Wstęp do programowania

Drzewa

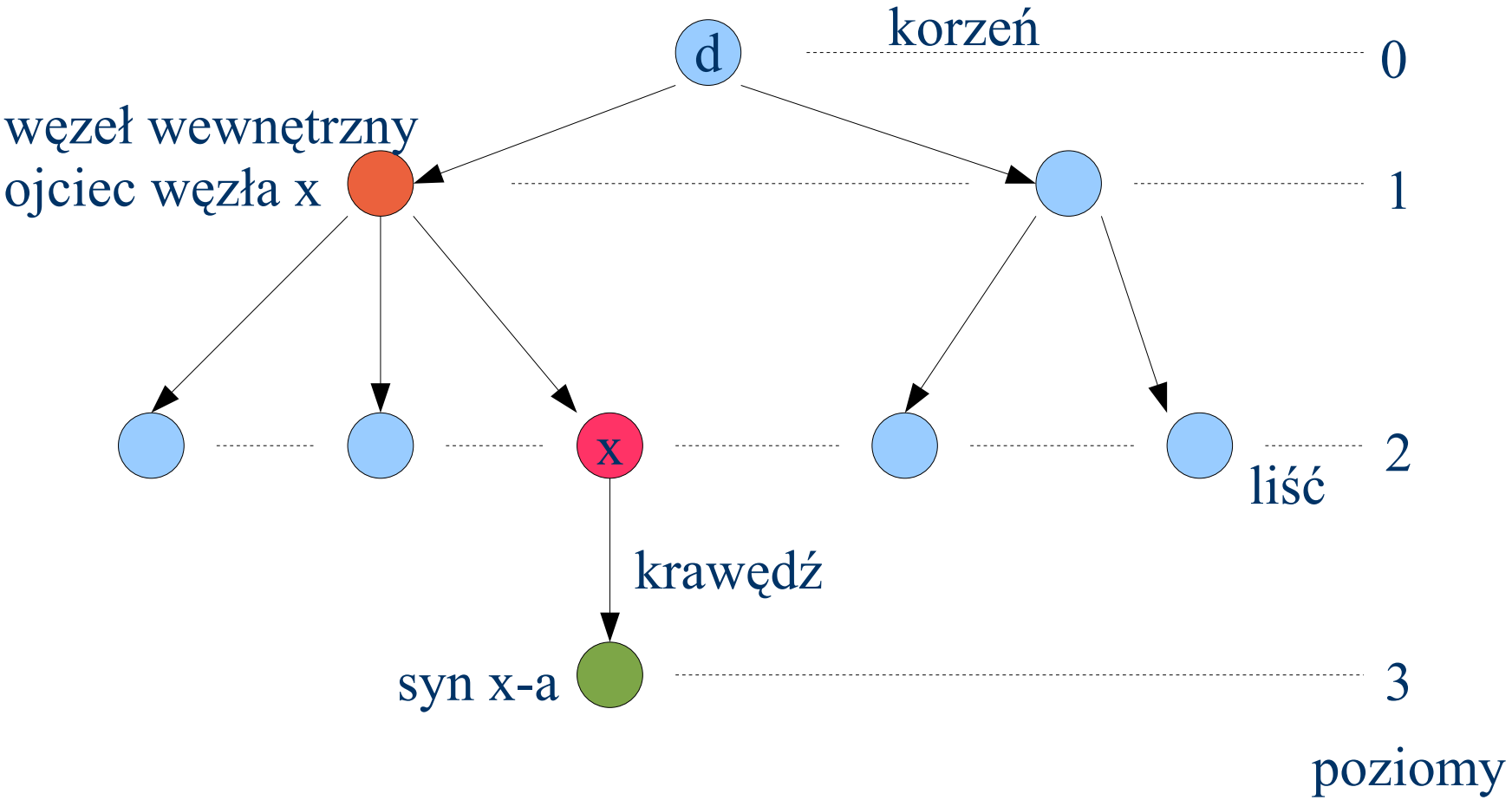
Drzewa

- Drzewa definiują matematycy, jako spójne nieskierowane grafy bez cykli.
- Równoważne określenia:
  - Spójne grafy o n wierzchołkach i n-1 krawędziach
  - Grafy, w których miedzy każdymi dwoma węzłami istnieje dokładnie jedna ścieżka
- Dla informatyków drzewa w zasadzie są takie same, ale zawsze dodatkowo wskazujemy węzeł początkowy, przez który „wchodzimy” na drzewo – korzeń drzewa. Ponadto wszystkie krawędzie w naturalny sposób są zorientowane od korzenia do liści.

Definicja drzewa formalnie

- Podobnie, jak z listami, zdefiniujemy zbiór D drzew o wartościach w zbiorze A rekurencyjnie, jako najmniejszy zbiór spełniający dwa warunki:
  - nil $\in$ D, czyli drzewo puste należy do D;
  - jeśli a $\in$ A oraz ciąg $D=D_1,...D_m$ składa się z drzew ze zbioru D, to para T=(a,D) jest drzewem ze zbioru D. Mówimy wtedy, że T jest ojcem dla każdego z drzew D_k, zaś te z kolei są jego synami.

