CVIČENÍ 1

Téma: Indukce a rekurze/rekurence v programování a v matematice
Cíle: Zvládnutí techniky důkazu jednoduchých tvrzení matematickou indukcí, induktivně definované struktury a jejich vlastnosti, rekurzivní návrh funkcí pracujících s těmito strukturami.

Úvod do předmětu:

Seznamte studenty:

1. s Vaší maličkostí – uveďte na sebe kontakt a konzultační hodiny
2. se střídáním proseminářů a cvičení
3. s bodováním (viz edux)

Matematická indukce:

1. silná indukce - má jiný indukční krok:

slabá:	z předpokladu platnosti S(n) dokážeme platnost S(n+1)
silná indukce:	z předpokladu platnosti S(0), S(1), ..., S(n) dokážeme platnost S(n+1)

Příklad na důkaz indukcí: Mějme následující induktivní definici funkce f : N → N:

$$f(0) = 0,\ f(1) = 1$$

$$\left.\begin{aligned} f(2n) &= f(n) \\ f(2n+1) &= f(n) + f(n+1) \end{aligned}\right\} \text{pro } n = 1, 2, 3, \ldots$$

Pro všechna n ≥ 0 dokažte pro tuto funkci ***f(n)*** platnost tvrzení:

V(n) = "f(n) je sudé, právě když je n dělitelno třemi"

2. Zavedeme (neúplně) ADT ListInt induktivně takto:
   - prázdný seznam je ListInt (vytvoří se pomocí **`initL()`** a testuje pomocí predikátu **`emptyL(L)`**)
   - je-li x celé číslo a L je ListInt, pak přidáním prvku x před začátek seznamu S (výsledek operace **`cons(x,L)`**) je rovněž ListInt
   - nic jiného než to, co vzniklo použitím výše uvedených pravidel, není ListInt

   Máme dále k dispozici operace **`first(S)`** a **`last(S)`**, které vrací první, resp poslední znak neprázdného seznamu S, a operace **`butFirst(S)`** a **`butLast(S)`,** které vrací zbytek seznamu S po odebrání prvního, resp. posledního prvku.
   Definujte nyní induktivně následující operace se seznamy:
   - **`length(L)`** – délka seznamu L (počet jeho prvků)
   - **`max(L)`** – maximální hodnota prvku v (neprázdném!) seznamu L
   - **`ordered(L)`** – predikát testující, zda prvky seznamu L tvoří neklesající posloupnost

   Na základě těchto definic vytvořte odpovídající rekurzivní funkce.

3. Zavedeme (neúplně) induktivní formou ADT String takto:
   - prázdný řetěz je řetěz (vytvoří se pomocí **`initS()`** nebo ″″ a testuje pomocí predikátu **`emptyS(S)`**)
   - je-li c libovolný znak a S řetěz, pak také výsledek připojení znaku c na začátek (výsledek operace **`addFirst(c,S)`**) nebo na konec (výsledek operace **`addLast(S,c)`**) je řetěz
   - nic jiného …

   Máme dále k dispozici operace **`first(S)`** a **`last(S)`**, které vrací první, resp poslední znak neprázdného řetězu S. Definujte nyní induktivně následující operace se řetězy:

- **concat(S,R)** – spojí za sebe řetězy S a R
- **lesseqStr(S,R)** – vrací **true**, je-li řetěz S lexikograficky menší nebo roven řetězu R

4. Silná indukce není nic jiného než slabá indukce pro tvrzení P(n) = S(0) & S(1) & … & S(n)
Procvičit dokazování jednoduchých vlastností na několika příkladech týkajících se stromů.
a) Zavedeme **pravidelný strom stupně 2** (**PSS2**) následující „množinovou“ definicí jako:
   - samostatný uzel (list) je PSS2
   - uzel (kořen) s podřízenou dvojicí PSS2 {L, R} je rovněž PSS2
   - nic jiného než struktura vzniklá použitím výše uvedených pravidel není PSS2

b) Připomeneme názvosloví: **listy** stromu jsou uzly, které nemají žádné podřízené stromy, ostatní jsou **vnitřní uzly; hloubka uzlu** je jeho vzdálenost od kořene; **výška uzlu** je maximální vzdálenost od listů; **výška stromu** je rovna výšce jeho kořene
c) Vhodným použitím indukce dokážeme následující
   - **Tvrzení 1**: PSS2 s n (≥0) vnitřními uzly má n+1 listů.
   - **Tvrzení 2**: PSS2 s n (≥0) vnitřními uzly má 2.n hran.
   - **Tvrzení 3**: Výška PSS2 s n (≥1) listy je nejvýše rovna n-1.
   - **Tvrzení 4**: PSS2 o výšce n (≥0) má celkem nejvýše 2^n listů.

