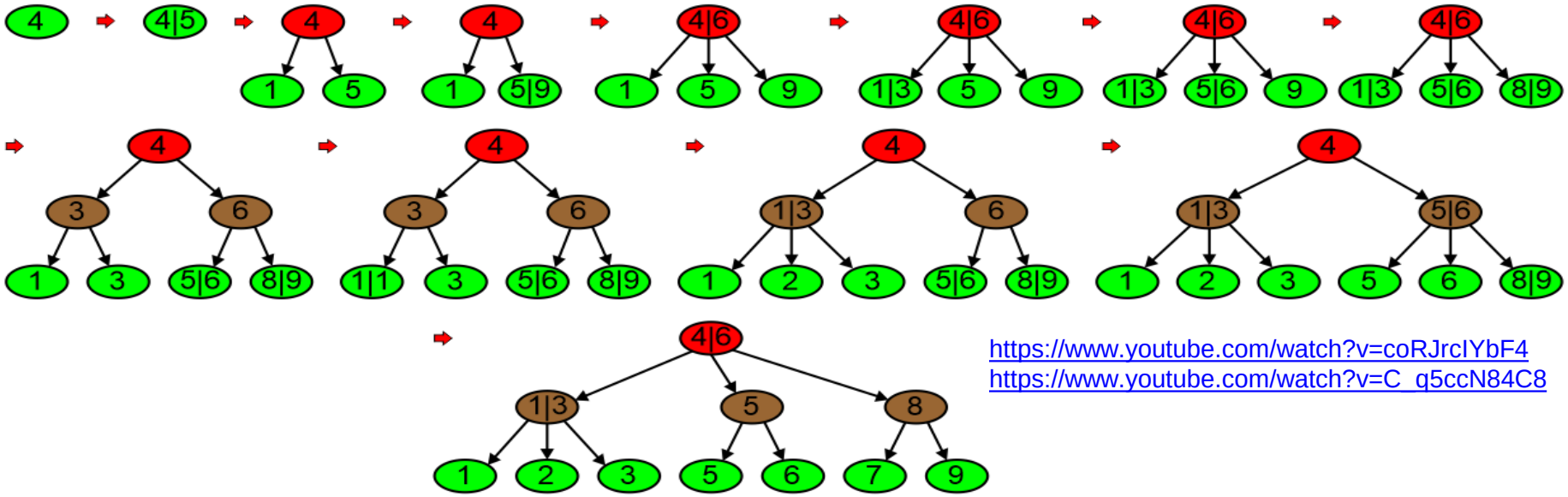
# TWORZENIE B-DRZEWA 3. STOPNIA

**Tworzenie B-drzewa** dla ciągu kluczy jest złożonym procesem wymagającym wykonywanie operacji przywracających takie uporządkowanie drzewa (https://www.cs.usfca.edu/~galles/visualization/BTree.html):

Dodawanie nowego elementu do B-drzewa składa się z kilku kroków:

1. Idź od korzenia do odpowiedniego liścia w drzewie poruszając się po węzłach następująco:
   - w lewo po gałęziach, jeśli klucz jest mniejszy lub równy od lewej wartości klucza w węźle,
   - w prawo po gałęziach, jeśli klucz jest silnie większy od prawej wartości klucza w węźle,
   - w kierunku środkowej gałęzi, jeśli klucz jest silnie większy od lewej wartości i mniejszy lub równy od prawej wartości klucza.
2. Dodaj element do liścia w sposób uporządkowany, jeśli nie przechowuje jeszcze dwóch wartości.
3. Jeśli już zawiera 2 wartości, podziel go na dwa węzły, a środkową wartość przekaż do rodzica, a jeśli nie istnieje, utwórz go. Rodzic będzie wskazywał te dwa węzły, jeśli nie zawierał dwóch wartości.
4. Jeśli rodzic zawierał dwie wartości, również się dzieli i środkową wartość przekazuje do swojego rodzica, a jeśli takowy nie istnieje, utworzy go. Jeśli istnieje i jest pełny, rekurencyjnie do góry powtarza ten krok.

