STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Dátové štruktúry a algoritmy

Abstraktné dátové typy

3. 3. 2021

letný semester
2020/2021

Abstraktné dátové typy (ADT)

- Všeobecný model pre dátový typ (dátovú štruktúru) vyjadrený pomocou abstrakcie:
  - Určíme operácie s dátovým typom a ich vlastnosti
  - Abstrahujeme od konkrétnej implementácie
- ADT môžeme implementovať rôznymi spôsobmi bez toho, aby to ovplyvnilo správnosť behu programu-algoritmu, ktorý ADT používa

Dátový typ

- Každá hodnota v programe má dátový typ
- Množina použiteľných dátových typov je určená použitým programovacím jazykom
- Dátový typ premennej určuje
  - **Množinu hodnôt**, ktoré možno dátovým typom reprezentovať
  - **Vnútornú reprezentáciu v počítači** (využitie-kódovanie v pamäti)
  - **Prípustné operácie**, ktoré možno nad hodnotami daného typu vykonávať

Jednoduché dátové typy

- Boolean
  - Množina hodnôt {**true**, **false**}
  - Reprezentácia v pamäti ako 1 byte
  - Prípustné operácie: NOT, AND, OR
- Štandardne v jazyku C:
  - char, int, float, double
  - ukazovateľ (smerník)
  - Zložené: pole, struct, union

Abstraktný dátový typ vs. dátová štruktúra

- Abstraktný dátový typ
  - Množina typov údajov a operácií, ktoré sú špecifikované nezávisle od konkrétnej implementácie
  - Reprezentuje model zložitejšieho dátového typu
  - Abstraktný model

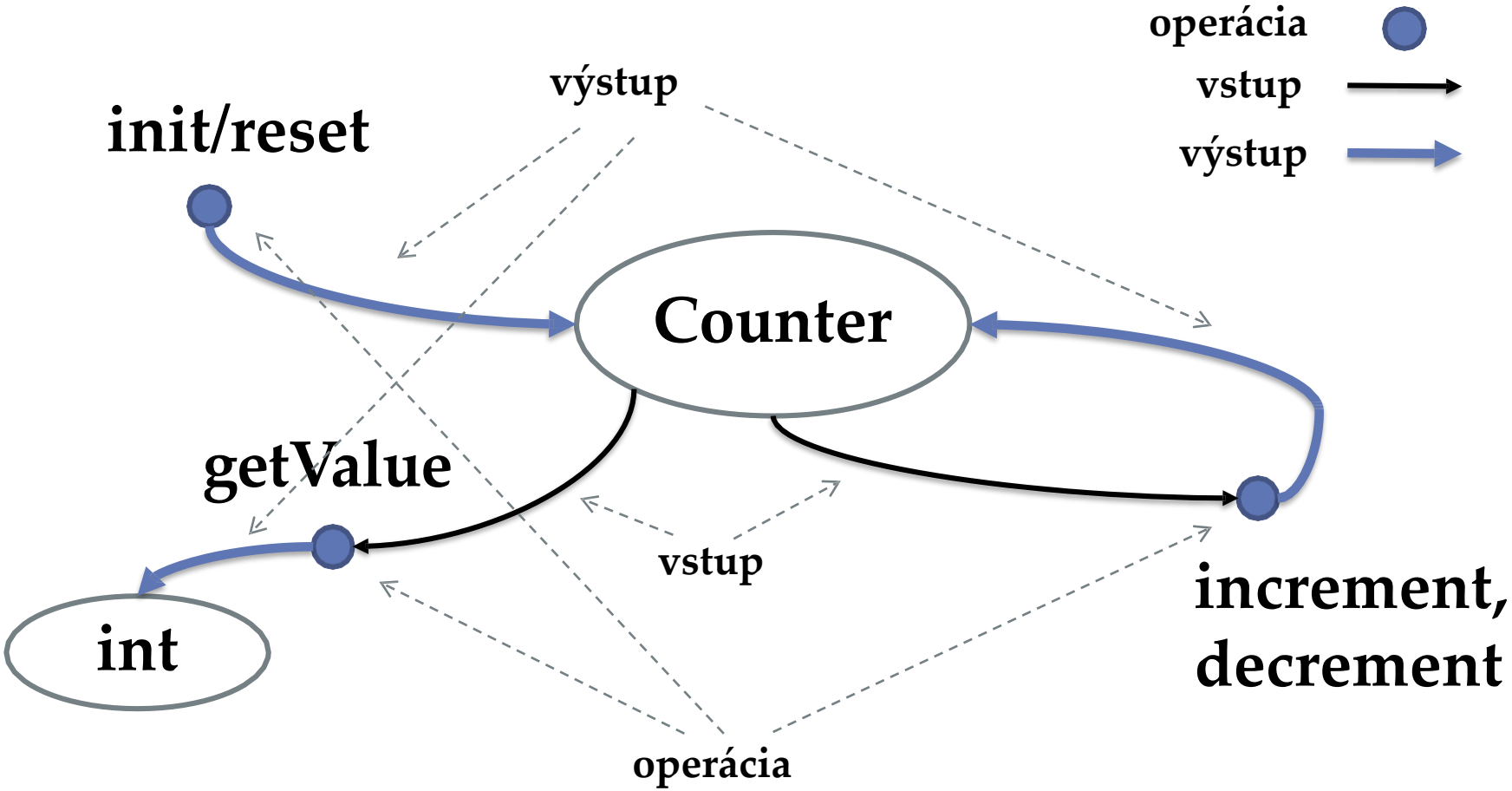
- Dátová štruktúra
  - Implementácia ADT v programovacom jazyku
  - Reprezentácia typov údajov v ADT
  - Voľba algoritmov pre implementáciu operácií ADT

Definícia A D T

- Formálne
  - Signatúra a axiómy
- Programátorsky
  - Definícia rozhranie s operáciami

- Ukážka: **Počítadlo – A D T**
  - Stevard v lietadle počíta cestujúcich ...