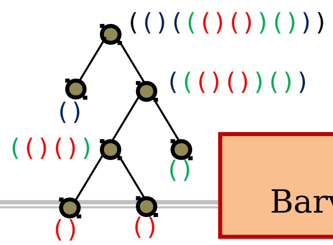
5. Předpokládejme, že PSS2 jsou implementovány obdobně jako binární stromy, každý uzel má ale nejen složky **`left`**, **`right`**, ale navíc i složky **`depth`**, **`height`**, **`count`**. Navrhněte rekurzivní algoritmy pro:
– výpočet hodnot složky **`height`** (výška) každého uzlu zadaného PSS2
– výpočet hodnot složky **`count`** (počet uzlů příslušného podstromu) každého uzlu zadaného PSS2
– výpočet hodnot složky **`depth`** (hloubka) každého uzlu zadaného PSS2
Návod: začněte rekurentními definicemi příslušných parametrů, které budou vycházet z induktivní definice PSS2, předpokládejte existenci predikátu **`Boolean list(T)`.**

Induktivní definice dalších typů stromů a jejich reprezentace

6. Jak induktivně definovat
   – binární strom
   – n-ární strom
   – pravidelný strom stupně r
   – (obecný) kořenový strom
   – uspořádaný kořenový strom

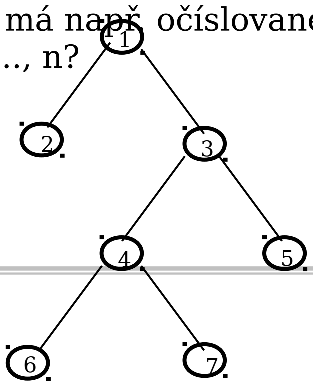
Pro každý typ stromu nakreslit charakteristický příklad, zjistit základní (elementární) alternativu a pravidlo konstrukce obecného případu

7. Jak bychom mohli vyjádřit strukturu nějakého stromu jistého typu pomocí řetězu znaků?



POZOR
Barvy jen rozlišují úroveň

8. Jak úplně jinak by se dala vyjádřit struktura stromu, který má např. očíslované uzly 1, 2, …, n?



	Pravidla vyjádřit opět induktivním způsobem korespondujícím s příslušnou definicí daného typu stromu. Kromě struktury je třeba pamatovat i na identifikaci uzlu / uložení nějaké hodnoty.		

Vzorová řešení (většiny) úloh

POZOR - programy jsou psané v pseudokódu (a-la Java)

Matematická indukce:

1. úplná indukce - má jiný indukční krok:

normálně:	z předpokladu platnosti S(n) dokážeme platnost S(n+1)
úplná indukce:	z předpokladu platnosti S(0), S(1), ..., S(n) dokážeme platnost S(n+1)

Příklad na důkaz indukcí: Mějme následující induktivní definici funkce f : N → N:

$$f(0) = 0,\ f(1) = 1$$

$$\left.\begin{aligned} f(2n) &= f(n) \\ f(2n+1) &= f(n) + f(n+1) \end{aligned}\right\}\ \text{pro } n = 1, 2, 3, \ldots$$

Pro všechna n ≥ 0 dokažte pro tuto funkci ***f(n)*** platnost tvrzení:

V(n) = "f(n) je sudé, právě když je n dělitelno třemi"

Řešení:

Zjevně platí V(0) i V(1), v indukčním kroku předpokládáme platnost V(0),V(1), ..., V(n) a dokazujeme V(n+1). Podle definice je f(n+1) = f(2k) = f(k), pro n+1=2k, kde k≤n, takže f(n+1) je sudé, právě když k je dělitelno 3, což platí právě když n+1 = 2k je dělitelno 3. Pro n+1= 2k+1 máme f(n+1) = f(2k+1) = f(k) + f(k+1). Aby součet byl sudý, musí mít oba členy stejnou paritu, ale oba nemohou být sudé, protože k a k+1 nemohou být obě dělitelná 3, tím pádem je k=3r+1, k+1= 3r+2, takže k+k+1 = 2k+1 = 6r + 3, což je dělitelno 3.☑

