# Lista

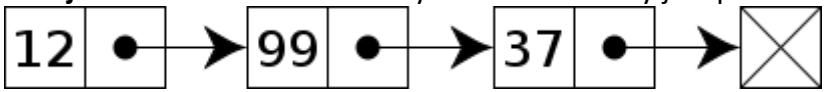
Lista - rodzaj kontenera - dynamiczna struktura danych, używana w informatyce. Składa się z podstruktur wskazujących na następniki i/lub poprzedniki.

Typowa lista jest łączona **jednostronnie** - komórki zawierają tylko odnośnik do kolejnej komórki. Innym przypadkiem jest lista **dwustronna**, gdzie komórki zawierają także odnośnik do poprzednika.

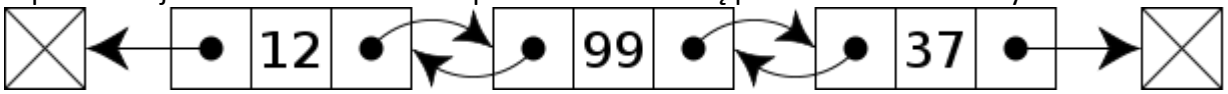
Popularna jest także lista zwana **słownikową**, która zazwyczaj jest wariacją listy jednostronnej. Z reguły stosuje się ją tam, gdzie elementy listy zawierają kilka pól z danymi, a kolejny element może rozszerzać pojęcie (definicję poprzedniego). Przykładem jest prosty translator tekstu, zrealizowany jako lista, gdzie każdy z elementów zawiera dane wyraz i definicja wyrazu - może się okazać, że definicja danego wyrazu ma swoje rozwinięcie (definicję) w pewnym innym elemencie, wówczas tam kieruje się dodatkowy łącznik.

Wgłębiając się w szczegóły implementacji listy za pomocą wskaźników można wyróżnić następujące rodzaje list:

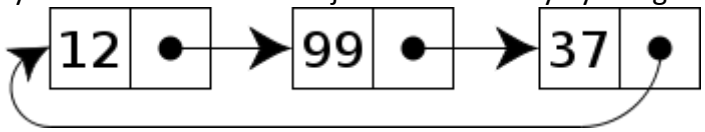
- **lista jednokierunkowa** - w każdym elemencie listy jest przechowywane odniesienie tylko do jednego sąsiada (następnika lub poprzednika).




- **lista dwukierunkowa** - w każdym elemencie listy jest przechowywane odniesienie zarówno do następnika jak i poprzednika elementu w liście. Taka reprezentacja umożliwia swobodne przemieszczanie się po liście w obie strony.




- **lista cykliczna** - następnikiem ostatniego elementu jest pierwszy element, a poprzednikiem pierwszego ostatni. Po liście można więc przemieszczać się cyklicznie. Nie ma w takiej liście charakterystycznego ogona (ani głowy), często rozpoznawanego po tym, że jego następnik jest pusty (NULL).



**SPOSÓB IMPLEMENTACJI**: tablicowa i **wskaźnikowa**.

## Wskaźnikowa

W tej implementacji każdy obiekt na liście musi (co nie było konieczne w wersji tablicowej) zawierać dodatkowy element: wskaźnik do innego obiektu tego typu. Wynika to z faktu, że to wskaźniki są podstawą nawigacji w tym typie listy, a dostęp do jej elementów jest możliwy wyłącznie przez wskaźnik.

**PODSTAWOWE METODY:**

**ADD - Dopisanie elementu** czyli wstawienie nowego elementu do listy (jeśli chcemy dbać o to by lista była posortowana to od razu przy wstawianiu zachowujemy odpowiedni porządek). Wstawianie polega na dopisaniu czyli utworzeniu i odpowiedniemu powiązaniu nowego elementu z już istniejącymi w liście. Podczas tej czynności element wstawiany jest w odpowiednim miejscu w liście – należy zawsze zadbać by nie pogubić już istniejących powiązań.