Počítadlo (Counter) - signatúra



Počítadlo (Counter) - axiómy

- Opisujú vlastnosti – význam (sémantiku) operácií prostredníctvom ekvivalencie výrazov

  Pre všetky C $\in$ Counter platí:

  getValue (init) = 0
  getValue (increment(C)) = getValue(C) + 1
  getValue (decrement(C)) = getValue(C) - 1

Počítadlo - Programátorské rozhranie

- Definícia rozhrania s operáciami

```
int getValue();
void increment();
void reset();
```

- Možná implementácia:

```
int value = 0;
int getValue() { return value; }
void increment() { value++; }
void reset() { value = 0; }
```

Počítadlo - Programátorské rozhranie

- Definícia rozhrania v jazyku C

```
int getValue(struct Counter *c);
void increment(struct Counter *c);
void reset(struct Counter *c);
```

- Možná implementácia v jazyku C:

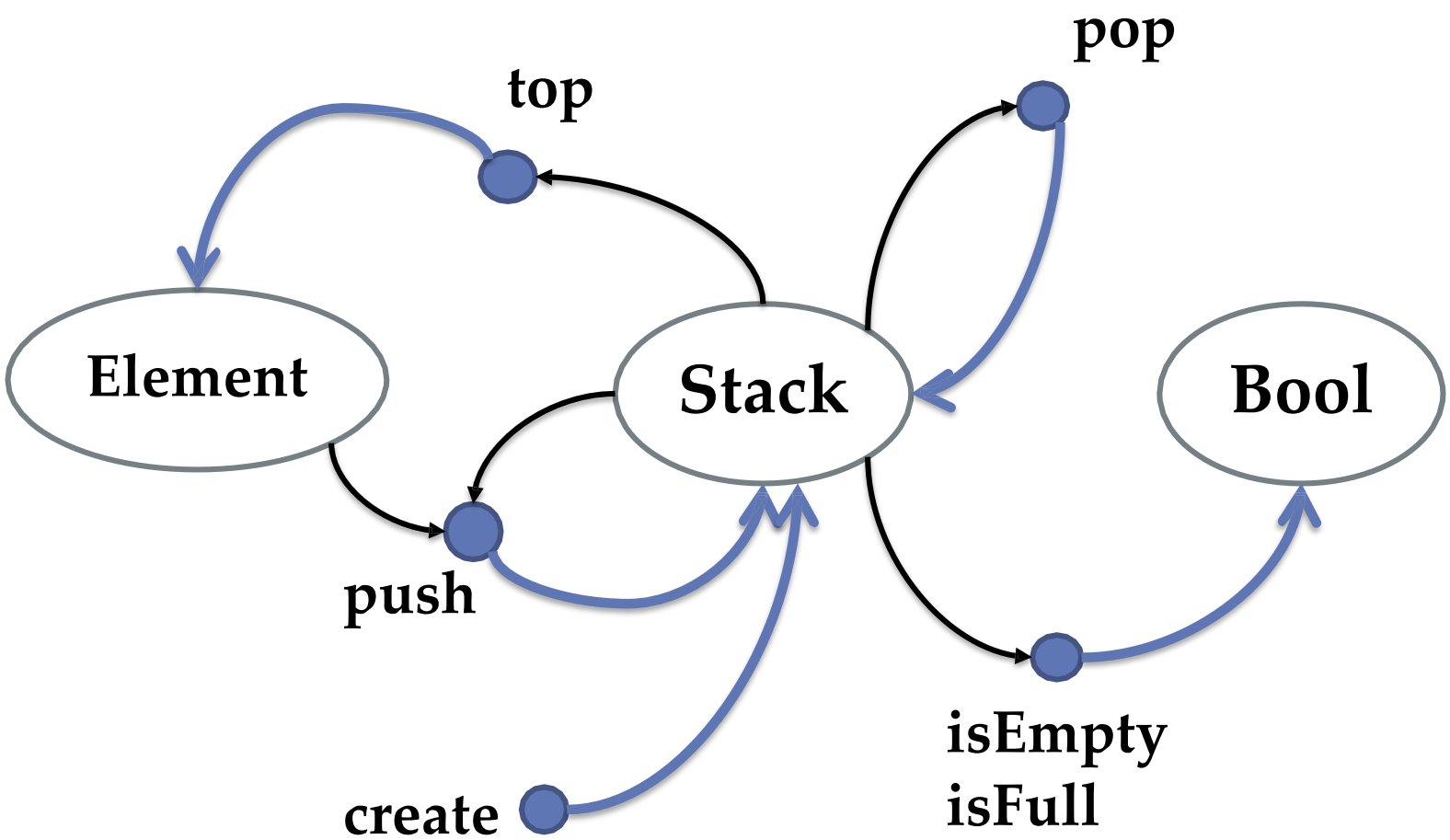
```
struct Counter
{
  int value;
};

int getValue(struct Counter *c) { return c->value; }
void increment(struct Counter *c) { c->value++; }
void reset(struct Counter *c) { c->value = 0; }
```

Ďalšie ADT (s obmedzeným prístupom)

- **Zásobník (Stack)**
  - Push (vlož na vrch zásobníka)
  - Pop (vyber z vrchu zásobníka)
  - LIFO – Last In, First Out
- **Rad-Front (Queue)**
  - Enqueue (pridaj na koniec do radu-fronty)
  - Dequeue (odober zo začiatku radu-fronty)
  - FIFO – First In, First Out

Zásobník (Stack) - Signatúra



Zásobník (Stack) - axiómy

Pre všetky $S \in Stack$, $e \in Element$ platí:

$isEmpty(create) = true$

$isEmpty(push(S,e)) = false$

$pop(create) = error$

$pop(push(S,e)) = S$

$top(create) = NULL$

$top(push(S,e)) = e$

Zásobník (Stack) - implementácia poľom

Stack S:

create(S)

```
    top(S) ← 0
```

push(S,x)

```
    if isFull(S)
        then error ”overflow”
        else top(S) ← top(S) + 1
             S[top(S)] ← x
```

pop(S)

```
    if isEmpty(S)
        then error ”underflow”
        else top(S) ← top(S) – 1
             return S
```

Zásobník - Implementácia poľom v jazyku C

- Lenivá implementácia, globálna premenná

```
int stack[MAX], head;

void push(int v) { stack[head++]=v; }
int pop(void) { return stack[--head]; }
void create(void) { head = 0; }
int isEmpty(void) { return !head; }
```

STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Dátové štruktúry a algoritmy

Vyhľadávanie

3. 3. 2021

letný semester
2020/2021

Vyhľadávanie

- Vstup:
  - Postupnosť: $a_1, a_2, a_3 \ldots a_n$
    $k(a_i)$ označíme kľúč k_i prvku a_i
  - Hľadaný kľúč x
  - Čo sú kľúče?

    Definičný obor D – reťazce, reálne čísla, dvojice celých čísel, ...
  - Relácia = (rovnosti) – relácia ekvivalencie nad D
  - Usporiadanie kľúčov < (binárna relácia nad D)

    Lineárne usporiadaná množina K (total ordering)
    Pre $k_1, k_2 \in D$ budeme písať, že $k_1 \leq k_2$ akk $k_1 < k_2$ alebo $k_1 = k_2$.
- Výstup:
  - Index res $\in \{1,2,\ldots,n\}$ takého prvku, že $k(a_{res}) = x$, alebo 0 ak taký prvok neexistuje.