2. Zavedeme (neúplně) ADT ListInt induktivně takto:
   - prázdný seznam je ListInt (vytvoří se pomocí **`initL()`** a testuje pomocí predikátu **`emptyL(L)`**)
   - je-li x celé číslo a L je ListInt, pak přidáním prvku x před začátek seznamu S (výsledek operace **`cons(x,L)`**) je rovněž ListInt
   - nic jiného než to, co vzniklo použitím výše uvedených pravidel, není ListInt

Máme dále k dispozici operace **`first(S)`** a **`last(S)`**, které vrací první, resp poslední znak neprázdného seznamu S, a operace **`butFirst(S)`** a **`butLast(S)`**, které vrací zbytek seznamu S po odebrání prvního, resp. posledního prvku. Definujte nyní induktivně následující operace se seznamy:

- **`length(L)`** – délka seznamu L (počet jeho prvků)
- **`max(L)`** – maximální hodnota prvku v (neprázdném!) seznamu L
- **`ordered(L)`** – predikát testující, zda prvky seznamu L tvoří neklesající posloupnost

Na základě těchto definic vytvořte odpovídající rekurzivní funkce.

length(L) = 0 – je-li L prázdný seznam
          = length(rest(L)) + 1 – jinak ☑

max(L) = first(L) – je-li emptyL(rest(L)), tj. seznam obsahuje pouze jeden prvek
        = first(L) – je-li first(L) >= max(rest(L))
        = max(rest(L)) – jinak ☑

	Lepší vyjádření se inspiruje tím, jak se běžně maximum počítá, tj. pamatováním „dosavadního maxima“. Všechnu práci provede pomocná funkce, které zadáme první prvek jako dosavadní maximum a zbytek seznamu ve druhém argumentu. Pomocná funkce pak projde zbytek až do konce.
	max(L) = maxPom(first(L), rest(L))
	maxPom(Amx, L) = Amx – je-li L prázdný seznam
	= maxPom(Amx, rest(L)) – je-li Amx >= first(L)
	= maxPom(first(L), rest(L)) – jinak ☑
	
	ordered(L) = true – pokud L neobsahuje žádný prvek
	= ordPom(first(L), rest(L))
	ordPom(x, L) = true – je-li L prázdný seznam
	= false – je-li x > first(L)
	= ordPom(first(L), rest(L)) – jinak ☑
	
	`int length (ListInt L) {`
	`  if (emptyL(L)) return 0;`
	`  else return length(rest(L))+1;`
	`}`
	
	`int max (ListInt L) { // jaká je složitost tohoto algoritmu???`
	`  if(empty(rest(L))) return first(L);`
	`  else if (first(L) >= max(rest(L))) return first(L);`
	`  else return max(rest(L));`
	`}`
	
	`int max ListInt L) {`
	`  return maxPom(first(L), rest(L));`
	`}`
	`int maxPom (int Amx, ListInt L) {`
	`  if (emptyL(L)) return Amx;`
	`  else if (Amx >= first(L)) return maxPom(Amx, rest(L));`
	`  else return maxPom(first(L), rest(L));`
	`}`
	
	`Bool ordered (ListInt L) {`
	`  if (emptyL(L)) return true;`
	`  else return ordPom(first(L), rest(L));`
	`}`
	`Bool ordPom (int x, ListInt L) {`
	`  if (empty(L)) return true;`
	`  else if (x > first(L)) return false;`
	`  else return ordPom(first(L), rest(L));`
	`}`

3.	Zavedeme (neúplně) induktivní formou ADT String takto: - prázdný řetěz je řetěz (vytvoří se pomocí **`initS()`** nebo ˝˝ a testuje pomocí predikátu **`emptyS(S`**)) - je-li c libovolný znak a S řetěz, pak také výsledek připojení znaku c na začátek (výsledek operace **`addFirst(c,S)`**) nebo na konec (výsledek operace **`addLast(S,c)`**) je řetěz - nic jiného … Máme dále k dispozici operace **`first(S)`** a **`last(S)`**, které vrací první, resp poslední znak neprázdného řetězu S, a operace **`butFirst(S)`** a **`butLast(S),`** které vrací zbytek řetězu S po odebrání prvního, resp. posledního znaku. Dále předpokládáme,

že jednotlivé znaky jsou uspořádány („abecení uspořádání“). Definujte nyní induktivně následující operace se řetězy:

- **`concatS(S,R)`** – spojí za sebe řetězy S a R
- **`lesseqS(S,R)`** – vrací **`true`**, je-li řetěz S lexikograficky menší nebo roven řetězu R

concatS(S,R) = R – je-li S prázdný řetěz
= S – je-li R prázdný řetěz (tuto větev možno vynechat)
= addFirst(first(S), concatS(butFirst(S),R)) – jinak