**DEL - Usunięcie elementu** jest odwrotne do wstawiania.

**VIEW – Wypisanie wszystkich elementów** polega na wypisaniu na ekranie wszystkich elementów listy w kolejności od pierwszego do ostatniego.

**METODY DODATKOWE:**

**COUNT – Ilość elementów** informuje ile elementów znajduje się na liście.

**GET – Wyświetlenie konkretnego (o zadanym numerze) elementu** z listy – użytkownik podaje, że chce zobaczyć element 5 i o ile taki istnieje na liście zostaje wyświetlony.

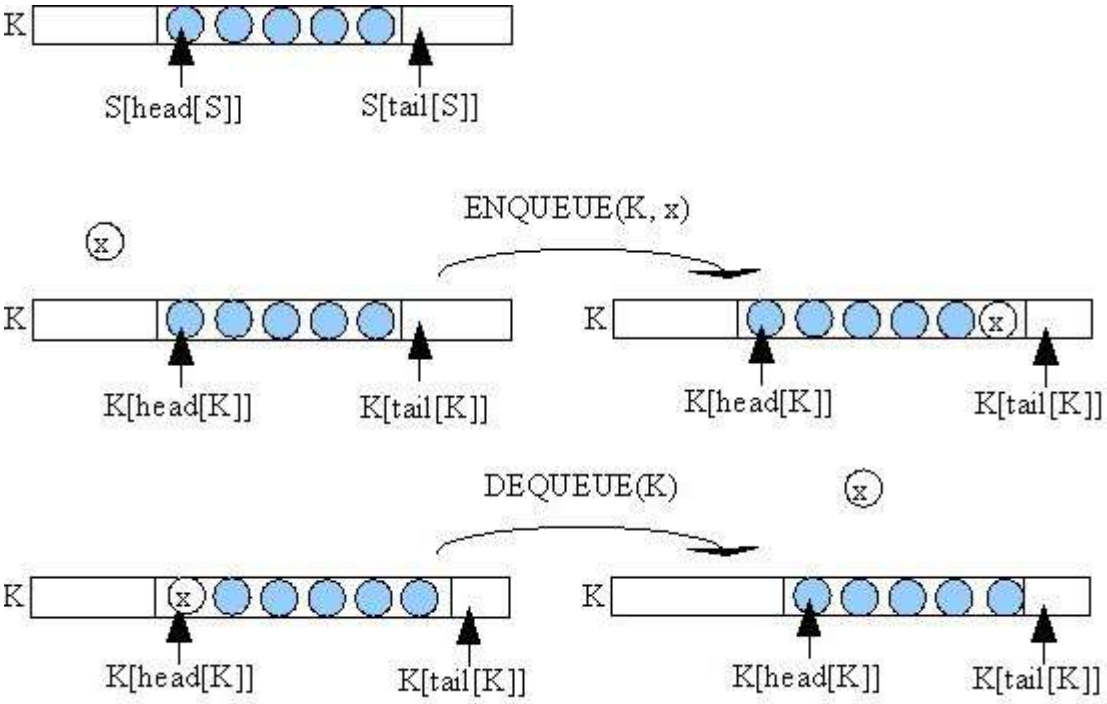
# Kolejka

Kolejka (ang. **queue**) – liniowa struktura danych, w której nowe dane dopisywane są na końcu kolejki, a z początku kolejki pobierane są dane do dalszego przetwarzania (bufor typu **FIFO**, *First In, First Out*; pierwszy na wejściu, pierwszy na wyjściu).

Specjalną modyfikacją kolejki jest kolejka priorytetowa – każda ze znajdujących się w niej danych dodatkowo ma przypisany priorytet, który modyfikuje kolejność późniejszego wykonania. Oznacza to, że pierwsze na wyjściu niekoniecznie pojawią się te dane, które w kolejce oczekują najdłużej, lecz te o największym priorytecie.