[4]-[5]-[1]-[9]-[6]-[3]-[6]-[8]-[3]-[1]-[2]-[5]-[7]



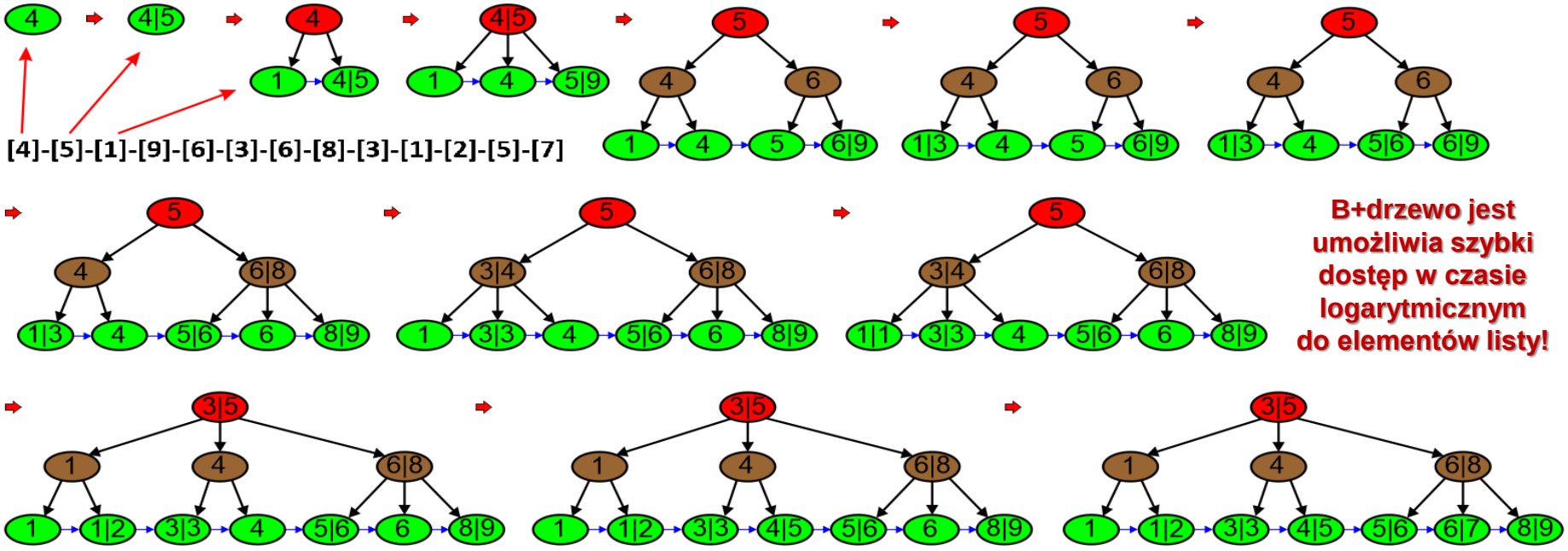

https://www.youtube.com/watch?v=coRJrcIYbF4
https://www.youtube.com/watch?v=C_q5ccN84C8

# TWORZENIE B+DRZEWA 3. STOPNIA

**Tworzenie B+drzewa** dla ciągu kluczy jest złożonym procesem wymagającym wykonywanie operacji przywracających takie uporządkowanie drzewa (https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html):

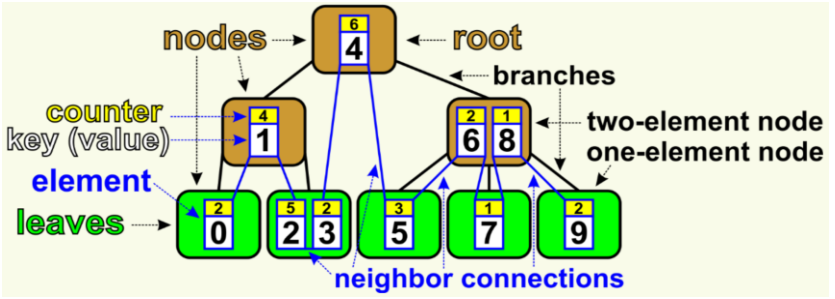
Dodawanie nowego elementu do B+drzewa składa się z kilku kroków: https://www.youtube.com/watch?v=_nY8yR6iqx4