Jiná možnost je připojovat na konec S znaky ze druhého řetězu
concatS(S,R) = S – je-li R prázdný řetěz
= R – je-li S prázdný řetěz (tuto větev možno vynechat)
= addLast(concatS(S, butLast(R)), last(R)) – jinak

Pro lexikografické srovnání nejprve vysvětlit na příkladech, jak vlastně funguje.
lesseqS(S,R) = true – je-li S prázdný řetěz
= false – je-li R prázdný řetěz
= true – je-li first(S) < first(R)
= false – je-li first(S) > first(R)
= lesseqS(butFirst(S), butFirst(R)) – jinak

```
String concatS (String S, String R) {
  if (emptyS(S)) return R;
  else if (emptyS(R)) return S;
  else return addFirst(first(S), concatS(butFirst(S), R));
}

Bool lesseqS (String S, String R) {
  if (emptyS(S)) return true;
  else if (emptyS(R)) return false;
  else if (first(S) < first(R)) return true;
  else if (first(S) > first(R)) return false;
  else return lesseqS(butFirst(S), butFirst(R));
}
```

4. Silná indukce není nic jiného než slabá indukce pro tvrzení P(n) = S(0) & S(1) & ... & S(n)
Procvičit dokazování jednoduchých vlastností na několika příkladech týkajících se stromů.
a) Zavedeme **pravidelný strom stupně 2** (**PSS2**) následující „množinovou“ definicí jako:

- samostatný uzel (list) je PSS2
- uzel (kořen) s podřízenou dvojicí PSS2 {L, R} je rovněž PSS2
- nic jiného než struktura vzniklá použitím výše uvedených pravidel není PSS2

b) Připomeneme názvosloví**:** **listy** stromu jsou uzly, které nemají žádné podřízené stromy, ostatní jsou **vnitřní uzly, hloubka uzlu** je jeho vzdálenost od kořene, **výška uzlu** je maximální vzdálenost od listů, **výška stromu** je rovna výšce jeho kořene

Tvrzení 1: PSS2 s n (≥0) vnitřními uzly má n+1 listů.

- základní PSS2 (samostatný uzel) má 0 vnitřních uzlů a 1 list, tvrzení tedy platí pro základní případ PSS2
- nechť tvrzení platí pro PSS2 s 0, 1, ..., n (≥0) vnitřními uzly, mějme PSS2 T s (n+1) vnitřními uzly, přitom je n+1 ≥1. Strom T tedy musí být jiný než

základní PSS2, takže má kořen a podřízenou dvojici podstromů L a R, které mají n1 (0 ≤ n1 ≤ n), resp. n2 (0 ≤ n2 ≤ n) vnitřních uzlů, přičemž platí (kořen+n1+n2) = n+1 neboli n1+n2 = n (nakreslit obrázek!!!). Podle indukčního předpokladu má tedy podstrom L právě n1+1 listů a podstrom R právě n2+1 listů, takže strom T má n1+n2+2 = n+2 = (n+1) + 1 ☑

Důkaz prostou indukcí podle n – liší se induktivní krok:

- nechť tvrzení platí pro PSS2 s n (≥0) vnitřními uzly, mějme PSS2 T s (n+1) vnitřními uzly, přitom je n+1 ≥1. V T nalezneme takový vnitřní uzel u, jehož oba potomci x a y jsou listy – takový zde musí existovat (proč?). Pokud listy x a y ze stromu T odebereme, stane se uzel u sám listem a vzniklý strom T’ má tedy jen n (≥0) vnitřních uzlů. Podle indukčního předpokladu má strom T’právě (n+1) listů. Od T’ se dostaneme ke stromu T zpětným přidáním původních dvou listů x a y, přičemž z uzlu u se opět stane vnitřní uzel. Počet listů stromu T je tedy (n+1)+2–1 = (n+1)+1, což dokazuje platnost našeho tvrzení. ☑

Tvrzení 2: PSS2 s n (≥0) vnitřními uzly má 2.n hran.