Pojęcia związane z drzewami



Definicje

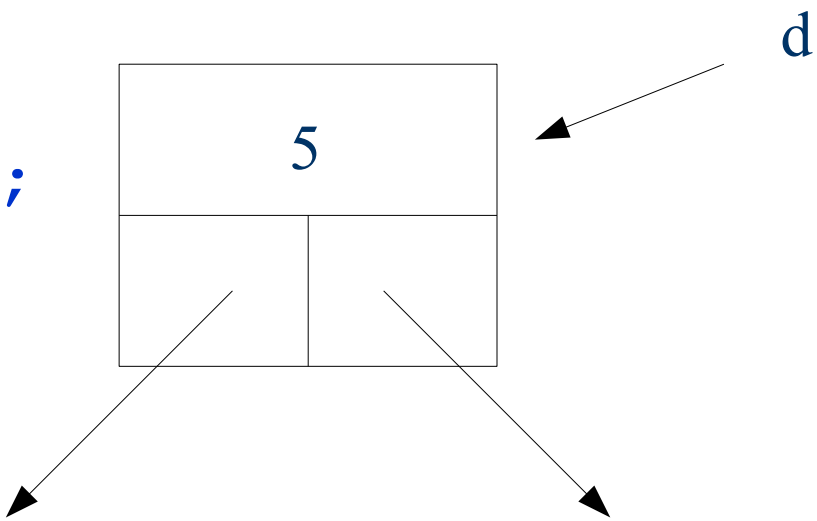
- Głębokość węzła – poziom, na którym się znajduje
- Wysokość drzewa – max z głębokości węzłów
- Stopień węzła – liczba synów
- Stopień drzewa – max z liczby synów
- Drzewo binarne – drzewo stopnia <=2
- Liść – węzeł stopnia 0
- Węzeł wewnętrzny – węzeł stopnia >0
- Przodek x – każdy węzeł na ścieżce od korzenia do x
- Potomek x – każdy węzeł którego przodkiem jest x

Drzewa nieuporządkowane

- Jeśli w definicji drzewa przyjmiemy, że kolejność synów nie ma znaczenia, czyli zamiast ciągu synów weźmiemy ich zbiór, to dostaniemy drzewo nieuporządkowane. Zdarzają sie też drzewa hybrydowe, w których niektóre węzły mają uporządkowanych synów, a inne nie. Nie będziemy się nimi zajmowali.

Drzewa binarne

```
type drzewo = ^węzeł;
     węzeł = record
               w : typ;
      lsyn,psyn : drzewo
     end;
var
  d:drzewo;
```



Wyszukiwanie wartości w drzewie

```
function jest(x:typ; d:drzewo):Boolean;
begin
 if d=nil then jest := false
 else if d^.w=x then jest := true
 else jest := jest(x,d^.lsyn) or
              jest(x,d^.psyn)
end;
{leniwie}
```

Wyszukiwanie wartości w drzewie

```
function jest(x:typ; d:drzewo):Boolean;
begin
 if d=nil then jest := false
 else if d^.w=x then jest := true
  else if jest(x,d^.lsyn) then
                          jest:=true
       else jest:= jest(x,d^.psyn)
end;
{nie leniwie}
```

Liczymy liczbę liści w drzewie

```
function LiczbaLisci(d: drzewo): integer;
var lewo, prawo: integer;
begin
   if d = nil then LiczbaLisci:=0
   else if d^.lsyn=nil and d^.psyn=nil then
                                   LiczbaLisci:=1
      else begin
         lewo := LiczbaLisci(d^.lsyn);
         prawo := LiczbaLisci(d^.psyn);
         LiczbaLisci := lewo + prawo;
      end
end;
```

Obieg prefiksowy

- Często chcemy odwiedzić każdy węzeł drzewa. Sposób, w jaki to zrobimy może nam pomóc w rozwiązaniu konkretnego problemu.
- Poza omawianymi algorytmami przechodzenia grafów w przypadku drzew mamy możliwość zastosowania specyficznych procedur
- Od tej pory zajmujemy się drzewami binarnymi.

Obieg prefiksowy

```
procedure wypiszPreLP(d:drzewo);
begin
 if d<>nil then
    begin
      Write(d^.w);
      wypiszpreLP(d^.lsyn);
      wypiszpreLP(d^.psyn)
    end
end;
```

Obieg prefiksowy - PL

```
procedure wypiszPrePL(d:drzewo);
begin
 if d<>nil then
    begin
      Write(d^.w);
      wypiszprePL(d^.psyn);
      wypiszprePL(d^.lsyn)
    end
end;
```

Obieg infiksowy

```
procedure wypiszInLP(d:drzewo);
begin
 if d<>nil then
    begin
      wypiszInLP(d^.lsyn);
      Write(d^.w);
      wypiszInLP(d^.psyn)
    end
end;
```

Obieg infiksowy - PL

```
procedure wypiszInPL(d:drzewo);
begin
 if d<>nil then
    begin
      wypiszInPL(d^.psyn);
      Write(d^.w);
      wypiszInPL(d^.lsyn)
    end
end;
```

Obieg postfiksowy

```
procedure wypiszPostLP(d:drzewo);
begin
 if d<>nil then
    begin
      wypiszpostLP(d^.lsyn);
      wypiszpostLP(d^.psyn)
      Write(d^.w);
    end
end;
```

Obieg postfiksowy - PL

```
procedure wypiszPostPL(d:drzewo);
begin
 if d<>nil then
    begin
      wypiszpostPL(d^.psyn);
      wypiszpostPL(d^.lsyn)
      Write(d^.w);
    end
end;
```

Twierdzenia o obiegach

- Jeśli porządki w prezentowanych obiegach, to odpowiednio PreLP, PrePL, InLP, InPL, PostLP, PostPL, to
  - $PreLP=PostPL^{-1}$
  - $PostLP=PrePL^{-1}$
  - $InLP=InPL^{-1}$