**Podstawowe metody:**
**ENQUEUE –** powoduje umieszczenie elementu na końcu (TAIL) kolejki;
**DEQUEUE –** powoduje zdjęcie elementu z początku (HEAD) kolejki;

**IDEA:**



# Stos

**Stos** (ang. **Stack**) – liniowa struktura danych, w której dane dokładane są na wierzch stosu i z wierzchołka stosu są pobierane (bufor typu **LIFO**, *Last In, First Out*; ostatni na wejściu, pierwszy na wyjściu). Ideę stosu danych można zilustrować jako stos położonych jedna na drugiej książek – nowy egzemplarz kładzie się na wierzch stosu i z wierzchu stosu zdejmuje się kolejne egzemplarze. Elementy stosu poniżej wierzchołka stosu można wyłącznie obejrzeć, aby je ściągnąć, trzeba najpierw po kolei ściągnąć to, co jest nad nimi.

Stos jest stosowany w systemach komputerowych na wszystkich poziomach funkcjonowania systemów informatycznych, używane są przez procesory do chwilowego zapamiętywania rejestrów procesora, do przechowywania zmiennych lokalnych, a także w programowaniu wysokopoziomowym.

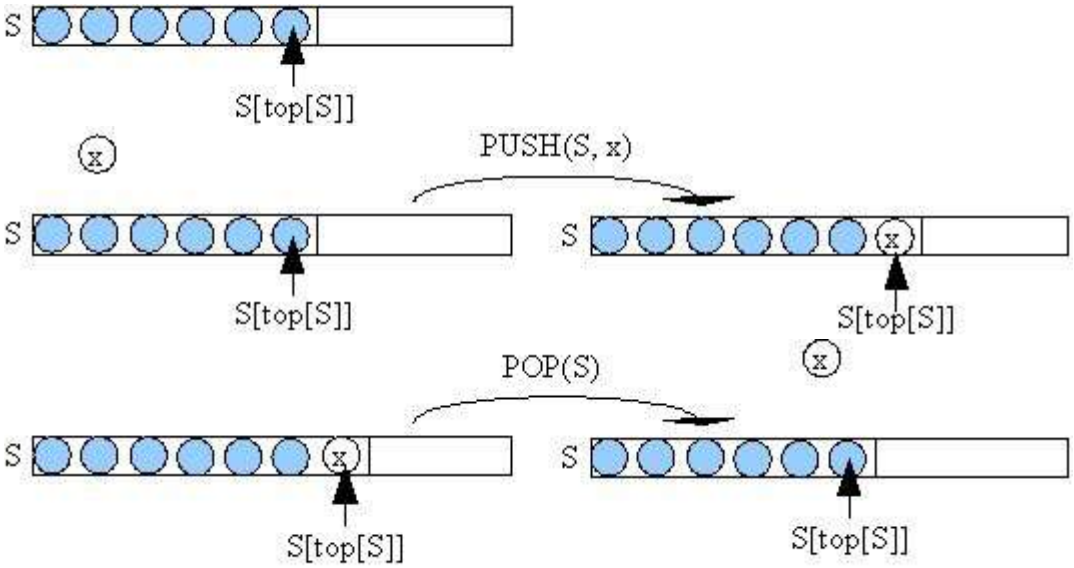
Przeciwieństwem stosu jest kolejka, bufor typu FIFO (ang. First In, First Out; pierwszy na wejściu, pierwszy na wyjściu), w którym dane obsługiwane są w takiej kolejności, w jakiej zostały dostarczone (jak w kolejce do kasy).