Lineárne vyhľadávanie

- O vstupnej postupnosti kľúčov nemám žiadne znalosti
- Algoritmus: **Prehľadám postupne všetky prvky**
- Čo ma zaujíma: Počet vykonaných operácií
- Vstup: postupnosť N čísel, hľadám x=33
  - hľadám y=42

- Počet vykonaných operácií:
  - najlepší prípad: O(1), najhorší prípad O(n), priemerne (n)

Lineárne vyhľadávanie - zdrojový kód

- Vyhľadanie podľa čísla

```
int hladaj(int *data, int n, int kluc)
{
    int i;
    for (i = 0; i < n; i++)
        if (data[i] == kluc)
            return i;
    return -1;
}
```

- Vyhľadanie podľa mena

```
struct Osoba
{
    char *meno;
    int vek;
};

int hladaj(struct Osoba *data, int n, char *kluc)
{
    int i;
    for (i = 0; i < n; i++)
        if (!strcmp(data[i].meno, kluc))
            return i;
    return -1;
}
```

Existuje lepší algoritmus pre tento problém?

- Nie

  ak o postupnosti kľúčov naozaj nič ďalšie neviem

- Uvažujme teraz, že by sme vstupnú postupnosť mali vzostupne usporiadanú – dodatočná informácia je **usporiadanie**
- **Hľadám y=42**

y

- **Skontrolovaním jedného prvku môžem vylúčiť z ďalšieho hľadania celý interval prvkov**
  - Najlepší prípad?

    kratší z intervalov vľavo alebo vpravo
  - Najhorší prípad?

    dlhší z intervalov vľavo alebo vpravo

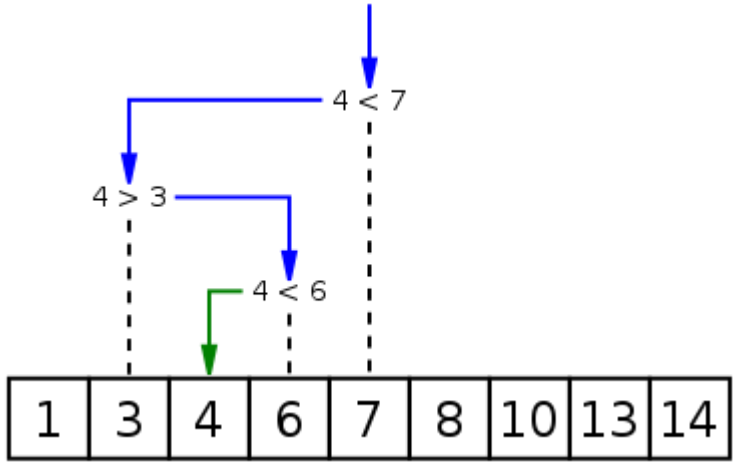
Princíp rýchlejšieho vyhľadávania

- Nejaká informácia navyše o vstupe, napr. usporiadanie
- Skontrolovaním **jedného prvku** môžem vylúčiť z ďalšieho hľadania celý interval prvkov
  - Najlepší prípad? kratší z intervalov vľavo alebo vpravo
  - Najhorší prípad? dlhší z intervalov vľavo alebo vpravo
- Čo keď nechcem riskovať lepší-horší prípad?
- Metóda pólenia intervalu (**binárne vyhľadávanie**)
  - Vždy **skontrolujem prvok v strede intervalu**, vktorom hľadaný kľúč ešte môže byť
  - Ak som našiel hľadaný kľúč, končím, inak pokračujem v zostávajúcej polovici intervalu
  - Koľko bude porovnaní?

Analýza zložitosti binárneho vyhľadávania

- **Algoritmus: skontrolujem prvok v strede intervalu,** v ktorom hľadaný kľúč ešte môže byť
  - Ak som našiel hľadaný kľúč, končím, inak pokračujem v zostávajúcej polovici intervalu

Príklad (hľadám 4):



- Koľko bude porovnaní?
  - Najlepší prípad: O(1)
  - Najhorší prípad?
    keď sa hľadaný kľúč v postupnosti nenachádza
  - Koľko razy môžem skrátiť interval na polovicu, až kým nedostanem posledný prvok, ktorý to teoreticky môže byť?
    O(log n)

Porovnanie O(n) vs O(log n)

- Lineárne vyhľadávanie
  - N prvkov = N operácie
  - 2N prvkov = 2N operácií
- Binárne vyhľadávanie
  - N prvkov = log N operácií
  - 2N prvkov = (log N) + 1 operácií

N	log N
10	4
1000	10
1 000 000	20
2 000 000 000	32

Existuje ešte rýchlejší algoritmus pre tento problém?

- Nie

  ak o postupnosti kľúčov naozaj nič ďalšie neviem

- Uvažujme teraz, že by sme pre vstupnú postupnosť mali **ešte nejakú dodatočnú informáciu**
  - Aká by to mohla byť?
- Distribúcia hodnôt kľúčov
  - Predstava o tom, koľko hodnôt ktorého kľúča sa môže v postupnosti nachádzať.
  - Príklad z nedávnej praxe:
    telefónny zoznam – počet priezvisk podľa začiatočného písmena

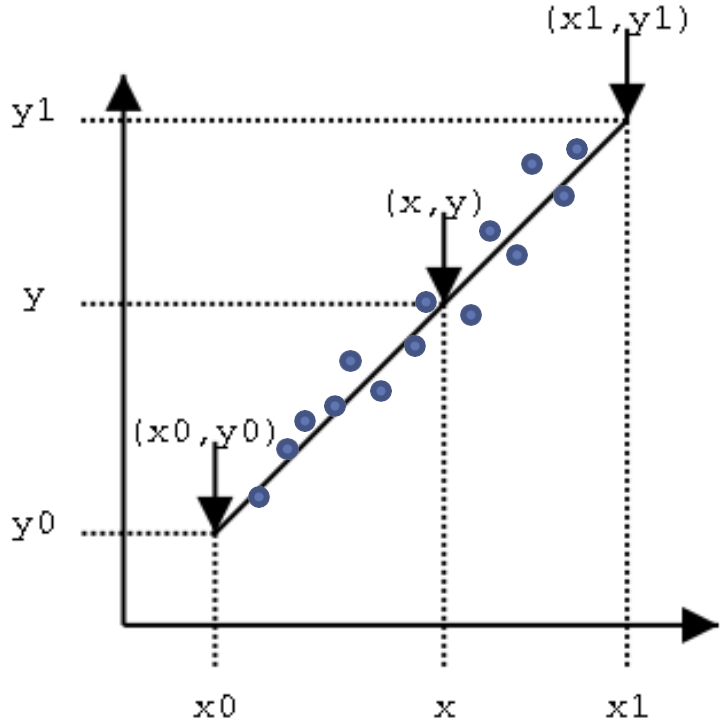
Interpolačné vyhľadávanie

- Predpokladajme rovnomernú distribúciu hodnôt kľúčov
- Na intervale $<x_0, x_1>$ nadobúdajú hodnoty $<y_0, y_1>$
- Hľadám kľúč y, ako určím čo najpravdepodobnejší výskyt - index x, taký, že $k(a_x)$ je blízko y?