- základní PSS2 má 0 vnitřních uzlů a 0 = 2.0 hran, tvrzení tedy platí pro základní případ PSS2
- nechť tvrzení platí pro PSS2 s 0, 1, ..., n (≥0) vnitřními uzly, mějme PSS2 T s (n+1) vnitřními uzly, přitom je n+1 ≥1. Strom T tedy musí být jiný než základní PSS2, takže má kořen a podřízenou dvojici podstromů L a R, které mají n1 (0 ≤ n1 ≤ n), resp. n2 (0 ≤ n2 ≤ n) vnitřních uzlů, přičemž platí (kořen+n1+n2) = n+1 neboli n1+n2 = n (nakreslit obrázek!!!). Podle indukčního předpokladu má tedy podstrom L právě 2.n1 hran a podstrom R právě 2.n2 hran, takže strom T má 2.n1 + 2.n2 + 2 = 2.n + 2 = 2.(n+1) hran. ☑

Tvrzení 3: Výška PSS2 s n (≥1) listy je nejvýše rovna n-1.

- základní PSS2 má 1 list a výšku 0, tvrzení tedy platí pro základní případ PSS2
- nechť tvrzení platí pro PSS2 s 0, 1, ..., n (≥1) listy, mějme PSS2 T s (n+1) listy, přitom je n+1≥2. Strom T má tedy alespoň dva listy a musí být jiný než základní PSS2, takže má kořen a podřízenou dvojici podstromů, které mají n1 (1 ≤ n1 ≤ n), resp. n2 (1 ≤ n2 ≤ n) listů, takže jejich maximální výška je podle indukčního předpokladu n1-1, resp. n2-1. Strom T bude mít výšku rovnou maximu z těchto dvou hodnot zvětšenému o 1 (nakreslit obrázek!!!). Maximální výšky se dosáhne při co největším rozdílu velikosti levého a pravého podstromu, tedy pro n1=1 a n2=n nebo n1=n a n2=1. Pak bude výška stromu T rovna (n-1)+1 = (n+1) – 1. ☑

Tvrzení 4: PSS2 o výšce n (≥0) má celkem nejvýše 2^n listů..

- základní PSS2 má výšku 0 a 2^0=1 list, tvrzení tedy platí pro základní případ PSS2
- nechť tvrzení platí pro PSS2 s výškou 0, 1, ..., n (≥0), mějme PSS2 T s výškou (n+1), přitom je n+1≥1. Strom T tedy musí být jiný než základní PSS2, takže má kořen a podřízenou dvojici podstromů, které mají oba (nebo aspoň jeden z nich) výšku n. Podle indukčního předpokladu obsahuje každý podstrom nejvýše 2^n listů, dohromady tedy má strom T maximálně 2. 2^n = 2^{n+1} ☑

5. Předpokládejme, že PSS2 jsou implementovány obdobně jako binární stromy, každý uzel má ale nejen složky **`left`**, **`right`**, ale navíc i **`depth`**, **`height`**, **`count`**.

Navrhněte rekurzivní algoritmy pro:
– výpočet hodnot složky **`height`** (výška) každého uzlu zadaného PSS2
– výpočet hodnot složky **`count`** (počet uzlů příslušného podstromu) každého uzlu zadaného PSS2
– výpočet hodnot složky **`depth`** (hloubka) každého uzlu zadaného PSS2

Návod: začněte rekurentními definicemi příslušných parametrů, které budou vycházet z induktivní definice PSS2, předpokládejte existenci predikátu **`Boolean list(T).`**

height(u) – výška uzlu závisí pouze na vzdálenosti listů pod tímto uzlem, takže vždy stačí uvažovat jen podstrom s kořenem u, neboli počítat výšku kořene stromu

height(u) = 0 – je-li u samostatný uzel (list)
= max (height(u_L), height(u_R)) + 1 – je-li u kořen stromu s postromy L a R, přitom u_L, resp. u_R
je kořen podstromu L resp. R ☑

count(u) – opět se počítá pro kořeny
count(u) = 1 – je-li u samostatný uzel (list)
= count (u_L) + count (u_R) + 1 – je-li u kořen stromu s postromy L a R, přitom u_L, resp. u_R je
kořen podstromu L resp. R ☑

depth (u, T) = 0 – je-li u kořenem stromu T
= depth(parent(u), T) + 1 – jinak, přitom parent(u) označuje rodiče uzlu u ve stromu ☑