**PODSTAWOWE METODY:**

**Push** - Powoduje umieszczenie wartości na szczycie stosu.

**Pop** -Powoduje zdjęcie wartości ze stosu.

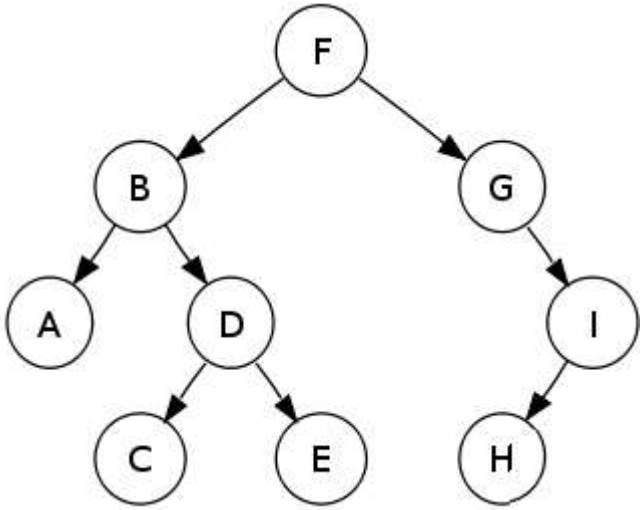
**IDEA:**



# Drzewa

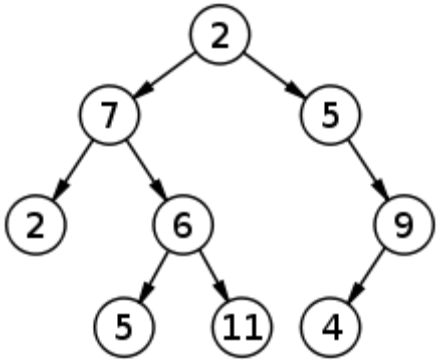
Drzewo - w informatyce to struktura danych reprezentująca drzewo matematyczne. W naturalny sposób reprezentuje hierarchię danych (obiektów fizycznych i abstrakcyjnych, pojęć, itp.) jest więc stosowane głównie do tego celu. Drzewa ułatwiają i przyspieszają wyszukiwanie, a także pozwalają w łatwy sposób operować na posortowanych danych.

**Budowa drzewa:**
Drzewa składają się z **wierzchołków** (węzłów) oraz łączących je **krawędzi**.
Jeśli drzewo nie jest puste, tzn. liczba wierzchołków jest większa od zera, jeden z nich jest wyróżniony i nazywany **korzeniem drzewa**; na rysunku jest oznaczony literą **F**.
Ciąg krawędzi łączących węzły nazywa się **ścieżką**. Istnieje dokładnie jedna ścieżka łącząca korzeń z wszystkimi pozostałymi wierzchołkami.
Liczba krawędzi w ścieżce od korzenia do węzła jest nazywana **długością** – liczba ta określa **poziom** węzła.
**Wysokością** drzewa jest największy poziom istniejący w drzewie. Np. korzeń znajduje się na 0. poziomie, węzły **A, D** i I na poziomie 2.; wysokość drzewa to 3.
Wszystkie wierzchołki połączone z danym wierzchołkiem, a leżące na następnym poziomie są nazywane **dziećmi** tego węzła (np. dziećmi wierzchołka **F** są **B** i **G**, natomiast wierzchołka **B: A i D**). Wierzchołek może mieć dowolną liczbę dzieci, jeśli nie ma ich wcale nazywany jest liściem. Liśćmi w przykładowym drzewie są **A, C, E, H**.
Wierzchołek jest **rodzicem** dla każdego swojego **dziecka**. Każdy węzeł ma dokładnie jednego rodzica, wyjątkiem jest korzeń drzewa, który nie ma rodzica.