$$\frac{y-y_0}{y_1-y_0} = \frac{x-x_0}{x_1-x_0}$$

$$x = x_0 + \frac{(y-y_0)*(x_1-x_0)}{y_1-y_0}$$

- Výpočtová zložitosť pre postupnosť s n prvkami
  - **O(log log n) krokov**



Nová požiadavka – dynamická množina

- Doteraz sme vyhľadávali v postupnosti, ktorú sme dostali celú na vstupe a ďalej sme ju nemenili
- Čo keby sme chceli postupnosť upravovať?
  - Uvažujme register osôb
  - Pridávať prvky (narodenie dieťaťa)
  - Vyhľadávať prvky (osoba podľa rodného čísla)
  - Odstrániť prvky (úmrtie)

- ADT: dynamická množina

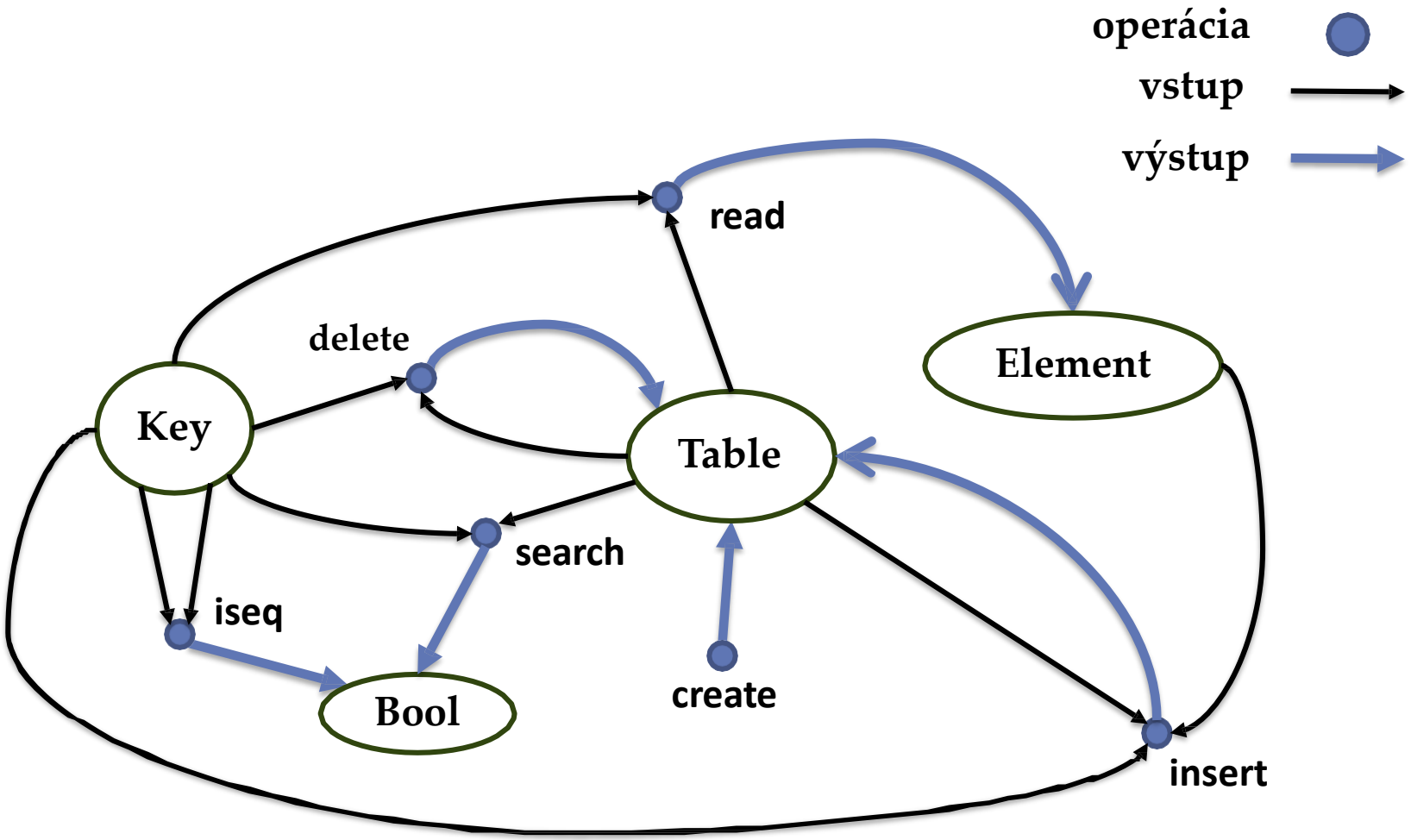
ADT - (Dynamická) množina

- abstraktný údajový typ množina
  - počet prvkov v údajoch typu množina sa často mení
  - najčastejšie operácie:
    insert – vložiť/pridať prvok do množiny
    search – vyhľadať prvok v množine
    delete – odstrániť prvok z množiny
- Nazývame aj **slovník (dictionary)**
- Napr: tabuľka symbolov v prekladačoch

Tabuľka

- skupina metód, ako implementovať slovník
- bežne aj synonymum pre slovník, najmä ak uvažujeme aj jeho implementáciu
- ale spravidla pomenovanie implementujúceho vektora
- určuje štruktúru pre jednotlivé údaje, ktorá združuje/asociuje hodnotu s kľúčom
- V pamäti sa údaje uchovávajú ako dvojica kľúč – hodnota = položka, prvok tabuľky
- K hodnote sa pristupuje pomocou kľúča – kľúč položku jednoznačne určuje

Slovník /Tabuľka - signatúra



Slovník / Tabuľka - axiómy

- Opisujú vlastnosti – význam (sémantiku) operácií prostredníctvom ekvivalencie výrazov

  Pre všetky $k, k_1, k_2 \in$ KĽÚČ, $e \in$ ELEMENT, $t \in$ TABUĽKA platí:

```
insert(k1,e1, insert(k2,e2,t)) =
    if(iseq(k1, k2))
       then insert(k1,e1,t)
       else insert(k2,e2,insert(k1,e1,t))
```

Slovník /Tabuľka - axiómy (2)

- Pre všetky $k, k_1, k_2 \in$ KĽÚČ, $e \in$ ELEMENT, $t \in$ TABUĽKA platí:

```
search(k, create) = false
search(k1, insert(k2,e,t)) =
  if(iseq(k1, k2))
          then true
          else search(k1, t)

read(k, create) = error_elem
read(k1, insert(k2,e,t)) =
if(iseq(k1, k2))
    then e
    else read(k1, t)
```