POZOR! Toto je obrácená rekurze – triviální případ není list ale kořen, což se projeví v návrhu algoritmu, tzn. tento postup odpovídá preorder zpracování, ostatní jsou postorder.

```
void SetHeights (Tree T) {
  if (list(T)) {
    T.height = 0;
  }
  else {
    SetHeights(T.left);
    SetHeights(T.right);
    T.height = max(T.left.height, T.right.height) + 1;
  }
}

void SetCounts (Tree T) {
  if (list(T)) {
    T.count = 1;
  }
  else {
    SetCounts(T.left);
    SetCounts(T.right);
    T.count = T.left.count + T.right.count + 1;
  }
}

void SetDepths (Tree T) {
  SDep(T,0);
}
void SDep (Tree T, int D) {
  T.depth = D;
```

```
  if (!list(T)) {
    SDep(T.left, D+1);
    SDep(T.right, D+1);
  }
}

// ALL IN ONE
void SetAll (Tree T) {
  SAll(T, 0);
}
void SAll (Tree T, int D) {
  T.depth = D;
  if (list(T)) {
    T.height = 0;
    T.count = 1;
  }
  else {
    SAll(T.left, D+1);
    SAll(T.right, D+1);
    T.height = max(T.left.height, T.right.height) + 1;
    T.count = T.left.count + T.right.count + 1;
  }
}   ☑
```

Induktivní definice dalších typů stromů a jejich reprezentace

7. Jak induktivně definovat
   - binární strom
   - n-ární strom
   - pravidelný strom stupně r
   - (obecný) kořenový strom
   - uspořádaný kořenový strom

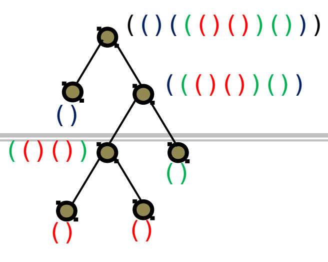
Binární strom je buď
- **prázdný**, nebo
- trojice: uzel (nazývaný **kořen**) a uspořádaná dvojice binárních stromů L (**levý podstrom**) a P (**pravý podstrom**)

n-ární strom je buď
- **prázdný**, nebo
- uzel (kořen) a jemu podřízená **uspořádaná** n-tice n-árních stromů T1, T2,...,Tn

Pravidelný strom stupně r je buď
- **samostatný uzel**, nebo
- uzel (kořen) a jemu podřízená **množina r** pravidelných stromů T1, T2, ..., Tr stupně r

(Obecný) kořenový strom je buď
- **samostatný uzel**, nebo
- uzel (kořen) a jemu podřízená neprázdná **množina** kořenových stromů T1, T2, ..., Tk

8. Jak bychom mohli vyjádřit strukturu nějakého stromu jistého typu pomocí řetězu znaků?



POZOR
Barvy jen rozlišují úroveň

Pravidla vyjádřit opět induktivním způsobem korespondujícím s příslušnou definicí daného typu stromu. Kromě struktury je třeba pamatovat i na identifikaci uzlu / uložení nějaké hodnoty.

Stačí použít libovolnou dvojici párových symbolů, např. závorky (a), nebo [a], nebo prostě 0 a 1, jedná se o tzv. **lineární kódování stromů**.

Binární strom
- **prázdný**
- **kořen**, L, P

```
()
(k RL RP)
```

n-ární strom
- **prázdný**
- kořen k a podstromy T1, T2,...,Tn

```
()
(k RT1 RT2 ... RTn)
```

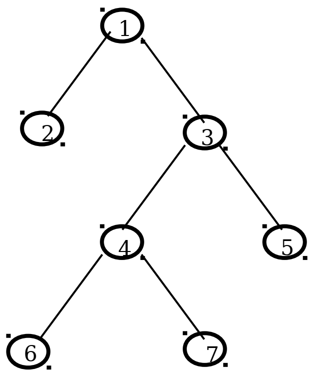
Pravidelný strom stupně r
- **samostatný uzel**
- kořen k a podstromy T1, T2, ..., Tr st

(i) nebo jen **i**
(k RT1 RT2 ... RTr)

(Obecný) kořenový strom - stejná reprezentace, jen pořadí podstromů nehraje roli

Pro každý typ stromu ilustrovat kódování na konkrétním příkladu.

9\. Jak úplně jinak by se dala vyjádřit struktura stromu, který má např. očíslované uzly 1, 2, ..., n?



Jiná vnější reprezentace stromu
- známe/zadáme počet uzlů ***N*** a počet hran ***M*** (uzly /hrany číslujeme 1, 2, ..., ***N*** / ***M***)
- strukturu vyjadřujeme "po částech" ve formě:
  - seznamu hran (tj. dvojic krajních uzlů)
  - seznamů potomků / sousedů jednotlivých uzlů
  - odkazy na předchůdce

Ilustrovat konkrétními výčty pro výše uvedený strom.