**Drzewo binarne – drzewo w którym stopień każdego wierzchołka jest nie większy od 3.**

# Binarne drzewo poszukiwań (bst)



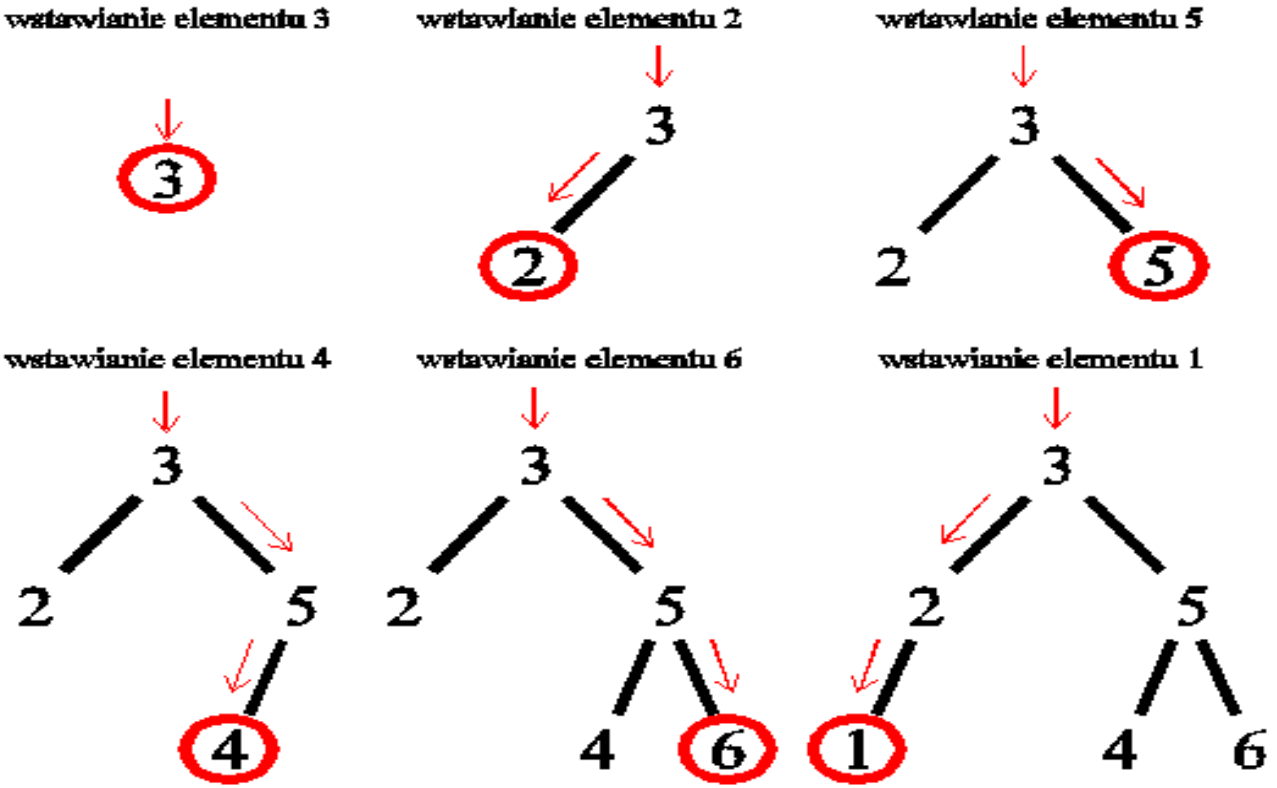

**Binarne drzewo poszukiwań** (ang. Binary Search Tree (BST)) – **dynamiczna** struktura danych będąca **drzewem binarnym**, w którym lewe poddrzewo każdego węzła zawiera wyłącznie elementy o kluczach nie większych niż klucz węzła, a prawe poddrzewo zawiera wyłącznie elementy o kluczach nie mniejszych niż klucz węzła.

**Takie drzewo można zrealizować za pomocą struktury danych** z dowiązaniami, w której **każdy węzeł jest obiektem (rekordem).**
Oprócz pola **Key**, przechowującego klucz (**wartość**) węzła, każdy węzeł zawiera pola **Left i Right**, wskazujące, odpowiednio, **lewego i prawego syna**.
Niektóre algorytmy (np. wyszukiwanie poprzednika i następnika) potrzebują również pola Parent, wskazującego ojca węzła.

Dla pełnego drzewa BST o n węzłach pesymistyczny koszt każdej z podstawowych operacji wynosi O(logn).
Jednak w skrajnym przypadku, gdy drzewo składa się z jednej gałęzi koszt ten wzrasta do O(n). W literaturze często też podaje się koszt wykonania operacji w uzależnieniu od wysokości drzewa h - wynosi on O(h). Przechodząc drzewo metodą in-order, uzyskuje się ciąg wartości posortowanych niemalejąco[1].