Slovník / Tabuľka - axiómy (3)

- Pre všetky $k, k_1, k_2 \in$ KĽÚČ, $e \in$ ELEMENT, $t \in$ TABUĽKA platí:

delete(k, create) = create

delete(k_1, insert(k_2,e,t)) =

if(iseq(k_1, k_2))

then t

else insert(k_2,e,delete(k_1, t))

Dynamická množina poľom

- Implementácia poľom – vektorom
- Pridanie/odstránenie jedného prvku do usporiadanej postupnosti vyžaduje (v najhoršom prípade) posunutie všetkých prvkov
- Teda vieme spraviť dátovú štruktúru, podporujúcu operácie:
  - **Pridanie prvku X** – vyžaduje O(N) operácií, kde N je počet prvkov v postupnosti
  - **Vyhľadanie prvku X** – vyžaduje O(log N) operácií, využijeme binárne vyhľadávanie
  - **Odstránenie prvku X** – vyžaduje O(N) operácií, kde N je počet prvkov v postupnosti

Dynamická množina spájaným zoznamom

- Implementácia spájaným zoznamom
- Pridanie/odstránenie jedného prvku do usporiadanej postupnosti vyžaduje (v najhoršom prípade) prehľadanie všetkých prvkov
- Teda vieme spraviť dátovú štruktúru, podporujúcu operácie:
  - **Pridanie prvku X** – vyžaduje O(N) operácií, kde N je počet prvkov v postupnosti
  - **Vyhľadanie prvku X** – vyžaduje O(N) operácií
  - **Odstránenie prvku X** – vyžaduje O(N) operácií, kde N je počet prvkov v postupnosti

Existuje rýchlejší algoritmus pre pridávanie?

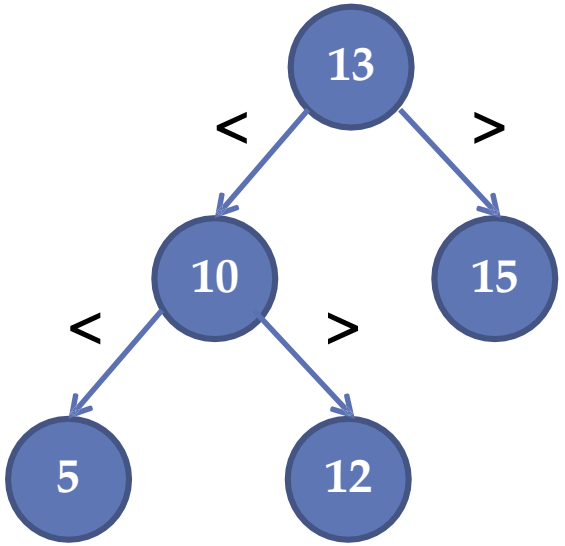
- Vyhľadávanie dokážeme v O(log N)
- Dokážeme prvky aj pridávať v O(log N) pri zachovaní času pre vyhľadávanie O(log N)?
- Spomeňme si na základný princíp rýchlejšieho (aj binárneho) vyhľadávania v usporiadanej postupnosti:

  Skontrolovaním jedného prvku môžem vylúčiť z ďalšieho hľadania celý interval prvkov

- Takéto „rozhodovanie“ vieme vhodne reprezentovať rozhodovacím stromom...

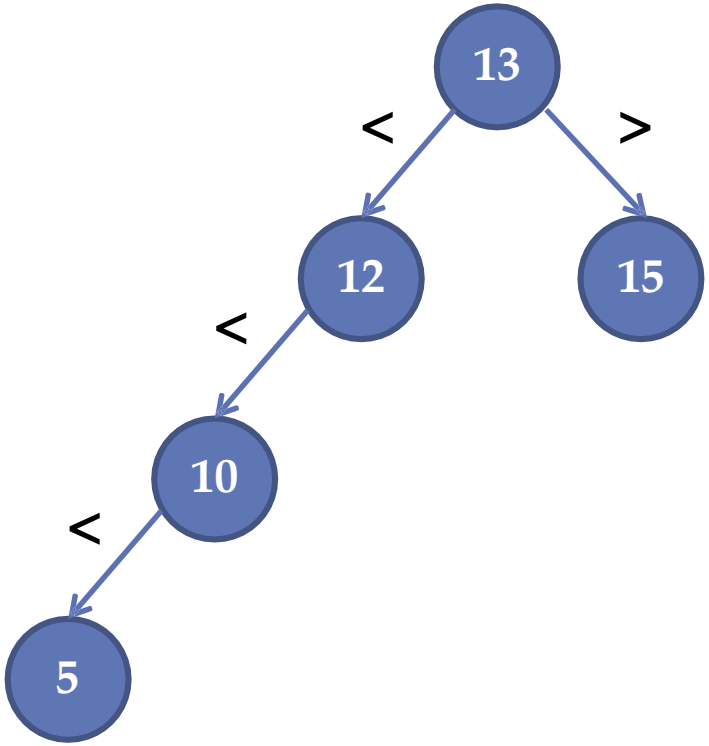
Binárny vyhľadávací strom

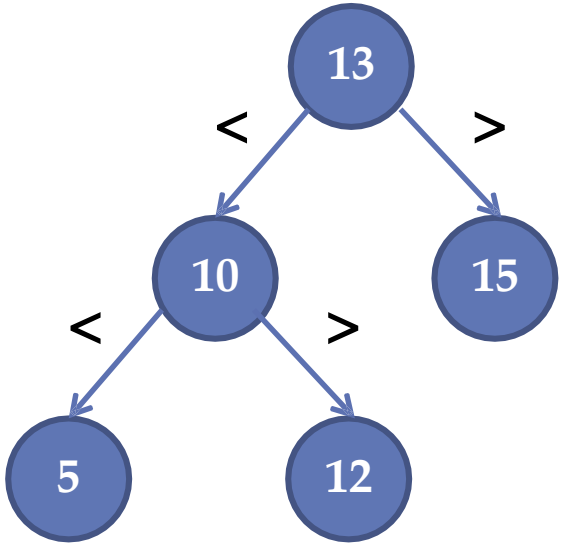
- Každý vrchol má hodnotu a (najviac) dvoch nasledovníkov:
  - Ľavého – kde je menšia hodnota
  - Pravého – kde je väčšia hodnota
- Tá istá množina prvkov môže byť v binárnom vyhľadávacom strome rozlične umiestnená vo vrcholoch
  - Dôležité však je, aby bola splnená podmienka usporiadania hodnôt:
  - ľavý nasledovník každého vrcholu má menšiu hodnotu, pravý väčšiu



Binárny vyhľadávací strom (2)

- Rôzna štruktúra stromu, rovnaké prvky, ktorý je lepší?





STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Dátové štruktúry a algoritmy

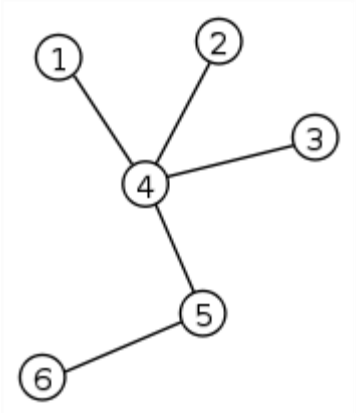
Binárne (vyhľadávacie) stromy

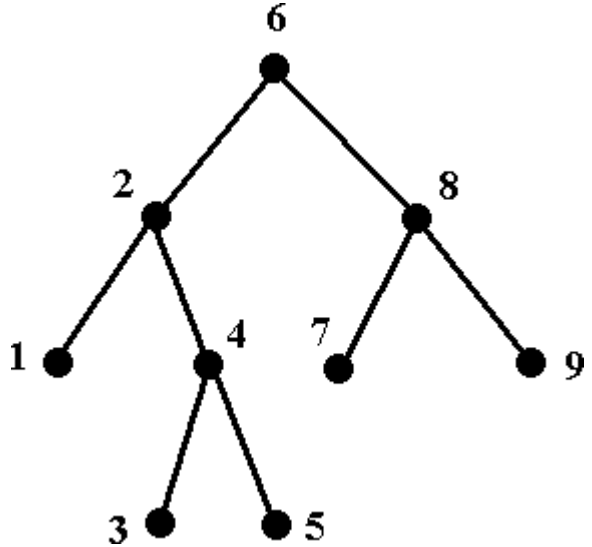
3. 3. 2021

letný semester 2020/2021

Strom – Definícia (teória grafov)

Strom – Súvislý neorientovaný graf bez cyklov



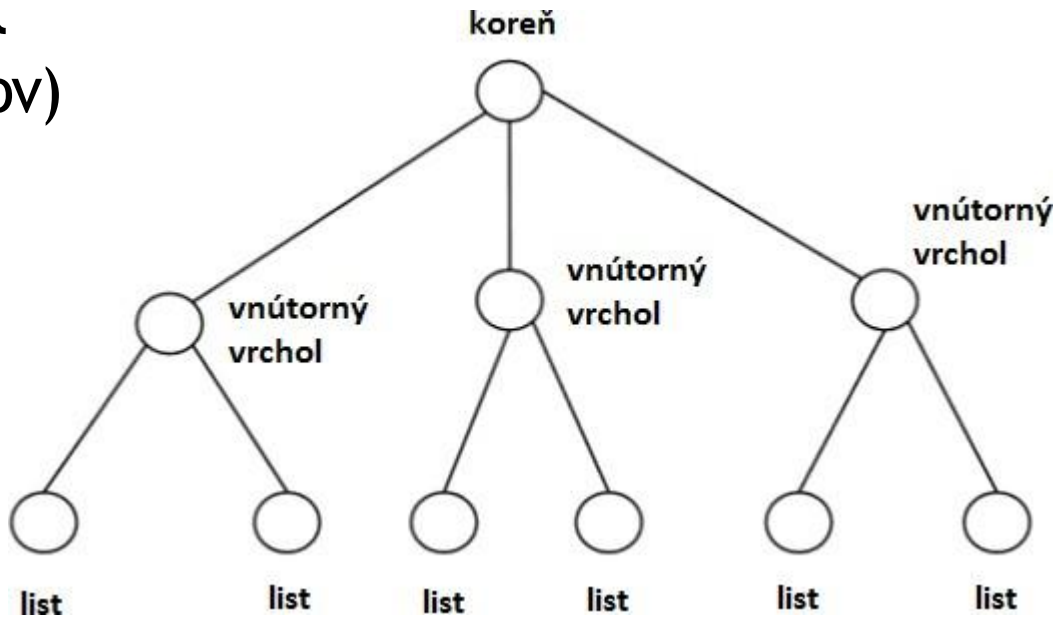


Graf G = (V, E)
V – množina vrcholov
E – množina hrán (dvojíc vrcholov)

Neorientovaný graf – hrany nemajú orientáciu (smer)
Súvislý graf – po hranách je možné prejsť z ľubovoľného vrcholu do ľubovoľného iného vrcholu v grafe
Cyklus – taký prechod po hranách, že začneme v nejakom vrchole, prejdeme po aspoň jednej hrane a skončíme v počiatočnom vrchole

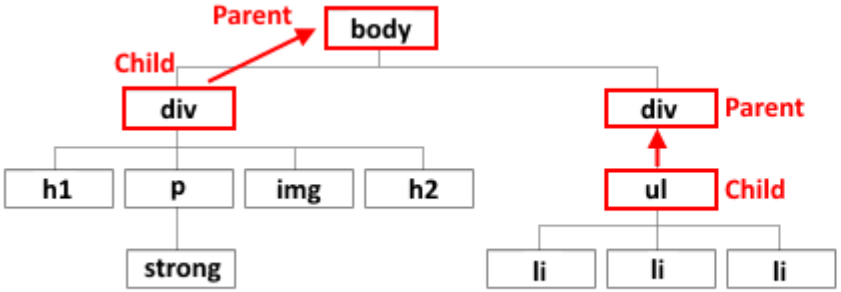
Zakoreňený strom

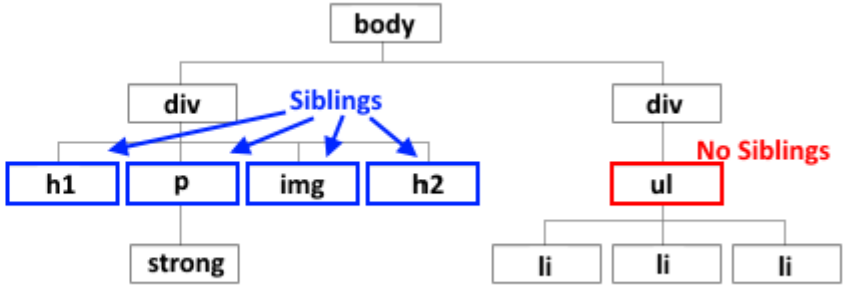
- Strom, v ktorom je význačný vrchol – **koreň** (root)
- Uvažujme vrchol **u**, ktorý leží na ceste z **koreňa** do **v**
  u nazývame **predchodca (ancestor) / rodič** **v**,
  resp. **v** je **nasledovník (descendant) / dieťa** **u**
- **List** – koncový vrchol
  (ktorý nemá nasledovníkov)
  - Ostatné vrcholy sú **vnútorné**
- Zvyčajne sa uvažuje orientácia hrán zhora dole (od koreňa k listom)