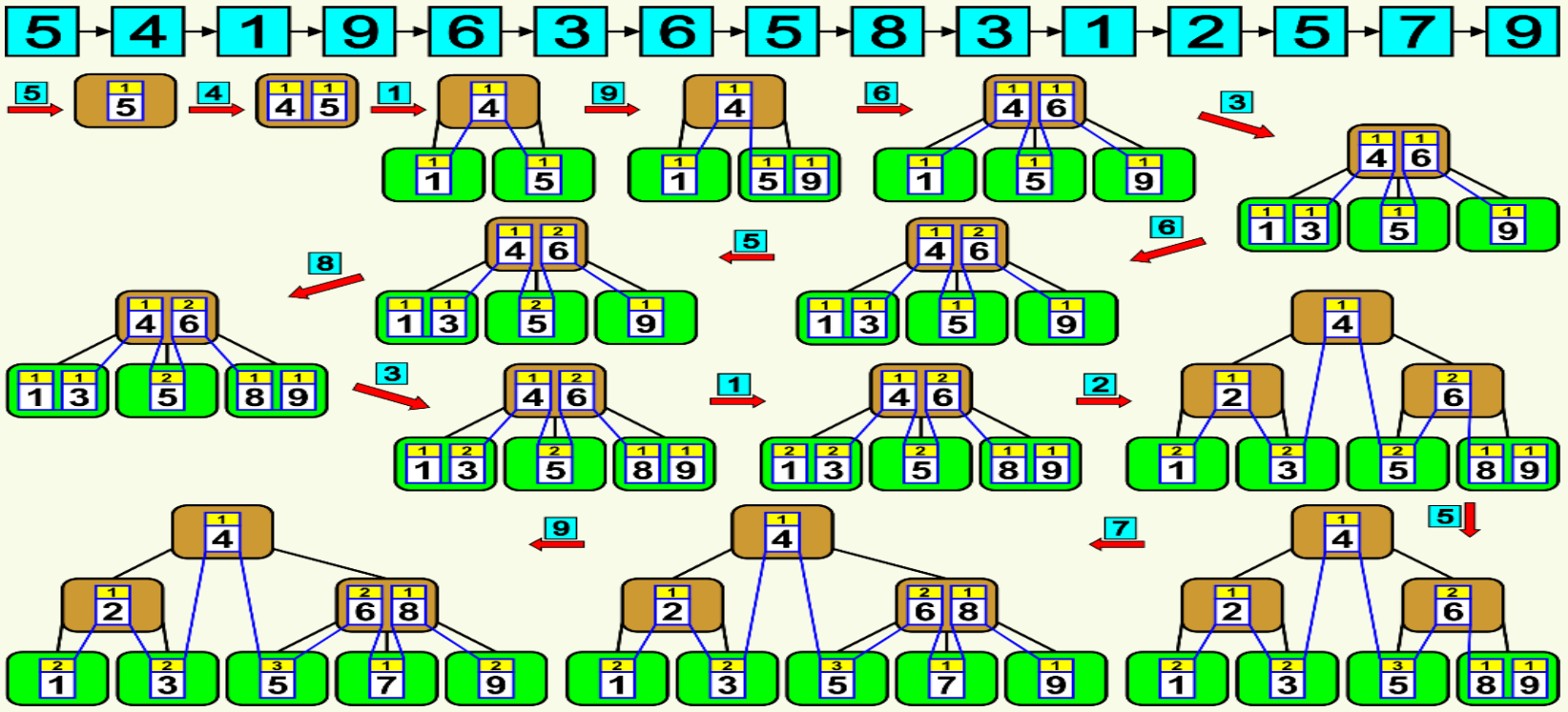
1. Idź od korzenia do odpowiedniego liścia w drzewie poruszając się po węzłach następująco:
   - w lewo po gałęziach, jeśli klucz jest mniejszy lub równy od lewej wartości klucza w węźle,
   - w prawo po gałęziach, jeśli klucz jest silnie większy od prawej wartości klucza w węźle,
   - w kierunku środkowej gałęzi, jeśli klucz jest silnie większy od lewej wartości i mniejszy lub równy od prawej wartości klucza.
2. Dodaj element do liścia w sposób uporządkowany, jeśli nie przechowuje jeszcze dwóch wartości.
3. Jeśli już zawiera 2 wartości, podziel go na dwa węzły, lewy zawierający mniejszą wartość od środkowego, a prawy zawierający środkowy i większy element. Połącz krawędzią te dwa węzły. Ponadto środkową wartość klucza skopiuj do rodzica, a jeśli nie istnieje utwórz go. Rodzic wskazuje te dwa węzły, jeśli nie zawierał dwóch wartości.
4. Jeśli rodzic zawierał dwie wartości, również się dzieli i środkową wartość przekazuje do swojego rodzica, a jeśli takowy nie istnieje, tworzy go. Jeśli istnieje i jest pełny, rekurencyjnie do góry powtarza ten krok.