## Operacje:

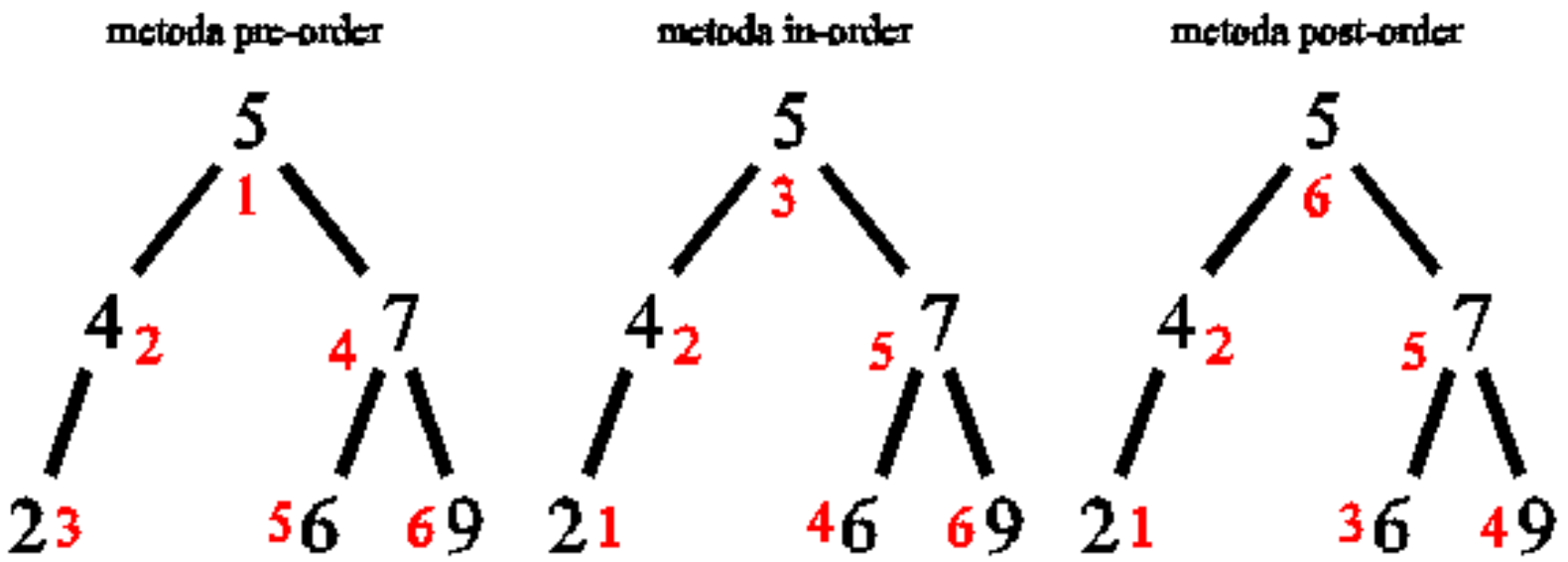
**Wstawianie elementów:**



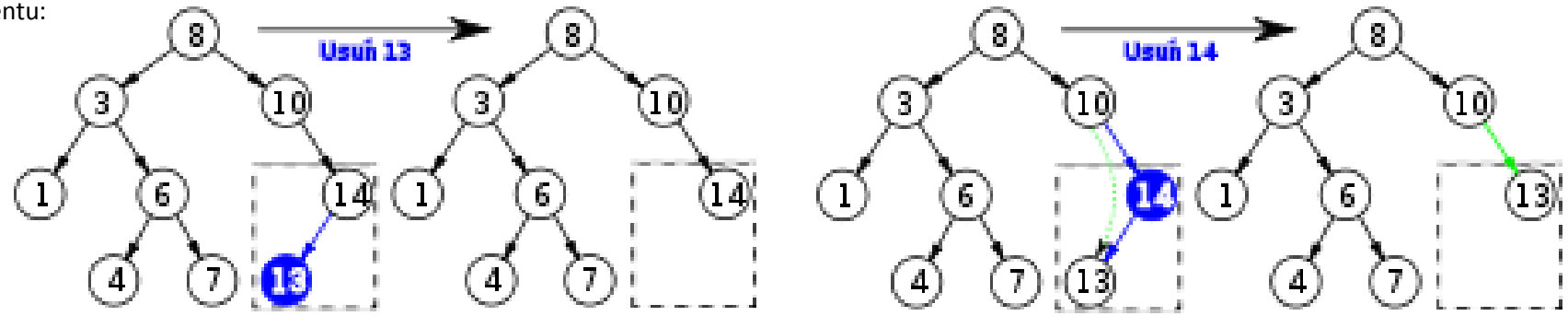
## Przeszukiwanie drzewa:

Przeszukiwanie drzewa jest niczym innym jak podróżą po jego wierzchołkach w odpowiedniej kolejności. Czasem nazywa się to także numerowaniem drzewa, jako że najczęściej przydziela się wierzchołkom numery zgodnie z kolejnością przechodzenia.
Są trzy metody przeszukiwania drzewa:

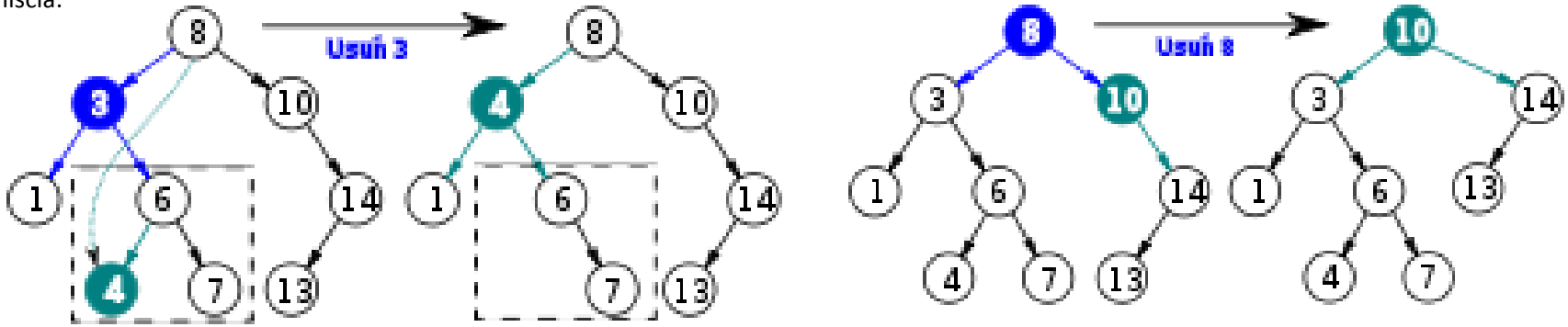
1. **Pre-order** (wzdłużna): korzeń, lewe poddrzewo, prawe poddrzewo.
2. **In-order** (poprzeczna): lewe poddrzewo, korzeń, prawe poddrzewo.
3. **Post-order** (wsteczne): lewe poddrzewo, prawe poddrzewo, korzeń.



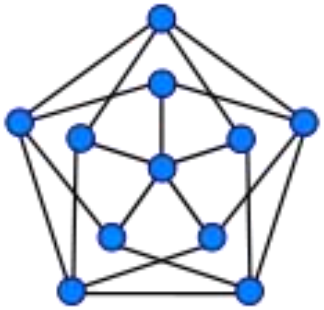
Usuwanie elementu:




Usuwanie liścia:



# Grafy



**Graf** – w uproszczeniu – **zbiór wierzchołków**, które mogą być **połączone krawędziami**, w taki sposób, że **każda krawędź kończy się i zaczyna w którymś z wierzchołków** (ilustracja po prawej stronie).