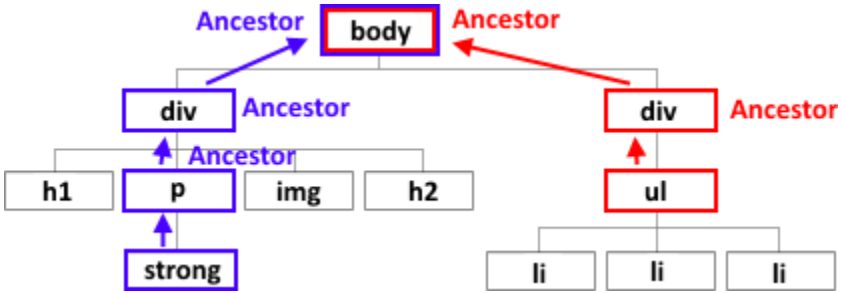
Zakoreňený strom (2) - príklad HTML

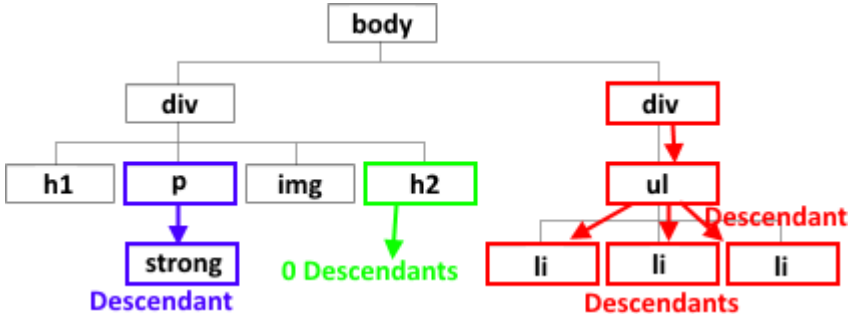
- **Rodič (parent)** – najbližší-priamy predchodca vrcholu
- **Dieťa / potomok (child)** – priamy nasledovník vrcholu
- **Súrodenci (siblings)** – vrcholy s rovnakým rodičom





- **Nasledovník / predchodca je aj nepriamy:**



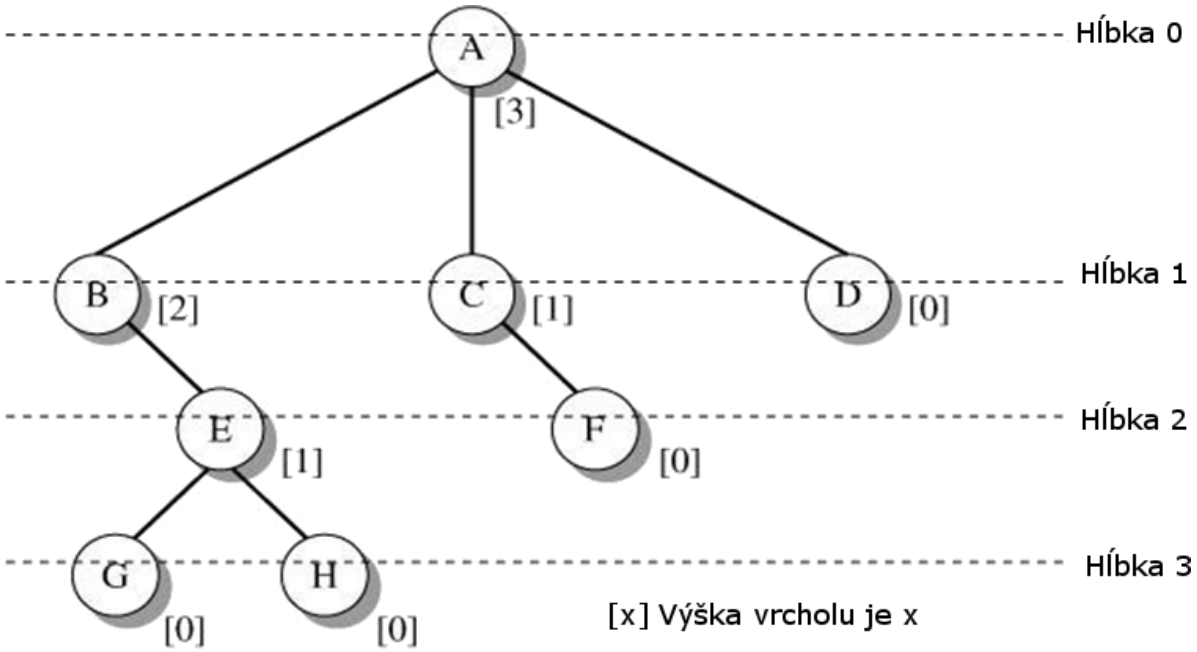


(Zakoreňený) strom – hĺbka, výška

Hĺbka vrcholu – počet hrán od koreňa stromu k danému vrcholu

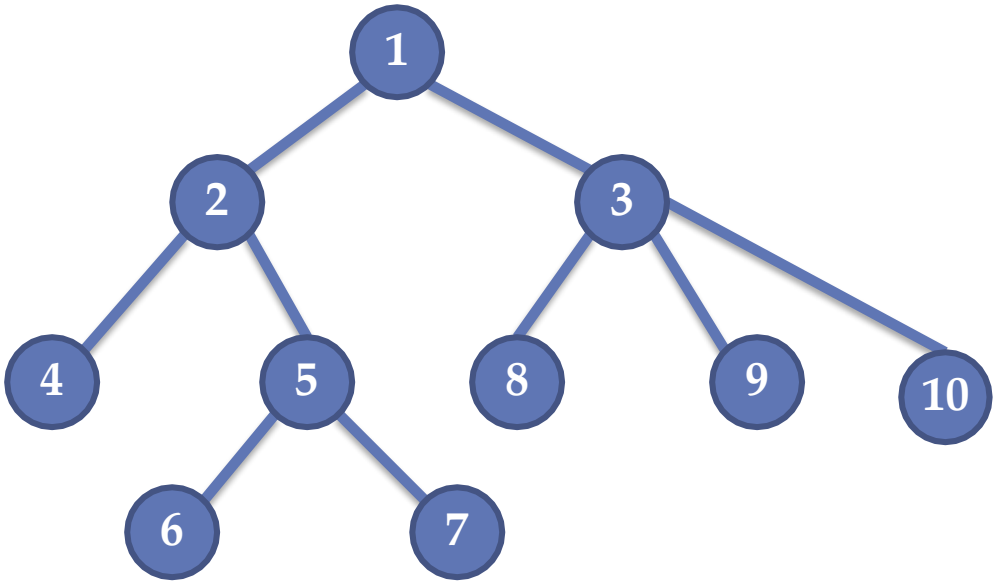
Výška vrcholu – dĺžka najdlhšej cesty z daného vrcholu k listu (koncovému vrcholu)