# AVB+DRZEWA 3. STOPNIA

**AVB+drzewa** łączą koncepcje binarnych drzew wyszukiwania (BST), B-drzew, B+drzew oraz sortujących list podwójnie wiązanych. Są samo-wyważającą i samo-sortującą się złożoną strukturą (grafem), który zapewnia szybki dostęp do wszystkich kluczy (pesymistycznie logarytmiczny w przypadku braku duplikatów, optymistycznie stały) przechowywanych w drzewie oraz stały dostęp do kluczy poprzednich i następnych według wybranej metryki. Dodawanie i usuwanie elementów do AVB+drzew jest procesem złożonym składającym się z kilku kroków (o złożoności maksymalnie logarytmicznej), gdyż te operacje wymagają czasami przywrócenie doskonałego wyważenia drzewa: http://home.agh.edu.pl/~horzyk/lectures/ci/CI-AGDSandAVB+trees.pdf



# BIBLIOGRAFIA I LITERATURA UZUPEŁNIAJĄCA


1. L. Banachowski, K. Diks, W. Rytter: „Algorytmy i struktury danych”, WNT, Warszawa, 2001.
2. M. Sysło: „Elementy Informatyki”.
3. A. Szepietowski: „Podstawy Informatyki”.
4. R. Tadeusiewicz, P. Moszner, A. Szydełko: „Teoretyczne podstawy informatyki”.
5. W. M. Turski: „Propedeutyka informatyki”.
6. N. Wirth: „Wstęp do programowania systematycznego”.
7. N. Wirth: „ALGORYTMY + STRUKTURY DANYCH = PROGRAMY”.
8. Binarne drzewa poszukiwań: https://pl.wikipedia.org/wiki/Binarne_drzewo_poszukiwań
9. Złożoność obliczeniowa: http://xion.org.pl/files/texts/mgt/pdf/M_B.pdf
10. Sortowanie w Pythonie: https://docs.python.org/2/howto/sorting.html?highlight=complexity
11. Symulator tworzenia B-drzew: https://www.cs.usfca.edu/~galles/visualization/BTree.html
12. Symulator tworzenia B+drzew: https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html
13. Symulatory: https://www.cs.usfca.edu/~galles/visualization/Algorithms.html
14. A. Horzyk, Associative Graph Data Structures with an Efficient Access via AVB+trees, In: 2018 11th International Conference on Human System Interaction (HSI), 2018, IEEE Xplore, pp. 169 - 175, DOI: 10.1109/HSI.2018.8430973
15. AVB+drzewa algorytmy: http://home.agh.edu.pl/~horzyk/lectures/ci/CI-AGDSandAVB+trees.pdf