Wierzchołki grafu zwykle są numerowane i czasem stanowią reprezentację jakichś obiektów, natomiast krawędzie mogą wówczas obrazować relacje między takimi obiektami. Krawędzie mogą mieć wyznaczony kierunek, a graf zawierający takie krawędzie jest grafem skierowanym. Krawędź może posiadać także wagę, to znaczy przypisaną liczbę, która określa na przykład odległość między wierzchołkami (jeśli na przykład graf jest reprezentacją połączeń między miastami). W grafie skierowanym wagi mogą być zależne od kierunku przechodzenia przez krawędź (np. jeśli graf reprezentuje trud poruszania się po jakimś terenie, to droga pod górkę będzie miała przypisaną większą wagę niż z górki).

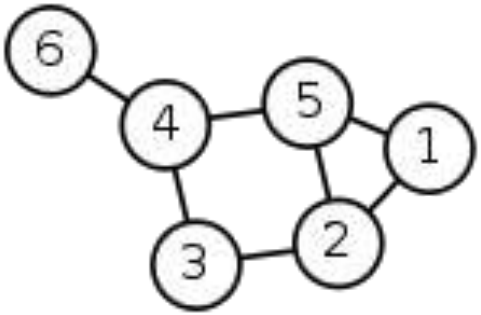
Wybrane **SPOSOBY REPREZENTACJI:**

**MACIERZ SĄSIEDZTWA**: Najprostszą ze struktur danych umożliwiających przedstawienie skomplikowanego grafu lub jego przechowywanie w pamięci komputera jest **macierz sąsiedztwa**, zawierająca dane na temat połączeń między wierzchołkami. Macierz jest rozmiaru $|V(G)|$ na $|V(G)|$, wyraz leżący z *i-tego* wiersza i *j-tej* kolumny zawiera wartość będącą liczbą krawędzi łączących *i-ty* i *j-ty* wierzchołek. Sposób ten pozwala na reprezentację zarówno grafów prostych, jak i grafów zawierających krawędzie wielokrotne oraz pętle własne. W przypadku grafów prostych wyrazami w macierzy będą wartości boole'owskie – *jest* krawędź, bądź *nie ma* krawędzi).

Graf odpowiadający tej macierzy

Macierz sąsiedztwa dla rozważanego wcześniej grafu nieskierowanego:

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$



**LISTA SĄSIEDZTWA (tutaj przydadzą nam się wskaźniki)**: Drugą popularną reprezentacją grafu są tzw. listy sąsiedztwa – dla każdego wierzchołka zapamiętywana jest lista sąsiadujących z nim wierzchołków, np.:

$v_1 \longrightarrow v_2, v_5$
$v_2 \longrightarrow v_1, v_5, v_3$
$v_3 \longrightarrow v_2, v_4$
$v_4 \longrightarrow v_3, v_5, v_6$
$v_5 \longrightarrow v_1, v_2, v_4$
$v_6 \longrightarrow v_4$

W implementacji tej metody stosuje się listy jednokierunkowe oraz jednowymiarową tablicę wskaźników o rozmiarze | *V*(*G*) | , gdzie i-ty element tablicy jest wskaźnikiem do początku listy przechowującej sąsiadów i-tego wierzchołka.

W odróżnieniu od macierzy sąsiedztwa, lista sąsiedztwa wymaga ilości pamięci proporcjonalnej do liczby krawędzi, także przejrzenie całego zbioru krawędzi jest proporcjonalne do jego rozmiaru. W stosunku do macierzy sąsiedztwa większą złożoność mają jednak operacje elementarne – sprawdzenie, czy $\{v_i, v_j\} \in E(G)$ wymaga czasu proporcjonalnego do mniejszego ze stopni wierzchołków, a np. usunięcie krawędzi – do większego z nich.

# ZADANIA:

ZAD1.Napisać implementacje listy prostej jednokierunkowej (realizującej wszystkie podstawowe i dodatkowe metody).
ZAD2.Napisać implementacje listy cyklicznej i dwukierunkowej realizującej podstawowe metody.
ZAD3.Napisać implementację stosu i kolejki realizujące podstawowe metody.