Výška stromu – výška jeho koreňa



Strom - Reprezentácia poľom

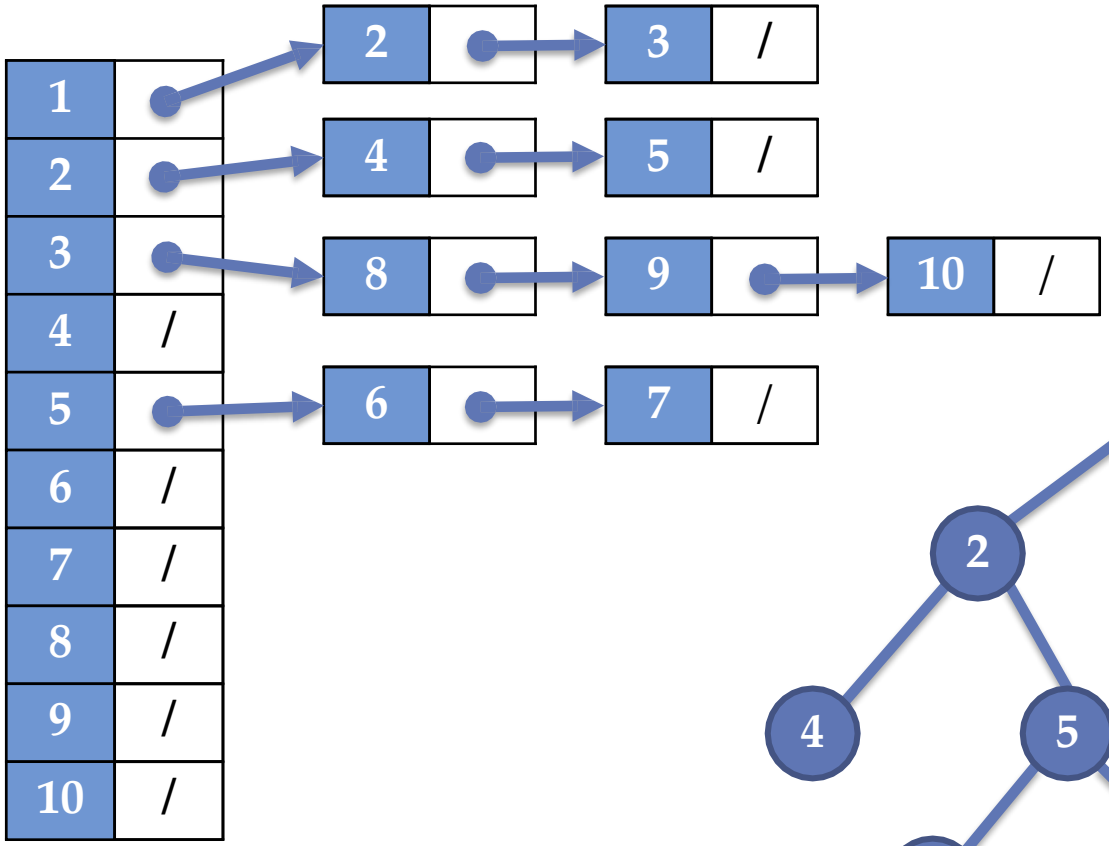
- Index do poľa = číslo vrcholu
- Hodnota prvku poľa = ukazovateľ na rodiča

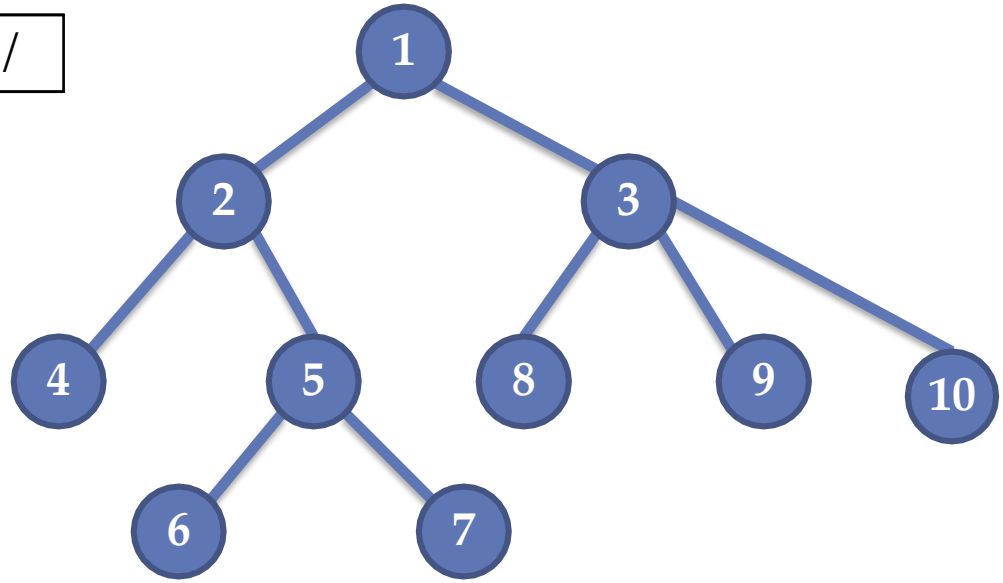


pole[i]	0	1	1	2	2	5	5	3	3	3
index	1	2	3	4	5	6	7	8	9	10

Strom - Reprezentácia spájaným zoznamom

- Každý vrchol má spájaný zoznam priamych nasledovníkov





Rôzne typy stromov (terminologicky)

- Strom (**príroda**)

- Strom (**teória grafov**)
  - Súvislý neorientovaný graf bez cyklov
  - Zakoreňený strom
- Strom (**abstraktná dátová štruktúra**)
  - Reprezentácia hierarchických vzťahov
- Strom (**teória množín**)
  - Čiastočne usporiadaná množina
    (keď nie je nutné, aby sa dali porovnať všetky dvojice prvkov)

Binárny strom

- Strom, v ktorom každý vrchol má najviac **dvoch priamych nasledovníkov** (potomkovia)
- Potomkovia sa označujú ako ĽAVÝ a PRAVÝ
- Rekurzívna definícia:
  - Jeden vrchol je binárny strom a súčasne koreň.
  - Ak u je vrchol a T_1 a T_2 sú stromy s koreňmi v_1 a v_2, tak usporiadaná trojica (T_1, u, T_2) je binárny strom, ak v_1 je **ľavý potomok** koreňa u a v_2 je jeho **pravý potomok**.